第34卷 第22期 计 算 机 工 程 2008年11月
Vol.34 No.22 Computer Engineering November 2008
·安全技术· 文章编号：1000—3428(2008)22—0182—02 文献标识码：A 中图分类号：TP309

Windows Vista 的栈保护机制

陈 扬，祝跃飞，梅 强
(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：Windows Vista 应用的栈保护机制降低了利用栈溢出漏洞的可能性。该文分析 Windows Vista 中与栈溢出漏洞利用相关的 3 个安全机制：栈溢出检测、安全结构化异常处理 (SAFESEH)和随机分配地址空间技术(ASLR)。结合实例研究 Windows Vista 抵御栈溢出漏洞被恶意利用的能力，针对不足之处提出了改进方法。

关键词：栈溢出检测；安全结构化异常处理；随机分配地址空间

Stack Protection Mechanisms in Windows Vista

CHEN Yang, ZHU Yue-fei, MEI Qiang
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】The mechanisms which proctect the stack in Windows Vista make it more difficult to exploit a stack overflow vulnerability. This paper analyzes three mechanisms which are close to exploiting stack overflow in Windows Vista. They are stack overflow detection, SAFESEH and Address Space Layout Randomization(ASLR). In allusion to a real vulnerability it analyzes Windows Vista's ability of resisting the stack overflow being exploited, and gives the improved methods.

【Key words】stack overflow detection; SAFESEH; Address Space Layout Randomization(ASLR)

随着计算机和信息安全技术的不断发展，缓冲区溢出漏洞已成为互联网和计算机系统最具威胁的漏洞。当前，缓冲区溢出漏洞主要可分为栈溢出漏洞和堆溢出漏洞。栈溢出漏洞有比较稳定的利用方法，分析和利用起来更加容易；而堆溢出漏洞由于其复杂的原理和大量未公开的数据结构，分析和利用起来都相对困难。微软公司于 2007 年 1 月正式发布了桌面操作系统 Windows Vista，应用了很多安全机制， 因此，Vista 下的漏洞被恶意利用的可能性大大降低。

1 栈溢出及常见的利用方法

程序在执行时，由于对函数保存在栈中的临时变量没有正确地验证长度，出现变量数据长度超过了其在栈中预分配空间长度的情况，进而导致过长的数据覆盖了栈中其他数据。如果被覆盖的是关键数据，会导致程序运行结果出错，甚至程序异常或触发恶意代码。

在程序执行的过程中，诸如函数返回地址、结构化异常处理的相关结构等关键数据都保存在栈中，只要利用溢出覆盖掉关键数据就可以控制程序流程，因此，栈溢出较易利用，通常有覆盖函数返回地址和覆盖异常处理结构 2 种方法。

2 Windows Vista 中与栈溢出相关的机制

2.1 栈溢出检测

针对覆盖函数返回地址利用栈溢出漏洞的方法，微软的 Visual Studio C++编译器提供了栈缓冲区溢出检测的选项：/GS。其原理是当函数开始执行时在栈中保存一个cookie，当函数返回前检查这个cookie是否被覆盖[1]。该机制的实现因函数是否有异常处理而有所不同，但原理没有变化。仅有栈溢出检测而无异常处理的函数汇编代码如下：

```
push    ebp   ; 函数开始时在栈中设置frame和溢出检测cookie。
mov ebp,esp
sub esp,10h
mov eax,dword ptr [___security_cookie]
mov dword ptr [ebp-4],eax
…
mov ecx,dword ptr [ebp-4]
add esp,8
call __security_check_cookie (401199h)
;函数返回前通过比较 cookie 来检测栈溢出
mov esp,ebp
pop ebp
ret
```

其中，*___security_cookie* 在进程创建时由函数*__security_init_cookie*()初始化，由系统时间和进程创建时间共同决定，很难被预测。

但并不是所有的函数都会被编译器加上栈溢出检测的代码，MSDN 中对/GS 选项的说明中给出 1 个规则，编译器在以下情况不对函数附加栈溢出检测的代码：

(1)函数本身没有局部变量，不存在栈溢出的可能。

(2)编译器的优化选项没有打开。

(3)函数的参数是不确定的，例如 *printf*，参数的数目是不确定的。

(4)函数有 naked 标记。

(5)函数声明为包含内嵌汇编代码。

(6)函数的变量的使用方法即使在发生溢出的情况下也不太可能被利用。

2.2 安全结构化异常处理

安全结构化异常处理(SAFESEH)是一项保护和检测和防

作者简介：陈 扬(1983－)，男，硕士研究生，主研方向：网络安全；祝跃飞，教授、博士生导师；梅 强，硕士研究生

收稿日期：2008-03-14 **E-mail：**cyandqcw@hotmail.com

止栈中的结构化异常处理函数地址被覆盖而导致栈溢出漏洞被恶意利用的技术。它并不是 Vista 的新技术，在 Windows Xp sp2 就已被引入。但由于需要.net 的编译器支持，而 Windows Xp sp2 系统自身所带的库和执行程序都是由非.net 编译器编译，因此在 Windows Xp sp2 下，栈溢出只要覆盖 *SEH* 的地址就能饶过所有的保护机制。而在 Windows Vista 下的系统库几乎全部带有 SAFESEH 支持，使得覆盖栈中的 *SEH* 结构来利用溢出漏洞的技术很难再应用了。

SAFESEH 本身的原理很简单，就是在链接器生成二进制文件的时候，把所有合法的 *SEH* 函数的地址解析出来，在文件里生成一张合法的 *SEH* 函数表，用于异常处理时候进行匹配检。

异常处理过程如下：*RtlDispatchException*()，首先察看 *VEH* 列表看其中是否有异常处理函数，如果有就调用 *VEH*；如果没 *VEH*，先确定 *_EXCEPTION_REGISTRATION* 结构在栈中且异常处理函数的地址不在栈中，满足条件的话调用 *RtlIsValidHandler*()，这个函数对 *SEH* 函数做最后检查，根据堆栈中 *SEH* 函数的地址，调用 *RtlLookupFunctionTable*()确认此函数地址是否属于一个 IMAGE 的地址空间。

(1)如果属于：读取 ntdll.dll 的加载模块数据内存对应的 *SEH* 函数表的加密地址、IMAGE 的开始地址、IMAGE 的长度、合法 *SEH* 函数的个数、解密 *SEH* 函数表的加密地址：如果该地址不为 0，代表该 IMAGE 支持 SAFESEH，根据合法 *SEH* 函数的个数，依次计算合法合法 *SEH* 函数的地址并和当前 *SEH* 地址进行比较，如果符合执行 *SEH* 函数，如果全不符合则不执行。如果该地址为 0，代表该 IMAGE 不支持 SAFESEH，只要该内存属于该 IMAGE 代码段范围内的代码都可以执行。

(2)如果不属于：调用*ZwQueryInformationProcess*()，通过分析其访问的结构[2]，其功能是检测进程是否开启了数据执行保护(DEP)，如果开启则认为异常处理函数不合法并报之用户，如果DEP未开启则正常执行。此处的DEP是指Windows利用代码实现的软DEP。

2.3 随机分配地址空间技术

随机分配地址空间技术(Address Space Layout Randomization, ASLR)表现为不同的 2 次会话中的进程加载同一个 dll 的地址是不同的，即 PC 重新启动后，同一个程序的进程空间里面同一个 dll 的加载地址将会发生变化，而且这个变化是随机的，这使以前漏洞利用时使用的固定跳转指令不存在，也使漏洞虽存在但更难稳定利用。

3 Windows Vista 栈溢出相关机制的弱点及实例

3.1 栈溢出检测的弱点

Windows Vista 的栈溢出检测可有效阻止函数返回地址被覆盖从而导致溢出漏洞被利用的情况。但并不是所有的函数都有栈溢出检测，MSDN 中给出的规则中除了第 1 条，剩下规则对应的函数仍然都有可能发生栈溢出。因此，这个栈溢出检测做的并不全面。

3.2 SAFESEH 的弱点

如果一个进程加载的所有模块都支持 SAFESEH，覆盖 *SEH* 修改异常处理函数的地址来控制流程就相当困难。Vista 下大部分系统库是支持 SAFESEH 的，因此，Vista 下的 SAFESEH 是非常强的。

但还存在问题：

(1)SAFESEH 需要.net 的链接器支持。通常 Windows 中大量的第三方程序和库不是使用.net 编译的。如果进程存在 1 个不支持 SAFESEH 的模块，就可能利用这个模块进行跳转，也就等于整个 SAFESEH 的机制失效，不过由于 Vista 下的随机分配地址空间技术，导致无法预先确定模块加载地址，因此，可以深层抵御这种情况下的 SEH 覆盖利用。

(2)SAFESEH 机制本身流程的问题，当 *SEH* 函数地址位于堆中时，需要 DEP 开启才能避免执行。而系统默认情况下，DEP 只对 Windows 的系统进程开启。这样就在进程的堆中充填恶意代码，进而利用栈溢出覆盖 *SEH* 实现跳转提供可能。

3.3 ASLR 存在的问题

ASLR技术需要Microsoft Visual Studio 2005 SP1 Beta以上版本的链接器支持(使用/dynamicbase选项[3])，而很多第三方的模块并不是VS2005 编译的，进程空间中仍然可能存在可以使用的固定跳转指令。

同时，ASLR 针对的是利用固定的跳转指令进行跳转执行 shellcode 的方法，如果有别的方法能够稳定跳转至 shellcode，就能够绕过这个机制来利用漏洞。

3.4 MS07-017 在 Windows Vista 下的利用

Microsoft Windows在处理畸形的动画图标文件(.ani)时存在缓冲区溢出漏洞，user32.dll 中的*LoadAniIcon*()函数在渲染畸形的动画光标文件(.ani)没有正确地验证ANI头中所指定的大小，导致栈溢出[4]。这个漏洞在Vista中也存在。

由反汇编存在漏洞的 user32.dll 文件可以看出存在问题的函数，其开始代码和结束代码如下：

```
LoadAniIcon()
mov edi, edi
push ebp
mov ebp, esp
sub esp, 50h
push ebx
mov ebx, [ebp+arg_0]
mov eax, [ebx]
push esi
push edi
…
pop edi
pop esi
pop ebx
leave
retn 14h
```

可以看出这里并没有栈溢出检测的代码。而 user32.dll 被编译时显然应用了/GS 选项。漏洞发生的原因是没有对光标文件中的“anih”字段长度做正确的验证，从而导致 *LoadAniIcon*()函数的栈缓冲区溢出。

分析这个函数没有被编译器附加缓冲区溢出检测代码的原因，可能是因为“anih”段的长度固定，而且由上层函数 *LoadCursorIconFromFileMap*()验证。因此，编译器认为这个缓冲区不会被溢出。

因为没有溢出检测，所以覆盖 *LoadAniIcon*()的返回地址就可随意改变流程了。而面对栈地址的随机性以及 ASLR 的深层保护，将 shellcode 置于栈中，很难找到一个稳定的跳转点来跳转至 shellcode。但如果结合 ie 来利用这个漏洞，就可以预先通过网页中的脚本占用大量堆空间，并在堆空间中填充大量的 nop 指令和 shellcode，然后再触发漏洞。此时只需

(下转第 186 页)