RE-实验吧/逆向观察&bitwise

原创

Alikas 于 2019-04-13 13:37:24 发布 222 收藏

分类专栏： 逆向 文章标签： writeup RE 实验吧

本文链接：https://blog.csdn.net/q83182034/article/details/89281223

Aliikas-0x05

题目：

实验吧*逆向观察*

实验吧 bitwise

1.逆向观察

File 一下 64位ELF。

```
rev50: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Li
nux 2.6.24, BuildID[sha1]=695ac65e2d54e1d05b327b59d37c2d6eb6aba299, not stripped
```

题目叫逆向观察，那我们就直接IDA看一下，进入mian函数，代码如下：

```
if ( argc <= 1 )
 {
   puts("usage ./rev50 password");
 }
 else
 {
   src = 'sedecrem';//8315162673632404845LL,按R字符转换一下
   v6 = 0;
   v7 = 0;
   v8 = 0;
   memcpy(&dest, &src, 9uLL);
   for ( i = 0; i <= 999; ++i )
   {
     if ( !strcmp(argv[1], (&dict)[i]) && !strcmp(&dest, (&dict)[i]) )
     {
       puts("Good password ! ");
       goto LABEL_10;
     }
   }
   puts("Bad ! password");
 }
LABEL_10:
  puts(&byte_40252A);
```

函数说明：

*1.memcpy():C 库函数 void *memcpy(void *str1, const void *str2, size_t n)*

从存储区 str2 复制 n 个字符到存储区 str1。

*2.int strncmp (const char * str1, const char * str2, size_t n);*

若str1与str2的前n个字符相同，则返回0；若s1大于s2，则返回大于0的值；若s1 小于s2，则返回小于0的值。

那要打印出Good password，那不就应该让`!strcmp(argv[1], (&dict)[i]) && !strcmp(&dest, (&dict)[i])`都为1嘛，那我们输入的就应该和dest一样…真的有这么简单吗？

```
./rev50 sedecrem
Bad ! password
```

emmmmmm这里我就想到了，储存方式的不同（大小端问题），故直接把字符串反转一下：

```
./rev50 mercedes
Good password !
```

2.bitwise

题目给出了python代码，如下：

```
    user_submitted = raw_input("Enter Password: ")

    if len(user_submitted) != 10:
      print "Wrong"
      exit()


    verify_arr = [193, 35, 9, 33, 1, 9, 3, 33, 9, 225]
    user_arr = []
    for char in user_submitted:
      # '<<' is left bit shift
      # '>>' is right bit shift
      # '|' is bit-wise or
      # '^' is bit-wise xor
      # '&' is bit-wise and
      user_arr.append( (((ord(char) << 5) | (ord(char) >> 3)) ^ 111) & 255 )

    if (user_arr == verify_arr):
      print "Success"
    else:
      print "Wrong"
```

那直接写出解密算法即可，这题不是很难0.0

```
s = [193, 35, 9, 33, 1, 9, 3, 33, 9, 225]
key=[0,0,0,0,0,0,0,0,0,0]
for i in range(10):
    for j in range(128):
        if s[i]==(((j<< 5) | (j >> 3)) ^ 111) & 255 :
            key[i]=chr(j)

flag= ''
for i in key:
    flag+=i
print flag
```

得：*ub3rs3cr3t*

补充：Big-Endian和Little-Endian的定义如下：

1) Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
2) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。
举一个例子，比如数字0x12 34 56 78在内存中的表示形式为：
1)大端模式：

低地址			高地址
0x12	0x34	0x56	0x78

2)小端模式：

低地址			高地址
0x78	0x56	0x34	0x12

可见，大端模式和字符串的存储模式类似。

3)下面是两个具体例子：

16bit宽的数0x1234在Little-endian模式（以及Big-endian模式）CPU内存中的存放方式（假设从地址0x4000开始存放）为：

内存地址	*小端模式存放内容*	*大端模式存放内容*
0x4000	*0x34*	*0x12*
0x4001	*0x12*	*0x34*

·32bit宽的数0x12345678在Little-endian模式以及Big-endian模式）CPU内存中的存放方式（假设从地址0x4000开始存放）为：

内存地址	*小端模式存放内容*	*大端模式存放内容*
0x400	*0x78*	*0x12*
0x4001	*0x56*	*0x34*
0x4002	*0x34*	*0x56*
0x4003	*0x12*	*0x78*

4)大端小端没有谁优谁劣，各自优势便是对方劣势：

小端模式：强制转换数据不需要调整字节内容，1、2、4字节的存储方式一样。
大端模式：符号位的判定固定为第一个字节，容易判断正负。

三、数组在大端小端情况下的存储：
以unsigned int value = 0x12345678为例，分别看看在两种字节序下其存储情况，我们可以用unsigned char buf[4]来表示value：
Big-Endian: 低地址存放高位，如下：

高地址	
buf[3] (0x78)	*低位*

高地址	
buf[2] (0x56)	
buf[1] (0x34)	
buf[0] (0x12)	*高位*
低地址	

Little-Endian: 低地址存放低位，如下：

高地址	
buf[3] (0x12)	*高位*
buf[2] (0x34)	
buf[1] (0x56)	
buf[0] (0x78)	*低位*
低地址	

总结：题略水0.0，但还是锻炼了自己的逆向思维和写脚本的能力

参考资料：https://blog.csdn.net/ce123_zhouwei/article/details/6971544