doi :10.3969/j.issn.1671-1122.2009.05.019

Windows 安全之 SEH 安全机制分析

徐有福[1]，张晋含[2]，文伟平[1]

（1. 北京大学软件与微电子学院信息安全系，北京 102600；2. 中国信息安全测评中心，北京 100085）

摘　要：SHE（结构化异常处理），是Windows操作系统所提供的对错误或异常的一种处理机制。但是，由于SEH设计存在缺陷，使得在存在缓冲区溢出时，攻击者可以利用SEH的异常处理链结构实施成功的攻击。所以，微软在不同版本的Windows操作系统中分别都SEH实现上做了相应的改动，分别出现了SafeSEH和SEHOP两个改进机制。但是，在改进之后的SEH机制中又分别出现了不同程度的缺陷。本文结合微软的GS、ASLR等技术对不同版本的SEH机制的实现以及其存在的安全缺陷进行深入研究，并针对安全缺陷提出相应的修改意见。

关键词：SHE；SafeSEH；SEHOP；GS；ASLR

中图分类号：TP316.7　　文献标识码：A

Windows Security: The gradual improvement of SEH mechanism

XU You-fu[1], ZHANG Jin-han[2], WEN Wei-ping[1]

(1.Department of Information Security, SSM, Peking University, Beijing 102600, *China;*

2.China Information Technology Security Evaluation Center, Beijing 100085, *China)*

Abstract: SEH(Structured Exception Handling) , a mechanism which is provided by Windows operating system to handle errors or exceptions. However, because of the limitations in SEH 's design, the attacker can make use of SEH exception handling chain to sucessfully attack the system when there is a buffer overflow bug. Therefore, changes of SEH implementation mechanism were made in accordance with different versions of Microsoft Windows operating system. As a result, two improved mechanisms - SafeSEH and SEHOP appeared. Nevertheless, different degrees of defects still exist in these improved mechanisms. This paper provides a profound research on the implementation of and security defects in different versions of the SEH mechanisms through technologies afforded by Microsoft, such as GS, ASLR. Also provided are some improvement advices of the corresponding security defects.

Key words: SEH; SafeSEH; SEHOP; GS; ASLR

0 引言

结构化异常处理（Structured Exception Handling，简称SEH）是一种Windows操作系统对错误或异常提供的处理技术。SEH是Windows操作系统的一种系统机制，本身与具体的程序设计语言无关。SEH为Windows的设计者提供了程序错误或异常的处理途径，使得系统更加健壮。

由于SEH的链式处理方式，在使用SEH的过程中出现了被攻击者能够利用的重大漏洞。以致在不同版本的Windows操作系统中一直对SEH的实现机制做相应的改进。

本文将对不同时期的SEH实现技术做相应的阐述并对其可能存在的安全隐患进行深入研究，并对产生安全隐患的原因进行分析以及提出一定的建议。然后，针对各种改进的SEH技术所做的安全性的改变进行比较。

1 SEH

本文主要研究SEH技术存在的安全隐患，所以对SEH的实现机制只做简单介绍。

SEH实际包含两个主要的功能：结束处理和异常处理。结束处理程序用以确保调用和执行结束处理代码段，而不管保护体代码段是如何退出的。即SEH中__try{}__finally{}结构中的Finally关键字所标出的，用以处理内存释放和信号量释放等操作。而异常处理则是通过__try{}__except{}结构来实现[1]。

1.1 SEH的实现机制

在Intel体系结构中，当进程发生异常时，系统会从用户态（Ring3级）切换的核心态（Ring0级），并进行Ring0的异常处理，比如发生缺页之类的异常Ring0可以正常处理。但是，有些异常是Ring0级处理不了的，比如存取无效内存地址，被0除等操作，这时系统就会把异常抛给该进程发生异常的线程的相关SEH处理程序进行处理。

异常处理过程如图1所示。

当发生异常时，系统定位到except块开头，并计算异常过滤器的值，异常过滤器只能有三种可能的值（定义在Windows的Excpt.h中，见表1）[2]。

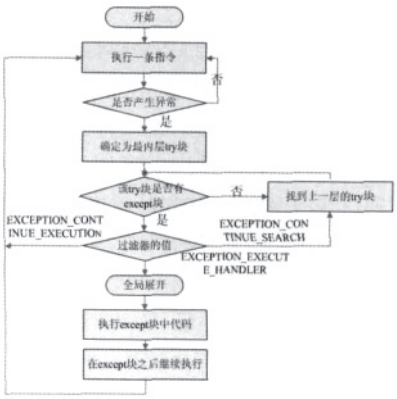

图1 异常处理过程

表1 标识符及其定义

标识符	定义值
EXCEPTION_EXECUTE_HANDLER	1
EXCEPTION_CONTINUE_SEARCH	0
EXCEPTION_CONTINUE_EXECUTION	-1

EXCEPTION_REGISTRATION结构：

```
_EXCEPTION_REGISTRATION STRUCT
    Struct EXCEPTION_REGISTRATION* Prev;
    DWORD Handler;
_EXCEPTION_REGISTRATION ENDS
```

在SEH技术中，每个线程的异常处理程序都对应其相应的EXCEPTION_REGISTRATION结构。这样EXCEPTION_REGISTRATION又称为线程的异常结构。EXCEPTION_REGISTRATION中的Handler域保存的是相应异常处理程序的调用句柄；Prev域保存的是另一个异常结构的指针（在过滤器的值为EXCEPTION_CONTINUE_SEARCH的情况下继续寻找异常处理程序就是通过Prev域来实现的），这样线程相关的异常结构就形成了一个异常结构链[3]。

EXCEPTION_REGISTRATION首节点的地址保存在线程的TEB（Thread Environment Block）的前4个字节中，FS寄存器指向当前线程的TEB，所以FS:[0]就指向当前线程异常结构链的头节点。异常处理回调函数过程见图2：

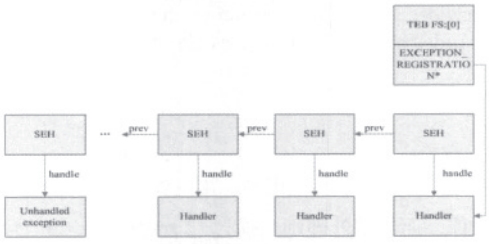

图2 异常处理回调函数过程

SEH中主要有两种异常处理回调函数：

（1）系统定义的缺省异常过滤器处理函数，或是用户通过调用SetUnhandledExceptionFilter API函数所安装的缺省异常处理函数。当每个异常过滤器都返回EXCEPTION_CONTINUE_SEARCH时就出现了未处理异常（Unhandled exception），这时就要调用缺省异常处理函数来处理异常。

（2）与线程相关的异常处理回调函数，正常处理所对应的异常。

线程相关的异常处理函数主要由RtlDispatchException API函数来完成查找分发。首先，RtlDispatchException函数通过TEB找到异常链的首部指针，然后利用该指针进行异常链的遍历以找到合适的异常结构节点，再通过RtlDispatchExceptionForException函数来实现对异常处理函数的调用。

异常处理函数返回EXCEPTION_CONTINUE_EXECUTION表明异常已经修复，如果返回EXCEPTION_CONTINUE_SEARCH表明这个异常在此异常结构节点不能被处理，RtlDispatchException继续查找异常结构链。异常结构链的最后一个异常节点通常对应缺省的异常处理程序，如果没有找到合适的线程相关异常处理函数，RtlDispatchException就会调用这个缺省的处理程序对异常进行处理[4]。

1.2 SEH技术存在的安全隐患

从Windows XP之后Windows采用了GS的栈保护机制，而该机制中主要通过在栈上保存的被调用函数的返回地址之前添加Security cookie来实现的。如果发生缓冲区溢出，在溢出函数返回地址的过程中同样也覆盖了Security cookie。在函数返回之前对Security cookie进行检查，如果发现Security cookie被修改，就终止进程的继续执行。

但是，Security cookie是一个可写的变量，如果攻击者可以将原来用以检测的Security cookie也给覆盖，那么在溢出的过程中就可以自己填写相应的cookie了。还有一种情况就是在检查cookie之前系统已经出现异常了，而产生异常的方法可以有多种，比如用伪造的地址覆盖栈上的函数返回地址就可以触发一个异常，因为检测cookie是在函数执行结束之后进行的，在此不介绍这两种攻击了。在检查Security cookie失败后，系统开始执行SEH异常。在发现检查失败的情况下系统开始查找相应的异常处理程序，并通过RtlDispatchExceptionForException函数来进行异常处理程序的调用。但是，当前状况是栈发生溢出，而在这种情况下执行任何代码都是十分危险的事[4]。

在这里就会出现两种情况，第一种情况，攻击者覆盖异常结构，并将异常结构导向他想要执行的代码（ShellCode）。因为SEH对异常处理程序的调用只是查找异常结构节点并利用异常结构中的Handler域进行定位而没有进行任何安全验证，并且异常结构链是存在进程的线程空间栈里面，当发生栈溢出的时候是非常危险的。当栈空间被破坏时异常结构很可能被覆盖并导向攻击者想要执行的程序空间。Windows操作系统曾经就出现许多的栈溢出漏洞，典型的像IIS WebDav、Rpc Locator等漏洞[4]。

在Windows 2003 Server中微软通过对SEH的修改试图阻止上面所述的攻击，即后面所要讲的SafeSEH技术。在修改后，Exception handler是被注册的并且所有的Handler地址被存储在载入配置目录的模块中。当异常发生时，在Handler执行之前把Handler的地址和注册Handler的列表进行对比，如果发现不匹配则终止执行该Handler。如果Handler的地址是在载入模块地址外面的话将继续执行，如果Handler的地址直接指向栈内存地址，那么Handler将不被执行。通过这个方法来阻止攻击者直接将Handler域覆盖一个栈地址，但是如果这个地址是一个堆地址的话Handler将被继续执行[5][6]。

Windows 2003 Server所做的修改同样存在很大缺陷。首先是，攻击者可以通过覆盖给Handler一个载入模块之外的地址，然后将该地址重定向回执行栈地址空间来实现攻击。

其次就是，攻击者可以利用堆溢出来进行攻击[5]。

第二种情况，缺省的异常处理函数指针被覆盖。导致这个问题的很大原因是相同版本的Windows操作系统的缺省异常处理函数指针存放的地址是相同的，而当攻击者利用系统中漏洞得到写内存的机会时可以将这个地址中的内容改为他所想要执行程序的地址，这样当系统调用缺省异常处理程序时就会执行攻击者想要执行的程序。攻击者可以利用这种方法攻击Windows 2000 IIS服务的.ASP漏洞。

另外，在Windows 95/98操作系统下可以利用SEH实现代码的权限提升。在Windows 95/98系统中，在核心态执行代码时，CS寄存器中段选择子值是28h，DS和SS中的段选择子值是30h。如果能够利用代码动态的将CS和SS等寄存器赋予上述核心态权限选择子值功能的SEH异常处理程序，那么在调用该代码异常处理程序返回后，就可以获得代码权限的提升。CIH病毒就是因为拥有ring0级的权限而具有很大的破坏性。在Windows 95/98之后的Windows操作系统都严格的将系统划分为用户模式和内核模式，要实现从用户模式到内核模式的转换必须借助CPU的某种门机制，这也弥补了SEH中存在的一些安全缺陷[4]。

1.3 对SEH存在安全缺陷的改进建议

（1）针对运行时动态获得异常处理程序的问题，在执行异常处理程序之前必须保证该异常处理程序没有被修改过，这就要求在执行异常处理程序（包括缺省异常处理程序）之前对异常结构进行严格的安全验证。

（2）如果可以的话，当异常发生时不要进行任何操作而使程序直接运行结束，不过相信微软是不会这么做的。

（3）针对Windows 2003 Server改进之后的SEH增加对堆内存地址的安全检查，以及对在载入目录模块的地址的判断不仅仅是一个在该地址之外就可以执行，而是要严格的进行Handler执行地址的详细比较。

（4）对SEH技术中一些敏感的地址和一些全局不变函数指针，比如系统缺省异常处理函数指针存放的地址和系统缺省异常处理函数指针，进行调用之前的一致性检测，确保这些敏感信息没有被重写[4]。

（5）对Windows 95/98系统在异常处理程序调用前后，进行相应代码权限的一致性检查[4]。

2 SafeSEH

2.1 SafeSEH的实现

SafeSEH是用一种保护和检测以及防止堆栈中的SEH被覆盖而导致被利用的增强型SEH技术。SafeSEH是微软在.net编译器中加入的/safeseh编译选项而引入的技术。编译器在编译时将PE文件的合法的异常处理函数解析成一张表并将该表存放在PE文件的数据块中，用于匹配检查。如果PE文件不支持SafeSEH，则表的地址为0。当PE文件被加载后，表中的内容被加密保存在ntdll.dll模块的某个数据区。在PE文件运行时如果发生了异常，则RtlDispatchException函数会检查该Handler是否在SEH的函数表中，如果不在，则说明该Handler是非法的，程序将终止，如果在将正常处理异常[7]。

SafeSEH在Windows Vista中广泛使用，但并不是Vista的新技术，SafeSEH在Windows XP SP2就已经被引入。只是由于SafeSEH需要.net编译器编译的image才能支持，而Windows XP系统的库和执行程序都是非.net编译器编译的，所以SafeSEH在Windows XP中并没有发挥应有的作用。而Windows Vista系统中绝大部分的程序都是通过.net编译器编译的，所以SafeSEH在Vista系统中才发挥了应有的作用。

SafeSEH是非常强大的，如果进程所加载的所有模块都支持SafeSEH的image，则通过覆盖SEH来攻击是不可能实现的。 当然这所需要的前提是所有模块都支持SafeSEH的image，但是实际情况确是，如果进程中存在一个程序不支持SafeSEH的image就等于整个SafeSEH机制失效[8]。

2.2 SafeSEH存在的问题

虽然Vista系统中大部分的组建都经过.net编译的，但是仍存在许多的组件没有加入该技术，进程中只要有一个不支持SafeSEH的组件并存在缓冲区溢出漏洞，那么攻击者就可以通过覆盖SEH节点控制进程流程。不过在Vista系统中，由于进程空间支持进程空间随机加载技术（Address Space Layout Randomization，简称ASLR）技术，在一定程度上抵御住了这种情况下的SEH覆盖攻击。

SafeSEH需要编译器的支持，为PE文件生成一张异常处理函数表，由其他编译器生成的PE文件就无法加入SafeSEH技术保护。

进程在加载PE程序时将SEH函数表可用的加密地址存在ntdll.dll中，攻击者可能会通过覆盖这个SEH表地址使得计算出来的SEH函数表的地址为0或是攻击者存放自己的编造的SEH函数表地址，从而绕过SafeSEH的保护。攻击者也可能直接跳到dll内存里实施攻击。

2.3 针对SafeSEH存在问题的改进建议

（1）主要问题是有很多的组件并不支持SafeSEH技术，所以要在将来的Windows系统中对组件进行.net编译，不过这是微软的事了。

（2）在执行异常处理程序之前再对异常处理程序的地址进行安全性检测，和SEH函数表地址检测实现双重保护。

（3）针对编译器的静态保护不够全面和彻底，如果改为实时的保护，能够解决这一问题。通过对异常处理链的检测，检查SEH链是否能正常搜索到最后一个节点来判断异常处理链是否被破坏。即在异常处理程序执行之前对整个SEH链进行遍历，如果能搜索到最后一个节点则执行该异常处理程序，否则终止该进程[7]。

（4）既然在SafeSEH中有了SEH函数表，为什么还需

要SEH异常链，已经把所有的合法的异常处理函数地址存在SEH函数表中了，这样在发生异常时可以直接在表里面查找对应的异常处理函数，从而避免因异常链而导致的漏洞。

3 SEHOP

针对SEH和SafeSEH存在的缺陷，微软在Windows Server 2008和Windows Vista SP1中加入了结构化异常处理覆盖保护(Structured Exception Handler Overwrite Protection，简称SEHOP)。该功能在Windows Server 2008默认打开，在Windows Vista SP1默认关闭需要手动打开。

3.1 SEHOP的实现

SEHOP通过动态检测异常调度函数来确保SEH链没有被破坏，而调度函数不依赖于二进制的元数据。SEHOP通过验证线程的异常处理链的完整性来抵御攻击者利用SEH覆盖技术，即从SafeSEH的编译时保护该为运行时保护。

大多数的基于堆栈的缓冲区溢出发生时，攻击者会在覆盖异常处理函数之前先覆盖异常链上指向下一个节点的指针地址。由于下一条记录被破坏了，异常处理链的完整性就被破坏了，所以通过验证异常处理链的完整性可以阻止覆盖SEH链的攻击。

SEHOP通过两个步骤来实现。首先，在线程的异常处理链中插入一个标志性的异常处理记录(FinalExceptionHandler)到链表的末尾，这一步发生在线程第一次开始在用户态执行的时候。然后，在用户态出现异常并转向内核处理时，检查该异常处理链的链尾标志性异常处理记录，如果该尾记录无效的话，异常调度器就会认为异常处理链遭到破坏，发生了SEH覆盖攻击。然后异常调度器安全结束进程的运行。如果链尾记录有效，则正常执行异常处理[9]。SEHOP异常链如图3所示：

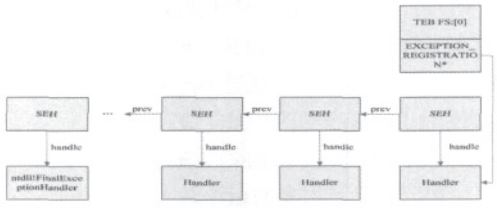

图3 SEHOP异常链结构

3.2 SEHOP存在的问题

攻击者通过伪造SEH链的链尾记录来实施攻击，但是在Windows Vista以后版本中采用了ASLR技术，ASLR技术使得攻击者不可能终止SEH链并将链导向伪造的链尾，除非攻击者攻破ASLR技术。

如果攻击者事先通过内存泄漏等漏洞获得了SEH链的信息的话，那么攻击者可以修复被破坏的SEH链，使得系统察觉不到SEH链已经被破坏了，从而成功攻破。不过有ASLR在FinalExceptionHandler的地址在内存中可能会随机分配，这样就给攻击带来很大麻烦[10]，但是如果ASLR的随机范围不够大的话，攻击者还是可以攻破的。

3.3 SEHOP改进意见

(1) SEHOP非常的强大，但是在借助ASLR的同时才能比较好的抵御攻击者伪造SEH链尾记录，如果在Vista SP0以前的系统上使用SEHOP，那么在没有ASLR时，还是可能被攻击的，所以在提升SEH的同时建议为其它版本系统也增加ASLR技术。

(2) 对SEH链中的每个异常处理函数设置安全验证标识，在执行异常处理函数时都要检查该异常处理函数的安全标识，而不只是在执行处理之前检查FinalExceptionHandler。

(3) 为了防止事先得到链表信息可以对SEH链表的信息进行隐藏，而不是静态的存在线程栈的FS:[0]的位置处。

4 结论

SEH机制作为一种异常处理机制在Windows各版本操作系统中得到了广泛的使用。本文主要针对SEH所存在的安全缺陷并被缓冲区溢出所利用进行分析和研究，并提出相应的解决建议。

微软在系统层面上做了很多的工作，为了对抗exploit技术提出了DEP、ASLR、GS以及SEH系列机制。Windows操作系统利用这些技术的相互关系和共同作用来保护系统安全。微软安全专家指出，会继续研究新的先进技术以缓解SEH系列机制存在的安全缺陷。 （责编 潘静）

参考文献：
[1] Matt Pietrek. A Crash Course on the Depths of Win32 Structured Exception Handling [J].The January 1997 issue of Microsoft Systems Journal, 1997.
[2] Jeffrey Richter, Christophe Nasarre.WINDOWS VIA C/C++ (Fifth Edition) [M]. Microsoft Press, 2008. 566-577.
[3] 段钢. 加密与解密(第三版)[M]. 北京:电子工业出版社,2008. 306-312.
[4] 齐雷,谢余强,程东年等. Win32 SEH异常处理机制分析[J]. 信息工程大学学报. Vol.5 No.2 Jun, 2004.
[5] David, Litchfield.Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server. Sep, 2003.
[6] 许治坤，王伟，郭添森等. 网络渗透技术[M]. 北京:电子工业出版社, 2005: 43-54.
[7] 彭建山,吴灏.Windows Vista内存保护关键技术研究[J]. 计算机工程与科学. Vol.29, No.12, 2007.
[8] Nagy B. SEH (Structured Exception Handling) Security Changes in XP SP2 and 2003 SP1[Z]. eEye Digital Security, 2006.
[9] Skape. Preventing the Exploitation of SEH Overwrites. Sep, 2006.
[10] Alexander Sotirov, Mark Dowd. Bypassing Browser Memory Protections. 2008.

作者简介：徐有福(1985-)，男，硕士研究生，主要研究方向：网络安全；张晋含(1969-)，男，助理研究员，博士，主要研究方向：操作系统安全性研究，漏洞分析，软件源代码缺陷分析等；文伟平(1976-)，男，副教授，主要研究方向：网络攻击与防范、恶意代码研究、信息系统逆向工程和可信计算技术等。