ciscn2021 ctf线上赛baby.bc wp（#超详细，带逆向新手走过一个又一个小坑）

原创

漫小牛 于 2021-05-17 23:09:54 发布 1007 收藏 10

分类专栏： CTF Writeup 文章标签： 安全

文章目录

引言

这是ciscn2021中的一道Reverse题，将附件进行下载，名称为baby.bc，文件名为baby，表示这道题的内容与baby有关（baby一般是刚出生的小孩子，至少是5岁以下）。扩展名为bc，有可能不是二进制文件。最初下发这道题时，给的是一个存在问题的baby.bc文件，直到当天19点多在qq群中群发消息后，更新了baby.bc。由于没有关注qq群发消息，比赛时并没有解出这道题的flag，为了搞清楚这道题出问题的地方，赛后再向主办方索取更新后的baby.bc，也未能获得。我们权且用旧的baby.bc文件来写writeup，但这丝毫不影响相关知识点的讲解和flag的求解。

题目附件：
链接：https://pan.baidu.com/s/13TheaHB7XcbGnBp-M1N5Rg
提取码：k4pm

第一步、分析文件类型

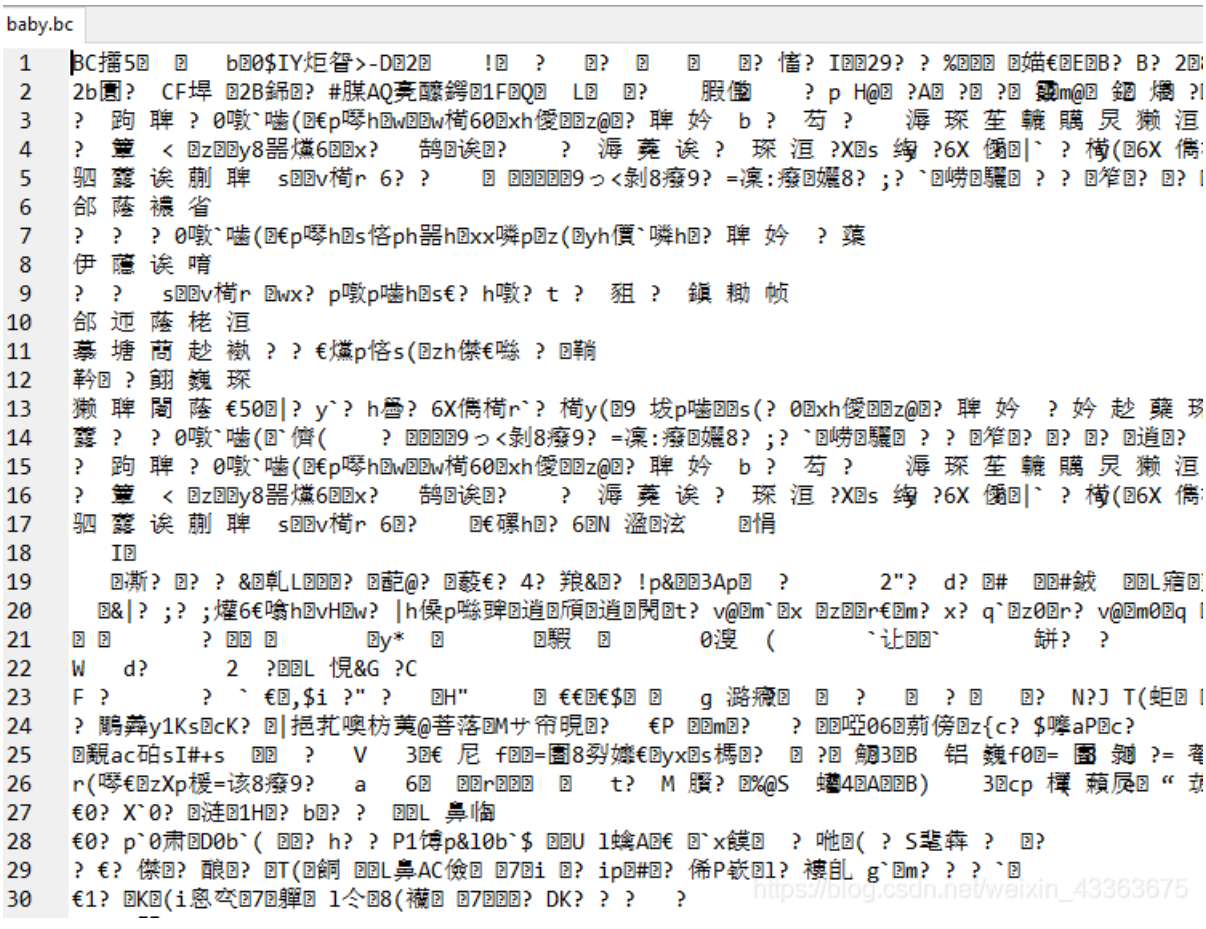
1.可执行文件判断

使用Exeinfo PE查看文件的类型，该工具主要用来查看文件是否为可执行文件，是否为动态链接库，是否有壳以及运行的操作系统等等。具体情况见下图：

结果显示，该文件并非可执行文件。

2.继续分析文件类型

既然不是可执行文件，那么是否为源文件呢？用任意的文本编辑器打开该文件，见下图：

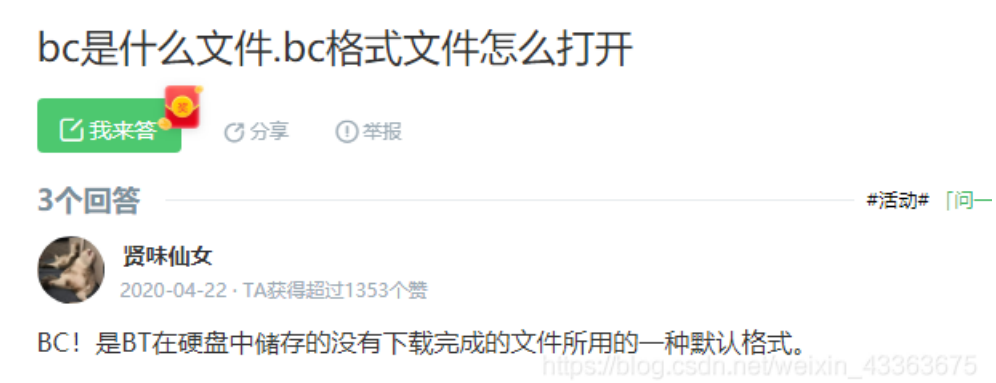


图中是大量的乱码，可知该文件也不是某种编程语言的源文件。

既不是二进制文件，也不是源文件，接下来应该如何继续分析该文件的文件类型呢？到这里也是很多新手所面临的**第一个坑：接下来如何分析文件类型？**

3.解决第一个坑-前奏

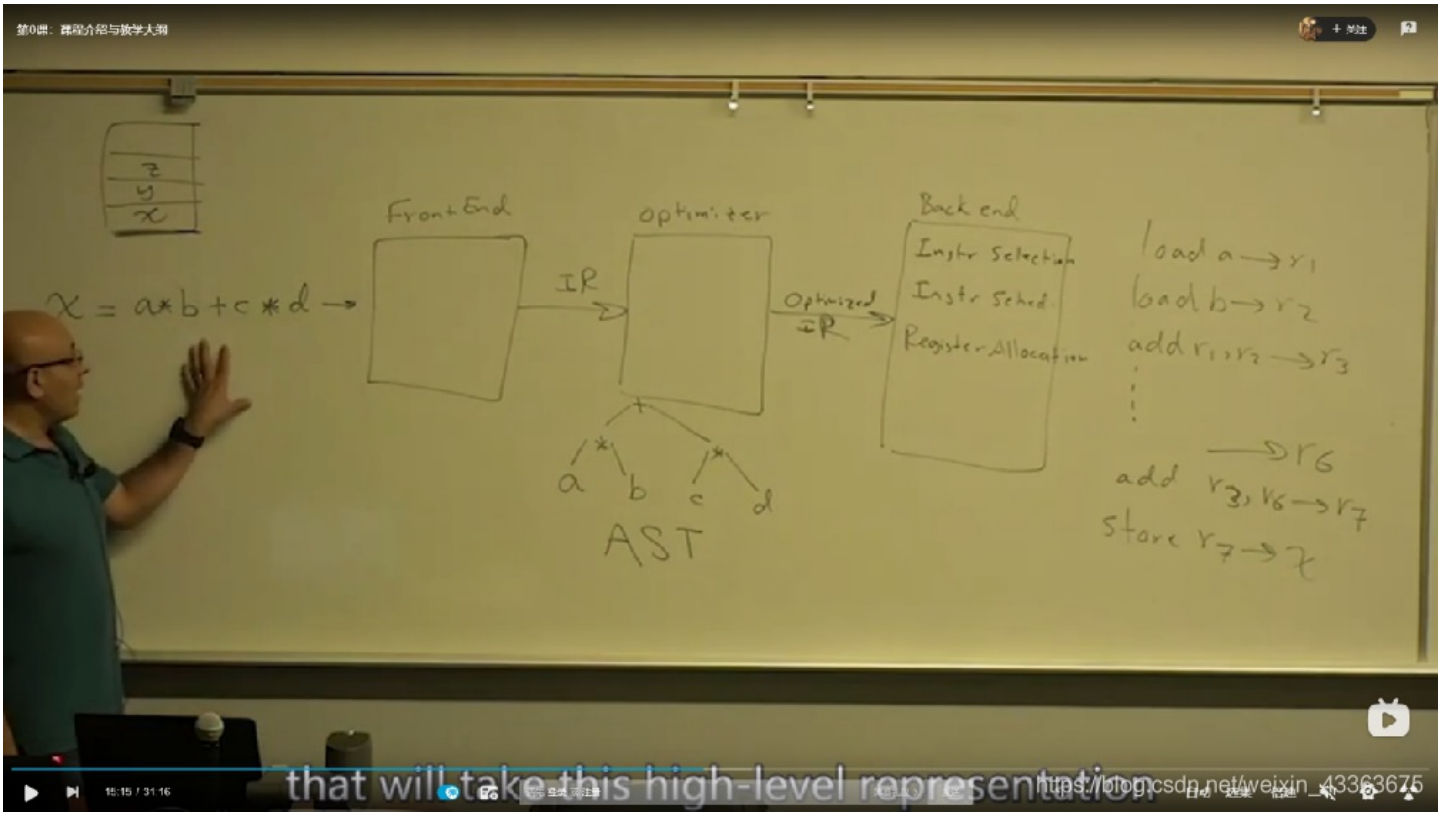
最通常的思路是拿百度看看以bc为扩展名的文件是什么文件？经过百度，我们可能会得到如下一些搜索结果：



既然百度到了这些可能的文件格式，那么，我们是否要沿着这种BT文件或Adobe Bridge的缓存文件进行分析呢？若打算对这些文件格式进行分析，就需要查找相关文档并对这些文件进行解析，并从文件的解析中找到flag。如果我们沿着这条路摸着黑走下去，将会渐渐的偏离主线和中心，那么除了这种思路，还有其他的思路分支吗？同时，带来的一个问题是，我们如何确定是选择亦或是抛弃当前的文件解析思路？我们先来解决第二个问题，请看下一小节。

4.reverse题的输入可能是什么？

为了解决第二个问题，即“我们如何确定是选择亦或是抛弃当前的文件解析思路？”，该问题的答案取决于文件是否符合reverse题型的输入，那么这个问题就等价的转化为“reverse题的输入可能是什么？”，首先想到的是二进制可执行文件（含动态链接库.dll和.so等），其次想到的是源文件，如.c。那么，除了以上两种最为典型的reverse输入外，还有其他文件格式吗？为了解决这个问题，需要简要插播一段儿编译器从源文件到可执行文件的过程，见下图：



为了省事儿，懒得重新画图了，这是我最近看的加州公立大学的一节老外的课，为什么搞ctf要研究compiler，尤其是compiler’s program analysis，including dataflow analysis，cfg analysis & pointer analysis（tail Front end），of course including optimizer。按ctf的行话来说，是reverse爸爸、pwn爷爷、misc儿子和web孙子，如果你懂上面的英文对应的关键技术，可以更为深入的搞自动化漏洞挖掘、污点分析、符号执行、程序切片等等，更为精确的程序分析有利于中后端优化和更为深入的漏洞解析，compiler（program analysis&optimizer）+pwn/reverse is definitely grandfather’s dad。
以上这幅图，给出了从源代码的x=a*b+c*d到一些load、add、store这些汇编的过程，很显然，除了我们刚才讨论的最前端的源码和接近可执行的汇编，还有很多IR，即中间表示，这些中间表示有可能是形如汇编的扁平化结构，也有可能是形如AST的结构，即抽象语法树。在源代码和可执行之间的IR可能存在很多级，如Open64中的IR就包括veri high whirl，high whirl，middle whirl，low whirl和very low whirl，而gcc则包括 GENERIC、GIMPLE和RTL，这些ir除了保存在编译器执行过程中的内存外，还可保存为中间文件。这里得出的结论是，compier的中间文件也可能是revserse题目的输入。那么现在我们再来回答“reverse题型的输入可能是什么？”这个问题，答案是：从源文件到二进制文件编译过程中所有可能的文件，如源文件.c、任何形式的ir、汇编.s、目标文件.obj、可执行文件elf等等。

5.继续解决第一个坑

沿着4的分析继续走，显然BT和ADOBE缓存这种文件跟compiler的source到binary之间的文件没有一毛钱的关系。那么，是不是到这里，我们就要调到坑里了（╥_╥)？到这里，该搬出来大家都知道却又用的不多的工具了，linux下的file。虽然，各种windows下的PE工具用的不亦乐乎，但有必要在解题过程中采用**“多平台、多工具、多版本”**的思维，权且称之为**互补性原理：不同平台下相同工具，具有互补性；不同平台下功能相似的工具，具有互补性；同一工具的不同版本，具有互补性。**

仅仅是使用了一个file命令，老鸟看了，显然会鄙视这种冠冕堂皇的包装，不扯了，直接给出跨过第一个坑的结果：

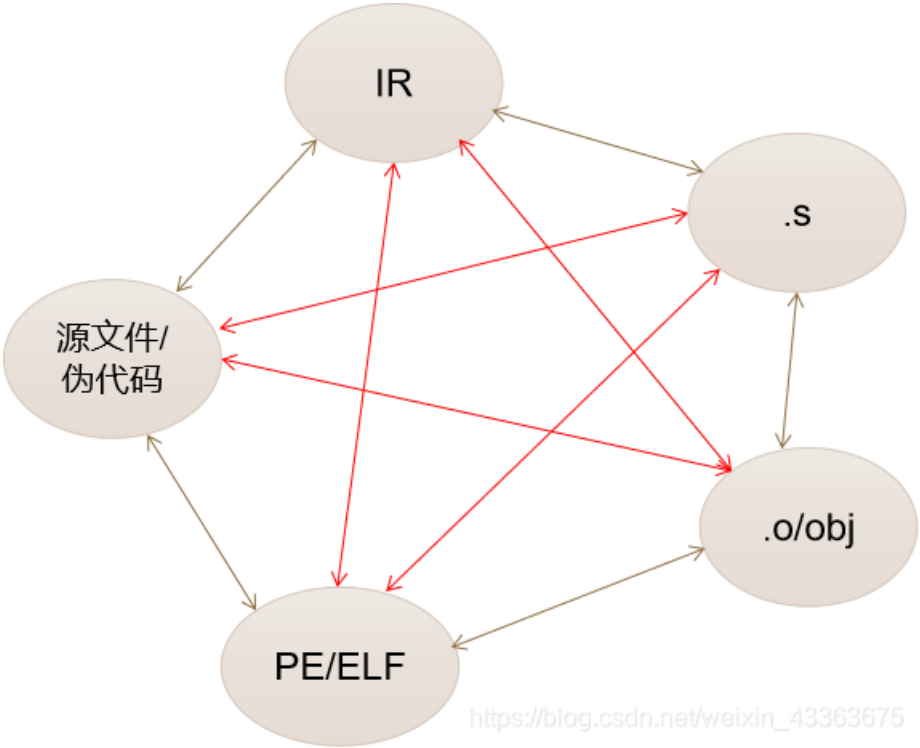
```
┌──(root💀kali)-[~/桌面]
└─# file baby.bc
baby.bc: LLVM IR bitcode
```

LLVM IR bitcode，某种形式的LLVM中间表示（如果不知道LLVM是编译器的就自己百度吧）。到这里，我们从baby.bc积累了第一条经验，**经验一：reverse题目最可能的形式是二进制文件，其次是源文件和某种形式的IR。**

第二步、明确分析的对象

1.确定“开始干”的优先级（二坑）

我们拿到了IR，面临了**第二个坑：怎么分析这个IR？**。在真正“开始干”之前，并不是直接开始分析我们拿到的文件，很多新手拿到IR文件后就直接解析IR格式，或者解析IR可以直接转换的格式，这种思路并不可取。那么，最关键的是要明确分析的对象是啥？这就看看IR可以转换为什么，以及这些转换后的东西又可以转换为什么吧。直接给图：



虽然画了个比较丑陋的图，但大家应该看的明白，大佬可以略过不要盯着细节，这幅图要告诉新手的是这一堆文件格式是可以相互转化的，只要拿到任何一个文件，就可以直接或间接的转换并得到其他所有的文件，换句话说，是一种“只拿了一把刀，就好像有了十八班兵器”的感觉，既然有个十八班兵器，问题就转化为这一节的标题，即“确定“开始干”的优先级？”。到这里，我们得到了第二条经验，**经验二：源代码/伪代码具有最高的优先级，其次是汇编（PE、ELF、.o、.obj的机器码和汇编一一对应，不考虑），最后是IR。**如果你是一个编译器的老程序员，也不建议直接分析各种层次的IR，IR包含了大量的中间信息，简洁度不但不如源文件，连汇编也比不了。

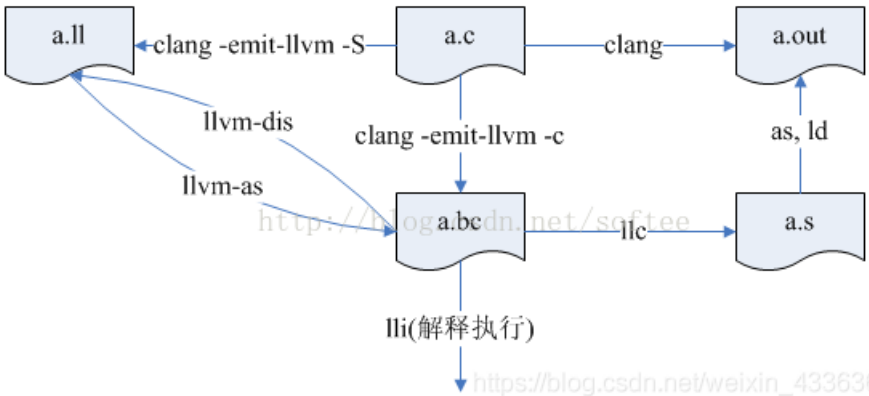
既然“开始干”的对象是源文件/伪代码，那么问题就转换为如何把LLVM的.bc转化为源码或伪代码。接着来看下一小节：

2.LLVM环境及代码转换

LLVM的安装可以参考https://zhuanlan.zhihu.com/p/102028114，说实话，这也是我第一次安装LLVM，安装方式有多种，如果不需要修改LLVM，可直接按编译好的二进制安装，或者使用包管理器进行安装，如进行源代码安装（wget http://release.llvm.org/9.0.0/llvm-9.0.0.src.tar.xz），需要cmake和make install，需要1-2个小时（呵呵，老鸟又开始鄙视了，工具都没准备好，还想着在24小时内编译LLVM环境）。
安装好后，就可以代码转换了，运行命令前，需要在命令行配置PATH路径，如export PATH="$PATH:/usr/local/llvm-9.0.0/bin"。

- 如果你想得到.s，可以执行命令：./llc baby.bc -o baby.s
- 如果你想得到.ll，可以执行命令：llvm-dis baby.bc -o baby.ll
- 如果你想得到可执行文件，可以执行命令：as baby.s

很多编译器可以从中间表示直接生成高级语言的c或fortran，如open64的whirl2c和whirl2f，但llvm是否支持从.ll文件或.bc文件到高级的.c呢？不多说，继续上图：



为了大家更清楚的看图，再归纳一下：

- a.c，源代码。
- a.bc，llvm的字节码的二进制形式。
- a.ll，llvm字节码的文本形式。
- a.s，机器汇编码表示的汇编文件。
- a.out，可执行的二进制文件。

既然从bc和ll无法到c，那么只能先生成可执行，再从ida到高级c的伪代码了。
顺便看看不太好看懂的.ll字节码，除了常见搞编译的看着顺眼外，一般人可能更喜欢看.s和.c。

```
; ModuleID = 'baby.bc'
source_filename = "baby.bc"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@map = global [5 x [5 x i8]] [[5 x i8] zeroinitializer, [5 x i8] zeroinitializer, [5 x i8] c"\00\00\04\00\00"
oinitializer], align 16
@row = global [5 x [4 x i8]] [[4 x i8] c"\00\00\00\01", [4 x i8] c"\01\00\00\00", [4 x i8] c"\02\00\00\01", [
01\00"], align 16
@col = global [4 x [5 x i8]] [[5 x i8] c"\00\00\02\00\02", [5 x i8] zeroinitializer, [5 x i8] c"\00\00\00\01"
@.str = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@input = common global [100 x i8] zeroinitializer, align 16
@.str.1 = private unnamed_addr constant [15 x i8] c"CISCN{MD5(%s)}\00", align 1
@input_unhex = common global [100 x i8] zeroinitializer, align 16
@steps = common global [100 x i8] zeroinitializer, align 16

; Function Attrs: norecurse nounwind readonly uwtable
define zeroext i1 @docheck() #0 {
  %vis = alloca [6 x i8], align 1
  %vis5 = alloca [6 x i8], align 1
  %1 = getelementptr inbounds [6 x i8], [6 x i8]* %vis, i64 0, i64 0
  br label %3

.thread54.preheader:                              ; preds = %.thread107
  %2 = getelementptr inbounds [6 x i8], [6 x i8]* %vis5, i64 0, i64 0
  br label %18

3:                                                ; preds = %0, %.thread107
  %indvars.iv102 = phi i64 [ 0, %0 ], [ %indvars.iv.next103, %.thread107 ]
  call void @llvm.lifetime.start.p0i8(i64 6, i8* %1) #8
  call void @llvm.memset.p0i8.i64(i8* align 1 %1, i8 0, i64 6, i1 false)
  %4 = getelementptr inbounds [5 x [5 x i8]], [5 x [5 x i8]]* @map, i64 0, i64 %indvars.iv102, i64 0
  %5 = load i8, i8* %4, align 1, !tbaa !1
  %6 = zext i8 %5 to i64
  %7 = getelementptr inbounds [6 x i8], [6 x i8]* %vis, i64 0, i64 %6
  %8 = load i8, i8* %7, align 1, !tbaa !4, !range !6
  %9 = icmp eq i8 %8, 0
  br i1 %9, label %10, label %17
```


第三步、IDA静态分析

1.main函数

见注释：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // rax
  unsigned __int64 v4; // rcx
  unsigned __int8 v5; // dl

  _isoc99_scanf(&unk_595, input, envp);
  if ( (unsigned int)strlen(input) == 25 )      // flag长度为25
  {
    if ( input[0] )
    {
      if ( (unsigned __int8)(input[0] - 48) > 5u )// 输入的第1个字符的ascii码不大于字符'5'
        return 0;
      v3 = strlen(input);
      v4 = 1LL;
      while ( v4 < v3 )
      {
        v5 = input[v4++] - 48;
        if ( v5 > 5u )                          // 输入的剩余24个字符的ascii码不大于字符'5'
          return 0;
      }
    }
    if ( (unsigned __int8)fill_number(input) && docheck() )// 需要两个函数返回为真
      printf("CISCN{MD5(%s)}", input);
  }
  return 0;
}
```

直接看注释，不解释。

2.fill_number函数

见注释：

```
char __fastcall fill_number(__int64 a1)
{
  char *v1; // rdi
  __int64 i; // rax
  char v3; // cl
  char v4; // cl
  char v5; // cl
  char v6; // cl
  char v7; // cl

  v1 = (char *)(a1 + 4);                        // 向后偏移四个字符
  for ( i = 0LL; i < 5; ++i )
  {
    v3 = *(v1 - 4);                             // 当前行第一个字符
    if ( map[5 * i] )                           // 既然名称是map，很可能是个二维数组
    {                                           // 分析每一个if/else，map不为0时，当前的输入字符必须是'0'；如果map为0
，当前的map字符为输入字符的ascii码减去'0'的ascii码
      if ( v3 != 48 )
        return 0;
    }
    else
    {
      map[5 * i] = v3 - 48;
    }
    v4 = *(v1 - 3);                             // 当前行第二个字符
    if ( map[5 * i + 1] )
```

```
    if ( map[5 * i + 1] )
    {
      if ( v4 != 48 )
        return 0;
    }
    else
    {
      map[5 * i + 1] = v4 - 48;
    }
    v5 = *(v1 - 2);                             // 当前行第三个字符
    if ( map[5 * i + 2] )
    {
      if ( v5 != 48 )
        return 0;
    }
    else
    {
      map[5 * i + 2] = v5 - 48;
    }
    v6 = *(v1 - 1);                             // 当前行第四个字符
    if ( map[5 * i + 3] )
    {
      if ( v6 != 48 )
        return 0;
    }
    else
    {
      map[5 * i + 3] = v6 - 48;
    }
    v7 = *v1;                                   // 当前行第五个字符
    if ( map[5 * i + 4] )
    {
      if ( v7 != 48 )
        return 0;
    }
    else
    {
      map[5 * i + 4] = v7 - 48;
    }
    v1 += 5;
  }
  return 1;
}
```

看一下map的初始值为：

```
unsigned char map[] =
{
    0,   0,   0,   0,   0,
 0,   0,   0,   0,   0,
    0,   0,   4,   0,   0,
 0,   0,   0,   3,   0,
    0,   0,   0,   0,   0
};
```

再画一个更直观的表：

0(map 为 0 时，map 的值为当前输入字符的 ascii 码减去'0'的 ascii 码)	0	0	0	0
0	0	0	0	0
0	0	4（map 为 4 时，对应的输入字符必须是'0'）	0	0
0	0	0	3（map 为 3 时，对应的输入字符必须是'0'）	0
0	0	0	0	0

综上，**fill_number函数的功能为：将5*5的map二维矩阵中的值为0时，转化为input字符对应的0-5，取值为0-5；不为0时，map值不变（map[2][2]=4,map[3][3]=3），且对应的输入值为字符'0'。**

3.docheck函数

见注释：

```
char docheck()
{
  __int64 v0; // rax
  __int64 v1; // rcx
  __int64 v2; // rcx
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v5; // rax
  __int64 v6; // rcx
  __int64 v7; // rcx
  __int64 v8; // rcx
  __int64 v9; // rcx
  __int64 v10; // rax
  char v11; // cl
  char v12; // cl
  char v13; // cl
  char v14; // cl
  __int64 v15; // rcx
  char result; // al
  char v17; // al
  char v18; // al
  char v19; // al
  char v20; // al
  char v21; // al
  int v22; // [rsp+0h] [rbp-10h]
  __int16 v23; // [rsp+4h] [rbp-Ch]
  int v24; // [rsp+8h] [rbp-8h]
  __int16 v25; // [rsp+Ch] [rbp-4h]
```

```
v0 = 0LL;
while ( 1 )
{
  v25 = 0;
  v24 = 0;
  v1 = (unsigned __int8)map[5 * v0];          // map二维数组中，每行不能有两个数相同
  if ( *((_BYTE *)&v24 + v1) )
    break;
  *((_BYTE *)&v24 + v1) = 1;
  v2 = (unsigned __int8)map[5 * v0 + 1];
  if ( *((_BYTE *)&v24 + v2) )
    break;
  *((_BYTE *)&v24 + v2) = 1;
  v3 = (unsigned __int8)map[5 * v0 + 2];
  if ( *((_BYTE *)&v24 + v3) )
    break;
  *((_BYTE *)&v24 + v3) = 1;
  v4 = (unsigned __int8)map[5 * v0 + 3];
  if ( *((_BYTE *)&v24 + v4) )
    break;
  *((_BYTE *)&v24 + v4) = 1;
  if ( *((_BYTE *)&v24 + (unsigned __int8)map[5 * v0 + 4]) )
    break;
  if ( ++v0 >= 5 )
  {
    v5 = 0LL;
    while ( 1 )
    {
      v23 = 0;
      v22 = 0;
      v6 = (unsigned __int8)map[v5];          // map二维数组中，每列不能有两个数相同
      if ( *((_BYTE *)&v22 + v6) )
        break;
      *((_BYTE *)&v22 + v6) = 1;
      v7 = (unsigned __int8)map[v5 + 5];
      if ( *((_BYTE *)&v22 + v7) )
        break;
      *((_BYTE *)&v22 + v7) = 1;
      v8 = (unsigned __int8)map[v5 + 10];
      if ( *((_BYTE *)&v22 + v8) )
        break;
      *((_BYTE *)&v22 + v8) = 1;
      v9 = (unsigned __int8)map[v5 + 15];
      if ( *((_BYTE *)&v22 + v9) )
        break;
      *((_BYTE *)&v22 + v9) = 1;
      if ( *((_BYTE *)&v22 + (unsigned __int8)map[v5 + 20]) )
        break;
      if ( ++v5 >= 5 )
      {
        v10 = 0LL;
        while ( 1 )
        {
          v11 = row[4 * v10];                 // 根据row数组，添加了一些约束条件
          if ( v11 == 2 )
          {
            if ( (unsigned __int8)map[5 * v10] > (unsigned __int8)map[5 * v10 + 1] )
              return 0;
          }
          else if ( v11 == 1 && (unsigned __int8)map[5 * v10] < (unsigned __int8)map[5 * v10 + 1] )
```

```
else if ( v11 == 1 && (unsigned __int8)map[5 * v10] < (unsigned __int8)map[5 * v10 + 1] )
{
  return 0;
}
v12 = row[4 * v10 + 1];
if ( v12 == 1 )
{
  if ( (unsigned __int8)map[5 * v10 + 1] < (unsigned __int8)map[5 * v10 + 2] )
    return 0;
}
else if ( v12 == 2 && (unsigned __int8)map[5 * v10 + 1] > (unsigned __int8)map[5 * v10 + 2] )
{
  return 0;
}
v13 = row[4 * v10 + 2];
if ( v13 == 2 )
{
  if ( (unsigned __int8)map[5 * v10 + 2] > (unsigned __int8)map[5 * v10 + 3] )
    return 0;
}
else if ( v13 == 1 && (unsigned __int8)map[5 * v10 + 2] < (unsigned __int8)map[5 * v10 + 3] )
{
  return 0;
}
v14 = row[4 * v10 + 3];
if ( v14 == 2 )
{
  if ( (unsigned __int8)map[5 * v10 + 3] > (unsigned __int8)map[5 * v10 + 4] )
    return 0;
}
else if ( v14 == 1 && (unsigned __int8)map[5 * v10 + 3] < (unsigned __int8)map[5 * v10 + 4] )
{
  return 0;
}
if ( ++v10 >= 5 )
{
  v15 = 0LL;
  while ( 1 )
  {
    v17 = col[5 * v15];             // 根据col数组添加了一些约束条件
    if ( v17 == 2 )
    {
      if ( (unsigned __int8)map[5 * v15] < (unsigned __int8)map[5 * v15 + 5] )
        return 0;
    }
    else if ( v17 == 1 && (unsigned __int8)map[5 * v15] > (unsigned __int8)map[5 * v15 + 5] )
    {
      return 0;
    }
    v18 = col[5 * v15 + 1];
    if ( v18 == 1 )
    {
      if ( (unsigned __int8)map[5 * v15 + 1] > (unsigned __int8)map[5 * v15 + 6] )
        return 0;
    }
    else if ( v18 == 2 && (unsigned __int8)map[5 * v15 + 1] < (unsigned __int8)map[5 * v15 + 6] )
    {
      return 0;
    }
    v19 = col[5 * v15 + 2];
```

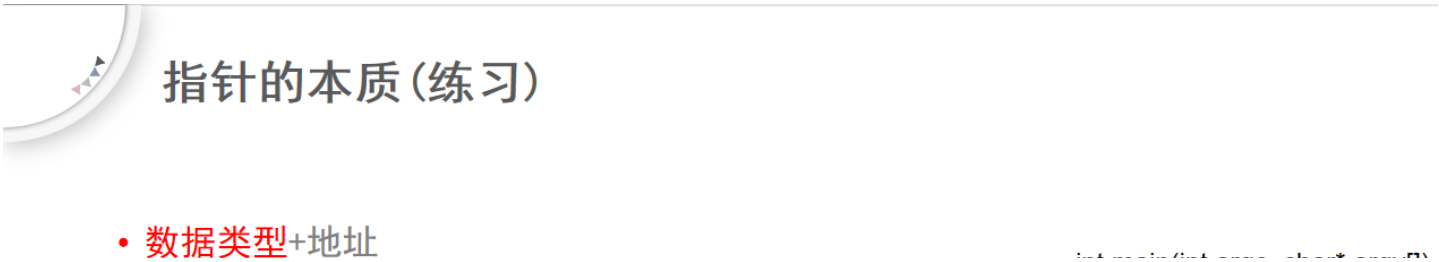
```
            if ( v19 == 2 )
            {
              if ( (unsigned __int8)map[5 * v15 + 2] < (unsigned __int8)map[5 * v15 + 7] )
                return 0;
            }
            else if ( v19 == 1 && (unsigned __int8)map[5 * v15 + 2] > (unsigned __int8)map[5 * v15 + 7] )
            {
              return 0;
            }
            v20 = col[5 * v15 + 3];
            if ( v20 == 2 )
            {
              if ( (unsigned __int8)map[5 * v15 + 3] < (unsigned __int8)map[5 * v15 + 8] )
                return 0;
            }
            else if ( v20 == 1 && (unsigned __int8)map[5 * v15 + 3] > (unsigned __int8)map[5 * v15 + 8] )
            {
              return 0;
            }
            v21 = col[5 * v15 + 4];
            if ( v21 == 2 )
            {
              if ( (unsigned __int8)map[5 * v15 + 4] < (unsigned __int8)map[5 * v15 + 9] )
                return 0;
            }
            else if ( v21 == 1 && (unsigned __int8)map[5 * v15 + 4] > (unsigned __int8)map[5 * v15 + 9] )
            {
              return 0;
            }
            ++v15;
            result = 1;                     // 必须的到这里
            if ( v15 >= 4 )
              return result;
          }
        }
      }
    }
  }
  return 0;
}
```

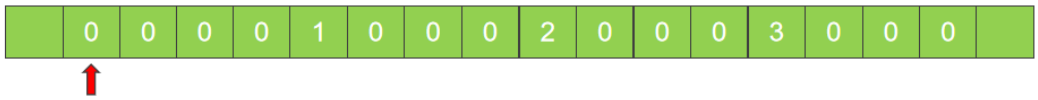
第四步、数独的确定（三坑）

代码中直接给出的注释表明，这是一道数独题，在这里，大佬一般不会告诉你，如何确定数独的过程，分析代码中需要扎实的c语言指针的功底，这也是大部分新手所面临的**第三个坑：怎么确定这是一道数独题？**

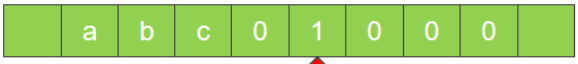
填坑的核心代码段是L33-L52，关键代码是形如*((_BYTE *)&v24 + v1) = 1;这样的代码，关键变量是v24、v25和map二维数组。

在将分析的思路之前，为了填上这个坑，需要新手补一下指针的本质，先看看下面这张图：





- 从p访问a[3]： *(int *)(p+4*3) 或 (int *)p+3



- 仅从a变量，去访问c： *((char *)&a-4+2)

```
int main(int argc, char* argv[])
{
    int a[4];
    a[0] = 0;
    a[1] = 1;
    a[2] = 2;
    a[3] = 3;
    char* p = &a[0];
}
```

```
int main(int argc, char* argv[])
{
    int a = 1;
    char b[4]="abc";
}
```


看图说话，不解释。

有人说指针的本质是地址，有人说指针的本质是一个变量，这些理解虽然正确，但新手很难接受。事实上，指针的本质是**数据类型+地址**，很多ida的代码涉及**指针的强制类型转化，比如从char指针转化为int指针、short指针，或把int指针、short指针转化为char指针。**

具体例子见下图：

指针的本质

- 数据类型+地址

int * p;　　(char*)p+i

```
(char)(*((char *)&v7 + i % v6) ^ v8[i])
```


看图说话，不解释。

在有个上面指针本质的基础后，分析的过程和思路为：

1） **填坑：不要把v24和v25看成两个变量，**v24是int型4个字节，v25是short型2个字节，要把它们**合二为一，**看成是内存中连续6个字节的空间；给新手提供保姆图：

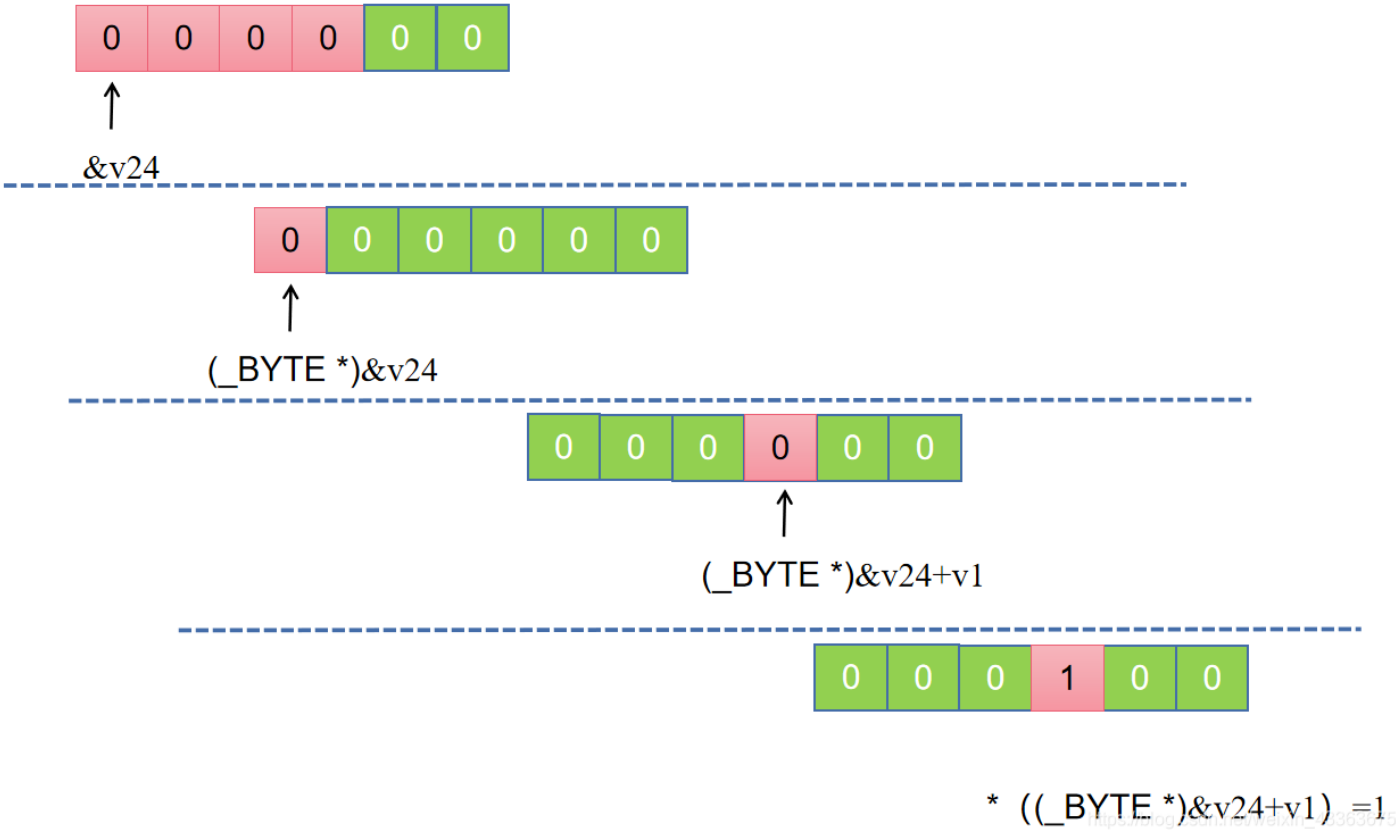
```
int v24;        // [rsp+8h] [rbp-8h]
int16 v25; // [rsp+Ch] [rbp-4h]
```

__int16 v25; // [rsp+Ch] [rbp-4h]

2）L35: v1 = (unsigned __int8)map[5 * v0]; 表示取得当前map值（8字节的map填入64字节的v1的低8位，高56位补0），v1为一个0-5的整数。

3）L36:if (*((_BYTE *)&v24 + v1))紧接L37:break；显然，if中的表达式 *((_BYTE *)&v24 + v1) 不能为真，必须为0，否则，直接跳到L190:return 0; 该表达式中，连续6个8字节空间已赋值为0（L33、L34），*((_BYTE *)&v24 + v1) 可能指向6个6空间中的任一个，当然也为0。

4）接着来到L38的*((_BYTE *)&v24 + v1) = 1;即将这6个位置中的第v1个位置变为1，如果再不懂的话，保姆式的给出下图，这是假设v1为3的情况下，将相应的字节置1。



5）如果上面的都看懂了，那么这道题已经解出来50%了，后面的很多代码都是一样的思路，接着看L39的v2 = (unsigned __int8)map[5 * v0 + 1];这行代码和L35一致，只是取得map当前行下一个元素的值，v2也为0-5之间的某一个值。L40：if (*((_BYTE *)&v24 + v2))，显然，该值应为false，满足该表达式为false的约束条件为：v1不等于v2，即map[0][0]不等于map[0][1]。

6）按上述思路继续分析，5次循环后，到L53条件为真，此时，**map矩阵中每行的元素为0-5中的值，且值不能相等。**

7）还是按照这种思路分析，L58到L77，分析略，结果为：**map矩阵中每列的元素为0-5中的值，且值不能相等。**

8）到这里，我们已经分析出了这是一道数独题。

c你理解了上面指针的本质，看后续代码就更为轻车熟路了，先看看row数组和col数组的布局吧：

```
unsigned char row[] =
{
    0,   0,   0,   1,
 1,   0,   0,   0,
 2,   0,   0,   1,
 0,   0,   0,   0,
 1,   0,   1,   0,
};
```

```
unsigned char col[] =
{
  0, 0, 2, 0, 2,
  0, 0, 0, 0, 0,
  0, 0, 0, 1, 0,
  0, 1, 0, 0, 1
};
```

10）row中的元素表示的是相邻的两个数的关系，如第一行第一个元素row[0]表示的是map[0][0]和map[0][1]的关系，如果这个值为1，表示左边的数大于右边的数；如果这个值为2，表示右边的数大于左边的数。再给个新手画一个保姆图吧，绿色的值为0、1或2，为1或2时，指明了两者的约束。

再给个对照代码：

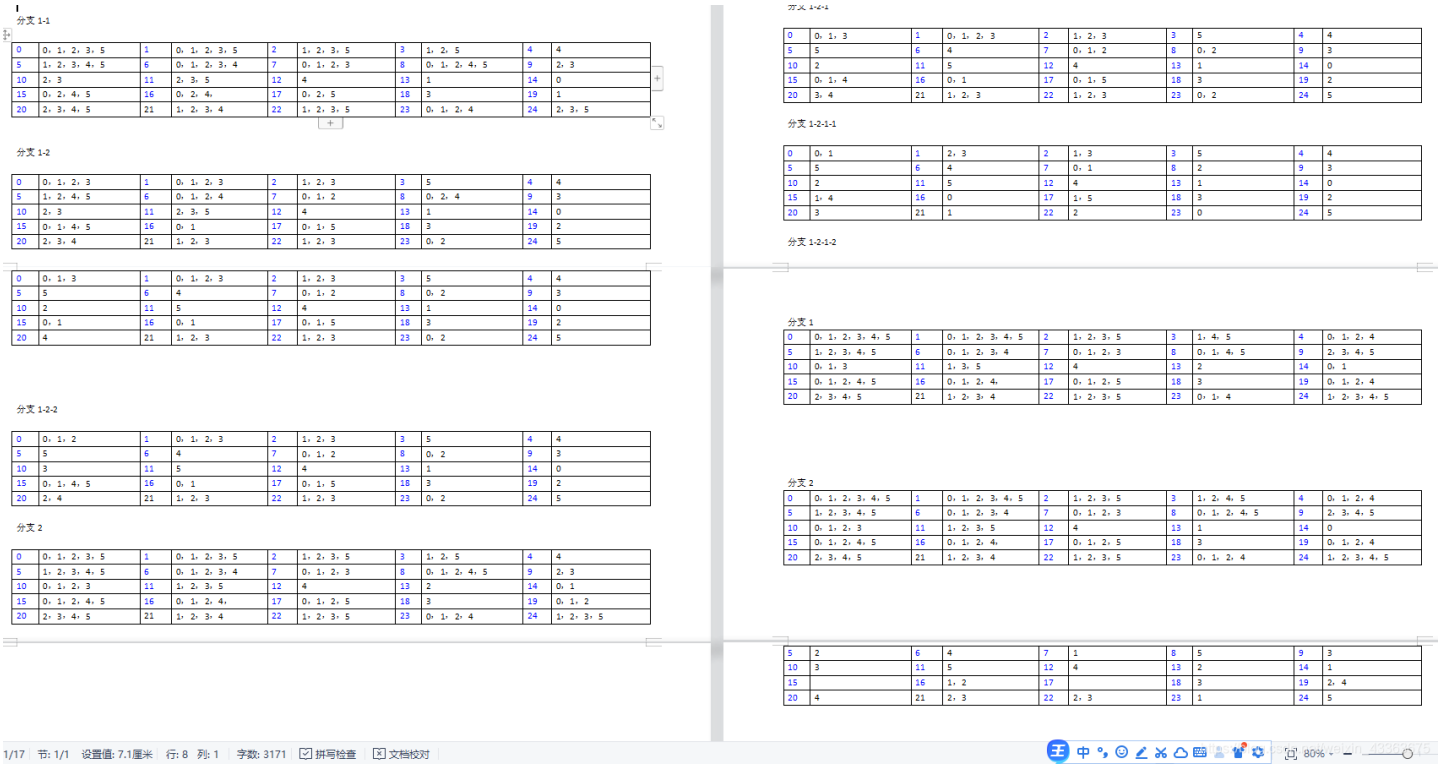
```
unsigned char row[] =
{
    0,                      0,   0,                      1/*map[3]>map[4]*/,
 1/*map[5]>map[6]*/,     0,   0,                      0,
 2/*map[10]<map[11]*/,   0,   0,                      1/*map[13]>map[14]*/,
 0,                      0,   0,                      0,
 1/*map[20]>map[21]*/,   0,   1/*map[22]>map[23]*/,   0,
};
```

11）col元素表示的是相邻的两个数的关系，也照此分析，对照代码为：

```
unsigned char col[] =
{
  0, 0,                     2/*(map[2]>map[7]*/, 0,                     2/*map[4]>map[9]*/,
  0, 0,                     0,                   0,                     0,
  0, 0,                     0,                   1/*(map[13]<map[18]*/, 0,
  0, 1/*(map[16]<map[21]*/, 0,                   0,                     1/*map[19]<map[24]*/
};
```

第五步、约束求解（四坑）

万事俱备只欠东风，有了初始值，有了初始条件，剩下的就是计算map数组并反推input数组了。map数组是0-5之间的任意一个数，就按数独填就好了，下面就看看失败的教训吧：



因为是0-5的数独题，脑子算就可以了，居然算了17页，这还不包括拿笔拿纸算的，后来又反复回溯到解题的每个阶段，到底问题出在哪里，然而，都没有找到问题所在。没想到最终也没有跳过这第四个坑。剩下的操作就更为离谱了，lindo、lingo这些工具开始搞，这虽然是很久以前数学建模的工具，但也还是很亲切和熟练，当然也有c语言爆破，下面也是失败的教训，爆破的部分c代码：

```
#include <stdio.h>

int p0[6] = {0, 1, 2, 3, 4, 5} ;
int p1[6] = {0, 1, 2, 3, 4, 5} ;
int p2[4] = {1, 2, 3, 5};
int p3[4] = {1, 2, 4, 5};
int p4[4] = {0, 1, 2, 4};
```

```
int p5[5] = {1, 2, 3, 4, 5};
int p6[5] = {0, 1, 2, 3, 4};
int p7[4] = {0, 1, 2, 3};
int p8[5] = {0, 1, 2, 4, 5};
int p9[4] = {2, 3, 4, 5};
int p10[4] = {0, 1, 2, 3};
int p11[4] = {1, 2, 3, 5};
int p12[1] = {4};
int p13[2] = {1, 2};
int p14[2] = {0, 1};
int p15[5] = {0, 1, 2, 4, 5};
int p16[4] = {0, 1, 2, 4};
int p17[5] = {0, 1, 2, 4, 5};
int p18[1] = {3};
int p19[4] = {0, 1, 2, 4};
int p20[4] = {2, 3, 4, 5};
int p21[4] = {1, 2, 3, 4};
int p22[4] = {1, 2, 3, 5};
int p23[4] = {0, 1, 2, 4};
int p24[5] = {1, 2, 3, 4, 5};


int main()
{
    int i=0;
    int a[25];
    for(; i<25; i++)
    {
        a[i] = 0;
    }
```


最终跳过这个坑的办法是这里：

既然这样就试图下载下来继续研究新旧两个文件的区别，由于链接过期了，找主办方要baby.bc的链接仍然未果，那就自己猜吧，数独题应该没有0吧，每行和每列的5个位置应该填1-5的某一个数字，且每行每列均不能有两个相同的数字。按这个思路，以下表为基础：

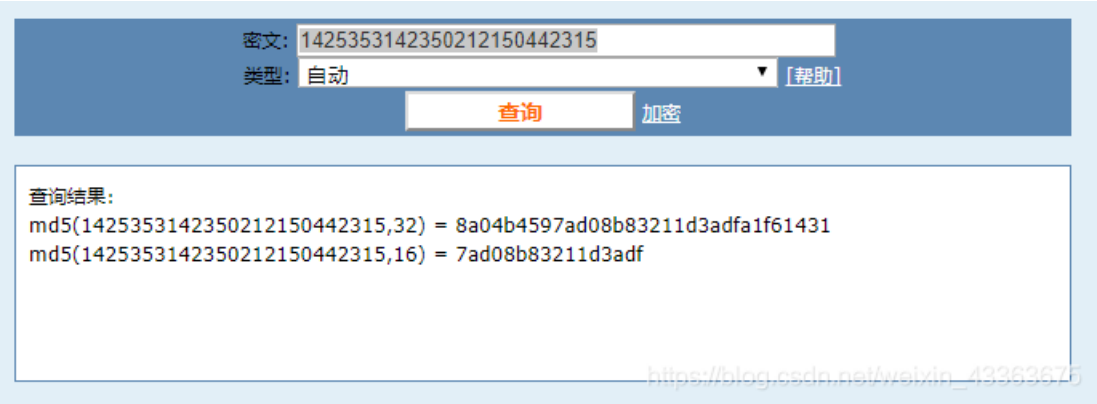
0		1		2		3		4	
5		6		7		8		9	
10		11		12	4	13		14	
15		16		17		18	3	19	
20		21		22		23	1	24	

以上一节中10）11）的map约束为条件手工算，5分钟内就可以搞定。真的是baby就可计算出来的结果，实现了内容与题目的呼应：

0	1	1	4	2	2	3	5	4	3
5	5	6	3	7	1	8	4	9	2
10	3	11	5	12	4	13	2	14	1

15	2	16	1	17	5	18	3	19	4
20	4	21	2	22	3	23	1	24	5

连成一串是**142535314235021215044231 5**（map[2][2]和map[3][3]按IDA之前的分析替换为0），接着md5加密吧，https://www.cmd5.com/的结果为：



当然，也有别人给出的高大上的解法，但baby级别的，人工求解肯定更快。

```
rowss = [[0x00, 0x00, 0x00, 0x01],[0x01, 0x00, 0x00, 0x00], [0x02, 0x00, 0x00, 0x01], [0x00, 0x00, 0x00, 0x00], [0x01, 0x00, 0x01, 0x00]]
colnms = [[0x00, 0x00, 0x02, 0x00,0x02], [0x00, 0x00, 0x00, 0x00, 0x00], [0x00, 0x00, 0x00, 0x01, 0x00], [0x00, 0x01, 0x00, 0x00, 0x01]]
from z3 import *
from hashlib import md5
s = Solver()
map = [[None]*5,[None]*5,[None]*5,[None]*5,[None]*5,]
for i in range(5):
    for j in range(5):
        map[i][j]=(Int("x%d%d"%(i, j)))
print(map)
s.add(map[2][2] == 4)
s.add(map[3][3] == 3)
for i in range(5):
    for j in range(5):
        s.add(map[i][j] >= 1)
        s.add(map[i][j] <= 5)
for i in range(5):
    for j in range(5):
        for k in range(j):
            s.add(map[i][j] != map[i][k])
for j in range(5):
    for i in range(5):
        for k in range(i):
            s.add(map[i][j] != map[k][j])
for i in range(5):
    for j in range(4):
        if rowss[i][j] == 1:
            s.add(map[i][j] > map[i][j+1])
        elif rowss[i][j] == 2:
            s.add(map[i][j] < map[i][j+1])
for i in range(4):
    for j in range(5):
        if colnms[i][j] == 2:
            s.add(map[i][j] > map[i+1][j])
        elif colnms[i][j] == 1:
            s.add(map[i][j] < map[i+1][j])
answer = s.check()
if answer == sat:
    print(s.model())
    m = s.model()
    flag = []
    for i in map:
        for j in i:
            flag.append(m[j].as_long())
    for i in range(len(flag)):
        flag[i] += 0x30
    flag[12] = 0x30
    flag[18] = 0x30
    flag = bytes(flag)
    print(md5(flag).hexdigest())
```

结果为：

```
======
[[x00, x01, x02, x03, x04], [x10, x11, x12, x13, x14], [x20, x21, x22, x23, x24]
, [x30, x31, x32, x33, x34], [x40, x41, x42, x43, x44]]
[x20 = 3,
 x01 = 4,
 x02 = 2,
 x13 = 4,
 x04 = 3,
 x23 = 2,
 x34 = 4,
 x14 = 2,
 x24 = 1,
 x10 = 5,
 x40 = 4,
 x21 = 5,
 x41 = 2,
 x42 = 3,
 x44 = 5,
 x30 = 2,
 x03 = 5,
 x31 = 1,
 x12 = 1,
 x43 = 1,
 x00 = 1,
 x32 = 5,
 x11 = 3,
 x33 = 3,
 x22 = 4]
8a04b4597ad08b83211d3adfa1f61431
```


可以看出，结果相同，到这里，第四个坑填完了。

这是从今年二月份开始断断续续搞逆向以来写的最长的一篇wp，深感“长江后浪推前浪、大爷被拍在了沙滩上”。

人生是一场马拉松，不是每一次努力都会有收获，但是，每一次收获都必须努力，这是一个不公平且不可逆转的命题。