Xen 硬件虚拟化下一种客户机进程追踪技术①

周继冬[1,2], 崔超远[2], 王儒敬[2], 陈祝红[1,2]

[1](中国科学技术大学 自动化系, 合肥 230027)
[2](中国科学院合肥物质科学研究院 智能机械研究所, 合肥 230031)

摘 要: Xen 硬件虚拟化(HVM)是运行于 Xen Hypervisor 上, 无需修改客户操作系统内核便可实现虚拟化的技术. 但 Xen HVM 在提升用户体验的同时, 也造成了 VMM 对客户机的干涉和理解程度的降低, 丧失了进程粒度级别的管理能力. 鉴于此, 提出了一种针对 Xen HVM 客户机的轻量级进程追踪技术原型 Xen HPT. 借助 VMM 掌握的客户机 ESP 寄存器信息, 从客户机外部隐式捕获其实时进程信息. 实验证明, 该技术是一种追踪客户机进程的有效方法.

关键词: Xen 硬件虚拟化; VMM; 进程; Xen HPT

Process Trace with Xen Hardware-Assisted Virtualization

ZHOU Ji-Dong[1,2], CUI Chao-Yuan[2], WANG Ru-Jing[2], CHEN Zhu-Hong[1,2]

[1](Department of Automation, University of Science and Technology of China, Hefei 230027, China)
[2](Institute of Intelligent Machines, Chinese Academy of Sciences, Hefei 230031, China)

Abstract: Xen Hardware-Assisted Virtualization(HVM) is a service that one can run a virtual machine which is a unmodified OS on Xen Hypervisor. HVM causes a loss of management of the process granularity for the reason of VMM's lack of interference and understanding of the guest OS. This paper presents a lightweight process trace method for Xen HVM named Xen HPT. With the assistance of guest OS register information (mainly the ESP) collected by VMM, Xen HPT can implicitly capture the real-time process information of target guest OS outside on Domain0 with little effect on guest OS.Xen HPT has been tested by Linux guest OS and result shows that the method is effective.

Key words: Xen HVM; VMM; process; Xen HPT

随着云计算的蓬勃发展, Iaas 云计算模式的虚拟化技术受到广泛关注[1]. 其中 Xen 是一款基于 X86 架构、性能相对稳定且占用资源相对较小的开源虚拟化技术[2]. 目前 Xen 主要支持两类虚拟化方法: 准虚拟化(PV)和硬件虚拟化(HVM). 它们分别采用软件和硬件的方法解决 Xen 虚拟化平台 Hypervisor 上运行的客户机操作系统(Guest OS)部分特权指令的执行问题. PV 通过修改 Guest OS 内核代码来替换掉上述不能被直接执行的指令, 获取运行状态信息. HVM 则将指令切换工作交由硬件 CPU 完成, 因而无需修改操作系统, 减轻了 Hypervisor 的负载. 但同时 Hypervisor 对 Guest OS 行为的理解和干涉程度也相应的降低.

现今 Hypervisor 能够有效地管理 Guest OS 的启、停及迁移等. 但其管理粒度是虚拟机(VM)级别的. 多数情况下, 如 Guest OS 负载查询、安全监控等需要对 Guest OS 中进程加以管理, 即管理粒度需要深入到进程级别. 关于 Guest OS 进程检测, Antfarm 和 Lycosid 开创了这方面的工作[3,4]. Antfarm 是 Lycosid 的检测模块, 用来在 Hypervisor 上统计 Guest OS 的实际进程数量. Lycosid 利用交叉视图校验技术和最小二乘回归分析法实现了进程识别模块. Lycosid 属于重量级进程检测, 适合需要处理大量数据, 进程活跃系统. 由于它是以概率的方式识别进程, 存在一定程度的漏检或误报的现象. Hay 等研发的 VIX 是一款功能强大的客户

① 基金项目:国家自然科学基金(31171456)
收稿时间:2012-12-06;收到修改稿时间:2013-01-04

机监视工具. 它提供了访问虚拟机中的进程列表、网络端口、已打开文件、已加载内核模块等各种信息的接口[5]. 但调用 VIX 需要暂停 Guest OS 的运行, 客户机友好性等方面存在不足. 此外, 它还要参考特定类型 Guest OS 对应的内核符号表 System.map 文件才能解析进程描述符的地址, 程序的可移植性也有所限制.

鉴于上述研究, 本文提出一种针对 HVM 客户机的进程追踪技术原型 Xen HPT(Xen HVM Process Trace). 在不影响客户机运行且不被客户机察觉的情况下, 借助 Hypervisor 读取的 Guest OS 寄存器信息(ESP 寄存器)及 Xen 提供的影子页表操作, 从 Guest OS 外部, 如特权域 Domain0 中隐式捕获客户机的实时进程以及进程之间的关联关系. 较之 Lycosid, Xen HPT 没有用到视图校验和数据处理, 因此更加轻量化. 与 VIX 相比, Xen HPT 采用了模块化的实现方法, 无需暂停客户机, 可移植性和复用性较强.

1 基本概念

1.1 Xen 虚拟化

Xen Hypervisor 位于操作系统与硬件之间, 为其上运行的操作系统内核提供虚拟化的硬件环境. 在 Xen 系统中存在一个轻量级软件层称为虚拟机管理器(VMM), 而虚拟机则被称为虚拟域(Domain). Xen 采用混合模式, 特权域 Domain0 辅助 Xen 管理其他 Domain, 提供相应的虚拟资源服务[6].

Xen 早期推崇的准虚拟化技术(Para Virtualization, 简称 PV)在满特征操作系统支持和性能方面表现优异. 但由于需修改 Guest OS 内核, 故大多商业产品不支持 PV, 且修改后的客户机的可移植性及友好性等也会有所下降. 为了解决准虚拟化效率低下以及虚拟化软件实现复杂等问题, Intel 和 AMD 先后在其产品中为虚拟化提供了额外的硬件支持. Xen 硬件虚拟化(Hardware-Assisted Virtualization, 简称 HVM)是在上述硬件技术的协助下运行操作系统于 Hypervisor 上的虚拟化技术. 由 CPU 自动维护不能虚拟化的指令, 避免了对 Guest OS 内核代码的修改或是执行指令的翻译工作[7].

1.2 影子页表

作为一个可同时运行多个客户机的虚拟化平台, Hypervisor 需将宿主机的机器内存动态地分配给各个客户机. 维护机器内存和客户机所见“物理内存”的映射关系, 使得“物理内存”在客户机看来和真实硬件上运行时一样.

在硬件虚拟化下, VMM 启动何种内存虚拟化方式取决于 CPU 是否支持 EPT 技术. 本文仅讨论 CPU 不支持 EPT 的情况, 即影子页表方式[8].

客户机有自己的一套页表将客户机虚拟地址(GVA)映射到客户机物理地址(GPA). 但 GPA 并不是宿主机机器(物理)地址(HMA). 要访问真实的机器内存还需一个 GPA→HMA 的转换机制. 这样访问内存数据就需经过 GVA→GPA→HMA 两次映射. Xen 使用的影子页表技术可消除额外的地址映射, 直接实现 GVA→HMA 的转换, 从而提高访问效率.

2 Xen HPT

本文提出的针对 Xen HVM 客户机的轻量级进程追踪技术 Xen HPT(Xen HVM Process Trace)的核心是定位客户机进程的 task_struct 结构, 从中获取我们感兴趣的信息.

理论上讲, 要读取 task_struct 结构中的成员, 需要将 linux 源码中声明该结构体的头文件 linux-source-3.2.0/include/linux/sched.h 及其依赖的其它头文件包含在 XenHPT 源码中. 但这样简单的引入会影响原有 Xen 源码的编译.

本文设计了一种模块化的实现方法, 将 XenHPT 分为前端和后端两个模块: ①前端模块的作用是通过读取的 ESP 定位器来定位 task_struct 结构的 HMA, 并且向后端请求进程信息; ②后端模块为服务容器, 由一系列进程信息反馈接口函数组成, 如 return_pid()/return_comm()分别返回进程号和名称等 task_struct 结构体取成员操作. 两个模块分别在 Xen 源码和 linux 源码中编译成目标文件, 之后放在一起联合编译, 生成最终的可执行文件: 虚拟机进程跟踪 process_trace. 该文件可以移植到 Domain0 的任意位置独立执行.

2.1 前端模块

1) task-struct 结构体

task-struct 是 linux 内核的进程描述符. 该结构包含了内核管理进程所需的全部信息. 这些信息能够完整的描述一个正在执行的程序, 如打开的文件、进程的地址空间、挂起的信号、进程的状态、运行时间等. 其 C 伪代码如下所示.

```
struct task_struct{
```

```
    volatile long state;
    void *stack;
    ……
    pid_t pid;
    ……
    struct task_struct *parent;
    struct list_head children;
    ……
    char comm[TASK_COMM_LEN];
    ……
};
```

从 Linux2.6 开始, task_struct 结构的存储方式由之前的将各个进程的 task_struct 结构存放在它们内核栈的尾端, 改为通过 slab 分配器动态地为每个进程分配 task_struct 结构. 而在进程内核栈栈底或栈顶创建一个新的结构体 thread_info[9]. 其 C 伪代码如下所示.

```
struct thread_info{
    struct task_struct *task;
    struct exec_domain *exec_domain;
    ……
};
```

其中第一个成员 *task 是指向该进程实际 task_struct 结构体的指针. 而上述 task_struct 结构体代码中第二个成员*stack 是指向相应 thread_info 结构体的指针. 进程之间的父子、兄弟关系也存放在 task_struct 中.

2) 客户机机器地址 HMA 的获取

本文提出的 XenHPT 技术, 利用 x86 平台下 ESP 寄存器来指示当前栈顶位置. 当启用 THREAD_SIZE 大小的栈时, 执行如下代码:

```
struct task_struct* get_current_task{
    struct task_struct *p;
    p = ~ (THREAD_SIZE-1)&esp;
    p =map_page(ctx, vcpu, p->task);
    return p;
};
```

一条“按位与&”运算 p = ~(THREAD_SIZE - 1) & esp 定位到当前进程的 thread_info, p->task 操作指向当前进程的 task_struct. 再通过调用 map_page()函数将 GVA 转换为 HMA. 函数返回值即为指向 task_struct 结构的 HMA 指针.

2.2 后端模块

定位了 task_struct 结构的 HMA 之后, 便可从中读取我们感兴趣的进程信息, 如进程号 pid、进程名称 comm、进程状态 state 等. 对于同为 x86_64 架构的 Hypervisor 的宿主机和 Guest OS, 读取操作较为直接. 例如通过如下所示取成员操作 p->pid 便可得到进程的 pid:

```
int returnpid(size_t *task_addr_machine)
{
    struct task_struct *p = (struct
    task_struct*)task_addr_machine;
    return  p->pid;
}
```

3 实验与分析

3.1 实验环境

对于本文提出的客户机进程追踪技术 Xen HPT 的效果我们进行了实验测试. 具体实验环境: 操作系统 OS、采用的 Linux 和 Xen 版本等如表 1 所示.

表 1 实验环境

名称	配置
OS	Ubuntu12.04-x86_64 Desktop
Linux 版本	3.2.0-33-generic-x86_64
Xen 版本	Xen-4.1.2
CPU	Intel(R)-Core(TM)-i7-2.93GHz ,8 核
其他	RAM: 4G； Disk: 1TB

3.2 实验结果与分析

使用 Xen HPT 对 HVM 客户机进程进行了追踪. 实验与分析如下. 以下图中方框标注“1”代表 Hypervisor 所在的宿主机 Domain0, “2”代表 Desktop 客户机. 图 1 展示了 Xen HPT 捕捉到的 thread_info 和 task_struct 交互关系这个观察点上的实现效果. 实线表示通过 thread_info 定位 task_struct, 虚线表示通过 task_struct 重定位 thread_info.

由图 1 可知, 通过 thread_info 定位得到的 task_struct(GVA): 0xffffffff81c0d020 正是客户机内核符号表 /boot/System.map-3.2.0-33-generic 中符号为 init_task 的地址, 即 0 号 swapper 进程的进程描述符地址[9]. 反之, 通过 task_struct 结构取成员 stack 操作, 重定位得到的 thread_info(GVA): 0xffffffff81c00000, 与 esp&运算定

位到的 thread_info 地址完全吻合. 以上说明 task_struct 的定位机制是正确的.

图 2 为在客户机内运行“ps axf”命令和在 VMM 中执行 process_trace 获得的进程列表. 从中可以观察到 Xen HPT 捕获的进程 pid、名称及进程之间关系的匹配效果.

如图 2 中虚线方框所示在虚拟机外部, 即 Domain0 “1”中, pid=1 号进程对应的进程名为 init, 它是 pid=0 号 swapper 进程的子进程. 即运用 XenHPT 捕获到的进程号和名称一一匹配. 又例如图中实线方框所示 Domain0 中 291 号 udevd 进程有两个子进程与从虚拟机内部, 即客户机“2”中获取的进程信息完全匹配. 以上实验证明本文提出的 Xen HPT 技术是行之有效的.

图 1　Xen HPT 获取的 thread_info 与 task_struct 交互关系

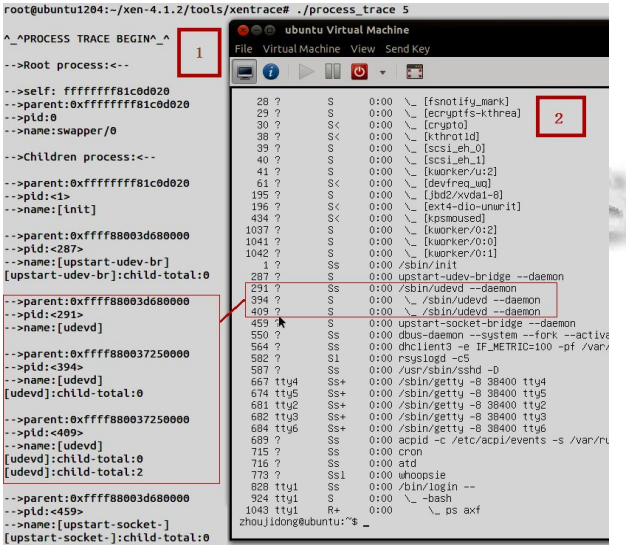

图 2　客户机内外获取的进程列表

4　结论

本文提出了一种在虚拟机外部隐式获取 Xen HVM 客户机实时进程信息的方法 Xen HPT. 通过在典型的 linux ubuntu 操作系统上进行测试, 证明 Xen HPT 是一款正确有效的进程追踪技术原型, 且采用模块化设计机制具有较强的可移植性和复用性.

在后续的工作中, 着重考虑 32 位和 64 位混合模式以及 Windows 操作系统. 此外, linux 内核版本多样化及客户机不可预测升级动作是 Xen HPT 面临的最大挑战, 因为 Xen HPT 对 linux 内核版本有针对性, 所以如何实现 Xen HPT 的复合性和鉴别客户机内核版本也是我们要努力的方向.

参考文献

1 虚拟化与云计算小组.虚拟化与云计算.北京:电子工业出版社,2009.

2 Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. Proc. of the 19th ACM Symp. on Operating Systems Principles. Bolton: ACM Press, 2003:164–177.

3 Jones S, Arpaci-Dusseau A, Arpaci-Dusseau R. AntFarm: Tracking Processes in a Virtual Machine Environment. Proc. Annual Usenix Tech. Conf. Berkeley: Usenix Assoc, 2006: 1–14.

4 Jones S, Arpaci-Dusseau A, Arpaci-Dusseau R. VMM-based Hidden Process Detection and Identification Using Lycosid. Proc. of the ACM International Conf. Virtual Execution Environments (VEE 08). New York: ACM Press, 2008: 91–100.

5 Hay B, Nance K. Forensic Examination of Volatile System Data using Virtual Introspection. ACM SIGOPS Operating Systems Review, 2008,42(3):74–82.

6 王磊,邹德清,金海.Xen 虚拟化技术.武汉:华中科技大学出版社,2009.

7 Rich U, Gil N, Dion R. Intel Virtualization technology. IEEE Computer, 2005,38(5):48–56.

8 The Xen Team. Xen Interface manual.Xen v3.0 for X86. http://oss.org.cn/ossdocs/server_storage/xen/interface/interface.html.2005.

9 Robert L. Linux Kernel Development. 3rd ed., New York: Addison-Wesley Professional, 2010:24–26.