

# 逆向新手踩坑指南之爬爬山能锻炼身体

转载

b10107 于 2017-12-23 21:53:00 发布 35 收藏

文章标签: [python](#) [c/c++](#) [操作系统](#)

原文链接: <http://www.cnblogs.com/HacTF/p/8094567.html>

版权

## 逆向新手踩坑指南之爬爬山能锻炼身体

首先坐下，打开电脑，平复一下心情，开始逆向分析。

对了，开始之前，首先推荐一下本站大神的逆向工程系统教程：[【传送门】](#)

好了，正文开始。软件运行一下是这样的：

逆向新手踩坑指南之爬爬山能锻炼身体

直接IDA打开，字符串窗口（shift+F12）：

逆向新手踩坑指南之爬爬山能锻炼身体

what a f\*\*\*\*\*ck? ? ? ! ! ! 这是啥？说好的字符串呢？说好的各种函数呢？

想了想忘了一件事情，先查查有没有壳：

逆向新手踩坑指南之爬爬山能锻炼身体

看到了刚才的小坑，顺便在看看有没有什么加密方式：

逆向新手踩坑指南之爬爬山能锻炼身体

没有，很好，直接upx脱壳，然后再次进入IDA【其实根据刚才IDA的提示也能看出来是upx的壳】：

逆向新手踩坑指南之爬爬山能锻炼身体

这次看到了字符串，然后根据关键字字符串交叉引用后来函数：

逆向新手踩坑指南之爬爬山能锻炼身体

直接F5：

```

1  _int64 main_0()
2  {
3  int v0; // edx
4  __int64 v1; // ST04_8
5  char v3; // [esp+0h] [ebp-160h]
6  int v4; // [esp+D0h] [ebp-90h]
7  int j; // [esp+DCh] [ebp-84h]
8  int i; // [esp+E8h] [ebp-78h]
9  char Str[104]; // [esp+F4h] [ebp-6Ch]
10
11  srand(0xCu);
12  j_memset(&unk_423D80, 0, 0x9C40u);
13  for ( i = 1; i <= 20; ++i )
14  {
15      for ( j = 1; j <= i; ++j )
16          dword_41A138[100 * i + j] = rand() % 100000;
17  }
18  sub_41134D("input your key with your operation can get the maximum:", v3);
19  sub_411249("%s", Str);
20  if ( j_strlen(Str) == 19 )
21  {
22      sub_41114F(Str);
23      v4 = 0;
24      j = 1;
25      i = 1;
26      dword_423D78 += dword_41A138[101];
27      while ( v4 < 19 )
28      {
29          if ( Str[v4] == 76 )
30          {
31              dword_423D78 += dword_41A138[100 * ++i + j];
32          }
33          else
34          {
35              if ( Str[v4] != 82 )
36              {
37                  sub_41134D("error\n", v3);

```

圈一里面是什么暂时不想管，直接从18行开始基本比较明显了，一个是屏幕输出函数printf，一个是从屏幕获得输入，圈二IDA本身分析得也很明显，就是判断长度的，字符串限制为19。22行跟进去以后是这样的：

```

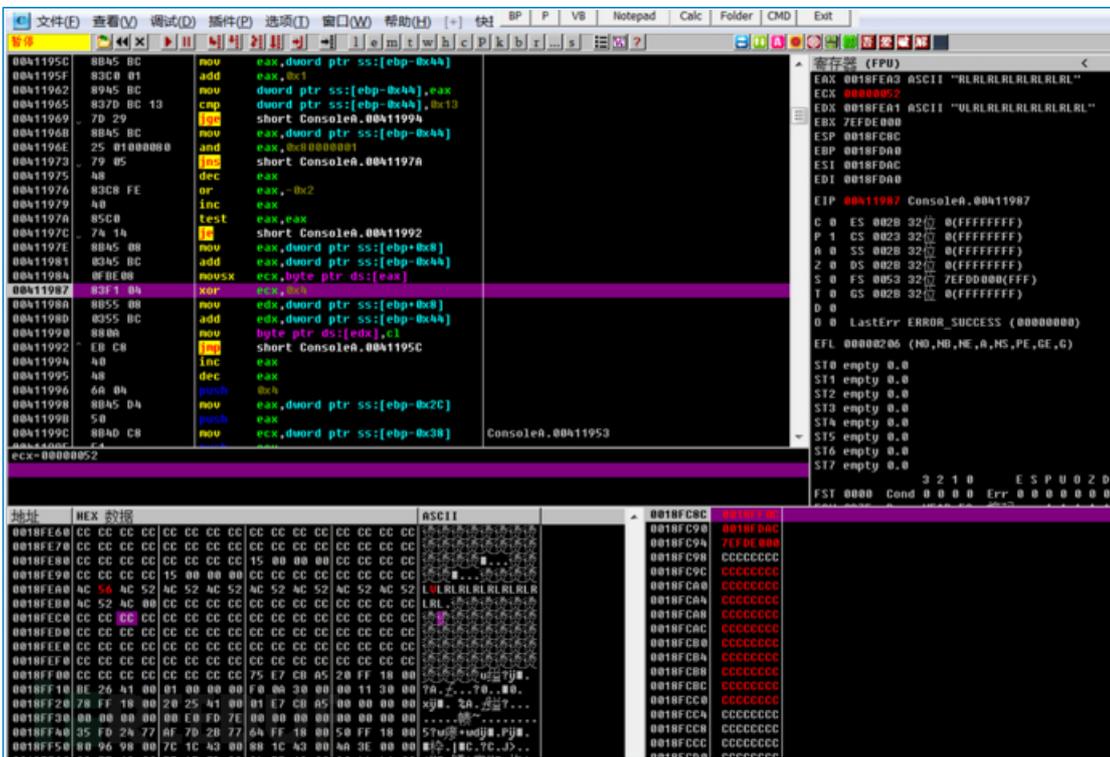
1  bool __cdecl sub_411750(LPCVOID lpAddress, int a2, int a3)
2  {
3  int v3; // ST1C_4
4  DWORD f10ldProtect; // [esp+D4h] [ebp-2Ch]
5  struct _MEMORY_BASIC_INFORMATION Buffer; // [esp+E0h] [ebp-20h]
6
7  VirtualQuery(lpAddress, &Buffer, 0x1Cu);
8  VirtualProtect(Buffer.BaseAddress, Buffer.RegionSize, 0x40u, &Buffer.Protect);
9  while ( 1 )
10 {
11     v3 = a2--;
12     if ( !v3 )
13         break;
14     *(_BYTE *)lpAddress ^= a3;
15     lpAddress = (char *)lpAddress + 1;
16 }
17 return VirtualProtect(Buffer.BaseAddress, Buffer.RegionSize, Buffer.Protect, &f10ldProtect);
18 }

```

这TM是啥？！！！！不过结合26行，31行，37行能看到类似于a=a+b一样的东西，是从dword\_41A138[]里面取值，然后直接加到dword\_423D78这个变量里面。

从29行和35行能判断这个输入的字符串一定是L或者R【根据ASCII码】，既然22行不懂，就直接先动态走一走：





这里是一个重点，函数在这里一直在41195C和411992之间循环，并且，在411987这一步直接和4做了异或，直接导致将我们输入的字符串所有偶数位做了异或。从这一点判断，这个函数应该是IDA伪代码的第22行，对字符串进行了处理：

```

18  sub_41134D("input your key with your operation can get the maximum:", v3);
19  sub_411249("%s", Str);
20  if ( j_strlen(Str) == 19 )
21  {
22  sub_41114F(); ←
23  v4 = 0;
24  j = 1;
25  i = 1;
26  dword_423D78 += dword_41A138[101];
27  while ( v4 < 19 )
28  {
29  if ( Str[v4] == 76 )
30  {
31  dword_423D78 += dword_41A138[100 * ++i + j];

```

处理方法就是偶数位和4做异或，根据IDA的判断，异或后的值只能是L或者R。于是再次输入的时候就该输入些什么了。经过反推，奇数位只能是L或者R，偶数位只能是V（代表R）或者H（代表L）。字符串输入什么变得清晰了。

直接让程序执行完，然后手动改栈里面的值，再返回：

地址	HEX 数据	ASCII
00411C8A	50	push eax
00411C8B	E8 BFF4FFFF	call ConsoleA.0041114F
00411C90	83C4 04	add esp,0x4
00411C93	C785 70FFFFFF	mov dword ptr ss:[ebp-0x90],0x0
00411C9D	C785 7CFFFFFF	mov dword ptr ss:[ebp-0x84],0x1
00411CA7	8B85 7CFFFFFF	mov eax,dword ptr ss:[ebp-0x84]
00411CAD	8945 88	mov dword ptr ss:[ebp-0x78],eax
00411CB0	6945 88 900100	imul eax,dword ptr ss:[ebp-0x78],0x190
00411CB7	8B8D 7CFFFFFF	mov ecx,dword ptr ss:[ebp-0x84]
00411CB0	8B15 783D4200	mov edx,dword ptr ds:[0x423D78]
00411CC3	039488 38A1410	add edx,dword ptr ds:[eax+ecx*4+0x41A138]
00411CCA	8915 783D4200	mov dword ptr ds:[0x423D78],edx
00411CD0	83B0 70FFFFFF	cmp dword ptr ss:[ebp-0x90],0x13
00411CD7	0F80 BF000000	jb ConsoleA.00411D9C
00411CD0	8B85 70FFFFFF	mov eax,dword ptr ss:[ebp-0x90]
00411CE3	0FB84C05 94	movsx ecx,byte ptr ss:[ebp+eax-0x6C]
00411CE8	83F9 4C	cmp ecx,0x4C
00411CEB	75 29	jnz short ConsoleA.00411D16
00411CED	8B45 88	mov eax,dword ptr ss:[ebp-0x78]
00411CF0	83C0 01	add eax,0x1
00411CF3	8945 88	mov dword ptr ss:[ebp-0x78],eax
00411CF6	694D 88 900100	imul ecx,dword ptr ss:[ebp-0x78],0x190
00411CFD	8B95 7CFFFFFF	mov edx,dword ptr ss:[ebp-0x84]
00411D03	A1 783D4200	mov eax,dword ptr ds:[0x423D78]
00411D08	038491 38A1410	add eax,dword ptr ds:[ecx+edx*4+0x41A138]
00411D0F	A3 783D4200	mov dword ptr ds:[0x423D78],eax
00411D14	EB 72	jmp short ConsoleA.00411D88
00411D16	8B85 70FFFFFF	mov eax,dword ptr ss:[ebp-0x90]
00411D1C	0FB84C05 94	movsx ecx,byte ptr ss:[ebp+eax-0x6C]
00411D21	83F9 52	cmp ecx,0x52
00411D24	75 3A	jnz short ConsoleA.00411D60
00411D26	8B45 88	mov eax,dword ptr ss:[ebp-0x78]
00411D29	83C0 01	add eax,0x1
00411D2C	8945 88	mov dword ptr ss:[ebp-0x78],eax
00411D2E	8B8D 7CFFFFFF	mov ecx,dword ptr ss:[ebp-0x84]

地址	HEX 数据	ASCII
0018FE60	CC CC CC CC CC CC CC CC	CCCCCCCC
0018FE70	CC CC CC CC CC CC CC CC	CCCCCCCC
0018FE80	CC CC CC CC CC CC CC CC	CCCCCCCC
0018FE90	CC CC CC CC 15 00 00 00	CCCCCCCC
0018FEA0	4C 52 4C 52 4C 52 4C 52	LRRLRLRLRLRLRLR
0018FEB0	4C 52 4C 00 CC CC CC CC	LRRLRLRLRLRLRLR
0018FEC0	CC CC CC CC CC CC CC CC	CCCCCCCC
0018FED0	CC CC CC CC CC CC CC CC	CCCCCCCC
0018FDA8		
0018FDAC		
0018FDB0	00000000	
0018FDB4	7EFDE000	
0018FDB8	CCCCCCCC	
0018FDBC	CCCCCCCC	
0018FDC0	CCCCCCCC	
0018FDC4	CCCCCCCC	
0018FDC8	CCCCCCCC	

从这里开始的一大长串我没有仔细分析，太多了，看得心烦，跟了几次以后发现总是在这里[411D97]跳转，方便起见直接这里下了断点，也在[411D9C]这里下了断点，之所以这样，是想随时关注着各个寄存器的变化和堆栈的变化：

The screenshot shows a debugger interface with the following components:

- Assembly Window:** Displays assembly instructions with their addresses and operands. Key instructions include:
  - 00411D97: `mov ConsoleA.00411C00`
  - 00411D9C: `mov eax, dword ptr ds:[0x423D78]`
  - 00411DA2: `push ConsoleA.00417B88`
  - 00411DA7: `call ConsoleA.0041134D`
  - 00411DB1: `push ConsoleA.00417B80`
  - 00411DB6: `call dword ptr ds:[<ucrtbased.system>]`
- Registers Window:** Shows the state of CPU registers. EAX is 00000000, ECX is 00001900, EDX is 00000000, and EIP is 00411D97.
- Hex Dump:** Located at the bottom, it shows memory addresses (e.g., 0018FE60) and their corresponding hex and ASCII values. The ASCII column shows some garbled characters.

然后结合IDA的整个流程：

```
Instruction Data Unexplored External symbol
IDA View-A Pseudocode-A Strings window Hex View-1 Struc
16 dword_41A138[100 * i + j] = rand() % 100000;
17 }
18 sub_41134D("input your key with your operation can get the maximum:", v3);
19 sub_411249("%s", Str);
20 if ( j_strlen(Str) == 19 )
21 {
22     sub_41114F();
23     v4 = 0;
24     j = 1;
25     i = 1;
26     dword_423D78 += dword_41A138[101];
27     while ( v4 < 19 )
28     {
29         if ( Str[v4] == 76 ) ①
30         {
31             dword_423D78 += dword_41A138[100 * ++i + j];
32         }
33         else
34         {
35             if ( Str[v4] != 82 ) ②
36             {
37                 sub_41134D("error\n", v3);
38                 system("pause");
39                 goto LABEL_18;
40             }
41             dword_423D78 += dword_41A138[100 * ++i + ++j];
42         }
43         ++v4;
44     }
45     sub_41134D("your operation can get %d points\n", dword_423D78);
46     system("pause");
47 }
```

能大概推测出这里应该就是根据输入的字符串判断是在圈一还是圈二。

处理完毕以后就能直接出结果了：

```
input your key with your operation can get the maximum:LRLRLRLRLRLRLRLRL
RLRL
your operation can get 312869 points
请按任意键继续. . .
```

还是我太天真，将输入提交以后发现并不正确。然后思路基本就卡到这里了，连续一两天没有进展，心里憋屈得不得了。后来再次看题目的时候发现题目是mountain climbing。英文翻译是爬山，爬山，爬山，爬山呢？！！于是我的思路就有变成了找山...

继续回到IDA来看吧，因为已经没什么可以看的了。这次就变成了分析前面所有没分析过的函数，首先就是dword\_41A138[]里面是什么？

```
10
11 srand(0xCu);
12 j_memset(&unk_423D80, 0, 0x9C40u);
13 for ( i = 1; i <= 20; ++i )
14 {
15     for ( j = 1; j <= i; ++j )
16         dword_41A138[100 * i + j] = rand() % 100000;
17 }
18 sub_41134D("input your key with your operation can get the maximum:", v3);
19 sub_411249("%s", Str);
```

从字面上理解rand()就是随机数函数，每次都产生不一样的随机数，然后根据这个去进行加数字的运算，这TM能算出来啥？！

然而事实告诉我我还是我太天真了(原谅我大学不是计算机专业，没好好学过C....)。

c语言里面有一个产生随机数的函数，rand()。但是在学习过程中发现这是伪机的，虽然叫随机，但是每一次结果都一样(下面图片来自随便一个搜索引擎，随便一搜一大把的讲解)：

```
rand和srand的用法
首先我们要对rand&srand有个总体的看法:srand初始化随机种子,rand产生随机数,下面将详细说明。
rand (产生随机数)
表头文件: #include
定义函数 :int rand(void)
函数说明:
因为rand的内部实现是用线性同余法做的,他不是真的随机数,只不过是因为其周期特别长,所以有一定的范围里可看成是随机的,rand()会返回一随机数值,范围在0至RAND_MAX 间。在调用此函数产生随机数前,必须先利用srand()设好随机数种子,如果未设随机数种子,rand()在调用时会自动设随机数种子为1。rand ()产生的是假随机数字,每次执行时是相同的。若要不同,以不同的值来初始化它.初始化的函数就是srand()。
```

于是思路一下子清晰了，srand()是一个固定的值(IDA里面对没有定义的数字后面会有一个u，undefined)，那么rand()产生的数字也是固定的!!! 根据代码描述，基本可以断定那个dword\_41A138[]就是山了，数组下标是这样的101,201,202,301,302,303...写成这样也许更直观：

```
1          101
2      201 202
3      301 302 303
4      401 402 403 404
5      501 502 503 504 505
6      601 602 603 604 605 606
7      . . . . .
```

从IDA里的27行到41行看：



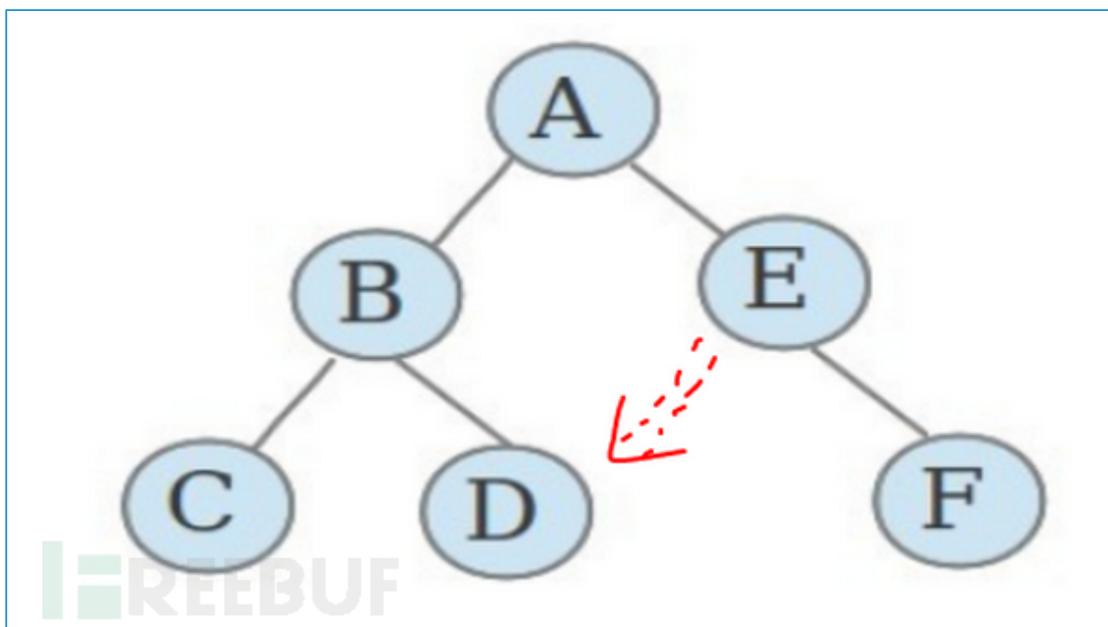
```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. #define STACKINITSIZE 100
5. #define STACKINCREASESIZE 20
6.
7. typedef char ElemType;
8. //树结构
9. typedef struct tree
10. {
11.     ElemType data;
12.     struct tree * lchild;
13.     struct tree * rchild;
14.     unsigned int isOut; //专为后序遍历设置的，0为不需要被输出，1为需要被输出
15. }TreeNode,*Tree;
16.
17. //栈结构
18. typedef struct stack
19. {
20.     Tree * base;
21.     Tree * top;
22.     int stacksize;
23. }Sqstack;
24.
25.
26. /*****栈的操作声明*****/

```

一共316行，相信我，这不是

我想要的，首先以我近乎于只穿着内裤的C语言功底读懂316行代码还需要很多时间，其次还要搞明白并熟练使用结构体变量。而且，经过耐心的学习以后发现二叉树遍历不能满足我的需求，因为二叉树遍历无法实现从E到D：



于是我的这道题目再次陷入深坑....

这里卡了一天半左右，没有任何思路，后来无聊，想了想一共有多少条路？我能不能手工遍历(我是一个天真善良吃苦耐劳的八道杠优秀代表)。从上往下数第二层有2条，走到第三层有4层，第四层有8条...不会算了，在纸上画到第五层的时候我已经有点崩溃了，又耐心思考了一个小时左右，耐着性子画到了六层，终于让我这个晕乎乎的脑袋看到了规律，2的几次方。从山顶到第二层有2条路，到第三层有4条路，是2的平方。第四层是2的3次方。因为每一个节点对应下一层的节点只有两个选择。我打开windows键，在运行处输入calc以后用科学计算器算了算2的19次方：

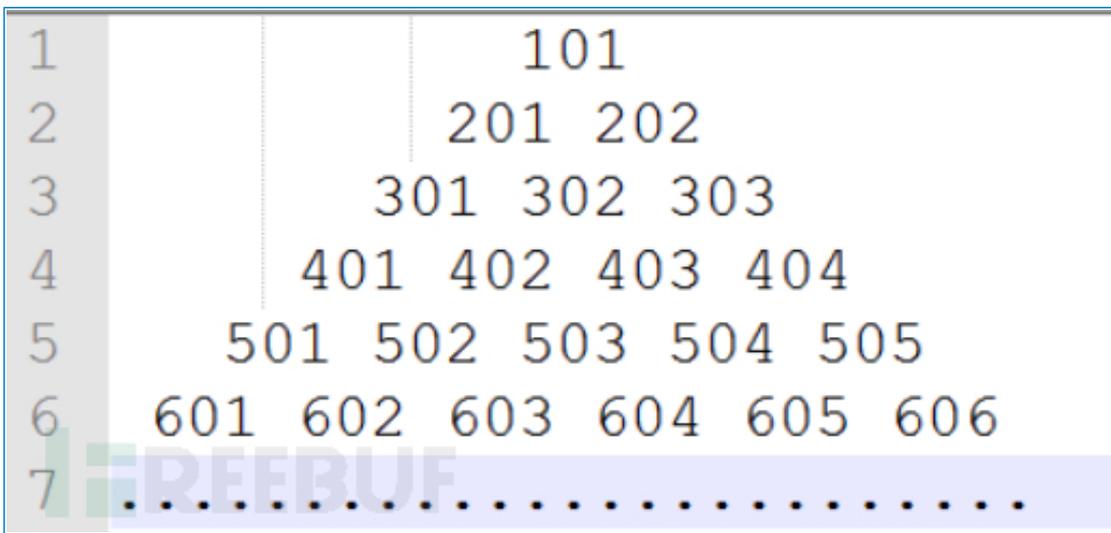


这是逼着我用程序来解这道题(此时心里有一万句脏话想说出口)。

既然用程序，问题就又来了，我如何表现每一种走法呢？这里我又卡住了，盯着自己在纸上画的所谓的山看了好久也没有想出来用什么办法来表现每一条路径。看看作者的源程序：

```
12 j_memset(&unk_423D80, 0, 0x9C40u);
13 for ( i = 1; i <= 20; ++i )
14 {
15     for ( j = 1; j <= i; ++j )
16         dword_41A138[100 * i + j] = rand() % 100000;
17 }
```

作者用了两个数字来写，i就是第几层，j就是在i层的第几个数。可是我表现不了从302到403啊，等等，302,403两个数层数和尾标都是差1，在看看整个山的分布：



找到了一个规律，每条路都是只能差1，比如这条路：101,201,302,403,503,604...这条路里，层数差1是固定的，可以用循环实现，问题是如何实现1,1,2,3,3,4这个路数。要么加0，要么加1。要么加0，要么加1。要么加0，要么加1。默念无数遍以后得到了这样一个数列01101，我嘞个王母娘娘，臣妾终于做到了~!!! 这TM不就是二进制么，0代表往左走一步，1代表往右走一步。（伟大的二进制~！伟大的十六进制~！）如果有6层，那么01101换算成十进制就是13，一共有32条路，00000也算一条路的话（十进制的0），11111就是最后一条路，十进制就是31，0到31一共32个数字，Nice，完美实现了所有路径的表示，此时，半天又过去了...于是代码实现就容易得多了，先定一个小目标，实现6层的山，用Python：

```
1 rodenum=1
2 for p in range(1,6,1):
3     rodenum *= 2
4 for i in range(0,rodenum,1):
5     tmp=str(bin(i)[2:])
6     rode=str(0)*(5-len(tmp))+tmp
7     print rode
```

Run temtest

01000  
01001  
01010  
01011  
01100  
01101  
01110  
01111  
10000  
10001  
10010  
10011  
10100  
10101  
10110  
10111  
11000  
11001  
11010  
11011  
11100  
11101  
11110  
11111

Process finished with exit code 0

已然实现了所有路径的表示方法，Nice!!!

下面面临着怎么把山里的数字给扒出来，从作者的定义方式来看，定义的数组的长度在2000多，否则下标放不下，简单，直接对照伪代码写出来C代码，然后自己跑一边就行，于是有了这段C代码：

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 int main()
4 {
5     int i,j;
6     long int mem[3000];
7     srand(12);
8     for (i=1;i<=20;++i)
9     {
10         for (j=1;j<=i;++j)
11             mem[i*100+j]=rand() % 100000;
12     }
13     for (i=1;i<=20;++i)
14     {
15         for (j=1;j<=i;++j)
16             printf("mem[%d] is: %d\n",i*100+j,mem[i*100+j]);
17     }
18 }
19
```

由于强迫症的限制，我就不想在windows上跑，没环境，不想装，伟大的linux干啥都行，为啥非要在windows上吊死，于是在我的kali上跑完以后是这样的（自己又给自己挖了一个坑）：

```
mem[1916] is: 27003
mem[1917] is: 64952
mem[1918] is: 14202
mem[1919] is: 17493
mem[2001] is: 74723
mem[2002] is: 94707
mem[2003] is: 84734
mem[2004] is: 74804
mem[2005] is: 63794
mem[2006] is: 11754
mem[2007] is: 76534
mem[2008] is: 78117
mem[2009] is: 47305
mem[2010] is: 15032
mem[2011] is: 99491
mem[2012] is: 93631
mem[2013] is: 18000
mem[2014] is: 13871
mem[2015] is: 46500
mem[2016] is: 40966
mem[2017] is: 74516
mem[2018] is: 33678
mem[2019] is: 92527
mem[2020] is: 54198
root@kali: ~/Desktop#
```

好了，数字找到了，也能按照自己的意愿去进行遍历了，剩下的就是融合C和Python在一起，用Python实现脚本计算（别问我为啥不用C）。

问题TMD又来了，我要如何把210个数字移到Python里，这里，我选择了裸奔的方式：手工输入！（C语言如何实现文件读写不会！不想学！）210个数字写到Python的字典里，比如：

```
'1601': 42706, '1602': 72933, '1603': 33675, '1604': 93746, '1605': 71468, '1606': 74564, '1607': 97708, '1608': 1701, '1701': 8531, '1702': 20415, '1703': 7051, '1704': 78070, '1705': 64471, '1706': 60301, '1707': 22895, '1708': 36, '1801': 65108, '1802': 10791, '1803': 35888, '1804': 56025, '1805': 24851, '1806': 74138, '1807': 93418, '1808': 1901, '1901': 2968, '1902': 98028, '1903': 52869, '1904': 6641, '1905': 60644, '1906': 70826, '1907': 35208, '1908': 79, '2001': 74723, '2002': 94707, '2003': 84734, '2004': 74804, '2005': 63794, '2006': 11754, '2007': 76534, '2008': 1
```

实现遍历的方法在前面已经写出来，还剩下如何根据路径把对应的数字进行相加，给一段关键代码：

```
for i in range(0, rodenum, 1):
    tmp = str(bin(i)[2:])
    rode = str(0) * (19 - len(tmp)) + tmp
    j = 0
    mountainlr = 1
    ournum = 0
    #print "current rode is:", rode
    while j < len(rode):
        mountainh = 100 * j + 200
        mountainlr += int(rode[j])
        mountainpoint = mountainh + mountainlr
        j += 1
        ournum += int(mountain[str(mountainpoint)])
        if ournum > biggestnum:
            biggestnum = ournum
            maxrode = str(rode)
    print biggestnum, maxrode, i
```

原理就是根据路径寻找字典里对应的点的值，然后相加，每条路径都产生一个值，不断的比较，只取最大的那个就行。

一阵开心地等待过后有了结果，然后根据最开始的对字符串进行处理的方式逆回去，提交答案以后又给了我当头一棒，我是有多天真！！为什么不对？后来我把Python跑出来的结果用题目的exe跑了一遍，发现得到的数字不一样.....郁闷。为啥不一样？同样是rand函数，固定srand。痛定思痛过后反思到了也许是不同的平台就不一样，然而我还是不想在电脑上装VC。

那怎么办？

让程序自己吐出来这个数组！

然后重新用OD运行这个exe，不过这次用F8走，为的是找到数组生成过程中产生的数字，于是我来到了这里：

00411BE4	8B85 7CFFFFFF	mov	eax,dword ptr ss:[ebp-0x84]	
00411BEA	83C0 01	add	eax,0x1	
00411BED	8985 7CFFFFFF	mov	dword ptr ss:[ebp-0x84],eax	
00411BF3	8B85 7CFFFFFF	mov	eax,dword ptr ss:[ebp-0x84]	
00411BF9	3B45 88	cmp	eax,dword ptr ss:[ebp-0x78]	
00411BFC	7F 2D	je	short ConsoleA.00411C2B	
00411BFE	8BF4	mov	esi,esp	
00411C00	FF15 4C214300	call	dword ptr ds:[<&ucrbased.rand>]	ucrtbody
00411C06	3BF4	cmp	esi,esp	
00411C08	E8 1AF5FFFF	call	ConsoleA.00411127	
00411C0D	99	cdq		
00411C0E	B9 A0860100	mov	ecx,0x186A0	
00411C13	F7F9	idiv	ecx	
00411C15	6945 88 900100	imul	eax,dword ptr ss:[ebp-0x78],0x190	
00411C1C	8B8D 7CFFFFFF	mov	ecx,dword ptr ss:[ebp-0x84]	
00411C22	899488 3BA1410	mov	dword ptr ds:[eax+ecx*4+0x41A138],edx	
00411C29	EB B9	jmp	short ConsoleA.00411BE4	
00411C2B	EB 9C	jmp	short ConsoleA.00411BC9	
00411C2D	68 30704100	push	ConsoleA.00417030	ASCII "input your key with your operation can get the maximum:"
00411C32	E8 16F7FFFF	call	ConsoleA.00411340	
00411C37	83C4 04	add	esp,0x4	
00411C38	8D45 94	lea	eax,dword ptr ss:[ebp-0x6C]	
00411C3D	50	push	eax	
00411C3E	68 7A704100	push	ConsoleA.00417074	ASCII "%s"
00411C43	E8 01F6FFFF	call	ConsoleA.00411249	

图里面这个不断的跳转就是在生成我们需要的这个数组，我们在411C29这个地方下断点，并且每次查看对应地址的内容：

The screenshot displays a debugger interface with two main sections:

- Assembly View:** Shows the disassembled code from the previous image. The instruction at address 00411C29 is highlighted in purple: `jmp short ConsoleA.00411BE4`. The instruction at 00411C2B is also highlighted: `jmp short ConsoleA.00411BC9`.
- Memory Dump:** A table showing memory addresses, hex data, and ASCII characters.
 

地址	HEX	数据	ASCII
0041A2CC	4D	00 00 00 00	.....
0041A2DC	00	00 00 00 00	.....
0041A2EC	00	00 00 00 00	.....
0041A2FC	00	00 00 00 00	.....
0041A30C	00	00 00 00 00	.....
0041A31C	00	00 00 00 00	.....
0041A32C	00	00 00 00 00	.....
0041A33C	00	00 00 00 00	.....
0041A34C	00	00 00 00 00	.....
0041A35C	00	00 00 00 00	.....
0041A36C	00	00 00 00 00	.....
0041A37C	00	00 00 00 00	.....
0041A38C	00	00 00 00 00	.....
0041A39C	00	00 00 00 00	.....
0041A3AC	00	00 00 00 00	.....
0041A3BC	00	00 00 00 00	.....
0041A3CC	00	00 00 00 00	.....
0041A3DC	00	00 00 00 00	.....
0041A3EC	00	00 00 00 00	.....
0041A3FC	00	00 00 00 00	.....
0041A40C	00	00 00 00 00	.....
0041A41C	00	00 00 00 00	.....
0041A42C	00	00 00 00 00	.....
0041A43C	00	00 00 00 00	.....
0041A44C	00	00 00 00 00	.....
0041A45C	FC 15	58 18	?..X
0041A46C	00	00 00 00 00	.....
0041A47C	00	00 00 00 00	.....

这就是我们需要的数组，总不能再跟210次，再写210个字典里的值吧？于是经过再次使用搜索引擎的技能，学会了条件断点记录到日志，这样，能把每次经过411C22这个点时的edx的值记录在案！！！！

顺便一说，在Python里的文件读写，我还是会一点的.....

记录的日志大概是这个样子的：

1	00411C22	COND: 0000004D
2	00411C22	COND: 000015FC
3	00411C22	COND: 00001858
4	00411C22	COND: 0000717C
5	00411C22	COND: 00000616
6	00411C22	COND: 00006626
7	00411C22	COND: 00003293
8	00411C22	COND: 000074E6
9	00411C22	COND: 00002ECD
10	00411C22	COND: 00005763
11	00411C22	COND: 00000FEE
12	00411C22	COND: 00006FD5
13	00411C22	COND: 00001239
14	00411C22	COND: 000008B5
15	00411C22	COND: 0000607B
16	00411C22	COND: 00006AEA
17	00411C22	COND: 00000C09
18	00411C22	COND: 0000465C
19	00411C22	COND: 00006185
20	00411C22	COND: 00000810
21	00411C22	COND: 0000690A
22	00411C22	COND: 0000523E
23	00411C22	COND: 00001469
24	00411C22	COND: 00002E01
25	00411C22	COND: 0000749D
26	00411C22	COND: 00000B8C
27	00411C22	COND: 000057A7
28	00411C22	COND: 00000D0D

经过这次的改变，我的Python脚本再次跑了起来，得到如下结果：

```
3 1 001 10 10 524276
3 1 001 10 10 524277
3 1 001 10 10 524278
3 1 001 10 10 524279
3 1 001 10 10 524280
3 1 001 10 10 524281
3 1 001 10 10 524282
3 1 001 10 10 524283
3 1 001 10 10 524284
3 1 001 10 10 524285
3 1 001 10 10 524286
3 1 001 10 10 524287
Process finished with exit code 0
```

然后根据那一连串的1001去反推回L和R，再反推回V和H，于是这次用我的字符串跑exe以后，exe跑出来的结果和我的Python一样了。然后提交分数，FFFFFF\*\*\*\*\*KKKKKKKK!!! 还不  
对!!! 为啥? 我都找到最大的路了，得到最大的分数了!

然后，我笑了，脑子高兴糊涂了，我提交了我的路径，正确!

七天过去了，做逆向真是打发时间的好帮手~

相信聪明的你已经看到了我所有犯过的错误，并且已经知道了我写的是哪里的writeup.....

exe链接: <https://pan.baidu.com/s/1o8LxYA2> 密码: jisg

转载于: <https://www.cnblogs.com/HacTF/p/8094567.html>