Vista 的抵御缓冲区溢出攻击技术研究 *

魏立峰, 江 荣, 赵 栋
(国防科学技术大学 计算机学院, 长沙 410073)

摘 要: 缓冲区溢出攻击是一种在互联网时代被广泛利用并危害严重的主要攻击方式。分析了缓冲区溢出攻击的基本原理,总结了缓冲区溢出攻击的关键步骤,并研究分析了 Windows Vista 的抵御缓冲区溢出攻击的四种关键技术,包括地址空间配置随机化(ASLR)、数据执行保护(DEP)、GS 栈保护和安全(structured exception handling, SEH)等技术;最后对 Windows Vista 抵御缓冲区溢出的整体效果进行了分析,指出了 Vista 仍然不能完全抵御缓冲区溢出攻击。

关键词: 缓冲区溢出; Vista; 地址空间配置随机化; 数据执行保护; GS 栈保护; 安全 SEH
中图分类号: TP301 **文献标志码**: A **文章编号**: 1001-3695(2010)05-1853-03
doi:10.3969/j.issn.1001-3695.2010.05.070

Research for resisting buffer overflow attack technologies of Vista

WEI Li-feng, JIANG Rong, ZHAO Dong
(*School of Computer Science, National University of Defense Technology, Changsha* 410073, *China*)

Abstract: Buffer overflow attack is a main attack method in internet, which has been widely used and could cause great damage. Analyzed the principle of buffer overflow attacks and summed up the key steps of buffer overflow attacks, researched and analyzed the four key technologies for resisting buffer overflow attack of the Windows Vista, including ASLR, DEP, GS stack protection, and safe structured exception handling (SafeSEH). finally analyzed the total effect of resisting buffer overflow of Windows Vista, and pointed out that Vista couldn't resist the buffer overflow attack absolutely.

Key words: buffer overflow; Vista; ASLR; DEP; GS stack protection; safeSEH

0 引言

随着信息技术的高速发展,计算机和互联网在全球迅速普及,人类社会已经进入了信息时代。然而,在人们享受信息时代带来便捷的同时,木马、病毒、蠕虫等不良因子却在时刻威胁着网络信息安全。网络突破是网络入侵的重要环节,漏洞利用是网络突破的重要技术,由于历史原因,缓冲区溢出漏洞普遍存在,已成为信息系统安全的主要威胁之一。操作系统的安全防护机制对缓冲区溢出漏洞利用有很大影响。如果能在操作系统层检测和阻止缓冲区溢出攻击,将大大提升系统的安全性。

Windows Vista 是微软公司作为 Windows XP 和 Windows 2003 的升级产品,是 2006 年推出的新一代操作系统。针对缓冲区溢出威胁,Windows Vista 采取了一系列安全措施,其中保护溢出点的具体措施有:地址空间配置随机化 ASLR(address space layout randomization)、数据执行保护 DEP(data execute prevention)、GS stack protection 和 SEH 的保护机制 safeSEH 等。这些安全措施构成了 Vista 的内存保护机制,从各个方面严格限制了缓冲区溢出漏洞利用技术的运用,给 Vista 的缓冲区溢出攻击造成了很大的困难。

1 冲区溢出攻击的基本原理分析

造成缓冲区溢出的根本原因是某些编程语言如 C 语言固有的不安全性。为了追求性能,在 C 语言中对指针和数组的操作没有进行自动的边界检查。C 语言中的许多对字符串操作的标准库函数都是不安全的,如 strcpy()、strcat()、gets()等。当利用这些函数向缓冲区写入大量数据时,就有可能超过缓冲区的容量而导致溢出,从而覆盖了与缓冲区相邻内存区的其他数据。如果写入的数据是攻击者精心构造的恶意代码 shellcode,返回地址等指针被覆盖成了指向攻击者恶意代码的指针,当调用该指针时,就将跳转并执行攻击者的恶意代码 shellcode,达到入侵的目的。

按照缓冲区的位置来分,缓冲区溢出主要分为堆栈缓冲区溢出、堆缓冲区溢出和静态数据区 bss 段溢出[1]。现在利用比较多的是堆栈溢出和堆溢出。

不失一般性,以 Windows 堆栈缓冲区溢出攻击来阐述缓冲区溢出攻击原理。在 Intel x86 体系结构下,栈是从内存高地址向低地址方向增长的,而对缓冲区写入操作则是从内存低地址向高地址方向进行的,如图 1 所示。因此,如果在往缓冲区写入数据之前没有进行越界检查,就有可能覆盖比缓冲区地址更高的系统的数据结构,如返回地址、函数指针、例外处理节点 SEH 等。

缓冲区溢出攻击就是构造超过缓冲区容量的特定数据,覆盖特定指针,如返回地址、堆头指针等,在特定的时候如程序返回时,使程序转向执行攻击者的恶意代码 shellcode,进而导致系统崩溃或使攻击者获得系统控制权等[2]。综上所述,缓冲

收稿日期: 2009-10-09; **修回日期**: 2009-11-16 **基金项目**: 国家"863"计划资助项目(2007AA01Z461)

作者简介:魏立峰(1973-),男,副研究员,博士,主要研究方向为计算机安全与系统软件(wei_lifeng@yahoo.com.cn);江荣(1984-),男,福建龙岩人,硕士研究生,主要研究方向为信息安全;赵栋(1985-),男,硕士研究生,主要研究方向为信息安全.

区溢出可以总结出几个关键步骤,如图 2 所示。完成一次完整的缓冲区溢出攻击,这几个关键步骤缺一不可,任何一个步骤被中断都将造成攻击失败。

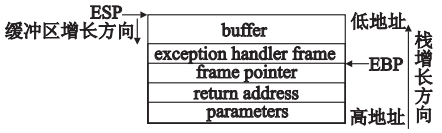

图1 栈结构示意图

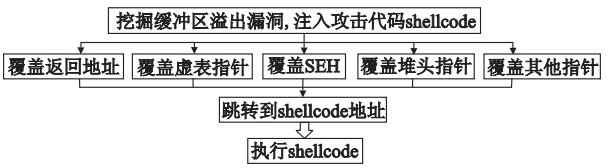

图2 缓冲区溢出攻击示意图

2 Vista 抵御缓冲区溢出攻击关键技术

2.1 地址空间分布随机化(ASLR)

ASLR 是一项通过将系统关键地址随机化,从而使攻击者无法获得需要跳转的精确地址的技术。通过第 1 章分析可知,一次成功的缓冲区溢出攻击需要跳转到一个固定的地址,通过运行该地址所包含的指令如 jmp esp、jmp eax 等,来达到改变程序流程的目的,否则只能造成程序报错或崩溃。同时,在改变了程序流程后,恶意代码往往需要调用一些系统函数如新建用户、打开某端口等,来达到某种攻击目的,这也需要找到这些函数的固定调用地址。在 Windows XP 及之前的操作系统中,这样的固定地址往往存在于某个系统 DLL 如 kernel32. dll、可执行文件本身、栈数据或 PEB 中。

ASLR 随机化的关键系统地址包括:PE 文件(exe 文件和 dll 文件)映像地址、堆栈基址、堆地址、PEB 和 TEB 地址。

当程序启动,将执行文件映射到内存时,系统将在原来映像基址的基础上加上一个取自 RDTSC counter 的 8 位随机数 α,作为新的映像基址。为了保证新的映像基址与原来的不同,这个随机数 α 从不为 0(当 α 为 0 时,系统将其修改为 1),所以映像基址将有 255 种可能。但这使 1 出现的概率是其他随机数的两倍。因此在 Vista SP1 中,微软改正了这个问题,将随机数 α 整除 254 并加 1,使 α 的取值范围限定为 1 ~ 254,保证了每个数值出现的概率为 1/254[3]。映像基址地址的随机化是在内核模块 ntoskrnl. exe 未公开函数 MiSelectImageBase ( ) 中完成的,具体细节伪代码如下[4]:

```
if((nt_header - >Characteristics & IMAGE_FILE_DLL) = = 0)
{
RelocateExe:
unsigned int Delta = (((RDTSC > >4)%0xFE) +1) *0x10000;
dwImageSize = image size rounded up to 64KB
dwImageEnd = dwImageBase + dwImageSize;
if (dwImageBase > = MmHighestUserAddress||
dwImageSize > MmHighestUserAddress||
dwImageEnd < = dwImageBase||
dwImageEnd > MmHighestUserAddress)
return 0;
if (arg0 - >dwOffset14 + Delta == 0)
return dwImageBase;
if (dwImageBase > Delta){
dwNewBase = dwImageBase - Delta;
}
else{
dwNewBase = dwImageBase + Delta;
if (dwNewBase < dwImageBase||
dwNewBase + ImageSize > MmHighestUserAddress)||
dwNewBase + ImageSize < dwImageBase + ImageSize)
return 0;
}
...
return dwNewBase;
}
```

具体过程如下:

a)计算随机偏移量 α;

b)验证原始映像基址和大小;

c)根据验证结果,计算新的映像基址;

d)验证新的映像基址。

ASLR 技术很好地打乱了系统中存在的固定地址,使得攻击者几乎不可能从进程的内存空间中找到稳定的跳转地址。如果通过暴力猜测的方法突破 ASLR 的防护,必须不断尝试,不仅成功率低,还可能会造成程序或系统的不稳定,从而引起管理员的警觉。

2.2 数据执行保护(DEP)技术

DEP 是一项阻止代码在标有 non-executable 标志位的内存页上执行的保护机制。DEP 分为软件 DEP 和硬件 DEP。硬件 DEP 需要 CPU 的支持,需要 CPU 在页表增加一个保护位 NX(no execute),来控制页面是否可执行。如果 CPU 不支持的话,Windows 将采用软件检查页面是否可执行的方法实现 DEP,但效果不是很好[5]。现代的 CPU 一般都支持硬件 NX,所以现在一般采用的是硬件 DEP。

在 Windows 中,系统默认包含执行代码和加载 DLL 文件的 txt 段的页面为可执行页面,其他的页面不包含执行代码。而在很多情况下,攻击者往往把要执行的恶意代码放在堆栈或堆的缓冲区中。在 Windows XP 及其之前的操作系统版本中,没有对代码执行区域进行限制,从而攻击者可以根据自身需要将恶意代码放置在任何方便的位置执行。DEP 机制采用硬件或软件的方法将这些敏感区域设置成为 non-executable 不可执行后,即使攻击者溢出成功,并跳转到恶意代码的地址,系统将弹出非法访问例外,恶意代码无法继续执行,从而有效地阻止了缓冲区溢出攻击。

2.3 GS 技术

GS 是一项检测缓冲区溢出的技术。该技术是 VC ++ 编译器提供的一个编译选项。如果在 Visual Studio 2005 编译中将该选项置上,编译器将增加一段额外代码,使程序在调用函数时,在缓冲区和函数返回地址增加一个 32 位的随机数 security_cookie,并将可能遭受攻击的指针备份在缓冲区之上,程序只使用备份的 SafeParameters,如图 3 所示。反汇编代码如下:

```
push ebp
mov ebp, esp
sub esp, 214h
mov eax, ___security_cookie
xor eax, ebp mov [ebp + var_4], eax
```

同时编译器在函数返回时增加额外代码,调用__security_check_cookie 函数检查 security_cookie 是否被修改,如果发现被修改,说明发生了缓冲区溢出,则执行 RtlUnhandledExceptionFilter 函数产生异常,并调用 NtTerminateProcess 函数终止当前进程并报错[6]。

Security_cookie 的值是在进程启动时随机产生,它的原始

存储地址只与程序映像基地址有关，它们之间的相对位移是一定的。由于 ASLR 技术，程序映像基地址是随机的，因此 security_cookie 的原始存储地址也是随机放置的，攻击者无法对其进行窜改。

当攻击者进行栈缓冲区溢出攻击时，要覆盖返回地址或其他指针以达到跳转的目的，在溢出缓冲区覆盖这些指针的同时，必定先覆盖修改了 security_cookie 的值，在程序返回对 security_cookie 进行检测时，就会发现缓冲区溢出。因此 GS 技术能够很好地检测基于栈的缓冲区溢出攻击。

2.4　SafeSEH 技术

SEH 即结构化异常处理，是 Windows 操作系统提供给程序设计者的强有力的处理程序错误或异常的武器。程序设计者可以根据自身需要，定义程序发生各种异常时相应的处理例程。如果程序源代码中使用了__try{}__except{}或者 Assert 宏等异常处理机制，编译器将最终通过向当前函数栈帧中安装一个 SEH 来实现异常处理。SEH 结构 exception handler frame 在栈中的位置如图 3 所示。在 SEH 节点链表中，每个节点都包含两部分，即下一节点地址和异常处理例程句柄。在内存中按从低到高的次序先后排列，每个节点分布在栈中的不同区域。栈中的多个 SEH 通过链表指针在栈内由栈顶向栈底串成单向链表，位于链表最顶端的 SEH 通过 TEB（线程控制块）0 字节偏移处的指针标志，如图 4 所示。当程序发生异常时，系统会中断程序，并首先从 TEB 的 0 字节偏移处取出距离栈顶最近的 SEH，使用异常处理函数句柄所指向的代码来处理异常。当该异常处理函数运行失败时，将顺着 SEH 链表依次尝试其他的异常处理函数。如果程序安装的所有异常处理函数都不能处理，最后一个节点的下一节点地址为 0xFFFFFFFF，异常处理例程为系统默认例程，通常这个例程会弹出一个错误对话框，然后强制关闭程序。

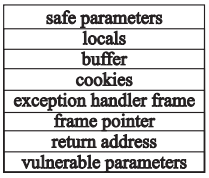

图3　采用/GS选项编译后的栈结构

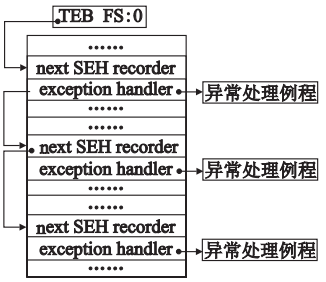

图4　SEH链表

通过精心构造，攻击者可以通过缓冲区溢出覆盖异常处理例程句柄，将其替换为指向恶意代码 shellcode 的地址，并触发相应异常，从而使程序流程转向执行恶意代码。

SafeSEH 就是一项保护 SEH 节点的技术。微软在编译器中加入了/SafeSEH 选项，通过反汇编可知，采用该选项编译的程序将 PE 文件中所有合法的异常处理例程的地址解析出来制成一张表，放在 PE 文件的数据块（LOAD_CON2FIG）中，用于匹配检查。在该 PE 文件被加载时，系统读出该表的地址，使用内存中 SharedUserData 的一个随机数加密，将加密结果、模块的基址、模块的大小、SEH 节点的个数生成一条 LISTEN-TRY，放入 ntdll.dll 的数据内存中（LdrpInvertedFunctionTable），即 SEHIndex 结构。对 SEH 节点地址表的加解密都通过 RtlDecodeSystemPointer 函数来完成。在 PE 文件运行期间，如果发生异常需要调用异常处理例程，系统会调用该函数解密获得了 SEH 节点地址表，进而通过逐个检查该例程在表中是否有记录：如果没有则说明该例程非法，不执行该例程并中断进程。通过严格的匹配检查，系统能够发现 SEH 节点是否被修改，从而能够很好地检测覆盖 SEH 的缓冲区溢出攻击。

3　Vista 抵御缓冲区溢出安全性分析

针对缓冲区溢出攻击的几个关键路径，Vista 通过 ASLR、DEP、GS 和 SafeSEH 等关键技术进行防护，在不同关键路径点给攻击者制造了层层障碍，增加了缓冲区溢出攻击难度，提升了系统的安全性。如图 5 所示，GS 栈保护技术保护了返回地址和函数指针，SafeSEH 技术保护了 SEH 节点，ASLR 技术使攻击者无法定位 shellcode 地址和系统调用地址，DEP 技术使位于栈和堆等位置的 shellcode 无法正常执行。这些技术各自独立又相互补充，构成了 Vista 抵御缓冲区溢出机制的完整体系。Windows Vista 抵御缓冲区溢出机制的工作流程可以总结为如图 6 所示。

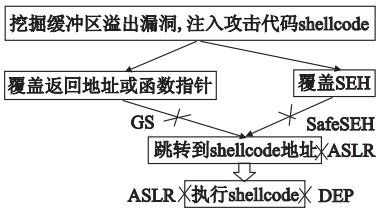

图5　Vista对缓冲区溢出攻击的阻断示意图

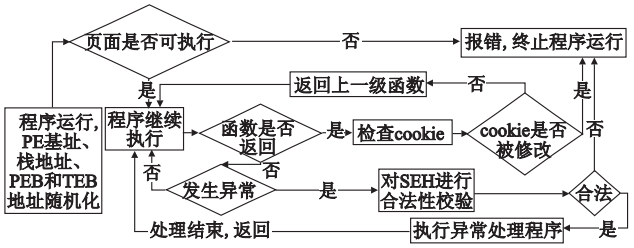

图6　Vista抵御缓冲区溢出攻击工作流程示意图

虽然 Windows Vista 能对大部分缓冲区溢出攻击起到很好的保护作用，但也存在着一些不足之处，主要表现在以下几个方面：

a）全面性。Vista 内存保护机制中的关键技术例如 GS、SafeSEH 等都需要编译器的支持，要求操作系统中文件使用/GS 和/safeSEH 等编译选项进行编译后才能发挥作用。而在操作系统组件中，仍有部分文件未被保护，这就使得攻击者有机可乘。这就存在了木桶效应，进程中只要存在一个没有被保护的缓冲区溢出漏洞，攻击者仍然可以进行攻击。

b）兼容性。考虑到与以前的 Windows 操作系统和第三方软件的兼容性，Vista 中部分组件的内存保护技术并没有开启。例如，由于历史的原因，第三方软件甚至 Windows 操作系统自身是允许在栈或者堆中执行代码的，而一旦将进行 DEP 保护，这些程序将无法运行。而攻击者往往利用这一点在第三方软件上寻找漏洞以绕过内存保护机制，来达到攻击的目的。ms07-017ANI 光标漏洞就是一个很典型的例子，攻击者通过在 IE 浏览器中构造畸形的 ANI 文件，将 shellcode 放置在不支持 ASLR 等技术 flash 插件中，就能够旁路 Vista 的内存保护机制。

c）性能。为了保证系统的安全性，Vista 在性能上作了一些折中。为了达到防护缓冲区溢出攻击，在内存保护技术中，系统增加了一些数据结构和程序运行时的校验，这必然对系统性能造成了一定的影响。例如，在极端情况下，GS 保护机制有可能导致 42% 的系统性能损耗[4]。

（下转第 1881 页）

(a)选随机数 $k' \in Z_q^*$,计算 $r' = k'r$;

(b)计算 $v' = H_2(m', r')$;

(c)计算 $U' = (v'/k')T$。

则在消息 m' 上的签名为(U', r')。

该签名(U', r')显然可以通过签名的验证等式 $e(U', r') = e(Q_{ID}, P_{pub})^{v'}$。因为:

$$e(U', r') = e((v'/k')T, k'r) = e((v'/k'k)d_{ID}, k'kP) = e(v'd_{ID}, P) = e(Q_{ID}, P_{pub})^{v'}$$

b)第二种攻击方法。Dai-Wang-Hu-Yun 的签名方案是可以普遍伪造的。敌手 A(不知道用户 ID 的私钥)可以冒充用户 ID(公钥 Q_{ID})对任意消息 m'签名,其操作如下:

(a)选随机数 $k' \in Z_q^*$,计算 $r' = k'P_{pub}$;

(b)计算 $v' = H_2(m', r')$;

(c)计算 $U' = (v'/k')Q_{ID}$。

则在消息 m'上的签名为(U', r')。该签名(U', r')显然可以通过签名的验证等式

$$e(U', r') = e(Q_{ID}, P_{pub})^{v'}$$

因为: $e(U', r') = e((v'/k')Q_{ID}, k'P_{pub}) = e(Q_{ID}, P_{pub})^{v'}$

4　结束语

本文分析了 Wen 和 Ma 提出的聚合签名方案和 Dai 等人提出的基于身份的签名方案的安全性,指出这两种签名方案都是不安全的,并给出了方案的伪造攻击。

参考文献:

[1] BONEH D, GENTRY C, LYNN B, *et al*. Aggregate and verifiably encrypted signatures from bilinear maps[C]// Proc of Advances in Cryptology-Eurocrypt. Berlin: Springer-Verlag, 2003: 416-432.

[2] XU Jing, ZHANG Zhen-feng, FENG Deng-guo. ID-based aggregate signatures from bilinear pairings[C]// Proc of CANS. Berlin: Springer-Verlag, 2005: 110-119.

[3] GENTRY C, RAMZAN Z. Identity-based aggregate signatures[C]// Proc of PKC. Berlin: Springer, 2006: 257-273.

[4] SONG J, KIM H, LEE S, *et al*. Security enhancement in Ad hoc network with ID-based cryptosystem [C]// Proc of ICACT. Berlin: Springer-Verleg, 2005: 372-376.

[5] BELLARE M, NAMPREMPRE C, NEVEN G. Unrestricted aggregate signatures[C]// Proc of ICALP. Berlin: Springer-Verlag, 2007: 411-422.

[6] LI J, KIM K, ZHANG Fang-guo, *et al*. Aggregate proxy signature and verifiably encrypted proxy signature[C]// Proc of Prov Sec. Berlin: Springer-Verlag, 2007: 208-217.

[7] CHENG X, LIU J, WANG X. Identity-based aggregate and verifiably encrypted signatures from bilinear pairing[C]// Proc of ICCSA. Berlin: Springer-Verlag, 2005: 1046-1054.

[8] SHAMIR A. Identity-based cryptosystems and signature schemes [C]// Proc of Crypto'84. New York: Springer-Verlag, 1985: 47-53.

[9] GUILLOU L, QUISQUATER J. A paradoxical identity-based signature scheme resulting from zero-knowledge[C]// Proc of Advances in Cryptology-CRYPTO'88, LNCS 403. Berlin: Springer-Verlag, 1990: 216-231.

[10] CHA J, CHEON J. An identity-based signature from gap Diffie-Hellman groups[C]// Procof PKC. Berlin: Springer-Verlag, 2003: 18-30.

[11] SAKAI R, OHGISHI K, KASHHARA M. Cryptosystems based on pairing [C]// Proc of Symposium on Cryptography and Information Security. 2000: 26-28.

[12] HESS F. Efficient identity based signature schemes based on pairings [C]// Proc of SAC, LNCS 2595. Berlin: Springer-Verlag, 2003: 310-324.

[13] BARRETO P, LIBERT B, MCCULLAGH N, *et al*. Efficient and provably-secure identity-based signature and signcryption from bilinear maps [C]// Proc of Asiacrypt, LNCS 3788. 2005: 515-532.

[14] WEN Y, MA J. An aggregate signature scheme with constant pairing operations[C]// Proc of International Conference on Computer Science and Software Engineering. [S.l.]: IEEE, 2008: 830-833.

[15] HESS F. Exponent group signature schemes and efficient identity based signature schemes based on pairings[R]. [S.l.]: Cryptology ePrint Archive, 2002.

[16] DAI G, WANG M, HU H, *et al*. An effective signature scheme based on tate pairing for mobile business[C]// Proc of International Conference on Wireless Communications, Networking and Mobile Computing. 2008: 1-4.

(上接第 1855 页)

d)机制缺陷。虽然 Vista 试图在各个环节增加缓冲区溢出的难度,并不能从完全杜绝缓冲区溢出攻击。Vista 安全研究员 Skape 指出,由于 PE 文件映像基址的随机只针对前 16 位,后 16 位不会改变,可以通过覆盖返回地址的后 16 位数据来绕过 ASLR 技术。已经陆续有针对 Vista 的缓冲区溢出攻击问题报道。

4　结束语

正所谓道高一尺,魔高一丈,缓冲区溢出攻击和防御技术都在不断的发展。虽然 Vista 试图从各个方面增加缓冲区溢出攻击的难度,阻断缓冲区溢出对系统的攻击,但是要想完全杜绝缓冲区溢出攻击几乎是不可能的。因此,针对缓冲区溢出攻击,除了增加缓冲区溢出的难度外,还应该在缓冲区溢出攻击成功的情况下,研究如何将危害降低到最小。

参考文献:

[1] COWAN C, WAGLE P, PU C, *et al*. Buffer overflows: attacks and defenses for the vulnerability of the decade[C]// Proc of DARPA Information Survivability Conference and Expo. 2003: 227-237.

[2] ONE A. Smashing the stack for fun and profit[J]. Phrack Magazine, 1996,7(49):14-16.

[3] WHITEHOUSE O. An analysis of address space layout randomization on Windows Vista[EB/OL]. (2007-02-22). http://www.symantel.com/avcenter/reference/Address-space-layout-Randomization.pdf.

[4] SOTIROV A, DOWD M. Bypassing browser memory protections[C]. 2008.

[5] Data execution prevention[EB/OL]. (2003)[2009-04-01]. http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/BookofSP1/b0de1052-4101-44c3-a294-4da1bd1ef227.mspx.

[6] GS (buffer security check)[EB/OL]. [2009-03-15]. http://msdn.microsoft.com/en-us/library/8dbf701c(VS.80).aspx.