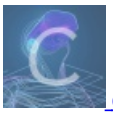
攻防世界 reverse-box

原创

Bxb0 于 2020-05-28 17:00:49 发布 191 收藏

分类专栏： ctf 文章标签： python

本文链接：https://blog.csdn.net/Bsecure/article/details/106408823



> **reverse-box**
>
> C语言
>
> python

C语言

题目下载下来，弄半天始终觉得有问题。百度发现题目少给了输出字符串。

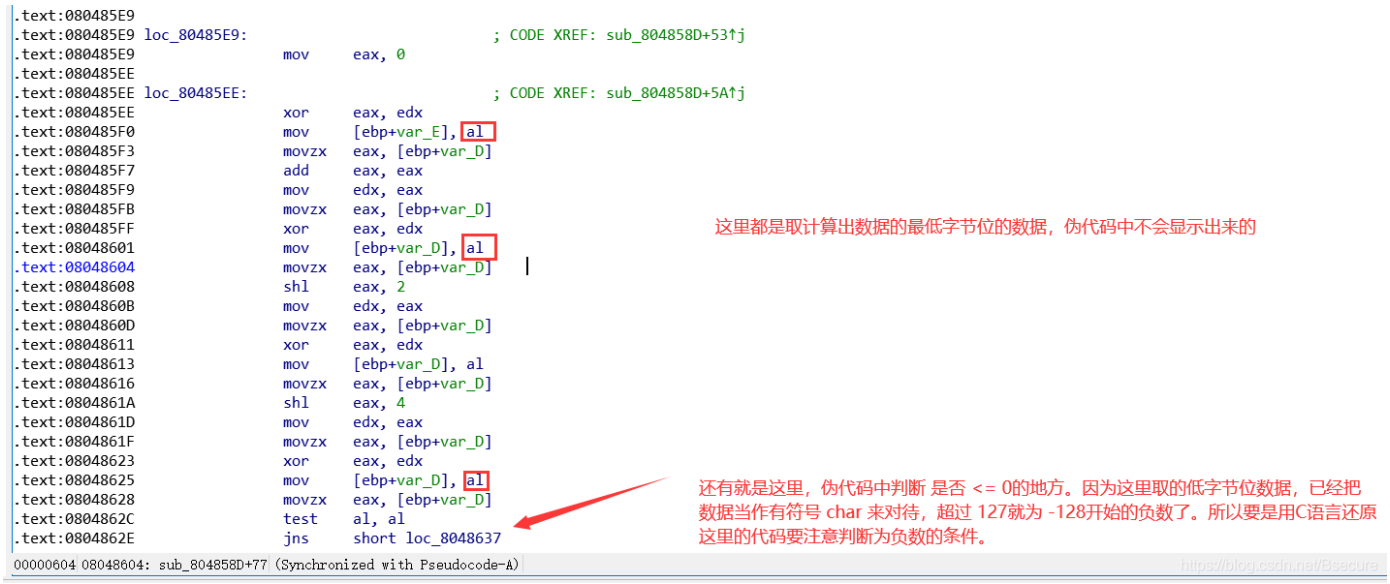
开始再次看题，被自己坑惨了。。。因为生成数据表的函数中自己只看到了生成的随机数赋值了给了数据表的第一次元素，那整个题就有点奇怪了。看了又看，才发先最后用到了随机数生成数据的，只是屏幕容不下了。。。

```
x
7; // [esp+1Ah] [ebp-Eh]
1Bh] [ebp-Dh]
1Bh] [ebp-Dh]
1Ch] [ebp-Ch]



 __int8)rand();




7;
u) == 0 )



8 ^ v8) ^ 2 * v8 ^ v8;
v4;



ned __int8)__ROR1__(v8, 4) ^ (unsigned __int8)__ROR1__(v8, 5) ^ (unsigned __int8)__ROR1__(v8, 6) ^ (unsigned __int8)__ROR1__(v8, 7) ^ (unsigned __int8)(v8 ^ *a1);
;

;
```

一直没有注意到这里。

另外就是先将每次生成索引值 v7 用C语言写出来后打印看看， 但大多数是负数，很大的数。这显然不对的，因为我们运行程序时输入的参数都是可打印字符 （ASCII: 32-127），伪代码不靠谱，果断去看看了汇编。

```
.text:080485DA                movzx   eax, [ebp+var_E]
.text:080485DE                test    al, al
.text:080485E0                jns     short loc_80485E9
.text:080485E2                mov     eax, 1Bh
.text:080485E7                jmp     short loc_80485EE
.text:080485E9 ; ---------------------------------------------------------------------------
```



其次就是 伪代码中的 __ROR__了，看了汇编知道了 ror 指令，将数据想右位移动指定的位数。以为只是简单的 >>，就这样写了整个题的爆破代码。但是始终不对，花了很时间，不甘心，就ida中动调，C语言中调试，每个步骤对比结果，才找到是 ror 这里错了。。。 原来 ror 是右移动位后会把多余的位移动到最左边，即一个圆圈转。

exp:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void generate(int a, int *result)
{
 int v7 = 1, v2 = 0, v3 = 0, temp = 0;
 int v4 = 0, v5 = 0, v8 = 1, v9 = 0;
 result[0] = a;
 do
 {
  v2 = v7 ^ 2 * v7;
  if((v7 & 0x80) == 0)
   v3 = 0;
  else
   v3 = 27;
  v7 = v2 ^ v3;
  v7 = v7 & 0xFF;

     v4 = (4 * (2 * v8 ^ v8) ^ 2 * v8 ^ v8)&0xff;
  v9 = ((16 * v4 ^ v4)&0xff);
     if ( v9 >= 0 && v9 <= 127 )
        v5 = 0;
     else
        v5 = 9;
     v8 = (v9 ^ v5);
  if(v7 < 127)
      result[v7] = (((v8 >> 4) | ((v8 & 0xF) << 4)) ^ ((v8 >> 5) | ((v8&0x1F) << 3)) ^ ((v8 >> 6) |
 ((v8&0x3F)<<2) ) ^ ((v8 >> 7) | ((v8&0xFF)<<1) ) ^ (v8 ^ a))&0xff;

 }while(v7 != 1);
}

int main(void)
{
 int *result = (int *)malloc(sizeof(int)*127);
```

```
int i = 0, value = 0, j = 0;
char a[] = "95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4
a";

for(i = 1; i < 256; i++)
{
 generate(i, result);

 if(result[84] == 0x95)
  break;
}

for(i = 0; i < strlen(a); i += 2)
{
 if(a[i] >= 48 && a[i] <= 57)
  a[i] -= 48;
 else
  a[i] -= 87;

 if(a[i+1] >= 48 && a[i+1] <= 57)
  a[i+1] -= 48;
 else
  a[i+1] -= 87;

 value = a[i+1] | a[i]*16;

 for(j = 32; j < 127; j++)
 {
  if(result[j] == value)
  {
   printf("%c", j);
   break;
  }

 }
}
}
```

```
TWCTF{5UBS717U710N_C1PH3R_W17H_R4ND0M123D_5-B0X}
--------------------------------
Process exited after 27.5 seconds with return value 96
请按任意键继续. . .
```

python

做完去看了看网上的 writeup， 发现都是用gdb写脚本搞的，然后去学习了下。

使用 gdb的 define 命令可以定义一系列的gdb指令。首先找到要下断点地址，这个在ida中很容易。

这里有2中方法，执行gdb脚本，一：直接输入defien命令，输入指令，最后执行。二：单独写成一个脚本文件使用 source 来执行。

这里直接写的脚本文件使用 source执行。

```
while($i<$total)
  b *0x80485b4
  b *0x8048707
  run T
  set $i=$i+1
  set *(char*)($ebp-0xc)=$i
  continue
  if ($eax==0x95)
    print $i
    x/127xb $esp+0x1c  #这里有一个知识点,最后总结。
    set $i=256
  stop
end
```

```
Breakpoint 2, 0x08048707 in ?? ()
$2 = 0xd6
0xffffcf1c:     0xd6    0xc9    0xc2    0xce    0x47    0xde    0xda    0x70
0xffffcf24:     0x85    0xb4    0xd2    0x9e    0x4b    0x62    0x1e    0xc3
0xffffcf2c:     0x7f    0x37    0x7c    0xc8    0x4f    0xec    0xf2    0x45
0xffffcf34:     0x18    0x61    0x17    0x1a    0x29    0x11    0xc7    0x75
0xffffcf3c:     0x02    0x48    0x26    0x93    0x83    0x8a    0x42    0x79
0xffffcf44:     0x81    0x10    0x50    0x44    0xc4    0x6d    0x84    0xa0
0xffffcf4c:     0xb1    0x72    0x96    0x76    0xad    0x23    0xb0    0x2f
0xffffcf54:     0xb2    0xa7    0x35    0x57    0x5e    0x92    0x07    0xc0
0xffffcf5c:     0xbc    0x36    0x99    0xaf    0xae    0xdb    0xef    0x15
0xffffcf64:     0xe7    0x8e    0x63    0x06    0x9c    0x56    0x9a    0x31
0xffffcf6c:     0xe6    0x64    0xb5    0x58    0x95    0x49    0x04    0xee
0xffffcf74:     0xdf    0x7e    0x0b    0x8c    0xff    0xf9    0xed    0x7a
0xffffcf7c:     0x65    0x5a    0x1f    0x4e    0xf6    0xf8    0x86    0x30
0xffffcf84:     0xf0    0x4c    0xb7    0xca    0xe5    0x89    0x2a    0x1d
0xffffcf8c:     0xe4    0x16    0xf5    0x3a    0x27    0x28    0x8d    0x40
0xffffcf94:     0x09    0x03    0x6f    0x94    0xa5    0x4a    0x46
```

因为可打印字符的ASCII: 32-127，所以我们打印出前128个数据即可。最后解密。python确实方便，各种方法直接调用即可。

```
list =  [0xd6,0xc9,0xc2,0xce,0x47,0xde,0xda,0x70
 ,0x85,0xb4,0xd2,0x9e,0x4b,0x62,0x1e,0xc3
 ,0x7f,0x37,0x7c,0xc8,0x4f,0xec,0xf2,0x45
 ,0x18,0x61,0x17,0x1a,0x29,0x11,0xc7,0x75
 ,0x02,0x48,0x26,0x93,0x83,0x8a,0x42,0x79
 ,0x81,0x10,0x50,0x44,0xc4,0x6d,0x84,0xa0
 ,0xb1,0x72,0x96,0x76,0xad,0x23,0xb0,0x2f
 ,0xb2,0xa7,0x35,0x57,0x5e,0x92,0x07,0xc0
 ,0xbc,0x36,0x99,0xaf,0xae,0xdb,0xef,0x15
 ,0xe7,0x8e,0x63,0x06,0x9c,0x56,0x9a,0x31
 ,0xe6,0x64,0xb5,0x58,0x95,0x49,0x04,0xee
 ,0xdf,0x7e,0x0b,0x8c,0xff,0xf9,0xed,0x7a
 ,0x65,0x5a,0x1f,0x4e,0xf6,0xf8,0x86,0x30
 ,0xf0,0x4c,0xb7,0xca,0xe5,0x89,0x2a,0x1d
 ,0xe4,0x16,0xf5,0x3a,0x27,0x28,0x8d,0x40
 ,0x09,0x03,0x6f,0x94,0xa5,0x4a,0x46]

flag = ""
s = "95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4a";
for i in range(0, len(s), 2):
    s1 = int(s[i:i+2], 16)
    flag += chr(list.index(s1))
print (flag)
```

总结：这道题收获还是很大。1：对ida中汇编语言的解读更熟悉了些，注意各种 al 取最低字节数据。 2：ror指令的了解，及对他的C语言用法：如 `ror a, 3` 那C语言为：`((a>>3) | ((a&7) << 5))`。 3：使用gdb写脚本。

> **gdb脚本中的 查看内存内容的方法：x/<n/f/u> n、f、u是可选的参数。**
> **n：显示的内存单元的个数，f：表示显示的格式，其中：s：字符串显示，x：按十六进制格式显示，d：按十进制格式显示变量**
> **u：按十六进制格式显示无符号整型，t：按二进制格式显示， o：按八进制格式显示，c：按字符格式显示变量。**
> **最后的u表示每个单元的大小，其中：b表示单字节，h表示双字节，w表示四字 节，g表示八字节。**
> **那上面脚本写的 x/127xb 表示将 127个的单字节单元的数据按16进制格式显示出来。**