**缓存区溢出攻击实验（2）**

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)

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

**简述**

背景介绍请参照实验（1）。

本实验是使用 Visual Studio （以下简称 VS）作为 IDE，重复上面的实验(1)。本来不应该单独作为一个实验来讲述，然而在实际操作的过程中发现了一些问题，因为 VS 是个比较强大的 IDE，因此在编译过程中多了一些安全性的考虑，所以对缓冲区攻击有了一定的防范，因此在这里做一些简单的补充。

**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)

下面放上最终版的代码，与实验(1)的源代码有一些不同：（建议看完下文再回头看代码）

**源代码**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

void overflow(const char\* input)

{

char buf[8];

printf("Virtual address of 'buf' = Ox%p\n", buf);

strcpy(buf, input);

}

void fun()

{

printf("Function 'fun' has been called without an explicitly invocation.\n");

printf("Buffer Overflow attack succeeded!\n");

}

int main(int argc, char\* argv[])

{

printf("Virtual address of 'overflow' = Ox%p\n", overflow);

printf("Virtual address of 'fun' = Ox%p\n", fun);

printf("Virtual address of 'main' = Ox%p\n", main);

char input[30] = "01234567\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc";

char add[5];

\*((int\*)add) = (int)fun;

add[4] = 0;

strcat(input, add);

overflow(input);

return 0;

}

**实验步骤**

实验步骤与实验(1)类似，下面只进行简单描述，具体不懂可参照实验(1)。

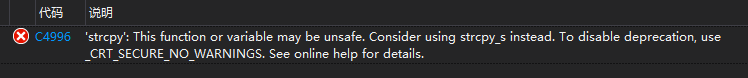
1. 先找到 fun 函数的入口地址 address\_fun，然后找到 main 函数中在 overflow 函数执行完毕之后返回的地址 address\_ret。
2. 找到 overflow 函数调用过程中，buf 所在地址 address\_buf，数出 address\_buf 与 address\_ret 之间的字节数，计算好准备写入缓冲区 buf 的数据的大小，以使能够让 overflow 执行完毕之后跳到写入的 address\_fun。
3. 修改写入缓冲区的数据，使其覆盖栈中其它的数据，让 address\_fun 正好覆盖 overflow 函数执行完毕之后返回的地址 address\_ret。执行程序之后，程序如果调用了 fun 函数，则说明缓存区溢出攻击成功。

**然而实验结果呵呵哒。下面将展示个人实验过程中出现的问题与解决方式。**

**遇到的错误与解决方式**

**错误 #1**

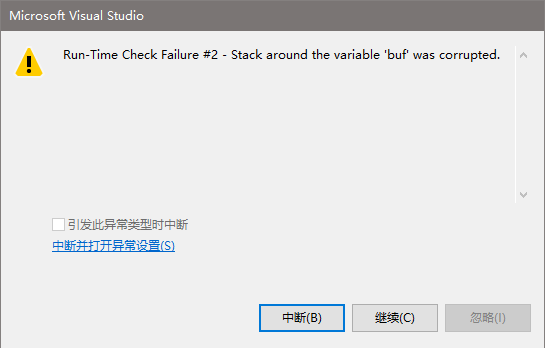
**解决方案生成出错**，报错如下：



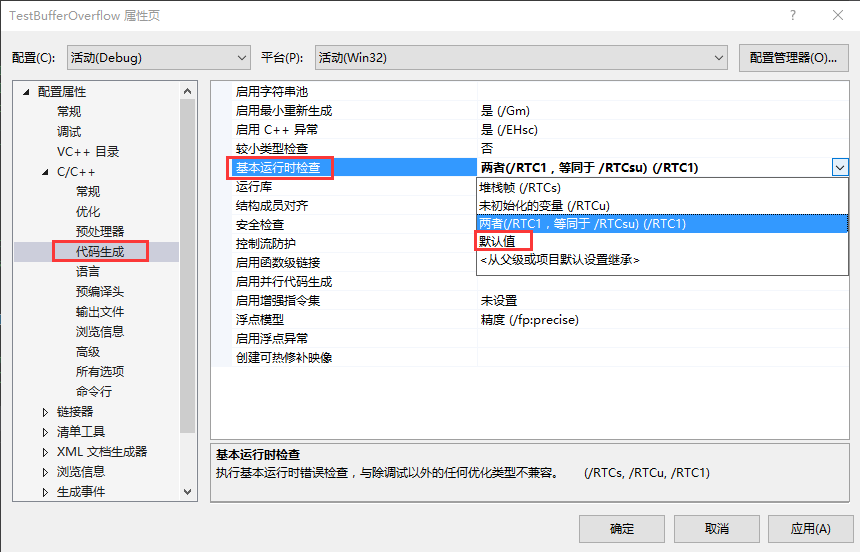
这个好解决，只是因为使用了不安全的函数，可以通过在文件头加入 #define \_CRT\_SECURE\_NO\_WARNINGS 来解决。

**错误 #2**

继续运行程序，报了**运行时检查失败**，如下：

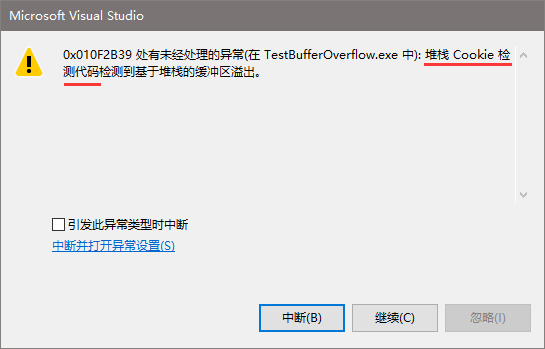


这是因为 VS 有一种**运行时检查机制**（具体原理见下文“思考”部分），关闭即可。右键点击项目，选择“属性”，将下图所示的 基本运行时检查 改为 默认值。

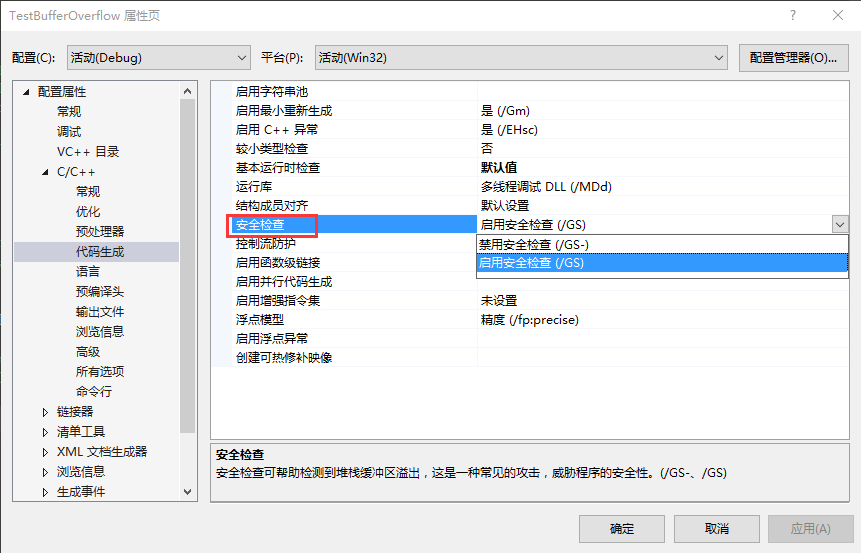


**错误 #3**

继续运行程序，报了**堆栈 Cookie 检测代码检测到缓冲区溢出**错误，如下：

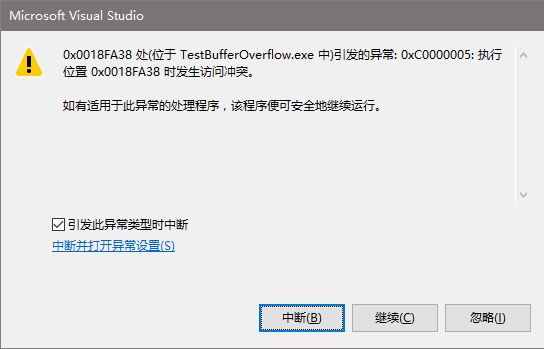


依然是因为 VS 有一种 **安全检查(\GS)机制**（具体原理见下文“思考”部分），仍然需要我们关闭。如下所示，选择 禁用安全检查(\GS)即可。



**错误 #4**

继续运行程序，报了**访问冲突**错误，如下：

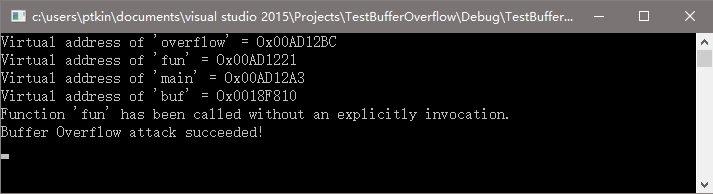


仔细检查代码、检查汇编语言里的逻辑地址、检查栈空间里的数据，我们会发现在栈空间里面，buf 与 overflow 函数执行完毕之后返回的地址之间，在我们解决上面两个问题之前相差 16 个字节，而在解决上面两个问题之后，它们之间只相差了 12 个字节。说明了 VS 的这两种机制（基本运行时检查机制 与 \GS安全检查机制）可能在栈空间里面使用了某种手段来保证访问不出错。这一点我们将在下文的思考里面讲述。这里略过原理。

既然知道了出错的地方，那我们就修改一下将要写入缓冲区的数据修改一下，让 overflow 函数执行完毕之后跳到我们想让它执行的函数 fun，即可。

**错误 #5**

继续运行程序，依然报了**访问冲突**错误。

这回又是哪个环节出现了问题？明明返回的地址对上号了，为什么依然访问冲突。在这里我们重新 review 代码与整个过程。很容易的我们就能发现，每次修改将要写入缓冲区的数据之后， build&run 的程序里面 fun 函数的入口地址都不一样:(   
在大概测了十几次之后，发现每次 fun 函数的入口都不同，而且没有什么固定的规律（如果有人能够发现请教我），没有什么好的解决思路，于是就修改了 main 函数，强行将 fun 函数的入口地址添加在了将要写入缓冲区的数据当中，于是解决了问题（虽然方法不太行）。   
结果如下：   


**done!**

**思考**

**基本运行时检查机制**

下面这篇博客已经写得很棒了~ 墙裂推荐~

* [关于VS2008中 Basic Runtime cheks的解释 - Renaissance的专栏 - 博客频道 - CSDN.NET](http://blog.csdn.net/brighttown/article/details/4183238)

这里对基本运行时检查机制做一个简单的概述：（总结自上面这篇博客）

1. **/RTCc开关**

检查在进行类型转换的保证没有不希望的截断（Truncation）发生。

1. **/RTCu开关**

这个开关的作用是打开对未初始化变量的检查。u代表未初始化的变量检查（uninitialized variable check）

1. **/RTCs开关**

这个开关是用来检查和Stack相关的问题。s代表堆栈检查（stack check）。

* 1. Debug模式下把Stack上的变量初始化为0xcc，检查未初始化的问题
  2. 检查数组变量的Overrun
  3. 检查ESP是否被毁坏

**\GS 安全检查机制**

这里也参考了一篇文章，来自老牌的看雪安全论坛，讲解十分详细，也是十分推荐：

* [【翻译】深入剖析编译器安全检查机制 - 看雪安全论坛](http://bbs.pediy.com/showthread.php?t=49873)

简单来说，就是

**/GS 参数在缓冲区和返回地址之间提供了“缓冲带”，或者称为cookie。如果溢出会覆盖返回地址，那么这个溢出必须先覆盖返回地址和缓冲区之间的cookie。**

当然，参考的这篇文章远远不止讲述了关于 \GS安全检查机制的由来与原理，更是深入剖析了缓冲区溢出问题（包括Windows XP中系统函数内部大量的带有“cookie”这个字符串的符号）另外，这篇文章是翻译作品，原文作者是 VS 开发团队的成员 Brandon Bray。