VC 中利用_GS开关防止缓冲区溢出 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/253/2021_2022_VC___E4_B8_AD_E5_88_A9_c97_253831.htm 缓冲区溢出通常表现为一个最为常见的漏洞而存在于今天的各种软件之中，黑客可以用恶意的输入，从而更改程序的执行流程，由此入侵相应的进程、电脑、或整个域。如果进程运行于一个高度受信的账户之下，如管理员或本地系统账户，那么黑客带来的破坏将是极其严重，并有潜在广泛传播的危险。近几年来爆发的一些"知名"病毒，如红色代码、冲击波、震荡波等等，都源于C/C 代码缓冲区溢出的结果。 从程序的角度来看，缓冲区溢出只是一个再简单不过的编程错误--都是关于复制一个内存区域的内容到另一个内存区域，而目标内存区域容量太小无法容纳。以下的代码作了简单的演示： char* source = "A reasonably long string". char dest[10]. ::strcpy(dest, source). 在本例中，源字符串的长度为25个字符（包括了空结束符），它对目标内存块来说，无疑太大了，而目标内存块声明在堆栈上；当此代码执行时，将会破坏掉原有堆栈，程序会因为一个访问违例而崩溃。如果此源内存块由外部第三方提供，那么就有可能存在一个漏洞，因为它允许传入函数的内存块以一种特定的方式修改堆栈。 当在C/C 中调用一个函数时，调用函数的返回地址被存放在堆栈中，因此在被调用函数执行完毕时，执行流程能重新返回到原处。如果调用了一个可能包含潜在缓冲区溢出的函数，返回地址可能会被修改，而且执行流程将会跳到缓冲区数据中指定的地方。通过改变函数的返回地址，攻击者可获取进程中任意位置的代码以执行，一般而言，

主要可以两种方式被利用：如果带有漏洞的程序是已知、且容易访问到的，攻击者可查找某函数的地址，这通常会在所有进程实例的一处固定地址处被找到；并修改堆栈，等着此函数被调用。 要执行的指令可作为缓冲区的一部分传递到进程地址空间，攻击者利用此来完成攻击。 防范缓冲区溢出 防范缓冲区溢出最简单的方式是限制复制的数据大小，使其不能大于目标缓冲区容量。虽然此方法看上去微不足道，但实际上，经验证明，要在那些大型的C/C 代码中，完全消除了缓冲区溢出的隐患，是件非常艰巨的任务。另外，使用如.NET或Java这样的受托管技术，也能极大地降低缓冲区溢出的危险，但把大型项目移植到此技术上，实施起来不太可能也不适当。 基于堆栈的缓冲区溢出可如此简单地被利用的原因在于，编译器生成的指令，会把函数的返回地址存储在堆栈中，但要认识到，编译器在这个问题中，只扮演了一个小小的角色。从Visual C .NET（7.0）开始，Visual C 开发小组采取了一种方法，可从编译器方面减少此类问题发生的机率，他们在堆栈中保存函数返回地址的数据之下，插入了一个带有已知数值的cookie，由此，如果缓冲区溢出改变了函数的返回地址值，同样也会覆盖这个cookie，而在函数返回时，一般会对这个cookie进行检测，如果检测到cookie已被修改，就会抛出一个安全异常，而如果这个异常未被处理，此进程就会终止。以下的代码演示了一个带有安全异常处理方法的简单程序： void _cdecl sec_handler(int code, void *) { if (code == _SECERR_BUFFER_OVERRUN) { printf("检测到一个缓冲区溢出。\n"). exit(1). } } int main() { _set_security_error_handler(sec_handler). //主程序代码在此省略。 } Visual C .NET 2003

（7.1）通过移动易受攻击的数据结构--（如异常处理方法的地址）--到堆栈中位于缓冲区之下的某个位置，增强了缓冲区溢出的保护力度。在编译器的7.0版本中，可通过破坏缓冲区与cookie之间的敏感数据，绕过安全cookie所提供的保护；然而，在新版本的编译器中，已把这些数据移到位于缓冲区下的一个区域，现在，想要通过修改这些数据而达到溢出，似乎是不太可能了。 在C 编译器6、7.0、7.1中，堆栈概念上的布局，并演示了堆栈由高地址向低地址空间方向增长，这也是当程序执行时，堆栈增长的方向。堆栈向下增长，正是导致缓冲区溢出的主要原因，因为溢出会覆写在比缓冲区更高的内存地址空间上，而此正是易受攻击数据结构的栖身之地。 除了把异常处理方法等信息移到堆栈中数据缓冲区之下，Visual C .NET 2003的链接器也把结构化异常处理方法的地址放到可执行文件的头部中。当异常发生时，操作系统可以检查堆栈中的异常信息地址，是否符合记录在文件头信息中的异常处理方法，如果情况不符，异常处理方法将不会执行。比如说，Windows Server 2003就可检查结构化异常信息，而此项技术也在Service Pack 2中移植到了Windows XP上。
而Visual C 2005（8.0）在此基础上又更进了一步，通常当有函数调用发生时，如果其中的一个本地缓冲区超出限度了，攻击者可能改写堆栈中在此之上的任何东西，包括异常处理、安全cookie、帧指针、返回地址和函数参数。而这些值的大多数被不同的机制所保护（如安全异常处理），但对一个有函数指针作参数的函数来说，仍有机会被溢出。如果一个函数接受一个函数指针（或结构、类中包含有函数指针）作为参数，攻击者就有可能改写指针中的值，使代码执行任何他想

要的函数。鉴于此，Visual C 2005编译器将分析所有可能存在此漏洞的函数参数，并复制一份函数参数--并不使用原有的函数参数，把它放在堆栈中本地变量之下。如果原有函数参数被溢出改写了，只要副本中的值仍保持不变，整个函数就不会被攻破。 应用缓冲区保护 只需简单地打开/GS编译器开关，就可启用缓冲区保护。在Visual Studio中，此开关可在"C/C "选项页的"代码生成"选项中找到。默认情况下，在Debug配置下为关，而在Release配置下为开。 如果用最新版本的编译器进行编译，并生成结构化异常信息，那么在默认情况下，安全结构化异常处理将是打开的，另外，也可以使用/SAFESEH:NO命令行选项来关闭安全结构化异常处理，在Visual Studio的工程设置中，是没办法关闭安全结构化异常处理的，但仍可在链接器中使用此命令行选项来完成。 /GS及更远的安全前景 仅仅是打开一个编译器开关，不会使一个程序彻底变得安全，但在安全漏洞以各种形式出现的今天，它将有助于使程序更加安全。基于堆栈的缓冲区溢出是安全漏洞中的一大类，但随着黑客攻击技术的不断更新，相信它的谢幕，还有一段很长的路要走。 在Microsoft正式的术语中，/GS和SAGESEH均为软件强制的数据执行保护（DEP），软件强制的DEP也能以硬件的方式实现，如在实现了此功能的CPU中，如果数据出现在被标记为"不可执行"的内存页中，将不会执行它。Windows XP SP2及Windows Server 2003现在已支持这些技术，目前市面上的大多数的32位CPU及全部的64位CPU，都支持No Execute（NX）这类安全增强技术。 任何一个好的安全系统，均有多层防范措施应对安全威胁。本文所涉及的编译器开关，它能防范或减少普通编码错误所

带来的安全隐患，而且它具有易于使用和低成本的特点，在这场没有硝烟的战争中，不失为一个好的解决方案，绝对值得你的程序采用。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com