0w1_CTF_Writeup

原创

Little_Fall 于 2019-03-06 19:58:11 发布 507 收藏

分类专栏： 题解

本文链接：https://blog.csdn.net/m0_37809890/article/details/86584741

提交题解阶段写的题解，因为当时仍属于比赛就发了私密文章，直到3月6日才想起来取消解密。

0w1_CTF_Writeup by LittleFall

0. 前言

1. 只写正确的思路，错误的思路以及探索的过程因为太多就不写了orz；

0.1 SIGN-签到

1. 关注公众号`0w1网络安全俱乐部`；
2. 发送`0w1NB!`；
3. 收到flag：`flag{0w1_nb_!_hohoho}`

1. Reverse

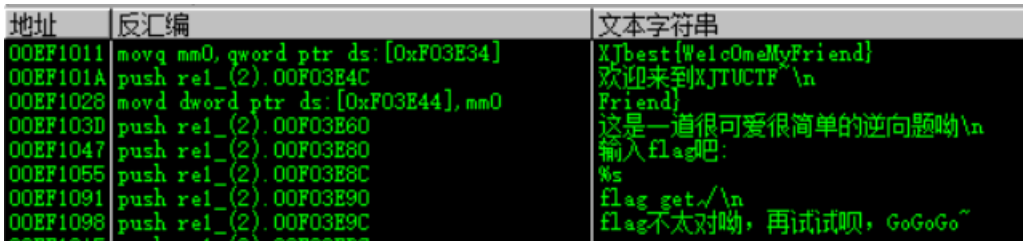
1.1 小试牛刀

工具

Ollydbg

流程

1. 使用OD打开程序；
2. 中文搜索引擎->智能搜索，列出全部字符串；
3. 其中`XJbest{We1c0meMyFriend}`长得很像flag；



4. 提交，成功。

1.2 有、意思

工具

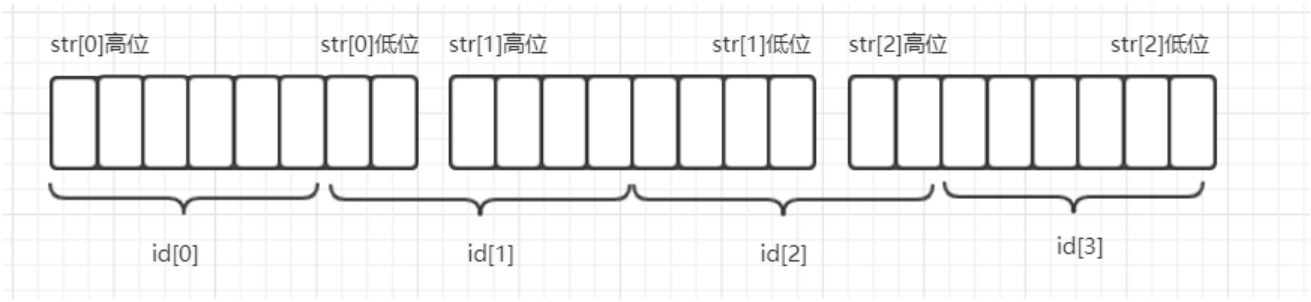
IDA
C++

流程

1. 使用IDA打开程序，F5大法列出反编译代码；
2. 读main函数，大致为：
    1. 读入字符串str；
    2. str经过某些操作（sub_411AB0函数）后得到一个字符串Dest；
    3. 处理Dest，方法是：

    ```
    for(j=0;j<strlen(Dest);j++)
     Dest[j]+=j;
    ```

    4. 拿Dest和字符串Str2比较，**如果在Dest的长度内都相等**，输出“rigth flag”(right?)，否则输出"wrong flag"。
3. 读sub_411AB0函数，函数接受字符串str和它的size（第三个指针参数没有用过，可以用来传递生成的Dest的长度），返回字符串Dest，内容大致为：
    1. 通过一些操作由str的size得到计算Dest的size，至少是$\lceil str.size/3 \rceil * 4$；
    2. **把str的每3个字符（字节）的24位拼到一起，然后每6位一组，作为4个下标从数组aAbcdefghijklmn里取值放到Dest里。** 当str位数不够时，下标默认为64；

    3. 给Dest补0（字符串结束符），返回Dest。
4. 读数组aAbcdefghijklmn，内容是`"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="`，第64位是`=`；
5. 读Str2，内容为：

    ```
    0x57, 0x46, 0x72, 0x6C, 0x5E, 0x5D, 0x54, 0x37,
    0x6D, 0x39, 0x58, 0x7A, 0x66, 0x64, 0x64, 0x88,
    0x73, 0x42, 0x4B, 0x56, 0x77
    ```

6. 坑点：Str2只有21字节，此时跑不出正确的结果。想到Dest的字节数是4的倍数，Str2后面必然还有东西，查看内存得到：

    ```
    01 00 00 00 57 46 72 6C  5E 5D 54 37 6D 39 58 7A
    66 64 64 88 73 42 4B 56  77 7F 58 50 00 00 00 00
    ```

    ```
    0x7F, 0x58, 0x50
    ```

    再后面全是0，应该不会有坑了；
7. 逆向：把str2逐字节减去0到23，然后在数组aAbcdefghijklmn中找到对应于这个字符的下标位置，一共24个。把这24个数字每4个拼接起来再分成3份，就得到了结果。
8. 运行如下代码，得到结果`XJbest{Cheers_Br0}`，提交成功。

代码

gen是生成函数（省略了不满3的倍数的情况），inv是逆向函数。

```
/* LittleFall : Hello! */
#include <bits/stdc++.h>
using namespace std;
using byte = unsigned char;
const char base[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
const int N = 24;
const byte target[N] = {
 0x57, 0x46, 0x72, 0x6C, 0x5E, 0x5D, 0x54, 0x37,
 0x6D, 0x39, 0x58, 0x7A, 0x66, 0x64, 0x64, 0x88,
 0x73, 0x42, 0x4B, 0x56, 0x77, 0x7F, 0x58, 0x50
};
string gen(string src)
{
    string des;
    for(int i = 0; src[i]; ++i)
    {
        char t[3] = {}, j;
        for(j = 0; j < 3 && src[i]; ++j)
            t[(int)j] = src[i++];
        des.push_back(base[t[0] >> 2]);
        des.push_back(base[((t[1] & 0xF0) >> 4) | (t[0] & 0x03) << 4]);
     des.push_back(base[((t[2] & 0xC0) >> 6) | (t[1] & 0x0F) << 4]);
     des.push_back(base[t[2] & 0x3F]);
    }
    for(int i = 0; des[i]; ++i)
        des[i] += i;
    return des;
}
string inv(const byte* des)
{
 byte id[N]; //index
 for(int i=0;i<N;++i)
 {
  for(int j=0;j<=64;++j) //search
   if(des[i]-i==base[j])
    id[i]=j, j=64;
 }
 string src;
 for(int i=0;i<N/4;++i)
 {
  int mask = 0;
  for(int j=0;j<4;++j)
   mask |= id[i*4+j] << 6*(3-j);
  src.push_back(mask>>16);
  src.push_back(byte(mask>>8));
  src.push_back(byte(mask));
 }
 return src;
}
int main(void)
{
 cout << inv(target) << endl;
    return 0;
}
```

1.3 CrackMe

工具

FileAnalysis
jadx-gui

流程

1. 用FileAnalysis分析CrackMe.jpg发现可能是APK；
2. 用jadx-gui打开，导出源代码；
3. 打开`sources/com/tay/crackme/MainActivity.java`，逻辑返回`this_stupid_challenge`；

```
27      public void check(String name, String pass) {
28          if (name.equals("0w1nb") && pass.equals("qweasdzxc")) {
29              Toast.makeText(this, "成功,this_stupid_challenge", 0).show();
30          } else {
31              Toast.makeText(this, "失败", 0).show();
32          }
33      }
```

4. 提交`flag{this_stupid_challenge}`，通过。

1.4 难以自拔

工具

FileAnalysis
IDA64
C++

流程

1. 用FileAnalysis分析re3，发现可能是linux下可执行的elf文件；
2. 使用IDA64打开，找到main，F5大法反编译；
3. 读main函数，大致为：
    1. 读入字符串str；
    2. 调用函数`sub_40111A(str)`；
    3. 调用函数`sub_401332(str)`；
    4. 释放str。
4. 读函数sub_40111A，函数接受一个参数str，可以认为无返回值，内容大致为：
    1. 定义了char变量s1，紧随其后定义了一堆char变量并赋值，且后文中出现了对s1的字符串操作，猜测可能是将s1当作字符串的首部。
    2. 遍历str，执行如下操作：

```
for(int i=0; str[i]; ++i)
{
 char c1 = char(str[i])*4;
 byte b1 = byte(str[i])>>6;
 byte b2 = (c1|b1)^i;
 str[i] = char(b2);
}
```

3. 如果str与s1相等，输出

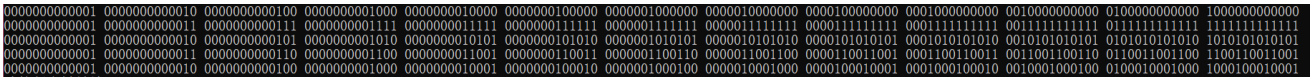
```
g00d j0b
but .....
```

5. 读函数sub_401332，函数接受一个参数str，可以认为无返回值，内容大致为：
   1. 定义char数组s[8]，逐个赋值，紧随其后定义三个__int64变量(64位)并赋值，由上一步的经验，猜测可能是把这些变量按字符接在了s数组后面，这样的话s共占
      8+ 3= 32 个字节；
      下面是我把大整数-1920493130842480203变成char型的过程，较为繁琐，也可以直接用men
   2. 对str数组进行如下操作：
      1. 先将大整数变为无符号类型：16526250942867071413；
      2. 变为16进制，0xe55909ac368e61b5；
      3. 每两位取值，注意：内存按小端存储，即应当按字节从低到高读。
         0xb5，0x61，0x8e，0x36，0xac，0x09，0x59，0xe5
      4. 大于0x80的，要将其减去0x100，因为char类型的取值是-0x80到0x7f：
      5. 变为10进制（可选）：
         -75，97，-114，54，-84，9，89，-27

```
for(int i=0; i<4; ++i)
    for(int j=1; j<strlen(s); ++j)
        str[j] = (str[j]&str[j-1]) & ~(str[j-1]&str[j]);
```

用逻辑计算或真值表可以发现，里面的操作与异或等价，相当于`str[j]^=str[j-1];`
连续执行四次，用二进制模拟之后发现，这个操作相当于每一个数异或了所有在自己之前且位置和自己相差为4的整倍的数，如下图所示：

64/8*



第12个数本来是1,0000,0000,0000，它与第8个数，第4个数，第0个数异或后变成了1,0001,0001,0001.

3. 如果str和s相等，输出`you g0t it!`。

6. 在读代码的过程中，手动复现了C++程序（见代码1），更好地理解了原程序，这里将两个子函数简记为deal1和deal2。注意：deal1的处理过程是独立的，但deal1会更改原字符串，导致影响deal2的结果；
7. 逆向deal1(记为inv1)：deal1的本质是对每个字符的独立处理，暴力检索即可，也可打表：运算更快，但编码更多；
8. 逆向deal2(记为inv2)：记操作之前的字符串叫做src，操作之后的字符串叫做des。deal2相当于`des[i]=str[i]^des[i-4]`，所以逆向时，`src[i]=des[i]^des[i-4]`；
9. 分别记deal1，deal2中的数组叫做s1,s2。对s1跑一遍inv1会得到一个假flag：`flag is: flag{7h15_15_4_f4k3_F14G_=3=_rua!}`,对s2跑一遍inv2，再跑一遍inv1会得到真正的flag：`flag{W0w_y0u_m4st3r_C_p1us_p1us}`，提交通过；（见代码2）
   实际上，在linux中运行原程序是无法得到第一个提示信息的，因为输入的假flag带有空格，而输入程序遇到空格就停止了，原程序复现版可以读入。

代码

1. 原程序复现版：

```cpp
/* LittleFall : Hello! */
#include<bits/stdc++.h>
using namespace std;
using byte = unsigned char;
void deal1(string &);
void deal2(string &);

int main(void)
{
    string str;
    cout << "input flag:";
    getline(cin,str);
    deal1(str); //flag is: flag{7h15_15_4_f4k3_F14G_=3=_rua!}
    deal2(str); //flag{W0w_y0u_m4st3r_C_p1us_p1us}
    return 0;
}
void deal2(string &str)
{
    const string s2 =   //little endian
    {
        -103,  -80, -121,  -98,  112,  -24,   65,   68,
           5,    4, -117, -102,  116,  -68,   85,   88,
         -75,   97, -114,   54,  -84,    9,   89,  -27,
          97,  -35,   62,   63,  -71,   21,  -19,  -43
    };
    for(int i=0; i<4; ++i)
        for(int j=1; j<(int)s2.size(); ++j)
            str[j] ^= str[j-1];
    if(str==s2)
        cout << "you g0t it!" << endl;
}
void deal1(string &str)
{
    const string s1 = {
        -103,-80,-121,-98,-124,-96,-53,-17,-120,-112,-69,-114,
        -111,-32,-46,-82,-44,-59,111,-41,-64,104,-58,106,-127,-55,
        -73,-41,97,4,-38,-49,61,92,-42,-17,-48,88,-17,-14,-83,-83,-33
    };

    for(int i=0; str[i]; ++i)
    {
        char c1 = char(str[i])*4;
        byte b1 = byte(str[i])>>6;
        byte b2 = (c1|b1)^i;
        str[i] = char(b2);
    }
    if(str==s1)
    {
        cout << "g00d j0b" << endl << "but ....." << endl;
        exit(0);
    }
}
```

2. 逆向程序：

```cpp
/* LittleFall : Hello! */
#include <bits/stdc++.h>
using namespace std;
using byte = unsigned char;
const string s1 =
{
    -103,-80,-121,-98,-124,-96,-53,-17,-120,-112,-69,-114,
    -111,-32,-46,-82,-44,-59,111,-41,-64,104,-58,106,-127,-55,
    -73,-41,97,4,-38,-49,61,92,-42,-17,-48,88,-17,-14,-83,-83,-33
};
const string s2 =   //little endian
{
    -103,  -80, -121,  -98,  112,  -24,   65,   68,
       5,    4, -117, -102,  116,  -68,   85,   88,
     -75,   97, -114,   54,  -84,    9,   89,  -27,
      97,  -35,   62,   63,  -71,   21,  -19,  -43
};
string inv1(const string &des)
{
    string src;
    for(int i=0;des[i];++i)
    {
        bool flag = 0;
        for(int ch=0;ch<256;++ch)
        {
            char v2 = char(ch)*4;
            byte v1 = (byte(ch)>>6|v2)^i;

            if(char(v1)==des[i])
            {
                src.push_back(ch);
                flag = 1;
                break;
            }
        }
        if(flag==0)
        {
            printf("?\n");
        }
    }
    return src;
}
string inv2(const string &des)
{
    string src = des;
    for(int i=0;des[i];++i)
        if(i>=4)
            src[i]^=des[i-4];
    return src;
}
int main(void)
{
 cout << inv1(s1) <<endl;
    cout << inv1(inv2(s2)) << endl;
    return 0;
}
```

2. Web

2.1 god is a flag

1. 打开网页；
2. 查看源代码，看到flag：`flag{love&peace}`；
3. 提交，成功。

2.2 快问我flag在哪

工具

BurpSuite

流程

1. 打开网页发现有一个超链接，指向`flag.html`；
2. 打开这个超链接，发现网页变成了`flaghere.html`，名字对不上；
3. 打开burpsuite抓包，发现flag.html的HTTPresponse头中有一项叫flag，值为`flag{more_love&peace}`
4. 提交，成功。

2.3 哈哈哈希

工具

BurpSuite
hashpump

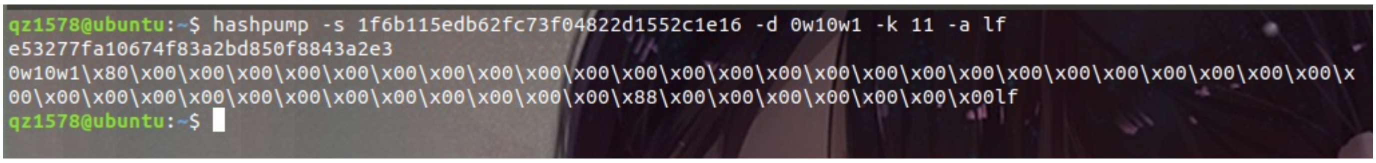
原理

哈希长度拓展攻击：

1. 简介：如果已知md5(secret)和secret的长度，则可以构造出一个attack_data和hash_value,使得md5(secret.attack_data)=hash_value，其中`.`表示php的字符串连接运算.
2. 应用场景：某些网页内部有一个secret，要求输入data和hash，当md5(secret.data)===hash时认为用户合法。
   当我们已经知道secret的长度，且有一组data和hash时，就可以构建出其它合法的data和hash，且旧data是新data的一个前缀。
3. 应用方式：linux下安装hashpump。
   输入`hashpump -s hash -d data -k secret_len -a add`
   其中-s之后跟原hash值（签名signature）；-d之后跟原data；-k之后跟secret的长度；-a之后跟需要追加的内容。
   会输出新的n_data和新的签名n_hash，此时md5(secret.n_data)===n_hash

流程

1. 打开网页，发现是一个登陆框，尝试登陆无果，查看源码无果，下载图片破解无果，搜索路径无果（有一个log但是并不会用）；
2. 登陆，用burpsuite抓包，发现cookie中有一项source=0比较奇怪，将其改成1之后发现网页返回了源码。
3. 分析逻辑如下：
    1. 有名为flag的变量保存flag，secret_key保存密文，secret_key的长度为17。在head中添加了hash_key;
    2. if嵌套，条件1：cookie有非空的getflag项；条件2：用户名是0w1且密码不是0w1；条件3：cookie的getflag项和md5(secret_key.username.password)全等。当条件123全部满足时，以html注释形式返回flag；
    3. 在cookie中设置了名为sample-hash的一项，值为md5(secret_key."0w1"."0w1");
    4. 当cookie的source项不为空时，返回<source_code>
4. 使用hashpump进行哈希长度拓展攻击：sample-hash（hash_key？）的值作为原签名，"0w10w1"作为原数据，11作为secret长度，"lf"作为添加的值。
   返回了新的签名和数据，如图所示。



```
qz1578@ubuntu:~$ hashpump -s 1f6b115edb62fc73f04822d1552c1e16 -d 0w10w1 -k 11 -a lf
e53277fa10674f83a2bd850f8843a2e3
0w10w1\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x88\x00\x00\x00\x00\x00\x00\x00lf
qz1578@ubuntu:~$
```

5. 用户名仍为0w1，密码变为`0w1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%88%00%00%00%00%00%00%00lf`（将`\x`换成了`%`），在cookie中添加`getflag=e53277fa10674f83a2bd850f8843a2e3`，提交。（因为各种bug失败了若干次，每次满怀希望看到"滚犊子"简直心态爆炸）
6. 网页返回陈独秀先生，在源码中看到flag：`flag{h4sh_1s_1nteresting}`，提交，成功。

2.4 逃亡计划

工具

BurpSuite
在线工具：16进制与文本转换
7-zip
python

流程

1. 打开网页，发现是个游戏，玩了一会计算每次的期望收益=
   $-2+(5e6+9*2e5+9*1800+9*300+9*20+9*5)/1e7=-1.2409035$
2. 氪分买hint，提示是robot，打开robots.txt->robots.php，得到一个二进制串，转成文本串保存为fangzhuaji.zip，解压得到caipiao6文件夹，是程序的源码。
3. 核心逻辑是随机产生7个数，与输入的数字使用`==`比较，如下：

```
for($i=0; $i<7; $i++){
 if($numbers[$i] == $win_numbers[$i]){
  $same_count++;
 }
}
```

   numbers使用json字符串传值，number初始是一个字符串，如下：

```
data: JSON.stringify({ action: "buy", numbers: numbers })
```

4. 在php中，0==false，(1->9)==true，true可以匹配除0之外的所有值。抓包后将numbers改成`"numbers":[true,true,true,true,true,true,true]`，现在获奖的期望就变成了
   $(9*5e6+9*2e5+9*1800+9*300+9*20+9*5)/1e7=2402124.1471485$
5. 钱够1e7后买票，提示需要计算RSA中`p=223398606,q=221141,e=23`时的d，而$d*$（我感觉还要高一些，可能

代码

python代码

```
p = 223398606
q = 221141
e = 23

def exeuclid(a, b):
 if b == 0:
  return 1, 0
 else:
  k = a // b
  x1, y1 = exeuclid(b, a % b)
  x, y = y1, x1 - k * y1
 return x, y

def inv(a, n):
 x = exeuclid(e,n)[0]
 x = (x%n+n)%n
 return x

d = inv(e,(p-1)*(q-1));
print(d);
```

3. misc

3.1 再也不用担心我1T的资源被妈妈发现啦[滑稽]

工具

7-zip
winhex

流程

1. 下载final.jpg，猜测可能有文件藏在里面，使用7-zip打开得到interim2.jpg；
2. 不能再做为压缩文件打开，使用winhex打开，发现最后（jepg文件结尾FFD9之后）还有一段二进制码，翻译为：flag=0w1_{D_hidden_picture}；

```
000A29A0  81 E0 FB DF 7B 1C 7F FF  D9 66 6C 61 67 3D 30 77  àûß{  ÿÙflag=0w
000A29B0  31 5F 7B 44 5F 68 69 64  64 65 6E 5F 70 69 63 74  1_{D_hidden_pict
000A29C0  75 72 65 7D 0A                                    ure}
```

3. 提交，成功。

3.2 找到女神的位置

工具

7-zip
百度识图

流程

1. 下载zip文件，使用7-zip解压得到若干图片；
2. 一头雾水，拿去百度识图，找到一篇帖子：谁说IT男没有春天！！黑客思维轻松追女神！！！；
3. 拜读完之后表示十分佩服，然后提交了flag：`flag{青田皇家风尚宾馆}`，通过。

3.3 和咩咩的PY交♂易

参考资料

PNG文件格式总结

工具

FileAnalysis
Wireshark
7-zip
winhex
python

流程

1. 下载文件pcapng，拿fileanalysis分析的得到可能是pcapng文件；
2. 使用wireshark将其打开，文件->导出对象->HTTP，得到三个文件：一个html，一个json，一个未知格式的文件multi；
3. html和json里的信息表面同目录下应该有一个zip文件，于是拿7-zip打开multi，里面有无格式的flag1和flag2；
4. 使用fileanalysis分析flag1，100%是png，打开后发现是一张下面全黑的图片。而flag是未知文件；
5. 使用winhex打开flag1，发现找不到文件结尾（IEND，通常为0000 0000 4945 4E44 AE42 6082）。而在flag2中发现了png文件结尾；
6. 把flag1和flag2拼接起来，得到全图如下：

7. 猜测可能用其它方式隐藏数据，用winhex打开这张图，运算发现，文件头部分的CRC校验不通过；
8. 暴力检索可能出现的修改方式（修改宽度或高度），发现可以将高度修改为0x2a7从而通过校验(见代码)；
9. 修改后打开图片，得到flag：`flag=0w1_{D_Bug_2333}`，提交后通过。

flag=0w1_{D_Bug_2333}

代码

```
import os
import binascii
import struct

misc = bytes([
   0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a, # PNG
   0x00, 0x00, 0x00, 0x0d, 0x49, 0x48, 0x44, 0x52, # IHDR
   0x00, 0x00, 0x02, 0x7d, 0x00, 0x00, 0x01, 0xfd, # 宽度, 高度
   0x08, 0x02, 0x00, 0x00, 0x00, 0x7c, 0xb4, 0x6c, # CRC(从0x7c开始)
   0xd3
  ])

print("width:")
for i in range(1024):
    data = misc[12:20] + struct.pack('>i',i) + misc[24:29]
    crc32 = binascii.crc32(data) & 0xffffffff
    if crc32 == 0x7cb46cd3:
        print(hex(i));

print("height:")
for i in range(1024):
    data = misc[12:16] + struct.pack('>i',i) + misc[20:29]
    crc32 = binascii.crc32(data) & 0xffffffff
    if crc32 == 0x7cb46cd3:
        print(hex(i));
```

4. Crypto

4.1 easyRSA

参考资料

RSA加密解密原理深度剖析（附CTF中RSA题型实战分析）

工具

在线工具：大数分解
在线工具：16进制与文本转换
python

原理

RSA是一种非对称加密算法，名字来源于三个提出人的首字母简写。

1. RSA密钥的产生
    1. 选择两个大质数p和q，令$n=p*q$，则n的欧拉函数值$\phi(n)=(p-1)*(q-1)$；
    2. 选择一个整数e，满足$1<e<\phi(n)$且$gcd(e,\phi(n))=1$；
    3. 由$d*e\equiv 1 \mod \phi(n)$计算出d；
    4. $\{e,n\}$为公钥，$\{d,n\}$为私钥。
2. RSA加密解密操作
    1. 设明文为m，密文为c;
    2. 加密：$c\equiv m \bmod n$；
    3. 解密：$m\equiv c \bmod n$。

操作流程

1. 下载文件，得到n，c，e；
2. 分解n得到p和q；
3. 计算phi = (p-1)*(q-1)；
4. 计算d，d为e模phi的逆元，由扩展欧几里得算法计算,代码如下；
5. 计算m，m=c^d mod n;
6. 将m换成16进制，在线转换字符串后得到答案。

代码

python代码

```
n = 96680893262749719063585923605496034909946397522735056426538437328033669985338725407066288126593756516300075860615430875794403057183717504851457447306140156633083633464717665528261926859256017272652664307449953412987821740904604553365689705011743849635723157599918552767507100280395180063522002901593200746511781873994890375020083085611566869100770683695224484271941945294625927525177329833816238993051883827270490888701647400705139719458839603911121670886621461477962756695933517067605502585093263105364157656616569412142054608104328580678323929679979565519112196637759017578061894491053281698814305675705405267996853890146089357120490439497571408105545524052389565330531551774572933411454975669533417114287608047710507040954477798160215276215461073854016379616429522281024330905150309086667463444035922619253072463547705157651517986446117491197566716259728676907938066078264795294480859631047697393915618747207695293572824906113748188758910397359108287298864195827028516965080379239555636330405629007780145398082209758357430968293569726020486275692386555639768669685423956454140718570994010780653677316026376448344385942572695314296414821620996843758704461761351805877928716785334936453371645867606673421687756618151460769388237553300" 
p = 31093551302922880999883020803665536616272147022877428745314830867519351013248914244880101094365815998050115415308439610066700139164376274980650005150267949853671653233491784289493988946869396093730966325659249796545878080119206283512342980854475734097108975670778836003822789405498941374798016753689377992355122774401780930185598458240894362246194248623911382284169677595864501475308194644140602272961699230282993020507668939980205079239221924230430230318076991507619960330144745307022538024878444458717587446601559546292026245318907293584609320115374632235270795633933755350928537598242214216674496409625928797450473
q = 31093551302922880999883020803665536616272147022877428745314830867519351013248914244880101094365815998050115415308439610066700139164376274980650005150267949853671653233491784289493988946869396093730966325659249796545878080119206283512342980854475734097108975670778836003822789405498941374798016753689377992355122774401780930185598458240894362246194248623911382284169677595864501475308194644140602272961699230282993020507668939980205079239221924230430230318076991507619960330144745307022538024878444458717587446601559546292026245318907293584609320115374632235270795633933755350928537598242214216674496409625928997877221
c = 16850291008885829563431507024437740955656763713973630808218636900322777193640732178355779562427916216230520043644690397638594867789766546629085276987756216748714238530802734163981640105508182049700201890889620286034239102908258162198730553309738665218384965706595206243398838764099038362326440552514400350028653126267431590053700184504322536314835976677103389968011107618167279707741058474750958193204554080177773854887274759789996536695082750552943248377982115815292889994783719639155566616548644187818328800875356110899571596192047292784487756985594050514884835309988781137228304278079266793242411411822389035676820424101453455518894421588951578757989909037151057826820838864616613070635834476961688286871269561479558864933838055135576041790290509816787550549456078663531957936541084039392427238616519191523699239040029668739948118263910803181462604169784993771825406844097903572574908162031384993696344908975532277635635539812468916776134463901344778321431752489921616416980111959687921052018479760823227866233902424702267406858222181402631820242262286921593805576615916330720919450773341919878602624483851235994596472285621373691780690728044980494631362338563378173859779901455710422317953329955239881748954328198728321700296908480
e = 65537

def exeuclid(a, b):
 if b == 0:
  return 1, 0
 else:
  k = a // b
  x1, y1 = exeuclid(b, a % b)
  x, y = y1, x1 - k * y1
 return x, y

def inv(a, n):
 x = exeuclid(e,n)[0]
 x = (x%n+n)%n
 return x

d = inv(e,(p-1)*(q-1))

m = pow(c,d,n)
# print(pow(m,e,n)==c)
print(hex(m))
```

4.2 未知的短信

工具

百度一下orz

流程

1. 数据33 53 21 41 43 74 74 43 61 71 53 32；
2. 打开手机9键键盘，每两个数字分为一组，第一个表示按哪个键，第二个表示取这个键上的第几个字母；

3. 得到F L A G I S S M L E
4. 提交flag{flagissimple}，通过