中国科学 : 信息科学　2012 年　第 42 卷　第 5 期 : 617–633
www.scichina.com　info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS

论　文

一种轻量级安全可信的虚拟执行环境

陈浩, 孙建华*, 刘琛, 李海伟

湖南大学信息科学与工程学院, 长沙 410082
* 通信作者. E-mail: jhsun@aimlab.org

收稿日期: 2010–10–01; 接受日期: 2011–09–17
国家重点基础研究发展计划 (批准号: 2007CB310900)、国家自然科学基金 (批准号: 61173166, 60803130) 和中央高校基本科研业务费资助项目

摘要　针对传统 TCB (trusted computing base) 庞大复杂的问题, 设计并实现了一个最小化的 TCB 系统架构. 利用 CPU 的系统管理模式 (SMM) 提供的硬件隔离特性, 通过将应用程序中对安全敏感的代码放入虚拟环境中执行, 从而将应用程序本身非安全敏感部分代码、操作系统及其上运行的其他应用程序排除在 TCB 之外, 使得 TCB 的软件部分只包含安全敏感代码和虚拟执行环境包含的少量代码, 实现了 TCB 的最小化. 本系统的强隔离性使得在操作系统和部分硬件 (如 DMA、硬件调试器等) 被攻击者控制后, 依然可以保证安全敏感代码执行过程的隐秘性和执行结果的完整性. 同时, 本系统还为执行结果提供了细粒度的可靠验证, 保证结果是在本系统的保护下得到的, 没有被任何恶意程序篡改.

关键词　可信计算　可信计算基　系统管理模式　虚拟化　最小化 TCB

1　引言

计算机和通信技术的迅速发展使得信息安全日益重要, 人们越来越需要一个可以信赖的计算环境来保证用户信息的安全性、完整性和可靠性. 不仅需要保证计算机程序能得到正确完整的结果, 而且要保证其执行过程的隔离性以确保执行的结果不会被外界所窃取. 可信计算技术应运而生, 可信计算技术构建从信任根开始的一条信任链, 从根开始到硬件平台到操作系统再到应用层, 逐级认证逐级信任, 最终把这种信任扩展到整个计算机系统. 可信计算方法的可信性建立在以 TPM 为信任根和从信任根开始的信任链上, 若要对一个应用程序的完整性和正确性进行度量, 就必须对整个信任链上的每一级都进行度量, 从这个应用程序信任链的根部开始, 采取回退方法逐级验证. 由于安全应用程序是运行在操作系统之上的, 传统的 TCB 必须包含操作系统, 以及其他系统级程序. 然而, 操作系统代码量庞大, 且运行时状态和数据也并不确定, 这使得对整个信任链的校验过程变得相当复杂而且对结果正确性校验也变得相当困难. 因此, 一个精心设计和执行的系统可信基对系统的安全是至关重要的, TCB 越小, 其中需要校验的部分也就越少, TCB 所包含的 bug 也越少, 这使得整个检测的过程也变得更简单, 检测的结果也更可靠 [1]. 因此, 如何使 TCB 更小成为可信计算研究 [2] 的热门课题.

本文利用系统管理模式 (SMM) 提供的隔离环境实现了一个 TCB 最小化的系统架构. 该最小化的 TCB 架构可以有效地保护应用程序中安全敏感代码的安全, 它在操作系统被入侵的情况下, 依然可

引用格式: 陈浩, 孙建华, 刘琛, 等. 一种轻量级安全可信的虚拟执行环境. 中国科学: 信息科学, 2012, 42: 617–633, doi: 10.1360/112010-1008

以保证安全敏感代码执行结果的可信性和完整性. 同时, 系统对执行结果提供了远程的可验证性. 远程的验证主机可以对系统执行的结果进行校验, 确信该结果是在本系统的保护下执行得到的, 并且该结果未被攻击者窃取和篡改.

2 相关知识简介

2.1 系统管理模式 (SMM)

Intel x86 架构定义了 4 种处理器工作模式: 实模式、保护模式、虚拟保护模式和系统管理模式 [1)], 这 4 种模式之间可以相互转换. 和其他模式相比, 系统管理模式 (SMM) 并不是用来执行操作系统或者用户应用程序的, 它的主要作用是用来进行低层次的硬件管理 (如电源管理和热调节等), 通常情况它是由 BIOS 进行安装的. SMM 拥有自身的内存空间和执行环境, 默认情况下这些空间对于 SMM 以外运行的代码是不可见的. 处理器切换到 SMM 后, 会将之前模式的整个状态保存, 并直到退出 SMM 后才恢复. 整个 SMM 的执行不会受到任何程序的干扰. 这种强的隔离特性有利于建立一个动态信任根, 实现最小化的 TCB.

系统管理模式内存空间 (SMRAM) 主要用来存放进入 SMM 之前的处理器状态信息和 SMI 中断处理程序及其相关数据. SMRAM 定义了 3 个地址空间: 兼容 SMRAM (C_SMRAM)、高内存段 (HSEG) 和顶端内存段 (TSEG) . 默认情况下, SMM 的内存空间为 C_SMRAM 区域. SMRAM 的内容只对执行在 SMM 中代码可见. 这种隔离性是由芯片进行保护的, 任何在非 SMM 模式下对 SMRAM 的访问, 都被重定向到 VGA 的帧缓冲区. C_SMRAM 内存主要包含状态保存区域和系统管理模式中断 (SMI) 处理程序, 其余的空间供 SMI 处理程序存储数据和堆栈使用.

2.2 可信计算基 (TCB)

可信计算基是可信计算领域一个重要的概念. 正如文献 [3] 所描述的, TCB 是和系统安全紧密相关的一些软件和硬件的组合体, 而对系统安全无关的软件和硬件则不包含在其中. 也就是说 TCB 是对计算机系统安全至关重要的所有软件、固件和硬件的集合. 它是保证一个系统安全所需的最小组件集合.

在通常的情况下, 一个计算机系统的 TCB 都相当的庞大, 包含硬件、BIOS、引导程序、以及整个的操作系统. TCB 本身的巨大导致了很难为一个远程验证者提供准确的验证. 因为如果一个远程验证要信任安全代码的执行结果是正确的, 首先必须信任该代码所属的应用程序, 然后还要信任和其同时运行的其他应用程序, 最后还必须信任操作系统及硬件. 这使得整个验证的内容相当复杂, 操作系统本身的错综复杂, 也导致了很难保证验证的结果一定是真实可信的. 同时, 整个验证包含如此多的内容, 也很可能将系统的一些重要信息泄露给远程主机 [4]. 一个精心设计的可信计算基对整个系统的安全至关重要. 现代的系统都致力于减小 TCB 的大小, 从而使得对安全敏感代码的可信验证变得可行.

3 系统设计

3.1 对抗模型

对系统构成最大安全威胁的主要是恶意攻击软件, 如操作系统层的 Rootkit, 它们运行在操作系统

1) Intel®64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide

的最高特权级别, 可以监视和控制整个操作系统的运行, 任意读取和篡改运行在操作系统上的各种系统软件 (包括操作系统本身) 的运行状态. 这是为保证安全敏感应用程序执行结果正确性和完整性的最大威胁, 也是 TCB 中必须包含操作系统的主要原因.

本系统主要设计就是用来对抗上述高特权级别的攻击模型. 在硬件层次上, 本系统可以防范硬件攻击, 比如 DMA 攻击, 以及挂载硬件 Debugger 攻击等. DMA 攻击的主要方式是利用 DMA 操作内存的能力, 绕过系统的保护直接对内存进行读写和控制, 从而达到窃取或者篡改安全敏感应用程序执行结果的目的. 对于复杂的硬件攻击, 比如对 CPU 和内存间的高速总线进行监视的攻击等, 不属于本系统保护范围. 但是, 由于该类攻击实现难度大, 实施条件苛刻, 在实际环境中基本上可以忽略.

3.2 设计目标

根据本系统的特点, 以及系统对抗的攻击类型, 本系统主要实现如下目标:

1) 隔离性. 隔离性是实现动态信任根的首要因素, 只有具备隔离性的系统, 才可以在系统启动之后采用动态的方式建立信任根. 本系统为安全敏感代码的执行提供一个完整的、具有很强隔离性的、硬件支持的虚拟执行环境, 保证安全敏感代码执行的隐秘性和完整性. 该环境将操作系统及其上的软件和 DMA 等硬件设备隔离, 并为安全敏感代码的执行提供一块由芯片保护的内存区域. 该内存区域在退出虚拟执行环境后, 在操作系统下不可见, 并由芯片对该区域进行保护. 这样可有效防止恶意的攻击者在退出虚拟执行环境后, 对执行内容的中间过程数据进行读写操作. 在虚拟执行环境中, 禁用了所有中断包括不可屏蔽 NMI 中断, 以及硬件调试器. 同时, 进入该环境的方式是通过触发 SMI 中断, 此类型中断属于不可屏蔽的外部中断, 独立于处理器的其他中断和异常处理程序, 由本地 APIC 直接触发, 攻击者很难对该中断进行劫持.

2) 最小化的 TCB. 本系统的目标之一是实现一个最小化的 TCB. 利用 SMM 提供的隔离环境, 在系统启动后动态的建立信任根, 可以将启动时的 BIOS, 运行时的操作系统, 系统的软件都排除在 TCB 之外, TCB 的软件部分只包含需要保护的安全敏感代码和本系统提供的额外几百行代码. TCB 的最小化简化了程序执行结果验证的复杂性, 增强了对安全敏感应用程序度量的可靠性. 验证者不需要对整个操作系统进行检测, 只需要对需要保护的安全敏感代码和额外的几百行代码进行校验就可以保证执行结果的可信性.

3) 细粒度的可验证保护. 可信计算中最重要功能是保护安全敏感应用程序的执行, 并提供远程的可验证性, 即远程主机可以对执行结果的正确性和完整性进行度量, 以确保应用程序在执行过程中信息未被窃取和篡改. 要保证执行结果的可信性, 远程主机需要对整个 TCB 进行验证. 传统的 TCB 过于庞大, 粗粒度的远程验证过程无法保证验证的可靠性和可信性. 最小化的 TCB 实现使验证更加准确, 更具有针对性. 同时, TCB 验证中不包含操作系统和其他软件等额外的部分, 也可以保证系统运行的信息不会泄露给远程主机, 保证系统其他信息的安全性.

3.3 系统整体结构

本系统使用 x86 架构的 SMM 模式, 在传统计算平台上实现了最小化的 TCB 基础架构. 系统的执行过程如下, 首先利用 SMI 中断机制暂停整个执行环境 (包括不可信的操作系统), 执行安全敏感代码, 然后恢复原有执行环境, 将结果传递回应用程序. 整个系统的结构如图 1 所示, 需要保护的安全敏

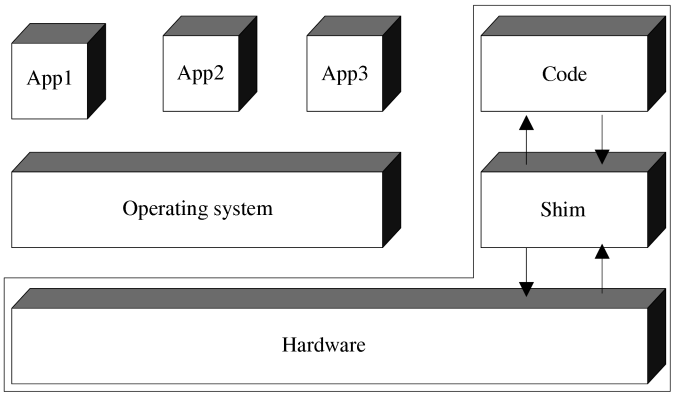


图 1　系统整体结构

Figure 1　Aegis architecture

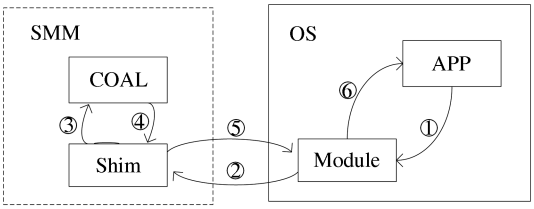


图 2　系统执行流程图

Figure 2　Execution flow

感代码称为应用程序逻辑块 COAL (chunk of application logic). 为 COAL 提供运行时支持和运行结果安全验证功能的模块称为 shim. shim 属于最小化 TCB 中的一部分, 为 COAL 的运行屏蔽了硬件底层的复杂性, 方便应用程序开发者开发高效的 COAL 代码. 一个最小化的 TCB 会话过程开始于触发 SMI 中断, 结束于执行 RSM 指令退出 SMM 模式, 并恢复到原有的运行环境. 整个 TCB 为图中细线框内的部分, 包括平台上的硬件, shim 和 COAL.

从功能结构上讲, 系统由数据交互、会话管理、远程验证、运行支持 4 部分组成. 其中数据交互是应用程序与系统内部模块进行交互的媒介, 主要为应用程序提供一个访问接口, 负责将输入参数、COAL 等传递给系统内部运行, 在执行完毕后负责将执行的结果返回给用户, 这部分是通过一个 Linux 内核模块实现. 会话管理是执行安全敏感代码的管理模块, 是整个系统的核心, 它负责安全敏感代码执行环境的构建, 运行时的安全保护, 以及运行完成后的处理工作. 远程验证主要是为外部实体提供结果的可验证性, 能够为远程主机提供程序运行结果的可信验证, 证明安全敏感代码被安全地执行, 并且结果是真实可信未被篡改的. 运行支持主要是对 COAL 的运行过程提供需要使用的一些基本的应用开发库, 包括参数处理、内存管理、加密算法等, 还包括供开发人员进行调试的串口通信库.

3.4　系统执行流程

基于 SMM 方法的 TCB 最小化系统的处理流程图如图 2 所示, 整个处理过程分为以下几个步骤:

1. 接收输入参数和 COAL. 应用程序将 COAL 和输入信息, 通过内核模块提供的设备接口传递到系统内, 内核模块为其分配内存, 并将 COAL 和输入参数放置到内存中的指定位置.

2. 进入 SMM. 应用程序在确认内核模块对输入信息初始化完成后, 向内核模块的控制接口写入命令, 内核模块触发 SMI 中断, 进入 SMM, 控制权转交给 shim 执行敏感代码, 系统进入受保护的虚拟执行环境.

3. 建立会话. 首先由 shim 进行整个系统执行环境的初始化, 建立 GDT, LDT 表、TSS 任务段描述符, 进入 32 位保护模式执行环境, 然后初始化 COAL 的执行环境, 限制 COAL 的运行权限, 对操作系统进行保护, 防止恶意或者出错的 COAL 对操作系统进行修改. 然后, shim 将控制权转交给 COAL 执行.

4. 执行 COAL. COAL 在隔离环境中运行, 执行完成后将输出结果依次写入到 COAL 内存的指定位置, 然后跳转到 shim 执行.

5. 结束会话. COAL 执行完毕后得到需要的执行结果, 然后由 shim 进行会话后续的处理工作. 首先, shim 对 COAL 的整个内存映像 (包含其自身), 输入输出参数进行 Hash 运算得到一个验证散列值, 并将该散列值保存在 SMRAM 内存中, 以便以后为应用程序或远程验证者提供结果的验证服务. 其次, 为了保证 COAL 自身和执行过程中中间数据的隐秘性, shim 在进行 Hash 运算后对 COAL 的内存映像进行清空. 最后, shim 执行退出指令, 将控制权重新返还给操作系统中的本系统内核模块.

6. 内核模块读取输出结果并将结果返回给应用程序.

以上是最小化 TCB 架构的一个完整执行过程, 整个执行过程从步骤 2 进入 SMM 之后, 到步骤 5 退出之前都是在无操作系统的隔离的虚拟执行环境中进行的, 因此保证了安全敏感代码的执行不会受到不可信平台上操作系统和恶意代码的干扰和破坏. 图 2 中右边方框内为应用程序与本系统进行交互的部分, 包含应用程序和内核模块, 该部分操作所产生的结果都是可以被验证的, 即若此部分被恶意程序入侵并篡改, 应用程序和远程验证者可以证明产生的结果是不可信的, 所以 TCB 中只包含左半部分, 右半部分为系统提供辅助支持, 并不包含在最小化的 TCB 中.

4 系统实现

整个 TCB 最小化架构的内存分布如图 3 所示, 右边部分为 shim 的内存布局, 左边为 COAL 的内存布局. SMRAM 内存区域主要有 3 部分: C_SMRAM, HSEG 和 TSCG. 其中 C_SMRAM 大小为 128 KB, HSEG 和 TSCG 部分最大可以达到 256 MB, 在传统的计算平台上, 一般使用 C_SMRAM 内存区域, 所以本架构的 shim 部分主要按照 C_SMRAM 对内存进行划分. 本系统是按照最小化的原则进行设计, 所以实现的系统自身很小, 在实际的使用中, 也只是对 SMRAM 内存的前 64 KB 进行划分. COAL 的内存由内核模块根据实际 COAL 的大小进行分配. 在进入隔离环境后, 由 shim 控制跳转执行, 其内存布局如图左半部分所示. 下面对各组成部分进行详细说明.

4.1 数据交互

数据交互模块负责会话前接收应用程序的输入信息和会话后将敏感代码输出信息传递给应用程序, 并为应用程序提供启动会话控制功能. 数据交互过程的实现是通过一个 Linux 内核模块来完成, 它在内核中建立一个设备模型, 应用程序通过操作该设备模型的设备文件来实现信息的传递. TCB 最小化设备模型 Mini_TCB 具备 4 个属性 inputs, outputs, COAL 和 control. 应用程序可以通过对这 4 个设备文件进行读写操作来传递命令和信息.

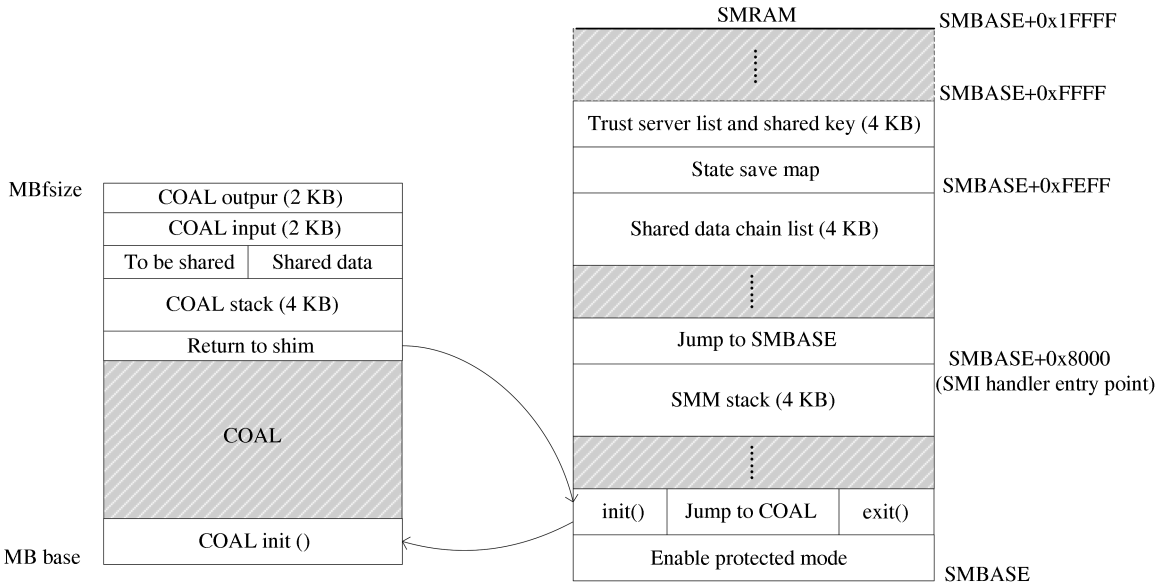


图 3　系统组成部分的内存分布图

Figure 3　Memory layout of aegis in SMM

inputs 属性是为应用程序提供输入参数的传递服务, 包含 inputs_read 和 inputs_write 两个操作. inputs_write 定义了应用程序向 inputs 设备文件写入输入参数时的操作. inputs_read 操作是为了方便应用程序检查输入参数是否写入正确, 以及查看已写入内存中输入参数的内容.

outputs 属性为应用程序提供读取和写入输出结果的服务, outputs 也有 outputs_read 和 outputs_write 两个操作, outputs_read 操作读取输出参数并传递给应用程序. outputs_write 操作为应用程序提供输出参数清空功能, 以便于下一次读取, 同时也能防止输出信息泄漏. 用户向 outputs 写入任意值, outputs_write 操作将 outputs 的内存区域置空值.

COAL 属性是为应用程序传递 COAL 的可执行映像, 它负责将 COAL 拷贝到指定内存地址. 默认为 COAL 分配的内存空间是 64 KB, COAL 映像从分配内存的基地址开始存放.

control 属性主要用来启动和控制会话过程. control 属性定义了 3 个控制参数 i, g 和 r, 参数 i 用来初始化 SMI 中断环境. 当应用程序向 control 设备文件中写入 i 值后, 内核模块会读取并设置 SMI 相关寄存器, 允许设备触发 SMI 中断. 参数 g 用来触发 SMI 中断, 启动本地会话. 应用程序在写入 i 值确信触发 SMI 中断的条件都具备后, 向 control 写入 g 值, 此时内核模块向 I/O B2 端口 (高级电源控制寄存器) 写入值触发 SMI 中断. 参数 r 是启动一个需要远程验证的会话. 系统在 SMM 中可以通过读取 B2 端口的值, 确定执行不同的操作.

整个数据交互由 Linux 内核驱动模块实现, 总共包含 816 行代码, 由于其操作的行为都是最终可验证的, 所以它并不包含在 TCB 之中.

4.2　会话管理

会话建立过程是从触发 SMI 中断进入 SMM 后开始的, 应用程序通过操作 control 设备文件启动会话, 整个操作系统被暂停执行, 系统进入隔离的虚拟执行环境, shim 获得系统的完全控制权, 并启动会话建立模块. 会话建立模块首先初始化 shim 执行环境, 按功能需要对 SMRAM 内存进行划分与分

配, 加载 smm_gdt 表, 将控制寄存器 CR0 的 PE 标志位置位开启保护模式, 然后利用长跳转进入保护模式执行. 在 SMM 中进入保护模式后, 内存寻址由段和偏移值的实模式方式转变为段选择子加偏移值的保护模式寻址方式, 利用段描述符程序代码可以访问系统的任意内存位置, 并可以跳转到系统的任意位置处执行, 而不用受限于 SMM 中对操作 1 MB 以上空间的指令需加覆盖前缀 (override prefix) 的限制. 进入保护模式后, 整个 TCB 最小化系统的地址空间如图 4 所示, shim 地址空间被定义到全局描述符表中, 特权级为 ring 0, 可访问整个 4 GB 的内存空间. COAL 的代码和数据空间被定义到了局部描述表中, 特权级为 ring 3, 只能访问其本身的内存区域, 以防止恶意或者执行错误的 COAL 对 shim 或者原操作系统内存空间的破坏.

COAL 的基地址和内存大小是由内核模块根据用户传递的 COAL 进行动态设置的, 根据这些信息设置好 COAL 的 LDT 表后, shim 将控制权转移给 COAL. shim 的执行环境位于保护模式的 Ring 0 下, 具有最高的权限, 而 COAL 的特权级为 Ring 3, 在 x86 的系统架构中, 特权级 0 不能直接把控制权转移到特权级 3. 为了能够跳转到 COAL 执行, 我们利用 IRET 中断返回指令, 通过人工建立中断返回场景, 实现程序控制流的转移. 具体的实现方法是在初始堆栈 init_stack 中人工设置一个返回环境, 即把 COAL 的 TSS 段选择符加载到任务寄存器 LTR 中, COAL 的 LDT 段的选择符加载到 LDTR 中以后, 把 COAL 的用户栈指针和代码指针以及标志寄存器值压入栈中, 然后执行中断返回指令 IRET. 该指令会弹出堆栈上的堆栈指针作为 COAL 的用户栈指针, 恢复预先设置的 COAL 标志寄存器的内容, 并且弹出栈中代码指针放入 CS:EIP 寄存器中, 从而开始执行 COAL 的任务代码, 完成从特权级 0 到特权级 3 代码的控制转移. 至此, 会话建立工作完成.

COAL 执行完成后, 系统的控制权需要转移回 shim 进行会话的拆除, 也就是控制权要从 ring 3 转移回 ring 0. 在 Linux 系统中特权级从低到高的转移主要采用了系统调用的方式, 用户程序通过库函数调用中断 int 0x80, 并在 EAX 寄存器中指定系统调用的功能号, 即可进入内核的管态, 使用内核资源. 在 SMM 环境下, 中断都是被禁用的, 我们采用了调用门方式在不同的特权级之间实现受控程序控制转移. COAL 通过调用门返回 shim, shim 进行后续的清理工作. 首先, shim 对 COAL 的内存进行清空, 以保护 COAL 及其中间执行过程中所产生数据的隐秘性, 防止恢复到操作系统后被恶意攻击者读幼胯 shim 设置 SMI_EN 标志位中的 EOS 位以保证系统可以再次触发 SMI 中断, 然后执行 WBINVD 指令对处理器的 Cache 进行清空. 执行 WBINVD 指令主要是为了防止 SMM Cache 攻击 [2)]. 系统最后执行 RSM 指令退出 SMM. 至此, 整个会话拆除过程完成. 会话管理是 TCB 最小化系统的核心部分, 程序最终实现的代码量是 192 行.

4.3 远程验证

COAL 执行完毕后, 系统还要为安全敏感代码以外的外部实体提供执行过程和结果的可验证性证明. 验证的任务是使程序的外部实体能够确信安全敏感代码是在隔离的虚拟执行环境中未被干扰的执行, 并且相应的执行结果是正确可信的. 远程验证是可信计算需要完成的一项重要任务. 在具有 TPM 硬件支持的系统上, 验证过程一般通过硬件完成. 然而本系统主要是为没有硬件支持的传统计算平台设计的, 因此设计了一个软件方式实现的远程验证过程. 执行安全敏感程序的一端, 需要向网络的另一端证明, 安全敏感的代码是在一个安全的可信环境下执行并且传递给另一端的结果是在该环境下产生并且未被篡改. TCB 最小化系统采用了一个 challenge-response 协议实现远程验证. 协议的主要实

2) Rafal W, Joanna R. Attacking SMM Memory via Intel®CPU Cache Poisoning. http://invisiblethingslab.com/resources/misc09/

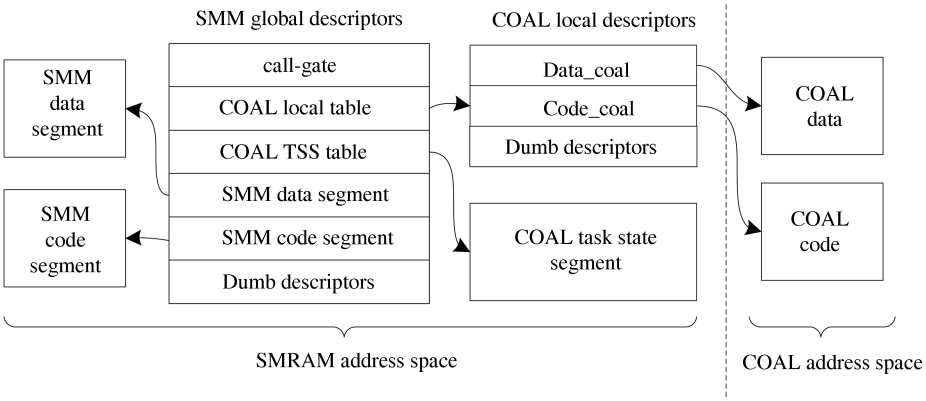


图 4　地址空间分配图

Figure 4　Address space allocation

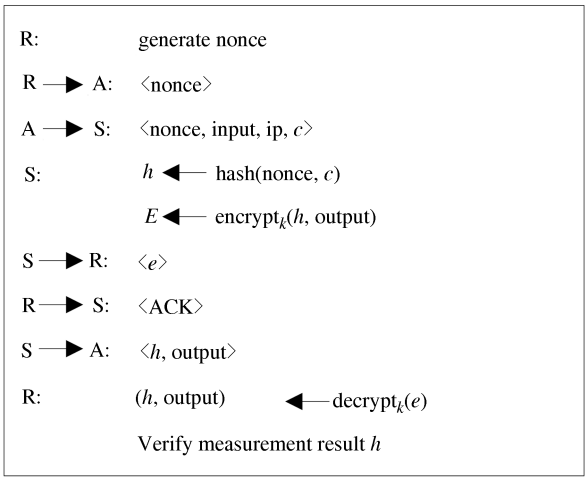


图 5　远程验证协议

Figure 5　The remote authentication protocol

现过程如图 5 所示, 其中 R 代表远程验证者, A 代表应用程序, S 代表 shim, c 代表 COAL:

首先, 远程验证者产生一个随机数 nonce, 然后远程验证者将包含 nonce 的 challenge 发送给不可信平台上的应用程序, 应用程序收到 challenge 后将 nonce、远程验证者的 IP、安全敏感代码 COAL、以及其输入参数发送给内核模块启动一个需要远程验证的会话. 启动会话后, COAL 在安全的隔离环境 SMM 中执行, 执行完毕后产生相应的输出结果. 然后控制权转移给 shim, shim 将随机数 nonce, COAL 的内存映像与输出结果, 以及此最小化 TCB 系统的 ID 信息进行 Hash 计算, 得到散列值 h. 系统选用了常用的散列算法 SHA-1 生成 Hash 值. 虽然现在已有方法获得简单 SHA-1 的碰撞, 但是对于复杂散列碰撞值的寻找依然很困难 [5]. 并且 SHA-1 一直被 TCG (可信计算组织) 选用作为 TPM 验证测量算法. 然后, shim 产生一个 1024 的 RSA 密钥对, 并用密钥对中的私钥对 h 进行签名, 然后将签名连同输出、公钥、系统 ID 一同发送给远程主机. 远程主机收到后会给 shim 发送一个 ACK 回复, shim 收到回复后进行结束会话工作. 在规定的时间内, shim 未收到回复, 则 shim 会重新发送一次信息, 连续三次超时, shim 将直接退出会话, 并返回给应用程序验证失败信息. 服务器收到验证信息后,

首先利用发送过来的公钥对签名进行解密, 然后利用已有的 COAL 信息, nonce 和发送过来的输出结果和 ID 重新进行 Hash 计算, 将两个散列值进行比对, 如果相同则证明结果未被篡改并且是真实可信的, 并且由 ID 信息可知结果是在该 TCB 最小化系统中执行得到的.

整个过程服务器每次发送的 nonce 是不同的, 并且 nonce 的熵值足够大, 这样可以有效的防止攻击者进行重放攻击. 不可信平台上发送给服务器信息的整个过程都在 SMM 中进行的, 所以保证了攻击者无法在对发送的数据进行截取和篡改.

4.4 运行时支持

应用程序的开发需要借助开发库的支持. 但是在 SMM 隔离环境中, 无法使用操作系统提供的各种开发库. 本系统为 COAL 开发提供了包括内存管理, 加密算法, 输入输出参数管理的运行时支持. 运行时库是选择性地加入到最小化 TCB 中, 即只有被 COAL 调用的库才被加入到 TCB 中.

在内存管理设计上, 并没有采用现代操作系统常用的段页式内存管理, 而只是采用了段式内存管理. 主要原因有以下几条, 首先, 本着最小化的原则, 需要保护的安全敏感代码一般都比较短小, 除去远程验证模块, 包含所有库的 shim 也只有 9 KB. 所以整个系统需要内存空间很少, 并不需要页式的大内存管理, 采用段式管理可以简化系统设计. 第二, 采用页式管理程序编译时需要进行页对齐, 这就导致 COAL 的执行映像可能包含额外空间, 容易被恶意攻击者注入恶意代码. 第三, TCB 最小化系统中只有一个 COAL 任务执行, COAL 整个执行映像都放在已经分配好的连续内存空间中, 所以不需要采用为多任务分配内存的页式内存管理方式, 使用分段式的线性地址到物理地址的转换就可以满足 COAL 的运行.

4.5 多会话交互

TCB 最小化系统在执行时会暂停操作系统的运行, 复杂的 COAL 会占用较长的 CPU 时间, 因此会对原操作系统上运行的程序产生影响, 特别是对实时性要求较高的程序. 为了既能够保证 COAL 完成所需的任务, 又能使执行的时间足够短, 本文提出了多会话技术. 多会话技术是指在保证可信的条件下, 多个系统会话间信息的传递和共享. 多个会话之间可以共享数据, 并且共享的数据只能被指定的 COAL 查看, 在控制权转移到操作系统后依然可以保证共享数据的隐秘性. 利用多会话技术可以将一个大的 COAL 分解为几个小的部分, 从而降低了一次 COAL 执行的时间, 而且分解后, 可以将其中安全性要求不高的部分分离出来, 交由操作系统来处理, 从而使 TCB 的大小减小到最低, 充分符合系统最小化的设计要求.

多会话技术实现的关键是对共享数据的安全隐秘存储, 首先是保证会话结束后, 数据不会被操作系统的恶意攻击者窃取, 其次是多个 COAL 之间共享的数据, 不会被其他的 COAL 读取和修改. 这两部分是利用 SMRAM 的隐蔽性和设置 COAL 的访问权限来实现的. 由前文所述, SMRAM 在设置了 D_LCK 位后, 在非 SMM 模式下对 SMRAM 的访问都会被转向到显存的位置, 这保证了会话结束后, 共享数据对操作系统是不可见的. 在前面章节提到根据系统安全要求 COAL 的执行环境被设置为 ring 3 级别, 并且只能访问其本身的内存空间, 所以 COAL 自身是无法读取系统在 SMRAM 中存放的共享数据, 这保证了多会话技术实现的第二个要求.

系统为多会话共享数据分配的空间如图 3 所示, 共享数据结构由共享数据及可访问共享数据 COAL 的 Hash 值组成, 该结构以链表的形式存放在 SMM 的状态存储区以下的 4 KB 位置处. 共享数据只能由 shim 进行管理和分配. 整个多会话数据共享的过程如图 6 所示.

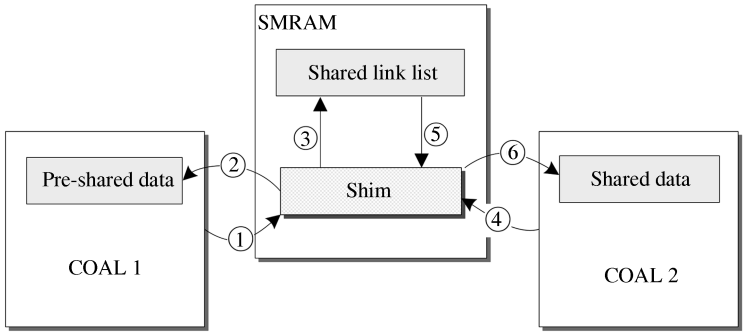


图 6　多个会话的数据共享过程

Figure 6　Data sharing process of multiple invocations

4.6　COAL 的构造

COAL 是一段逻辑独立的 32 位可执行代码, 为了更清楚的了解 COAL 的构造, 图 7 描述了一个包含输入和输出功能的简单 COAL 示例. 首先, 一个 COAL 应该包含头文件 coal.h, 该头文件除定义了输入输出管理函数的声明, 还包含了关于 COAL 参数空间、共享数据空间以及堆栈大小位置等基本信息的宏; 其次, COAL 的入口点函数为 coal_main, 类似于 C 语言的 main 函数, 是进入 COAL 后第一个被执行的函数. 输入参数由内核模块写入到 COAL 内存的指定位置, COAL 开发者只需要调用参数库中 pm_read 函数就可以进行读取. 如果开发者需要给调用 COAL 的应用程序输出信息, 可以使用参数库中一系列输出函数如 pm_append, 这些函数将会按照系统设置的参数格式将输出参数保存到 COAL 内存的指定位置. 在退出 SMM 之后, 应用程序可以使用 outputs_read 函数从 sysfs 文件系统的 outputs 设备文件中读取.

要将应用程序提取为 COAL, 首先要对应用程序进行划分, 划分为安全敏感的特权部分和非特权部分. 这部分工作可以利用文献 [6,7] 提到的手动方法或者文献 [8] 提到的自动方法完成. 每一个 COAL 都是逻辑独立的, 在对应用程序划分完成后, 使用本文工具就可以从特权部分将安全敏感部分的函数代码提取出来, 该工具可以将函数中用到的所有逻辑部分都提取出来, 比如一个全局变量, 或者调用的程序中的其他函数, 使之成为一个可以完全独立运行的应用程序逻辑. 将程序划分为特权和非特权部分并不是安全领域的一个新问题, 文献 [6] 提供一个应用开发接口, 利用该接口编译后的程序能够实现不同特权等级的划分, 缺点是开发人员要依赖于一个特定的接口. 文献 [8] 提出了一种基于程序静态分析的技术, 来自动化程序特权等级划分的过程, 大大简化了开发人员的干预, 但仍然需要用户提供少量的程序注释来帮助静态分析过程. 我们认为 COAL 是一种粗粒度的程序划分, 在对安全有特定需求的环境下, 借助于现有方法, 不会给开发人员带来过多的困难.

5　应用与性能分析

5.1　应用实例

TCB 最小化架构可以为多种类型的安全敏感应用程序提供可靠的安全保护. 为了对本系统的实际应用价值进行详细说明, 本节将系统应用到 3 个需要执行安全敏感任务或者处理安全敏感代码的程序上. 在这些实例中使用了前面提到的各种技术, 如远程验证、多会话交互等.

```
1  #include “coal.h”
2  void coal_main(void) {
3          int a, b, c;
4
5          pm_init(0);  // parameters initialization
6
7          a = *((int *) pm_read(1));
8          b = *((int *) pm_read(2));
9          c = a+b;
10
11         pm_append((unsigned char *) &c, sizeof (int));
12         return;
13 }
```

图 7 简单 COAL 示例

Figure 7 A simple COAL example

5.1.1 内核 Rootkit 检测

本文的 Rootkit 检测系统类似于文献 [9], 主要通过对内核的代码段, 内核调用表, 内核的模块列表周期性的进行 Hash 计算, 将得到的值与系统正常运行的值进行对比来检测内核 Rootkit. 不同的是本系统将这部分对比计算设计为 COAL 放入到 TCB 最小化架构中执行, 这样有效避免了文献 [9] 对时间的苛刻要求. 同时, 在 SMM 中将结果用公钥加密发送给远程主机, 可以有效地防止 Rootkit 对结果的篡改. 利用密钥的隐蔽性保证远程主机可以通过解密, 证明结果是在本系统中执行后得到, 因为 Rootkit 无法取得保存在 SMRAM 中的密钥也就无法伪造加密数据. 最后, 远程管理员通过将结果和正确值进行对比, 就能检测出系统是否被 Rootkit 入侵.

5.1.2 软件注册保护程序

因为商业软件的注册验证算法都是保密的, 本文编写了一个简单的注册验证系统, 该系统使用机器的 MAC 地址作为密钥, 使用 3-DES 算法对用户名进行加密生成注册码. 注册码的生成, 以及与用户输入进行对比的过程被设计为 COAL 执行. 整个验证过程都在 TCB 最小化系统中进行, 生成的注册码不会离开系统以外, 会话结束以后整个 COAL 及其中间结果都将被清空. 因此攻击者无法利用调试工具对软件注册过程进行监视, 也就无法通过逆向工程得到注册算法.

5.1.3 FTP 密码验证程序

为了说明多会话技术在实际中的应用, 本文将 TCB 最小化架构应用到 FTP 服务器的密码验证过程中, 利用两个系统会话保护用户 FTP 登陆密码的安全.

首先, 客户端用户向服务器请求登录, 服务器接收到用户的登录请求后, 启动密码验证的第一个会话执行 COAL1, COAL1 利用 RSA 加密算法产生密钥对 K, K^{-1}, 然后将私钥 K^{-1} 和 COAL2 的 Hash 值以共享数据结构的形式保存在 COAL 的预共享区域中. 然后将 K 在 SMM 中利用远程验证模块直接传送给客户端, COAL1 执行完毕后, shim 提取共享数据块, 并保存在 SMRAM 的共享链表处, 然后清除 COAL 内存的秘密信息并退出会话. 客户端接收到公钥 K 后, 用 K 加密登录的用户名和密码生成 e, 然后将 e 发送给服务器. 服务器收到后, 启动第二个会话验证用户登录. 首先 shim 在共享链表上查找到相应的共享信息以及整个服务器的用户账户信息, 然后写入到 COAL2 内存的共享

区域内, 接着跳转到 COAL2 执行. COAL2 首先读取服务器的私钥 K^{-1} 并解密用户登录信息, 然后将用户发送的账户密码信息和服务器存放的信息进行对比, 并把结果返回给服务器, 服务器根据结果决定用户是否登录成功, 至此用户登录过程结束. 整个过程服务器产生的私钥和解密的登录账户和密码的原文都只是在 TCB 最小架构的会话中出现, 这样有效地保护了 FTP 登录用户密码信息不被恶意攻击者窃取.

5.2 性能评测

本系统的 TCB 最小化架构是基于 Intel x86 系统管理模式实现的 (AMD 中也有类似特性的系统管理模式, 根据其硬件特性进行相应的修改, 同样可以移植到 AMD 平台上). 测试所使用的硬件环境是, 系统测试安装主机: 方正 E330 主机, 2.6 GHz 主频的 CPU, Intel 845 芯片组, 512 MB 内存, 瑞昱 RTL8139 网卡. 远程验证主机: Intel P4 超线程主机, 512 MB 内存. 测试主机和远程验证主机处于同一个局域网.

最小化 TCB 系统在实现安全敏感代码隔离保护和满足最小化要求的同时, 也带来了相应的时间性能损耗. 本节以传统 TCB 的运行时间为基准对系统及上文提到的 3 个应用实例所产生的时间开销进行详细分析. 因为系统是在隔离环境中运行, 所以无法使用常用的时间测量工具对系统的时间开销进行测量, 因此本文首先采用时间指令 RDTSC 来计算各部分消耗的 CPU 周期, 然后将 CPU 主频转化为相应的毫秒时间, 从而获得系统各部分的时间开销. CPU 主频从 /proc/cupinfo 中读取.

5.2.1 系统整体性能评测

为了更好地对 TCB 最小化系统应用实例的时间开销进行说明, 本文对系统本身以及各部分组件的运行时间进行测量分析. 一次系统会话的执行时间为系统在 SMM 隔离环境中执行的时间. 为了测量系统会话的时间, 在触发 SMI 中断即进入 SMM 指令之前和之后分别执行一次 RDTSC 指令, 其差值即为 CPU 周期数. 为了更好地对最小化 TCB 系统的时间开销进行说明, 本实验中将 COAL 只设置成一句访问调用门返回 SMM 的代码, 通过这种方式测量了在 COAL 没有做任何操作情况下的时间损耗, 也就是 COAL 之外系统自身的执行时间.

首先, 本实验测量了为 COAL 分配从 0 KB 到 128 KB 不同内存大小时系统的执行时间, 结果表明, 系统的时间开销随着为 COAL 分配内存的增加而呈线性增大, 所以开发者为 COAL 选择合适的内存大小可以有效提高系统的性能, 降低时间开销.

其次, 本实验执行了一系列的基准测试, 测量了整个 TCB 最小化架构也就是 shim 各部件的时间. 实验中将 shim 分为 3 部分, 每个实验执行 30 次, 取其平均值作为每次实验的结果. 实验结果如表 1 所示, 从表中可见, shim 的主要时间开销来自于对 COAL 的 Hash 计算.

SMI 中断时间是指在触发 SMI 中断后, 处理器保存当前系统状态的时间以及在退出 SMM 后, 处理器恢复保存的状态的时间. SMI 中断处理时间是指 shim 的 SMI 工作处理时间, 主要包括初始化 COAL 的执行环境, 如建立 GDT, LDT 表等, 以及在退出会话前的后期处理工作, 如设置 SMI 寄存器, 清除 COAL 内存等等. Hash COAL 时间是指 shim 对 COAL 的整个内存进行 Hash 计算的时间, 是整个 shim 中消耗时间最多的部分.

5.2.2 内核 Rootkit 检测评测

本实验中, Rootkit 检测程序运行在一个已经被入侵的主机上. Rootkit 检测程序在系统的后台每

表 1 shim 中各部分组件的时间开销

Table 1 Breakdown of aegis shim overhead

Aegis shim parts	Time (ms)
SMI (interrupt)	0.021
SMI handler	0.019
Hash of COAL (64 KB)	42.98
Total query latency	43.02

表 2 Rootkit 检测程序时间开销

Table 2 Overhead of rootkit detector

Task	Standard time (s)	Rootkit detect (s)	Overhead
Unzip kernel	86.063	87.762	1.97%
Copy kernel	100.979	102.604	1.61%

隔 5 秒钟运行一次. Rootkit 选用的是 adore-ng 0.56[3)], adore-ng 是一个 Linux 内核级 Rootkit. 它利用 Linux 的可装载模块 (LKM) 安装, 修改系统调用的入口地址, 劫持正常的系统调用. 通过对比正常系统调用表的 Hash 值, Rootkit 检测程序成功检测出 adore, 并将结果发送给远程管理员.

为了评估 Rootkit 检测程序的性能, 本实验采用了 2 个有代表性的任务对系统性能进行测量. 第一个是计算密集型的任务, 解压内核源码包; 第二个是 I/O 密集型的任务: 将解压缩的源码文件夹拷贝到另一个磁盘. 实验使用 Linux 中的 time 命令计算每一个任务的执行时间. 实验结果如表 2 所示, 可以看到, 周期性的运行 Rootkit 检测程序带来的时间开销很小, 时间延迟在用户可接受的范围内.

另外, 表 3 给出了 Rootkit 检测软件各个部分的时间开销. 从表中可以看出, 整个程序中最耗时间的是 shim 处理过程和对内核进行 Hash. 由先前得到的数据可知, shim 这部分的时间开销可以通过减小为 COAL 分配内存的大小进行优化. 在一些没有严格检测要求的情况下, Hash 内核这部分时间可以通过减少对内核某些关键部分的 Hash 来实现, 比如只检测内核调用表.

5.2.3 软件注册保护程序评测

本文分别开发了使用和不使用 TCB 最小化系统保护的两个版本程序. 在未使用保护的程序中, 可以轻易地通过 gdb 调试器得到关键注册函数, 在单步跟踪分析后便可以得到注册算法. 相反, 在有 TCB 最小化系统保护的程序中, 通过 gdb 调试器只能看到输入和输出函数, 整个注册码的生成过程都在隔离环境中运行, 对攻击者不可见. 因此, 本系统可以在一定程度上有效保护软件不被恶意破解.

在实验中为 COAL 分配的内存大小为 32 KB. 实验测量了在 TCB 最小化系统保护下的整个注册验证会话的时间开销, 同时也测量了整个会话中各个部分的时间. 实验结果如表 4 所示, 从表中可以看出, 整个验证过程的主要时间开销来自于 shim 和计算注册码的 3-DES 算法.

5.2.4 FTP 密码验证程序评测

因为我们修改的 FTP 程序只是在用户登陆过程中加上了基于 TCB 最小化架构保护的密码验证功能, 所以对 FTP 数据传输的性能并没有影响, 程序的主要性能损耗来自于建立连接的验证过程. 在

3) http://stealth.openwall.net/rootkits/

表 3 Rootkit 检测软件各个部分的时间开销

Table 3 Breakdown of rootkit detector overhead

Rootkit detector	Time (ms)
Shim	41.2
Hash kernel	38.3
Remote authentication	1.1
Total latency	80.6

表 4 注册验证 COAL 的各部分时间开销

Table 4 Breakdown of license verification COAL overhead

Register module	Time (ms)
Shim	20.3
Get MAC address	0.4
3-DES	12.2
Total latency	32.9

表 5 FTP 密码验证 COAL 的时间开销 COAL1(64 KB)

Table 5 FTP password authentication overhead-COAL1 (64 KB)

Parts	Time (ms)
Shim	38.7
Generate key	329.2
Send public key	1.3
Total time	369.2

表 6 FTP 密码验证 COAL 的时间开销 COAL2 (16 KB)

Table 6 FTP password authentication overhead-COAL2 (16 KB)

Parts	Time (ms)
Shim	10.2
Decrypt	7.9
Total time	17.9

第一个实验中, 本文已经测量了用户登录过程的时间开销, 再加上密码验证功能后, 整个登录时间为 392.5 ms, 而正常的登录时间只需要 3.2 ms. 从结果上看, 虽然时间开销有所增大, 但是仍小于 1 s, 因此仍然在用户可以容忍的范围内.

实验一中修改后的 FTP 程序的时间开销主要来自于服务器端和客户端两个部分. 服务器端的时间开销则来自于两个密码验证 COAL 的执行时间. 为了对这部分时间开销进行详细说明, 本节第二个实验测量了服务器两个 COAL 的整体执行时间和其中各部分的执行时间, 结果如表 5 和 6 所示, 主要的时间消耗来自于 1024-bit RSA 密钥对的产生. 这部分时间可以通过选用不同的加密算法和不同大小的密钥对进行优化. 客户端的时间延迟主要来自于获得密钥和利用密钥进行加密的过程, 在第三个实验中对该部分进行了测量, 测量的结果大约需要 4.2 ms, 相比服务器的时间开销基本可以忽略.

6 相关工作

6.1 TCB 最小化研究现状

从可信计算的概念提出以来, 研究学者一直致力于减小可信计算基 (TCB), 他们将各种最新的技术应用到可信计算中, 从软件或硬件的方面来减小 TCB 中所包含的内容, 从而实现对安全应用程序可靠的验证. Nizza[1] 利用虚拟化技术和微内核技术来减小 TCB 的大小. Nizza 将应用程序中安全敏感的部分抽出变为一个 AppCore, 并将其放入一个基于 L4 微内核的环境中执行. 应用程序的非安全敏感部分被放入到虚拟机中执行, 利用虚拟机中宿主机和客户机间良好的隔离性有效的降低了 TCB 的大小, TCB 中只包含虚拟机和微内核. Nizza 成功的将原有的百万数量级的 TCB , 降低到十万数量级. 此外, Terra[10]、Proxos[11] 等都是采用类似 Nizza 的方法来减小 TCB. 这些利用虚拟化技术减小 TCB 的方法, TCB 必须包含虚拟机, 因此 TCB 的大小仍然非常庞大.

很多研究采用加装可信硬件的方式保护安全敏感代码. 其中最具代表性的是 Dyad 的 HW 架构 [12] 和 IBM 4758[13]. 它们将安全敏感应用程序放入可信硬件中执行, 从而有效减小了 TCB 的大小. 这种利用可信硬件减小 TCB 的方式, 因成本较高和适用范围不广等特点, 导致了其实际应用性并不高. 文献 [14~16] 利用 AMD 硬件虚拟化技术实现了一个真正意义上的最小化 TCB 架构 Flicker. 系统在发出硬件虚拟化指令 SKINIT 后, 会启动一个由硬件保护的安全内存 (SLB), 由该内存加载虚拟机核心组件, 从而有效地保护虚拟机不被恶意软件修改. Flicker 利用硬件虚拟化提供的这种强隔离特性, 将应用程序的安全敏感代码放入 SLB 中执行, 实现了一个只包含的几百行代码 TCB. 但是 Flicker 只能工作在具有硬件虚拟化支持的平台上. 本文提出的 TCB 最小化技术具有更强的通用性. TrustVisor[17] 提出了一种类似于 Flicker 的 TCB 执行环境, 该环境基于硬件虚拟化技术和 TPM 硬件, 主要目的是改进 Flicker 中较大的性能开销.

6.2 SMM 相关研究

文献 [18] 利用 SMM 提升 OpenBSD 系统用户的权限. 在进入 SMM 之后, 系统会将处理器的上下文信息保存在相应内存中, 通过在 SMM 中修改保存的操作系统安全级别变量的值, 使得在退出 SMM 后将该值写入到系统中, 成功的实现提升当前用户为特权级. 文献 [19] 详细解释了关于 x86 架构中 SMM 的一些细节特性并对于如何利用 SMM 实现一种基于硬件保护的恶意软件提供思路. 通过实验证明在 SMM 模式下运行的程序可以操纵整个系统内存, 并指出通过在 SMM 中修改内存的方式更改原有操作系统的结构构造一个功能强大的高特权级的 Rootkit. 本文主要目标是利用 SMM 实现一个最小化 TCB 系统. HyperCheck[20] 提出一种利用 SMM 进行程序完整性检测的方法, 其功能只是本文所设计系统的子集, 应用分析小节中所提到的内核 Rootkit 检测应用类似于 HyperCheck, 只是功能上简化.

7 总结

本文以保护应用程序安全、减小 TCB 的大小为主要研究目标, 利用 SMM 所具有的良好隔离特性实现一个最小化的 TCB. 围绕这个主题, 本文设计并实现了一个能够有效保护安全敏感代码执行安全的最小化 TCB 系统架构, 该系统具有规模小、安全性高、性能损失较少的特点, 并能为验证者提供精确可靠的细粒度的结果验证.

参考文献

1 Lenin S, Calton P, Hermann H, et al. Reducing TCB complexity for security-sensitive applications: three case studies. ACM SIGOPS Oper Syst Rev, 2006, 40: 161–174

2 Shen C X, Zhang H G, Feng D G, et al. Survey of information security. Sci China Ser E-Inf Sci, 2007, 37: 129–150 [沈昌祥, 张焕国, 冯登国, 等. 信息安全综述. 中国科学 E 辑: 信息科学, 2007, 37: 134–140]

3 Lampson B, Abadi M, Burrows M, et al. Authentication in distributed systems: theory and practice. ACM Trans Comput Syst, 1992, 10: 265–310

4 Cong N. Dynamic root of trust in trusted computing. In: TKK T-110.5290 Seminar on Network Security, Helsinki, 2007

5 Wang X Y, Yin Y Q L, Yu H B. Finding collisions in the full sha-1. In: Proceedings of the 25th Annual International Cryptology Conference (CRYPTO). LNCS 4593. Berlin: Springer, 2005. 17–36

6 Kilpatrick D. Privman: A library for partitioning applications. In: USENIX Annual Technical Conference. Berkeley: USENIX Association, 2003. 9–14

7 Provos N, Friedl M, Honeyman P. Preventing privilege escalation. In: Proceedings of the USENIX Security Symposium. Berkeley: USENIX Association, 2003. 16–22

8 Brumley D, Song D. Privtrans: Automatically partitioning programs for privilege separation. In: Proceedings of USENIX Security Symposium. California: USENIX Association, 2004. 5–12

9 Seshadri A, Luk M, Shi E, et al. Pioneer: verifying integrity and guaranteeing execution of code on legacy platforms. In: Proceedings of the Symposium on Operating Systems Principals (SOSP). New York: ACM, 2005. 1–16

10 Garfinkel T, Pfaff B, Chow J, et al. Terra: a virtual machine-based platform for trusted computing. In: Proceedings of the Symposium on Operating Systems Principals (SOSP). New York: ACM, 2003. 193–206

11 Ta-Min R, Litty L, Lie D. Splitting interfaces: making trust between applications and operating systems configurable. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Berkeley: USENIX Association, 2006. 279–292

12 Yee B S. Using secure coprocessor. Dissertation for the Doctoral Degree. Carnegie Mellon: University of Pennsylvania, 1994. 89–91

13 Smith S W, Weingart S. Building a high-performance programmable secure coprocessor. Comput Netw J Comput Telecommun Netw, 1999, 31: 831–860

14 McCune J M, Parno B J, Perrig A, et al. Flicker: an execution infrastructure for TCB minimization. In: Proceedings of the EuroSys Conference (EUROSYS). New York: ACM, 2008. 315–328

15 Jonathan M M, Bryan P, Adrian P, et al. Minimal TCB code execution (extended abstract). In: Proceedings of the IEEE Symposium on Security and Privacy (SP). Washington: IEEE, 2007. 267–272

16 McCune J M, Parno B, Perrig A, et al. How low can you go? recommendations for hardware-supported minimal TCB code execution. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM, 2008. 14–25

17 McCune J M, Qu N, Li Y L, et al. TrustVisor: efficient TCB reduction and attestation. In: Proceedings of the IEEE Symposium on Security and Privacy (SP). Washington: IEEE, 2010. 143–158

18 Duflot L, Etiemble D, Grumelard O. Using CPU system management mode to circumvent operating system security functions. In: Proceedings of the 7th CanSecWest Conference. Vancouver, 2007. 245–253

19 Shawn E, Sherri S, Cliff Z. SMM rootkits: a new breed of OS independent malware. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. New York: ACM, 2008. 11–23

20 Wang J, Stavrou A, Ghosh A K. HyperCheck: A hardware-assisted integrity monitor. In: Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection (RAID). Heidelberg: Springer-Verlag, 2010. 158–177

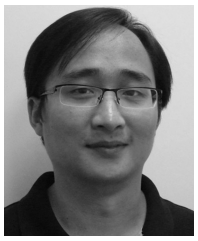
A light-weight, secure and trusted virtual execution environment

CHEN Hao, SUN JianHua*, LIU Chen & LI HaiWei

School of Information Science and Engineering, Hunan University, Changsha 410082*, China*
*E-mail: jhsun@aimlab.org

Abstract Traditional trusted computing base (TCB) contains the OS, device drivers, and all the applications, and the validation of the entire TCB is tremendously complicated. To solve this problem, we propose a TCB minimization architecture that leverages hardware isolation features such as system management mode provided by CPU, executing security-sensitive code of applications in a virtual environment to exclude these unsecure-sensitive code, OS and other applications out of TCB, which makes the TCB only include security-sensitive code and some management code of the virtual environment. Even if attackers control OS and part of hardware (DMA, hardward debugger, etc.), the isolated environment can guarantee the security and integrity of sensitive code execution. Meanwhile, the system provides reliable fine-grained validation, which convinces that the correct security-sensitive code is executed and the whole execution is protected by our system.

Keywords trusted computing, trusted computing base(TCB), system management mode (SMM), virtualization, TCB minimization



CHEN Hao was born in 1977. He received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan in 2005. Currently, he is an Associate Professor at School of Information Science and Engineering, Hunan University. His research interests include virtual machines, operating systems, distributed and parallel computing and security. Dr. Chen is a member of the IEEE.

SUN JianHua was born in 1977. She received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan in 2005. Currently, she is an Associate Professor at School of Information Science and Engineering, Hunan University. Her research interests include security and operating systems. Dr. Sun is a member of the CCF.