恶意代码安全虚拟执行环境研究*)

曹 跃 梁 晓 李毅超 黄 沾

(电子科技大学计算机科学与工程学院网络攻防实验室 成都 610054)

摘 要 入侵容忍的系统要求具有在现实环境中执行不安全程序且不遭受永久性伤害的能力。虚拟机技术提供了一种虚拟的可执行环境,能够满足这个需求。通过对操作系统调用接口资源的重命名的研究,在此基础上设计并实现了一种 Windows 平台下基于操作系统层的安全虚拟执行环境的体系结构。实验结果表明,该系统可以有效地模拟应用程序的各种运行行为和结果,并为后续分析提供充足的信息。经对比发现,基于操作系统资源虚拟化的安全执行环境技术比传统虚拟技术更灵活,消耗系统资源更少。

关键词 恶意代码,安全执行,虚拟机,命名空间虚拟化

Research on Malware Secure Virtual Execution Environments

CAO Yue LIANG Xiao LI Yi-Chao HUANG Zhan

(Laboratory of Network Attack & Defense, School of Computer Science and Engineering, UEST of China, Chengdu 610054)

Abstract Intrusion-tolerant and fault-tolerant systems require the ability to execute unknown programs in a realistic environment without leaving permanent damages. Virtual machine technique provides an execution environment that is both realistic and isolated, also meet this requirement. We present a new secure execution environments framework. After researched on rename mechanism of system call interface under operating systems, finally designed and implemented the system architecture of operating system level based secure execution environment under Windows. Central to our algorithm is namespace virtualization, which provides simulation of many runtime behaviors and results in applications with enough analysis information. Compared with traditional algorithms, our method is more flexible, and requires less system resource.

Keywords Malware, Secure execution, Virtual machine, Namespace virtualization

1 引言

虚拟机技术[1]是在一个物理主机上创建一个或多个可执行环境的技术。每个虚拟机代表了一个潜在的物理主机的实例,并且互不干扰。这种隔离的属性使虚拟机可以成为安全系统和错误容忍应用程序的基石。现有的主机入侵防御系统(HIPS)[2]的规则定制基本上都是以单一操作为单元,该操作将被禁止或者由用户手工操作决定处理方式。通常在这种情况下用户应用程序将不能继续正常运行或者用户被频繁打扰。因此针对单一操作规则存在的灵活性和合理性不足,规则的定制也复杂。为了改善访问规则的合理性,我们提出了基于安全虚拟执行环境的恶意行为分析系统。

沙盒[3]允许非安全代码运行在真实的系统环境中,但是不会在系统中留下任何的痕迹。基于操作系统级系统调用接口资源的重命名机制,完全满足构建一个沙盒的要求,它可以提供一个真实且隔离的执行环境。我们在 Windows 下实现了系统调用级安全虚拟执行环境(SVEE)。该原型系统尽可能多地共享利用了主机操作系统的资源,但是对这些资源的访问都是与主机操作系统隔绝的,不会对主机系统造成任何损害。

2 基本原理

实现 Windows 下安全执行环境的关键技术为命名空间虚拟化。Windows 为各种系统资源提供了多种类型的命名空间,例如文件、注册表、内核对象等。命名空间虚拟化技术就是通过在系统调用层上拦截调用[4],在用户进程请求访问各种资源时控制它们的命名,如文件的路径。最终通过重命名这些资源,使得这些资源对请求它们的进程来说是正常的,但是对于操作系统来说可能不存在或者仅仅是原始资源的一个拷贝。这样就到达了将被 SVEE 运行的用户应用程序与操作系统隔绝开的目的。

对于某些驻留在主机系统上的资源,应用程序有可能仅仅是访问其内容,但是不会修改其内容,因此在这种情况下 SVEE 没有必要将其复制。为了达到 SVEE 既共享主机系统的资源和环境,又能更改虚拟机中的资源并与主机系统隔绝开来的目的,写时拷贝机制被用在 SVEE 的实现中。当一个 SVEE 被创建时,它共享主机系统中的所有资源,但是不占用。当运行在 SVEE 中,进程仅仅是请求读取某个资源时,该进程的读请求就被简单地发送给主机操作系统,完成读取操作,这时 SVEE 并不占用任何系统资源。仅当进程请求更改某个资源时,SVEE 就会复制一个原资源到虚拟机中,该更改请求就会实施在这个资源拷贝上,而不会影响主机系统的资源。

本文的 SVEE 实现在 Windows 系统调用层,利用命名空间虚拟化和写时拷贝机制实现了资源访问重定向和隔绝,并且在此基础上构建了应用程序沙盒,用以实现对多个行为的捕获。因此下面将分别介绍资源访问的重定向和几种必需的

*)基金项目:国家科技基础条件平台工作基金资助项目(2003DIA7J051)。曹 跃 助教,在职硕士生,研究方向为计算机系统安全、计算机网络安全。

资源虚拟化的详细实现原理。

3 资源虚拟化

3.1 文件系统虚拟化

在本文的 SVEE 中，文件系统的虚拟化实现了将虚拟机中的文件/目录和一些特殊的设备文件(命名管道，油槽)与主机系统隔离。这主要是通过使用系统调用拦截技术拦截并控制了 Windows 的文件操作函数。

对于普通文件和目录，SVEE 使用了前面介绍的写时拷贝机制，其不同于 Windows 在内存管理中使用的以单个页面块为单位的写时拷贝机制。为了简化 SVEE 的实现就采用了以单个文件或目录对象为单位的写时拷贝机制。当进程以写入的方式打开一个文件的时候，整个目标文件都会被复制到 SVEE 的根目录下，且该文件的属性和其所有的父目录都会被复制。当进程要创建一个新的文件的时候，处理过程与以写入方式打开一个文件一样。对于特殊设备文件的虚拟化，如命名管道和油槽，处理方式与普通文件类似。当特殊设备文件被关闭，并且没有被任何进程占用时，其所对应的虚拟文件就被 SVEE 删除。

整个文件系统虚拟化的实现使用了系统调用拦截技术在 Windows 的系统调用层拦截并控制了相关的文件系统调用。当 SVEE 拦截到这些系统调用对应的请求时，首先会检查发起请求的进程是否为被虚拟化的进程。如果不是，则该系统调用被直接传递到 Windows 的原始系统调用函数；否则会根据不同的请求进行不同的处理：

(1)ZwCreateFile 和 ZwOpenFile

使用前一节论述的重定向算法处理 SVEE 中进程的文件创建和打开请求。这时需要做的就是根据需要对路径进行修改，使其指向 SVEE 根目录下的对应虚拟文件/目录，然后用重定向后的路径向 Windows 请求文件创建或打开操作，最后返回文件/目录的句柄。由于在以后的文件操作中，文件/目录都是由句柄来标识的，因此在这一步中就必须要将文件路径和其句柄对应地记录起来。

(2)ZwQueryInformationFile 和 ZwQueryDirectoryFile

系统调用函数 ZwQueryInformationFile 用来通过一个文件句柄来获得对应文件的各种信息。当一个文件被 SVEE 因为写时拷贝重定向后，进程通过传入该文件被打开或者创建的时候返回的句柄，调用 ZwQueryInformationFile 请求文件路径的时候，会得到指向 SVEE 根目录下的其对应的虚拟文件的路径。SVEE 必须要拦截该调用，并根据在处理文件创建，打开操作时所记录的句柄，来返回请求进程期望的结果。而系统调用函数 ZwQueryDirectoryFile 则是用来通过一个目录句柄返回在这个目录下的所有文件的。因此为了让发起请求的进程获得期望的结果，SVEE 需要根据在处理目录创建，打开操作时所记录的句柄，来返回请求进程期望的结果。

(3)NtSetInformationFile

该系统调用可能被用来删除或者重命名一个文件/目录，因此 SVEE 必须要对其进行处理，防止对主机系统的破坏。当请求是指向一个已经被写时拷贝过的文件/目录时，该请求可以直接传递给 Windows 的原始系统调用，然后在记录中根据请求类型修改文件名或者删除记录。否则如果是重命名操作，则进行写时拷贝和重定向并修改对应的记录，但如果是删除操作，则仅仅为目标文件/目录记录一个删除记录，并删除对应的记录。

3.2 注册表系统虚拟化

在 Windows 操作系统中，注册表几乎保存了与操作系统和应用程序等相关所有的配置信息。Windows 的注册表系统与文件系统十分类似。例如注册表中的项都有自己的路径，而在其他的操作中，项则是通过在创建、打开操作中返回的句柄来标识。注册表的虚拟化使用了命名空间虚拟化和写时拷贝。SVEE 为注册表的虚拟化创建了一个根注册表键“\HKEY_CURRENT_USER\VMStore”。所有的由于写时拷贝而被复制的注册表键和项等都会放到这个根下面。

整个注册表虚拟化的实现使用了系统调用拦截技术在 Windows 的系统调用层拦截并控制了相关的注册表操作系统调用。这里仅列出注册表操作的系统调用和文件操作的系统调用的对应关系，见表 1。

表 1 注册表与文件操作系统调用对应关系

注册表操作	文件/目录操作
ZwCreateKey	ZwCreateFile
ZwOpenKey	ZwOpenFile
ZwDeleteKey	ZwSetInformationFile
ZwDeleteValueKey	ZwSetInformationFile
ZwSetValueKey	ZwSetInformationFile
ZwEnumerateKey	ZwQueryDirectoryFile
ZwEnumerateValueKey	ZwQueryDirectoryFile
ZwQueryKey	ZwQueryInformationFile
ZwQueryVauleKey	ZwQueryInformationFile

在注册表中，可以将注册表键看作是文件系统中的目录，注册表项看作是文件系统中的文件，而注册表值看作是文件系统中的文件的内容。

3.3 内核对象虚拟化

Windows 在其内核中提供了多种命名对象，包括互斥对象、事件对象、信号量对象、定时器对象、区域对象和端口对象等。当然，文件、注册表等资源在内核中也是以对象的形式被对象管理器管理着。从这些内核对象的用途来看，大多数的进程都会用到它们。同时这些对象都是全局的，也就是说任何进程都可以访问它们。为了防止 SVEE 中的进程影响主机系统中的进程，SVEE 必须要对全局的内核对象进行虚拟化。

在 Windows 中内核对象是以目录的方式层次管理的，所有内核对象都有一个共同的根目录，以“\”表示。每个内核对象都可以有其自己的子目录。一般来说每种内核对象都有其固定的目录，如事件对象，互斥对象，信号量对象，区域对象等都会放在“\BaseNamedObjects\”目录下，而端口对象根据其用途不同会被放在“\Windows\”和“\RPC Control\”目录下。这样内核对象的命名空间虚拟化方法，通过重定向内核对象的根目录来达到虚拟化的目的。

现以事件对象为例，对应的系统调用拦截控制函数的实现如下：

(1)ZwCreateEvent

该系统调用函数用于创建一个新的事件对象或者打开一个已经存在的事件对象。该函数的第三个参数为指向 OJECT_ATTRIBUTES 结构的指针。当目标事件对象不存在时，需要做的就是将该对象的路径进行修改，使其前面加入 SVEE 的内核对象根目录，例如“\VM”，然后用重定向后的路

径向 Windows 请求创建事件对象，最后返回新创建事件对象的句柄。如果该对象已经存在，则有可能该事件对象是由系统关键进程创建的。而这些事件对象是进程运行所必需的资源，SVEE 直接将请求传递到 Windows 的原始 ZwCreateEvent 系统调用中。

(2)ZwOpenEvent

该系统调用函数用于打开一个已经存在的事件对象。该函数的第三个参数为指向 OJECT_ATTRIBUTES 结构的指针。首先 SVEE 在目标事件对象路径前加入 SVEE 的内核对象根目录，然后用重定向后的路径向 Windows 请求打开事件对象。如果成功则返回该事件对象的句柄；如果不成功，则 SVEE 将直接尝试打开请求的事件对象。如果打开成功，SVEE 需要将打开得到的句柄直接返回；不成功就返回打开失败信息。

因为各种内核对象的创建打开方式都是一样的，其它的内核对象的创建，打开操作系统调用在表 2 中列出，其对应的拦截控制函数可以参照 ZwCreateEvent 和 ZwOpenEvent 的实现。

表 2　各种内核对象创建/打开操作系统调用

对象类型	创建操作	打开操作
互斥对象	ZwCreateMutant	ZwOpenMutant
信号量对象	ZwCreateSemaphore	ZwOpenSemaphore
定时器对象	ZwCreateTimer	ZwOpenTimer
区域对象	ZwCreateSection	ZwOpenSection
端口对象	ZwCreatePort	ZwConnectPort

4　实现与测试

4.1　系统实现

SVEE 的性能瓶颈在于执行额外的拦截系统调用的指令，包括两个方面：拦截系统调用的瓶颈，包括进程到虚拟机的映射、额外内存的分配、重命名名字参数等等。文件和注册表拷贝瓶颈，只有在进程第一次以写方式打开文件或注册表时才发生。我们将评估拦截系统调用的瓶颈、命令行程序运行时瓶颈、在 SVEE 下交互式程序的启动并且以相同尺度来比较它在主机和在 VMware Workstation 5.0 的性能。测试平台：Pentium-4 2.4GHz，1024MB memory，Windows 2003 Server。

4.2　性能测试

为了测量系统拦截的瓶颈，我们首先关掉 SVEE 层，单独在主机上运行一系列 Windows 程序，并通过 RDTSC 指令计算在每个系统调用时消耗的平均 CPU 周期。然后我们打开 SVEE 层，在虚拟机中运行同样的程序，并作相同的测试。如表 3 所示，函数 NtOpenFile()带来了较大的瓶颈，这很大程度上是由当前重定向算法造成的。具体来说，当测试进程调用 NtOpenFile()函数打开一个文件，SVEE 层需要检测虚拟机本地是否存在该文件的副本，这是通过打开副本来实现的。如果打开请求失败，SVEE 层会重定向该函数打开主机原有的文件而不重命名。因此，这个系统调用对一个请求来说会被调用两次，所以导致性能下降。

表 3　系统调用时钟周期差异比较

系统调用名	原始(CPU 时钟周期数)	SVEE(CPU 时钟周期数)	差异(百分比)
ZwCreateFile	336325	439733	30%
ZwOpenFile	191780	300914	57%
ZwQueryDirectoryFile	141210	203615	44%
ZwQueryInformationFile	189161	347213	84%

总的来说，所有实验结果证明，SVEE 在系统开销、资源需求方面都优于现有的重量级虚拟机。SVEE 适合于支持轻量级平台虚拟机，用于用户模式下程序的安全保护和管理，而重量级虚拟机更适合支持需要完全隔离或不同操作系统程序环境的应用，如调试和测试。

结论　本论文主要的贡献是证实了在 Windows 平台上，通过系统调用拦截创建较强隔离的安全虚拟执行环境是可行的。当前，我们也通过实现自己的守护服务管理 API 以减少共享服务进程来改善主机和虚拟机之间的隔离情况，同时采用名字缓存用于减少拦截系统调用的开销。最后，我们将在 Windows 平台上研究进程移植技术以支持 SVEE 进程的检测点设置与回卷[5]。

致谢　感谢国家科技部科研院所专项基金委对本项课题的大力资助。同时，本文工作得到实验室学生王勋、柴方明、何子昂等给予的大力支持与帮助，对此表示衷心的感谢！

参考文献

1　Nanda S, Chiueh T. A Survey on Virtualization Technologies: [RPE Report]. State University of New York at Stony Brook, Février, 2005

2　Battistoni R, Gabrielli E, Mancini L V. A host intrusion prevention system for Windows operating systems. In: ESORICS'04, 2004

3　Lam L, Chiueh. Automatic extraction of accurate application-specific sandboxing policy. In: RAID 04, Proceedings of the International Symposium on Recent Advances in Intrusion Detection, 2004

4　Russinovich M, Cogswell B. Windows NT System-Call Hooking. Dr. Dobb's Journal, January 1997

5　Srinivasan S M, Kandula S, Andrews C R, Zhou Y. Flashback: A lightweight extension for rollback and deterministic replay for software debugging. In: Proceedings of the 2004 USENIX Technical Conference, 2004