**【信息安全】缓存区溢出攻击实验（3）**

本实验预计分 3 个小实验来做，本文是第一个实验。

1. [缓存区溢出攻击实验（1）](http://blog.csdn.net/PTkin/article/details/50968063)
2. [缓存区溢出攻击实验（2）](http://blog.csdn.net/PTkin/article/details/50968066)
3. [缓存区溢出攻击实验（3）](http://blog.csdn.net/PTkin/article/details/50995130)

本实验跪谢大神 [YSunLIN](https://github.com/YSunLIN" \t "_blank) 的帮助与指导 ~

不保证成功，一切皆有可能 ~

**背景介绍**

请参照实验（1）(2)。

**实验目的**

最终目的：**只给一个需要输入正确的序列号才能验证通过的exe可执行文件，要求在不知道源代码的情况下利用缓存区溢出攻击来破解该exe文件**。

注意！！！

**不知道源代码的情况下**

**不知道源代码的情况下**

**不知道源代码的情况下**

现实中，在实际的攻击中我们不可能直接拿到程序的源代码，因此最有可能的情况下就是只提供了可执行文件进行破解，打开的时候（如果exe没有防止反汇编的话）利用反汇编可以看到汇编语言，仅此而已。我们要做到的就是破解这个exe，**成功的衡量标准是通过序列号的验证**。

当然，由于我们只是一个简单的入门实验，所以在具体实验中这个exe文件的源码是有提供的，仅仅是用来提示我们到底要用什么方法才能破解成功。不过最终目的还是如上，

**实验环境**

* 系统

Windows 10 家庭中文版 (64 bit)

* IDE

[Visual Studio Community 2015](https://www.visualstudio.com/zh-cn/downloads/download-visual-studio-vs.aspx)

**源代码**

跟前面说的一样，在攻击中我们是不知道源代码的，但在本实验中我们提供了源代码仅用来提示大概可能有什么攻击的思路。

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <string.h>

int main() {

char in[20];

printf("Enter Serial Number\n");

scanf("%s", in);

if (!strncmp(in, "S123N456", 8)) {

printf("Serial Number is correct\n"); // 衡量成功的标准就是执行了该语句

}

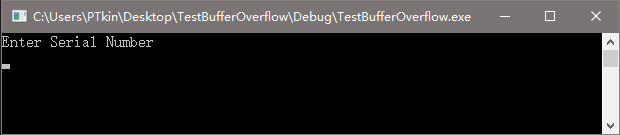
system("pause");

return 0;

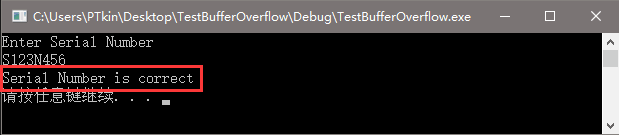
}

**程序截图**

* 正常开启



* 成功的标志是输出该行



**破解**

下面要开始思考如何破解了。虽然本实验的目的是利用 缓存区溢出攻击，不过这里还是主要考虑如何破解，不一定是利用缓存区溢出攻击。

以下破解思路大多归功于 [YSunLIN](https://github.com/YSunLIN" \t "_blank) ~

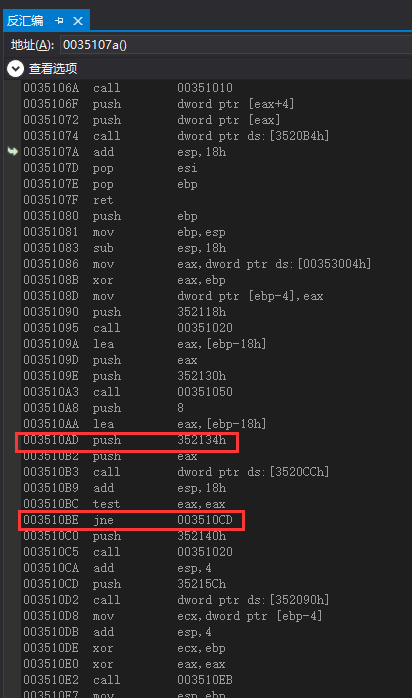
**破解方法一**

**猜测**

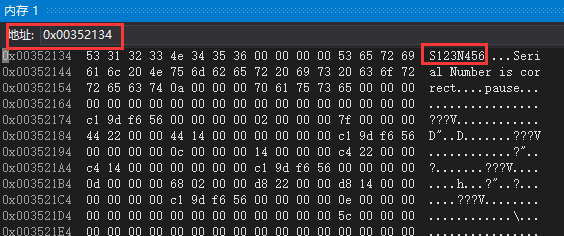
也许**匹配的字符串放在全局缓冲区**？

**破解过程**

1. 执行程序，(使用 VS 进行)反汇编得到汇编语言代码
2. 找到 jne 命令，该命令一般用来跳转，猜测这里就是判断之后的跳转语句
3. 找到 jne 之前的上一个类似 push 352134h 的语句，记住这个地址，一般这种形式可能是全局缓冲区的地址。如图：



1. 在内存中搜索地址 0x352134，可得到匹配的字符串 S123N456，如图：



**结论**

这个方法只是比较取巧，然而现在的密码基本不会用明文存储了。。输入的字符串也会经过一些处理才跟密码进行匹配，so :（

**破解方法二**

**思路**

调试的时候在汇编语言处加断点，直接修改寄存器。

这个倒是没有实现，实际上是尝试用 Visual Studio 进行反编译之后，在跳转地方加了断点，查看寄存器，可看不可修改，因此没有做进一步尝试。

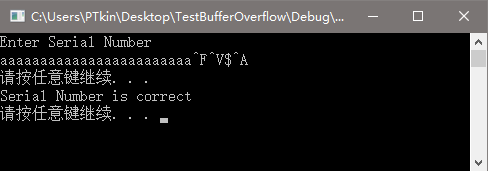
如果能找到能够对程序进行反编译、并且在执行过程中修改寄存器的工具的话，这个方法也许可行。

**破解方法三**

这个方法需要知道源代码，所以不能达到我们的最终目的。写在这里只是为了记录有这么个方法。

**使用实验（1）（2）的方法**，由于 main 函数执行完毕之后也需要返回，必然有个返回地址的跳转，我们在 if 判断语句打断点可以知道 print 语句的地址；在 return 处打断点可以知道 main 函数执行完毕之后的返回地址，搜索内存可以知道我们输入的数据与 main 函数返回地址之间相差的字节数，从而在输入数据的时候让其溢出，让 print 语句的执行地址覆盖 main 函数的返回地址。

实测可行（在知道源代码的情况下），由于跟实验(1)(2)方法类似，这里就不做过多阐述了。



**破解方法四**

**思路**

这个思路应该才是本实验想让我们实现的思路，采用**缓冲区溢出攻击**来破解。

类似前面的实验（1）和（2），我们想利用读入数据这一过程，将输入值拉长到可以覆盖栈里某些跳转的地址，达到跳转的目的。

这里有两个问题我们要解决，就是

1. **修改哪里的跳转地址？**

如果没有看到源代码，我们不知道里面有什么函数，不知道里面有什么跳转，所以这里考虑**利用 main 函数的 ret 来进行跳转**。

1. **跳转到哪里？**

既然是输入一个值如果正确则通过验证，这里必然有判断之后进行跳转的过程。我们考虑**跳转到判断的另一个分支**。

ok，思路有了，我们进行实验。

**过程**

1. 执行程序，(使用 VS 进行)反汇编得到汇编语言代码
2. 找到 j 开头的汇编指令，该指令一般用来跳转，猜测这里就是判断之后的跳转语句，得到判断之后跳转的地址
3. 在汇编语言代码中找到 ret 指令，打上断点，程序运行结束之前我们可以劫到 main 函数的返回地址（这里可能要在内存中搜索 esp）

**为什么这里没有图**

因为实际操作中有个问题，就是寻到需要跳转到的地址之后我们要输入数据，让缓冲区溢出对 main 函数的返回地址进行覆盖，可是这里有几个点：

1. 汇编指令地址是 32 bit 的
2. 输入的字符串中每个字符占 1 byte，即 8 bit
3. 也就是说我们要用 4 个字符来表示这个地址
4. 输入字符串都以 ASCII 码方式来读取
5. ASCII 码能表示的字符只有 128 个

**坑爹的地方来了！**

假如地址中出现 7F（十进制的 127）之后的地址，我们是无法用 ASCII 方式表示出来的…

po主卒。(┬＿┬)

**最后几句话**

1. 攻击不一定成功
2. 一边猜测一边尝试着攻击
3. 运气肯定是需要的
4. 好的工具可以节省很多功夫
5. 信息安全水很深:-P