MBE lab1 writeup

原创

BUFFER.pwn 于 2020-12-22 09:10:06 发布 78 收藏

分类专栏： 逆向工程 文章标签： MBE 逆向 pwn writeup

本文链接：https://blog.csdn.net/qq_35793285/article/details/111504707

欢迎访问我的博客: http://yanzhichen.top/

MBE（Modern Binary Exploitation）是RPI（伦斯勒理工学院）开设的PWN（漏洞利用）课程，本着由浅入深的原则，从基本的逆向分析开始，涵盖了内存溢出，编写shellcode，字符串格式化等内容。

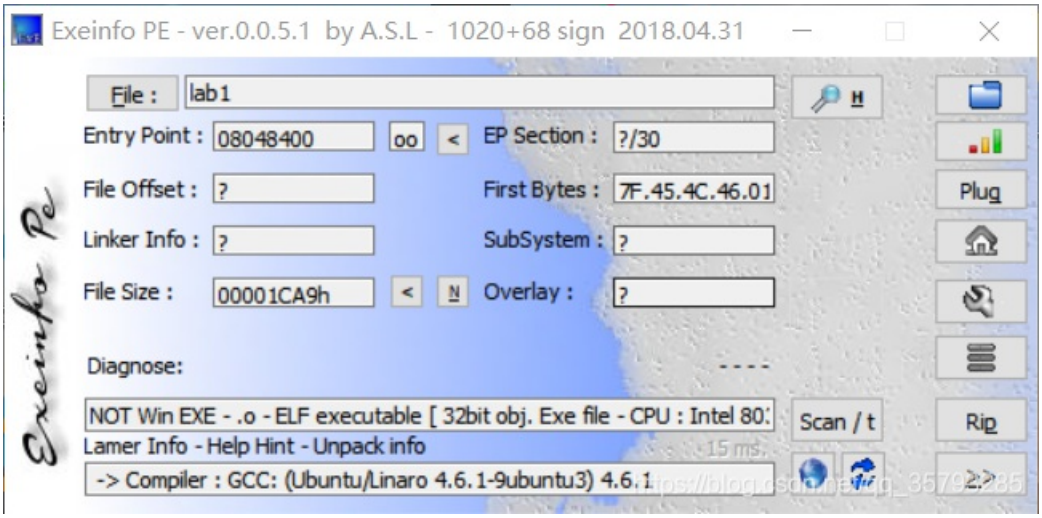
以下为 lab1 的 writeup

题目:

```
lab1 (C grade):
What is the password? Demostrate in person to a TA how you found it.
```

附件: 附件

首先，把它拖到 Exeinfo PE 里边看一下



嗯，32位，没有壳，那就直接拖到32位IDA里边F5，结果是这个样子滴。

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   unsigned int i; // [esp+20h] [ebp-28h]
4   int v5; // [esp+28h] [ebp-20h]
5   unsigned int v6; // [esp+3Ch] [ebp-Ch]
6
7   v6 = __readgsdword(0x14u);
8   printf("Enter password: ");
9   __isoc99_scanf("%s", &v5);
10  for ( i = 0; i < strlen(storedpass); ++i )
11  {
12    if ( *((_BYTE *)&v5 + i) != (*(_BYTE *)(i + 134520864) ^ (unsigned __int8)i) )
13    {
14      puts("Wrong!");
15      return 1;
16    }
17  }
18  puts("\nSuccess!! Too easy.");
19  return 0;
20 }
```


整体代码逻辑比较简单，双击一下第10行的“storedpass”，可以看到IDA跳转到了 .data 段的这个位置，并且在这个位置存放着一个字符串。

```
.data:0804A020 storedpass      db '5tr0vZBrX:xTyR-P!',0
```

这就好理解了，他就是。。嗯？？，第12行的 “134520864” 是什么鬼？？？双击一下试试，额，没反应，右键转成16进制看看：

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   unsigned int i; // [esp+20h] [ebp-28h]
4   int v5; // [esp+28h] [ebp-20h]
5   unsigned int v6; // [esp+3Ch] [ebp-Ch]
6
7   v6 = __readgsdword(0x14u);
8   printf("Enter password: ");
9   __isoc99_scanf("%s", &v5);
10  for ( i = 0; i < strlen(storedpass); ++i )
11  {
12    if ( *((_BYTE *)&v5 + i) != (*(_BYTE *)(i + 134520864) ^ (unsigned __int8)i) )
13    {
14      puts("Wrong!");
15      return 1;
16    }
17  }
18  puts("\nSuccess!! Too easy.");
19  return 0;
20 }
```

结果是"0x804A020"，咦。。有点儿眼熟，哦，就是 .data 段存放 storedpass 的位置呀，嗯，这下就全部理解了。

> storedpass存放的字符串的长度为 len，其中的每个字符所在的位置为 i（i从0开始）
> 用户输入的字符串为 v5
> 需进行匹配的对象为 storedpass存放的字符串中每个字符与其所在位置i异或组成的新字符串
> 将v5的前 len 个字符与 需进行匹配的对象 匹配，如果能全部匹配，则 Success！

开始写代码

```
storedpass = '5tr0vZBrX:xTyR-P!'
real_pass = ''
for i in range(len(storedpass)):
    real_pass = real_pass + chr(ord(storedpass[i]) ^ i)
print(f"real_pass is: {real_pass}")
```

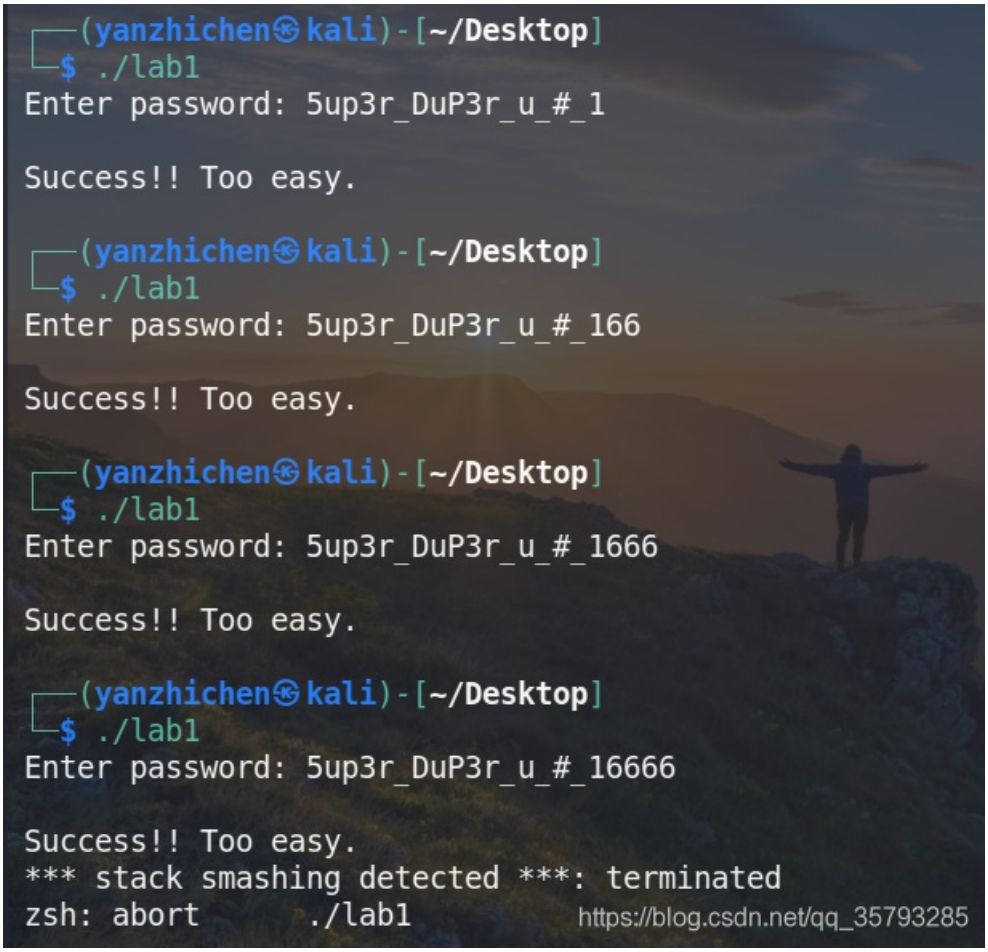
```
test ×
C:\Python36\python.exe F:/Python3_Projects/study/test.py
real_pass is: 5up3r_DuP3r_u_#_1 https://blog.csdn.net/qq_35793285
```

这里提示一下:

chr()：用一个整数作参数，返回一个对应的字符。

ord()：用一个字符做参数，返回一个对应的整数

最后测试一下:



发现结果是正确的，并且这个程序只会校验前 len(storedpass) 个字符，如果超过4个字符还会缓冲区溢出。

知道了目标设备能缓冲区溢出，那么就可以构造ROP，进而远程代码执行（RCE），拿到目标设备的shell，这些比较高级，还需要进一步的学习，随着学习的深入，以后再回过头来做补充。