# 基于硬件虚拟化技术的隐藏进程检测技术*

温 研[1]，赵金晶[2]，王怀民[1]
(1. 国防科学技术大学 计算机学院，长沙 410073；2. 北京系统工程研究所，北京 100101)

**摘 要**：随着越来越多的 PC 用户习惯于从互联网上下载和执行各类软件，潜在的自隐藏恶意代码已成为亟待解决的安全问题，而进程隐藏是这类恶意代码最常用也是最基本的规避检测的自隐藏技术。针对这个问题，提出了一种新的基于硬件虚拟化技术的隐藏进程检测技术——Libra。Libra 通过构造一个轻量级的虚拟机监视器(libra virtual machine monitor, LibraVMM)实现了从虚拟层隐式获取真实进程列表(true process list, TPL)的新技术。与已有的基于虚拟机技术的解决方案相比，Libra 具有两个特色，即动态的操作系统迁移技术和不依赖于操作系统的隐式进程自省技术。测试结果证明了 Libra 检测隐藏进程的完整性，具有很好的实用性。

**关键词**：虚拟机监视器；自隐藏恶意代码；硬件虚拟化技术；进程隐藏
**中图分类号**：TP309.5 **文献标志码**：A **文章编号**：1001-3695(2008)11-3460-03

# Detecting hidden process with hardware-assisted virtual machine monitor

WEN Yan[1], ZHAO Jin-jing[2], WANG Huai-min[1]
(1. *School of Computer, National University of Defense Technology, Changsha* 410073, *China*; 2. *Beijing Institute of System Engineering, Beijing* 100101, *China*)

**Abstract:** With more and more PC users were accustomed to download and execute programs from Internet, stealth malware had become a major threat to the PC computers. Process hiding was a powerful stealth technique commonly used by stealth malware to evade detection by computer users and anti-malware scanners. This paper proposed a new approach called Libra for detect hidden processes implicitly. Libra implemented a novel lightweight hardware-assisted VMM to obtain the true process list (TPL) from deep within the system. Compared to existing VMM-based approaches, Libra provides two unique advantages: dynamic OS migration and implicit introspection of TPL. The functionality evaluation shows the completeness and effectiveness of Libra.

**Key words:** virtual machine monitor(VMM); stealth malware; hardware-assisted virtualization; process hiding

自隐藏恶意代码利用各种技术使系统工具无法检测到其存在，进程隐藏则是这类恶意代码最常用也是最基本的规避检测的自隐藏技术，此类恶意代码已对计算机系统的安全构成了很大威胁。

多视图验证技术(cross-view validation)[1]是一种检测进程隐藏的有效机制，它分别从可信视图和非可信视图获取进程列表，两个列表集合的差集即为被隐藏进程。实现这种机制的关键就是如何获取真实准确的可信进程视图。已有的可信视图获取技术主要分为基于操作系统内核和基于虚拟机监视器(virtual machine monitor, VMM)两类。但是前者无法抵御内核态恶意代码的攻击。而现有的基于虚拟机监视器的检测技术则依赖于可被内核态恶意代码窜改的客户操作系统的内核数据结构，而且它们都是针对在虚拟机中重新安装部署的操作系统，不能检测实际需要保护的宿主计算机系统中的自隐藏恶意代码。针对这些问题，本文提出了一种新的基于轻量级虚拟化技术的隐藏进程检测技术——Libra。除了能够有效抵御内核态恶意代码的攻击外，与已有的基于虚拟机技术的解决方案相比，Libra 具有两大优势：a) 动态的操作系统迁移技术，准备检查隐藏进程时，Libra 可以动态创建 LibraVMM 并将运行中的操作系统迁移到 LibraVMM 之上，检测完毕后，可以动态退出 LibraVMM 从而将操作系统恢复到原有的本地运行方式，这种动态的迁移方式明显减少了对操作系统运行性能的影响；b) Libra 不依赖于操作系统的隐式进程自省技术，利用处理器管理进程地址空间的特征构造了不依赖于操作系统的进程枚举技术，使得所有的检测操作均在虚拟机监视器(LibraVMM)的层次完成，有效保护检测机制的安全性。

## 1 Libra 体系结构

如前所述，现有基于虚拟机监视器的检测技术实际检测是在虚拟中安装运行的操作系统中潜在的自隐藏代码，而不是 PC 用户真正关心的宿主操作系统。Libra 与这些基于虚拟机监视器的检测技术也有所不同，它无须先启动虚拟机监视器再启动虚拟机中安装的操作系统，而是通过动态地将运行中的宿主操作系统迁移到 LibraVMM 之上，真实再现了操作系统的动态信息，保证其真实性。

如图 1 所示。Libra 的核心组件为 LibraVMM，进程检测组件则在 LibraVMM 内实现。在 Libra 检测模型中，宿主操作系统运行于两种状态，即准备检测前的本地运行状态和迁移到

收稿日期：2008-01-01；修回日期：2008-03-24 基金项目：国家“973”计划资助项目(2005CB321801)；国家杰出青年科学基金资助项目(60625203)

作者简介：温研(1979-)，男，博士研究生，主要研究方向为信息安全与虚拟化技术等(celestialwy@126.com)；赵金晶(1981-)，女，助理研究员，博士，主要研究方向为信息安全；王怀民(1962-)，教授，博导，主要研究方向为分布式计算软件及应用、网络与信息安全等.

LibraVMM 之上后虚拟化运行状态( virtualized mode) 。在本地运行状态下,操作系统直接运行在硬件系统之上,所以没有任何性能损失。

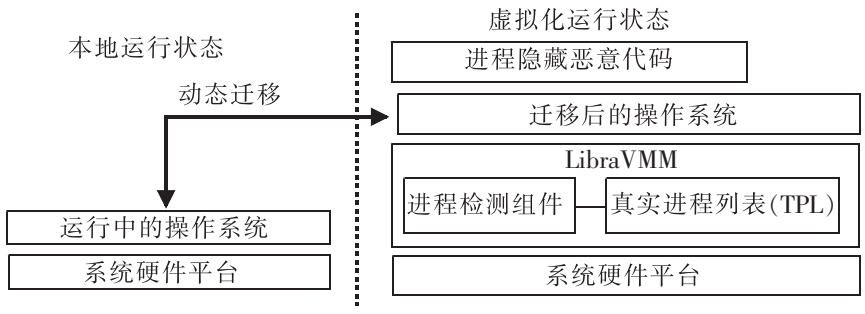


图 1　Libra 体系结构

若准备进行隐藏进程检测, Libra 将动态创建 LibraVMM,并将运行中的操作系统迁移到 LibraVMM 之上,进程检测组件负责提取真实的进程创建与退出信息。

进程检测组件的核心功能为获取 TPL,进程检测组件利用了 Intel x86 处理器管理进程地址空间( process address space) 的通用机制实现了 TPL 的获取。

对于所有 Intel x86 体系结构的处理器( 包括 AMD 的 x86 处理器) ,每个进程均有一个独立的进程地址空间( 32 bit 的处理器为 4 GB) 。进程基于逻辑地址访问内存,启动了段式内存管理的 x86 处理器首先将此逻辑地址转换为线性地址,接着处理器通过页式内存管理机制将由段式管理形成的线性地址映射为物理地址。x86 处理器的页式内存管理将进程地址空间分为固定大小( 一般为 4 KB) 的页面,每一个页面被映射到物理内存中的一页。进行线性地址到物理地址的转换时, x86 处理器的页式内存管理器利用页目录表—页表—页描述项这三层结构维护地址空间中从线性地址到物理内存页的映射, 32 bit 的线性地址被划分为三个部分,即 10 bit 的页目录表下标、10 bit 的页面表下标、12 bit 的页内地址偏移。x86 处理器中负责存放指向当前页目录表指针的是控制寄存器 CR3。寻址方式过程为从 CR3 取得页目录表的基地址,然后根据 10 bit 页目录表下标和 12 bit 的页内地址偏移得到的基地址,取得相应页面表的基地址,接着根据 10 bit 页表下标和之前得到的基地址,从页表中取出相应的页描述项,最后将页描述项中的页面基地址和线性地址中的 12 bit 页内地址偏移相加,得到物理页地址。

如前所述, CR3 寄存器的值指向了一个进程的地址空间的页目录表,因此,实际上 CR3 寄存器与进程形成了一一映射关系。Libra 基于 Intel 的硬件虚拟化技术( virtualization technology, VT) [2] 实现了 LibraVMM,而在创建了 LibraVMM 后,所有修改 CR3 寄存器的操作都将被 LibraVMM 捕获。在 LibraVMM 中实现的进程检测组件正是利用这一特性通过监视 CR3 寄存器的变化来获取 TPL。

## 2　Libra 实现

上一章描述的 Libra 的体系结构显然是与操作系统无关的,但考虑到目前 PC 平台最常见的配置是 Windows 操作系统与 Intel x86 处理器,所以 Libra 的原型系统首先在配置了支持 Intel VT 技术的处理器且在安装 Windows 计算机系统上实现。Libra 的隐式进程自省技术的详细描述参见文献[ 3] 。

Intel VT 技术于 2006 年推出,目前包括面向 x86 处理器的 VT-x 技术与面向安腾处理器的 VT-i 技术, Libra 针对普遍使用 x86 体系结构的处理器,因此 Libra 的原型系统基于 VT-x 技术实现,而本文后面提及的 VT 特指 VT-x 技术。Intel VT 技术的核心是给 x86 CPU 中各种特权指令的执行都增加可以进行自陷( trap) 的功能。VT 技术使得 CPU 运行在两种不同的模式下,即 VMX root 和 VMX non-root 模式。这两种模式都支持 0 ～3 的四个特权级。客户虚拟系统及其应用程序运行在 VMX non-root 模式下,虚拟机监视器运行在 VMX root 模式下。VT 引入了两种状态转换,从 root 模式转换到 non-root 模式称之为 VM entry;反之则称之为 VM exit。VM entry 和 VM exit 都会重新加载 Intel 的控制寄存器 CR3,以便 VMM 和 Guest OS 有不同的地址空间。与传统 x86 处理器相比, VT 处理器最大的不同就是当处理器在 non-root 模式下运行时,一些特权指令的执行将会导致 VM exit,执行 CPUID、mov CR3、RDMSR、WDMSR 等指令会无条件地触发 VM exit;而其他的指令、中断和异常是否触发 VM Exit 则由新引入的虚拟机控制结构( virtual machine control structure , VMCS) 管理。这种机制为 VMM 解释执行这些指令和异常提供了有效支持。

VM entry 会先加载客户系统状态区域的 VMCS,然后进入 VMX non-root,并且立即引发中断。这样, VMM 就可以通过对 IDT 的修改,在系统进入 VMX non-root 的第一时间触发事件,以便模拟事件的传递。

VM exit 则会保存 CPU 状态到客户系统状态区域,并且加载 host 状态区域的信息后返回 VMM。所有的 VM exit 使用公共的进入 VMM 的入口点( entry point) ,并且每次 VM exit 都会提供引发 VM exit 的详细原因,以便 VMM 进行相应的处理。

通过 VMCS 的设置, LibraVMM 可以配置处理器,使得每次 CR3 被修改均产生 VM exit 事件,从而使得进程检测组件捕获到此事件,并获取新的 CR3 寄存器的值以提取 TPL。通过动态加载 LibraVMM 及其进程检测组件来获取 TPL 的具体运行流程主要包括以下几个步骤:

a) 加载 LibraVMM 驱动。只有在内核态( ring 0) 才能执行 Intel VT 技术引入的新指令和处理各类 VM exit 事件( 包括修改 CR3 导致的 VM exit 事件) ,因此必须以 Windows 内核驱动的形式实现 LibraVMM 以使其能够进入操作系统内核执行特权指令。动态加载 LibraVMM 即利用 Windows 提供的标准接口加载 Libra 驱动程序并利用 I/O 控制指令通知其开始执行创建 LibraVMM 的操作。

b) 在所有逻辑处理器上启用 VMX 模式。Libra 通过设置 CR4 控制寄存器中为 VT 新增的相应控制位来启动 VMX 模式。接着, Libra 在以 4 Byte 为边界对齐的非分页内存上创建 VMXON 区段,最后通过传递 VMXON 区段的物理地址执行 VMXON,此操作必须在所有的逻辑处理器上均执行一遍。所谓逻辑处理器,就是从操作系统的角度,能够提供完整处理器计算功能的处理器单元。例如,对于多核处理器,每个核即为一个逻辑处理器;而对于超线程( hyperthread) 处理器,每个超线程就是一个逻辑处理器。

c) 设置 VMCS。通过执行 VMXON 启用 VMX 后,处理器将处于 VM root 模式。此时, Libra 将在非分页内存上创建 VMCS 并设置相应的版本标志符以便使得相应特权操作激发 VM exit 事件。进程检测组件需要监视 CR3 寄存器的变化,所以 Libra 设置 VMCS 是为了使 CR3 的修改产生 VM exit 事件。为此, Libra 需要将 VMCS 中的 CR3-target count 比特位设为零。之后, Libra 通过调用 VMWRITE 来初始化 VMCS 的客户虚拟

系统状态( guest-state) , 主要包括当从 VMX root 模式切换到 VMX non-root 模式( VM entry) 时需要的处理器执行上下文等。最后, Libra 将设置栈指针寄存器( ESP) 与指令寄存器( EIP) 以控制客户虚拟系统将要执行的指令。

d) 从 VMX root 模式切换到 VMX non-root 模式( VM entry) 。通过以上复杂的准备工作, Libra 已经完成了 LibraVMM 的创建; 接着通过调用 VMLUNCH 将处于在本地运行状态宿主操作系统动态迁移到 LibraVMM 之上; 最后被迁移的操作系统将在虚拟化状态下继续运行。

e) VM exit 事件处理。通过在 c) 中对 VMCS 作了相应配置, 所有在 VMX non-root 模式下对 CR3 的修改操作将会触发 VM exit 事件, 而 LibraVMM 的进程检测组件将会捕获该事件, 并提取进出系统中相应的进程信息, 利用 CR3 寄存器的变化侦测进程信息的算法将在第 3 章中详细描述。

f) 在所有逻辑处理器上禁用 VMX 模式。在结束隐藏进程的检测后, Libra 将结束 LibraVMM。LibraVMM 首先释放占用的所有内存资源, 接着在所有处理器调用 VMMOFF 以禁用 VMX 模式。至此, Libra 完成了将操作系统从虚拟化状态切换回本地化运行状态的状态转换。

## 3 实验

测试 Libra 的计算机系统是配置了支持 Intel VT 技术的处理器的 Thinkpad T61 笔记本电脑, 处理器为 Intel Core2 Duo 7300 2. 00 GHz, 内存为 2 GB, 硬盘为 120 GB。

为了验证 Libra 检测隐藏进程的有效性, 测试使用了五种基于不同进程隐藏技术的常用恶意代码, 包括 Aphex、Hacker Defender、FU、FUTo 和 phide_ex。Aphex[4] 通过修改每个进程的导入函数地址表( import address table) 来钩挂函数 ZwQuerySystemInformation, 从而将欲隐藏的进程从 ZwQuerySystemInformation 返回的进程列表中删除。Hacker Defender[5] 则是将函数 NtQuerySystemInformation 开始几个字节的代码修改成跳转到自己代码的 “jmp ”指令。FU[6] 则是采用了所谓直接内核对象操作技术( direct kernel object manipulation, DKOM) 将欲隐藏的进程从 Windows 内核的活动进程列表( active process list) 中删除。FUTo[7] 是 FU 的增强版本, 除了修改活动进程列表外, 它还将欲隐藏的进程从内核结构 PspCidTable 中删除, 而 PspCidTable 正是保存进程和线程标志符的列表。黑客 PE386 则提出了一种新的基于 KDOM 的进程隐藏技术, phide_ex[8] 则是其提供的实现范例。

测试同时比较了已有 Anti Rootkit Group[9] 推荐的检测工具, 包括 F-Secure BlackLight 2. 2、DarkSpy 1. 0. 5、IceSword 1. 20、RkUnhooker 3. 0、UnHackMe 2. 5. 5、GMER 1. 0、KProcCheck 0. 2、bitdefender_antirootkit-BETA1、Process Hunter 1. 1 和 TaskInfo 6. 2。

测试结果显示所有被测试的恶意代码检测工具均可检测到 Aphex, Hacker Defender 和 FU, BlackLight、Icesword、UnHackMe 和 TaskInfo 不能检测到 FUTo, 而所有的这些检测工具均不能检测到 phide_ex, 只有 Libra 能检测到所有这五种方式隐藏的进程。

对于 Libra 而言, 为了获取 TPL 而需要监视 CR3 寄存器的修改, 每次进程切换均会导致 CR3 寄存器值的改变, 进而引发 VM exit 事件。为了测试由此导致性能下降, 本文在 Windows 平台下实现了著名的 UNIX 的内核测试工具 forkwait, forwait 通过重复创建大量进程来测试进程创建/退出/切换对系统性能的影响。下面的代码清晰地说明了 forwait 的工作原理:

```
if ( argc > 1) return 0;
PROCESS_INFORMATION ProcessInfo;
STARTUPINFO StartupInfo;
memset ( &StartupInfo, 0, sizeof ( StartupInfo) ) ;
for ( int i = 0; i < 10000; i + + ) {
   CreateProcess ( NULL, " ForkWait. exe child", NULL, NULL,
FALSE, 0, NULL, NULL, &StartupInfo, &ProcessInfo) ;
   WaitForSingleObject ( ProcessInfo. hProcess, INFINITE) ;
   CloseHandle ( ProcessInfo. hProcess) ;
   CloseHandle ( ProcessInfo. hThread) ;
}
```

如上述代码所示, 本文的测试用例中进程创建数为 10 000, 在本地运行模式下 forwait 耗时 35. 926 s 测试同时比较了 forwait 在 VMware Workstation 6. 0、VirtualPC 2007 和 Parallels Workstation 2. 2 中的运行结果, 如表 1 所示。作为轻量级的虚拟机监视器, LibraVMM 对系统性能的影响远远小于系统级虚拟机。

表 1 Forwait 性能测试比较

| 模式 | 运行时间/s | 性能下降比例/% |
|---|---|---|
| VMware Workstation | 46. 869 | 30. 46 |
| VirtualPC | 51. 154 | 42. 39 |
| Parallels Workstation | 48. 620 | 35. 33 |
| LibraVMM | 41. 324 | 15. 03 |

## 4 结束语

针对 PC 平台, 为了在已有的宿主操作系统下有效检测进程隐藏恶意代码, 本文基于硬件虚拟化技术构造了支持操作系统动态迁移的轻量级虚拟机监视器 LibraVMM, 并在 LibraVMM 中实现了不依赖于操作系统的隐式的隐藏进程检测技术, 为自隐藏恶意代码的检测技术提供了有效支持。本文的测试结果证明了 Libra 能够准确获取真实的进程列表( TPL) 并能检测目前所有流行的进程隐藏恶意代码, 充分说明了本文提出的解决方案的有效性。

参考文献:


[1] ANG Y M, BECK D, VO B, *et al*. Detecting stealth software with strider ghost buster[C]//Proc of International Conference on Dependable Systems and Networks (DSN ’05). 2005: 368-377.

[2] UHLIG R, NEIGER G, RODGERS D, *et al*. Intel virtualization technology[J]. IEEE Computer, 2005, 38(5): 48-56.

[3] WEN Yan, ZHAO Jin-jing, WANG Huai-min. Implicit detection of hidden processes with a local-booted virtual machine[C]//Proc of the 2nd International Conference on Information Security and Assurance (ISA ’08). 2008.

[4] Aphex: AFX Windows rootkit[EB/OL]. [2007-11-21]. http://www.iamaphex.cjb.net.

[5] Hacker defender[EB/OL]. [2007-11-25]. http://hxdef.org/.

[6] Fuzen_op, FU rootkit[EB/OL]. [2007-11-23]. http://www.rootkit.com/project.php?id=12.

[7] SILBERMAN P, C H A O S. FUTo: bypassing blacklight and icesword[EB/OL]. [2007-11-26]. https://www.rootkit.com/newsread.php?newsid=433.

[8] PE386. phide_ex-untimate process hiding example[EB/OL]. [2007-11-24]. http://forum.sysinternals.com/printer_friendly_posts.asp?TID=8527.

[9] Anti Rootkit Group[EB/OL]. [2007-11-28]. http://www.antirootkit.com/blog/.