doi:10.11959/j.issn.1000-436x.2015103

虚拟机自省中一种消除语义鸿沟的方法

崔超远[1]，乌云[2]，李平[1,3]，张晓明[1]
（1. 中国科学院 合肥智能机械研究所，安徽 合肥 230031；
2. 中国科学院 安徽循环经济技术工程院，安徽 合肥 230088；3. 中国科学技术大学 自动化系，安徽 合肥 230027）

摘 要：虚拟机自省技术已经广泛应用于入侵检测和恶意软件分析等领域。但是由于语义鸿沟的存在，获取虚拟机内部信息时会导致其通用性和执行效率降低。通过分析现有语义鸿沟修复技术的不足，提出了一种称为 ModSG 的语义鸿沟消除方法。ModSG 是一个模块化系统，将语义修复分为 2 部分：与用户直接交互的在线语义视图构建和与操作系统知识交互的离线高级语义解析。二者以独立的模块实现且后者为前者提供语义重构时必要的内核语义信息。针对不同虚拟机状态和不同内核版本操作系统的实验表明，ModSG 在消除语义鸿沟上是准确和高效的。模块化设计和部署也使 ModSG 容易扩展到其他操作系统和虚拟化平台上。
关键词：语义鸿沟；虚拟机自省；模块化系统；可移植性
中图分类号 TP391 **文献标识码**：A

Narrowing the semantic gap in virtual machine introspection

CUI Chao-yuan[1], WU Yun[2], LI Ping[1,3], ZHANG Xiao-ming[1]
（1. Institute of Intelligent Machines, CAS, Hefei 230031, China;
2. Anhui Technology and Engineering Institute for Recycling Economy, CAS, Hefei 230088, China;
3. Department of Automation, University of Science and Technology of China, Hefei 230027，China）

Abstract: Virtual machine introspection(VMI) has been widely used in areas such as intrusion detection and malware analysis. However, due to the existence of semantic gap, the generality and the efficiency of VMI were partly influenced while getting internal information of a virtual machine. By analyzing the deficiencies of existing technology of semantic gap restoration, a method called ModSG was proposed to bridge the semantic gap. ModSG was a modularity system, it divided semantic restoration into two parts. One was online phase that interact directly with user to construct semantic views, the other was offline phase that only interact with operating system to parse high-level semantic knowledge. Both were implemented via independent module, and the latter provided the former with necessary kernel information during semantic view construction. Experiments on different virtual machine states and different kernel versions show that the ModSG is accurate and efficient in narrowing semantic gap. The modular design and deployment also make ModSG easily to be extended to other operating systems and virtualization platforms.
Key words: semantic gap; virtual machine introspection; modularity system; portability

1 引言

虚拟机自省（virtual machine introspection）[1]机制最早于2003年由Garfinkel和Rosenblum提出，它通过获取虚拟机所依托物理硬件级别的状态（如物理内存页、寄存器）和事件（如中断、内存读取），推导甚至控制虚拟机内部行为。这项技术已经在恶意软件分析[2]、入侵检测系统[3]和内存取证[4,5]等方面得到广泛应用。虚拟机自省要实现对虚拟机内部的细粒度监视，必须要从外部确定虚拟机中哪块区

收稿日期：2014-02-13；**修回日期**：2014-09-12
基金项目：中国科学院合肥物质科学研究院院长基金资助项目(YZJJ201329);国家自然科学基金资助项目(31171456, 61203373)
Foundation Items: Foundation of President of Hefei Institutes of Physical Science, Chinese Academy of Sciences(YZJJ201329); The National Natural Science Foundation of China(31171456, 61203373)

域是有意义的，这块有意义区域被安放在宿主机物理内存的哪块区域上，该区域存储的是“0”、“1”任意数量、任意组合的二进制字符串，如何将其还原成在虚拟机中看来是有语义含义的字符序列或数据结构。这一问题被称为虚拟化的语义鸿沟（semantic gap）[6]。

消除语义鸿沟在虚拟机自省技术中仍然是一个难题。已有的解决方案中，有通过解析虚拟机内核数据结构来重构语义视图的方法[2]，也有直接利用现成库函数获取虚拟机进程及加载模块等信息的方法[4]，还有通过对虚拟机内部进程生命周期跟踪来统计进程数量的方法[7]。不论哪一种都需要从虚拟机外部经由虚拟化平台这个特权层，获取寄存器、内存、磁盘等设备的低等级字节信息，然后将其还原为虚拟机内部高级语义信息。但是这种还原操作在云计算多租户多任务环境下，不得不频繁解析硬件状态和不同内核版本的客户操作系统，导致通用性差，实时性也不高。

本文针对现有技术的不足，采用模块化设计思想，提出了基于语义视图构建和内核语义解析相分离的语义鸿沟修复架构，二者功能上相互依存，实现上各自独立，能够有效提高系统可移植性和执行效率。

2 相关工作分析

为了消除语义鸿沟的影响，提高虚拟机自省的准确性和高效性，人们从虚拟化原理、操作系统原理以及自省程序代码等各个角度进行了尝试和探讨。

Garfinkel 等[1]最早提出虚拟机自省概念的同时也给出了一种消除语义鸿沟的方法，即采用 Linux 内核崩溃转储（kernel crash dump）[8]分析工具对内存、寄存器等设备进行检查，并结合称为操作系统接口库（OS interface library）的组件，解释虚拟机状态，这样获得的就不再是物理内存页内容之类的硬件状态，而是如进程代码段之类的操作系统语义。但是这一方法依据特定操作系统实现，而且需要编译带有调试符号的内核。

Jiang 等[2]利用 VMwatcher 从虚拟机外部进行恶意软件检测时，通过修复语义鸿沟实现其功能的。VMwatcher 把客户虚拟机内核数据结构定义（如文件、目录、进程）和函数语义（如文件系统驱动程序）用作模板来解释从 VMM 获得的底层 VM 状态，进行外部语义视图重构，获得包括虚拟机寄存器、内存、磁盘等状态的全部信息。但是客户机内核数据结构定义因内核版本的不同而有差异，该方法实施对内核敏感，不利于系统移植。

Hey 和 Nance[4]研发的 VIX（virtual introspection for Xen）提供了访问虚拟机内部进程列表、网络端口、已打开文件和已加载内核模块等各种信息的命令或接口，用于监视客户虚拟机。VIX 将读取到的字节信息还原成高级语义信息时，需要参照虚拟机内部文件实现其功能。例如，进程描述符的定位是通过参阅虚拟机内核符号表 system.map 来实现，这意味着 VIX 对虚拟机知识的依赖，智能化水平还有待提高。

康华[9]通过截获 CR3 寄存器获取当前进程的页目录地址，截获 ESP 寄存器获取当前进程的进程描述符地址，并将二者作为键值对查找对应的进程描述符读取进程信息，实现了 KVM 虚拟机的进程检测。但是，该方法中的键值对在进程活动频繁情况下会频繁更新，严重影响虚拟化平台的性能。同时，在线解析进程描述符依赖虚拟机内核版本，不利于系统移植。

同样是解析物理硬件状态消除语义鸿沟，Jones[7]提出的 AntFarm 框架是通过跟踪虚拟机内部进程生命周期（包括进程创建、销毁、切换等），统计虚拟机实际进程数量来实现的。进程地址空间的页全局目录 PGD 基地址存放 CR3 寄存器，进程创建、切换和销毁等状态变化会导致 CR3 寄存器值的变化。AntFarm 根据出现在 CR3 寄存器候选列表中 PGD 基地址数量，判断出了虚拟机中进程的实际数量。然而 AntFarm 只统计进程数量，无法得到描述进程的详细信息。另外，一台物理机上同时运行多个虚拟机时，必须同时有多个统计程序并行执行，这会加重平台的整体负载。

以上方法如 VMwatcher、VIX 和 AntFarm 都遵循 Pfoh[10]归纳的推导模式解决语义鸿沟。Dolan-Gavitt 等[11]提出的 Virtuoso 和 Fu 等[12]提出的 VMST 则另辟蹊径，从程序入手进行语义修复。如 VMST 在线内核数据重定向技术，绕开硬件状态获取和解析，直接生成自省程序代码。开发者利用 VMST 很容易将一个客户虚拟机内部的检查程序变成一个自省程序。但是这种方法的局限性也是明显的，首先，VMST 所依赖的系统调用、中断处理、上下文切换都离不开虚拟机操作系统知识，它对内核版本敏感；其次，VMST 不支持异步系统调用，不能读取被换出到硬盘的内存数据；另外 VMST 也不支持多核客户虚拟机。

3　系统设计

考虑到系统准确性、可移植性和执行效率，本文提出了一种模块化的语义鸿沟修复架构 ModSG，将修复功能分解为前后端 2 个模块，前端和用户交互，进行语义视图构建；后端离线解析不同版本内核结构，为前端提供语义视图构建时所需的操作系统信息。ModSG 系统架构及各模块间关系如图 1 所示。

3.1　语义修复后端模块

图 1 中右边部分为语义修复后端模块，是离线预处理操作，需要在语义修复前端启动之前执行并完成。后端主要由高级语义解析模块构成，用于解析客户虚拟机操作系统内核数据结构（如进程控制块），获取包括进程号、进程名之类的进程描述、系统调用和加载模块等信息，生成语义信息静态库，提供语义信息访问接口。生成的静态库用于接收来自前端传递的语义信息调用请求，并将请求处理结果向前端回复。

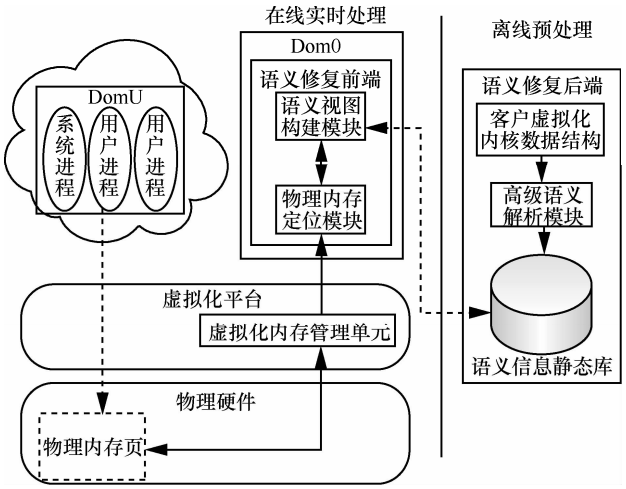


图 1　ModSG 系统架构

3.2　语义修复前端模块

前端模块包括物理内存定位和语义视图构建 2 部分。前者用于在语义修复任务启动之后，定位当前进程及模块在客户虚拟机所依托宿主实体机的物理内存地址；后者参照语义修复后端的语义信息静态库，解析物理内存，构建高级语义视图，还原客户虚拟机内部进程的语义内容。

本文提出的 ModSG 和国内外其他研究相比有如下优势。1）可移植性。在线语义视图构建和离线高级语义解析以不同模块实现。用户只与前端交互，不需依靠任何操作系统知识，因此前端对内核版本是不敏感的。后端尽管要面向各种系统内核，但以离线预处理实现，不会直接与用户交互，因而不影响前端执行，可以广泛地用在 Xen[13]、KVM[14]、QEMU[15]等虚拟化平台。2）准确性。ModSG 采用推导模式，把思路放在监视虚拟机硬件状态上。正如 Pfoh 等论述的，这是修复语义鸿沟最为可靠和防恶意篡改的方式。3）轻量性。前端进行语义视图构建时避免频繁解析虚拟机内核数据结构，只通过简单的函数调用即可，处理效率高，系统负荷较小。这一点也会在后面实验中得到证实。4）易实现。前后端在功能上相互依存，但实现上各自独立，这样可以限定排错范围，避免程序代码融汇带来的定义冲突。

4　系统实现

系统在开源虚拟化平台 Xen 上实现。Xen 上的虚拟域包括 Dom0 特权域和 DomU 非特权域，Dom0 可直接访问底层的物理资源。ModSG 的语义修复前端位于 Dom0 域，客户虚拟机位于 DomU 域。

4.1　虚拟化平台内存管理

Xen 通过虚拟化平台 Hypervisor 向客户提供并管理虚拟硬件资源[16]。为了让虚拟机使用一段隔离的、从零开始且连续的内存空间，Hypervisor 在操作系统原来的机器地址和虚拟地址之间引入了一层新的地址空间，即虚拟机物理地址空间。这样内存地址访问就存在 2 次地址转换：从客户机虚拟地址(GVA)到客户机物理地址(GPA)的转换和客户机物理地址到宿主机物理地址(HPA)的转换。Hypervisor 的虚拟机内存管理模块 VMMU 采用影子页表(SPT, shadow page table)技术，直接实现 GVA→HPA 的映射，提高了内存访问效率[17]。

4.2　客户虚拟机进程管理

Linux 内核把进程的列表存放在一个双向循环链表中[18]。链表中的每一项都是类型为 task_struct 的结构体，task_struct 包含了许多和进程描述有关的成员变量，如进程号（pid）、进程名（comm）、进程的状态（state）、进程运行的时间（utime、stime、gtime）、进程内存管理信息（mm）等。只要找到任意一个进程，就可以通过当前进程的 tasks 成员变量中 next 指针值来得到下一个进程的 task_struct 地址，如此依次遍历即可找到所有进程的列表。图 2 所示为进程 task_struct 结构之间的链表关系。

4.3　前后端模块实现方法

通过虚拟化平台只能获得客户机的二进制级

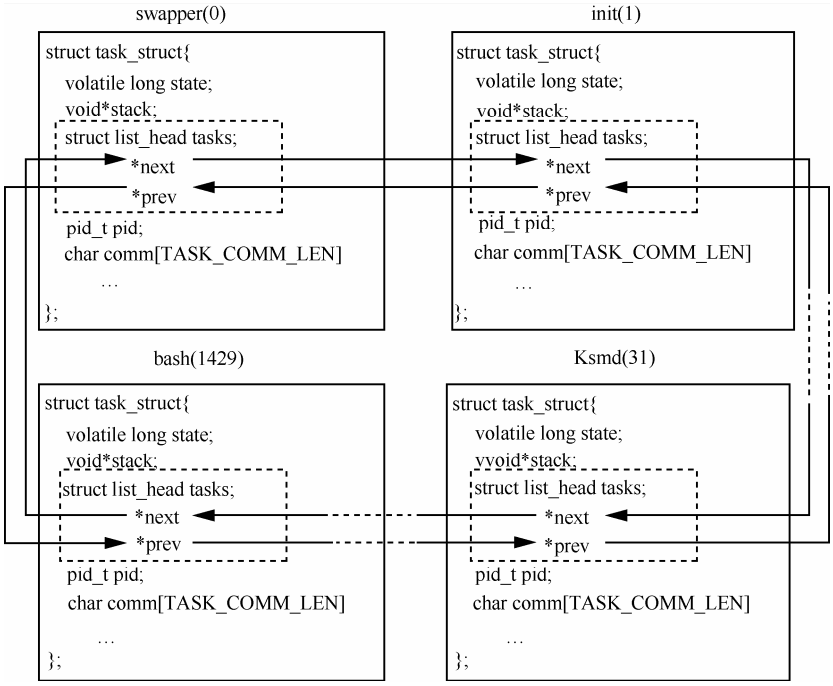


图 2　task_struct 链表关系

信息，必须将其转换成系统高级语义信息才能达到语义修复的目的。这个转化的工作由 ModSG 后端完成。图 3 所示为用类 C 伪代码给出的后端模块进行语义解析的例子。

```
1)    typedef unsigned long long guest_word_t;
2)    #define offsetof(TYPE, MEMBER)  ((size_t)  δ((TYPE *)0)->MEMBER)
3)
4)    #define  container_of(ptr,type, member)  ({        \
5)       const  typeof( ((type  *)0)->member  ) *__mptr = (ptr);     \
6)       (type  *) ( (char  *)__mptr  -  offsetof(type,member)  );})
7)
8)    struct  list_head  {
9)       struct list_head *next,  *prev;
10)   };
11)   unsigned  int  get_pid_offset(void);
12)   unsigned  int  get_comm_offset(void);
13)   unsigned  int  get_tasks_offset(void);
14)   guest_word_t  *get_task_struct_addr(guest_word_t  *taskds);
15)   guest_word_t  *get_next_tasks_addr(guest_word_t  *taskds_list);
```

图 3　后端模块语义解析

图 3 中，第 11)~13)行分别返回了 task_struct 结构体的进程号、进程名和 tasks 成员的偏移量。第 14)行根据 tasks 的地址使用内核中的宏定义（图 3 中第 4)行）来获得 task_struct 结构体的地址。第 15)行根据当前进程的 tasks（图 3 中第 8)行）的地址获得下一个进程的 tasks 成员地址。这些信息都保存在由后端生成的语义信息库中，供前端解析物理内存时使用。

前端模块直接与用户交互，执行在线语义修复，具体工作流程如图 4 所示。

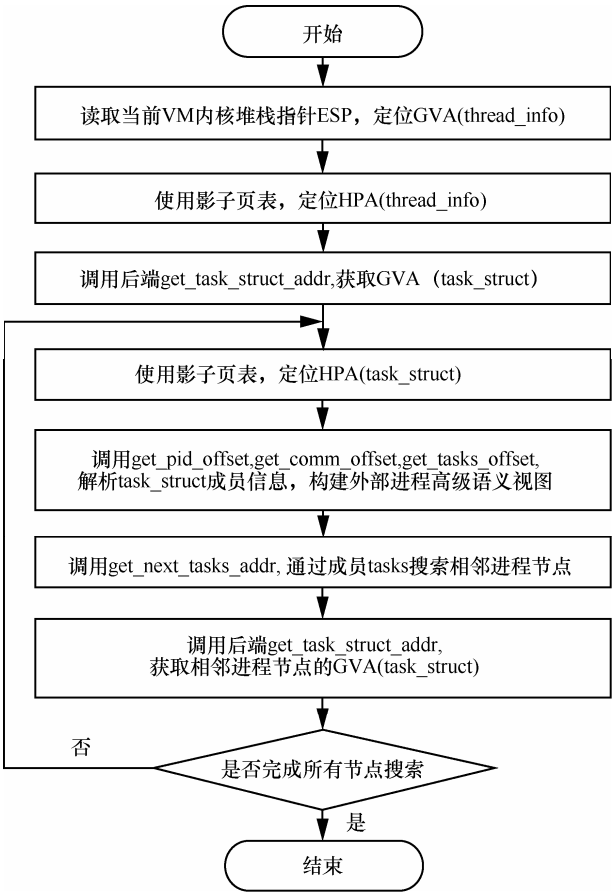


图 4　前端模块语义视图重构流程

前端在进行视图重构时，首先通过虚拟机内核堆栈指针和虚拟化平台影子页表定位进程所依

托的物理内存；其次调用语义信息库解析该物理内存，生成高级语义信息；最后遍历并解析所有进程对应的物理内存，构建虚拟机内部状态的语义视图。

5　实验结果分析

5.1　实验环境

本文通过实验验证 ModSG 的可行性，主要考察语义鸿沟修复的效果和运行效率。实验环境配置如表 1 所示，宿主机运行于 Intel Core i7 CPU（4 核，主频 2.93 GHz）、4 GB 内存的硬件之上。宿主机和虚拟机都是不同内核发行版本的 Linux 操作系统。

表 1　　**实验环境配置**

虚拟域	CPU	内存	OS	内核
Dom0	4 核	4 GB	Ubuntu12.04	3.2.0
Dom1	1 核	1 GB	Ubuntu12.04	3.2.0
Dom2	1 核	1 GB	CentOS6.4	2.6.32
Dom3	1 核	1 GB	Debian7.3.0	3.2.0
Dom4	1 核	1 GB	Fedora18	3.6.10

5.2　语义视图重构

下面以 CentOS6.4 为例，针对客户虚拟机内初始进程、新建进程和隐藏进程 3 种行为进行外部语义视图重构，通过与内部进程统计信息比较，验证 ModSG 的语义修复效果。

图 5 是某一时刻虚拟机内运行进程的内外视图比较，右边窗口是内部终端执行“ps -e”命令得到的当前进程信息。左边窗口是通过 ModSG 生成的外部语义视图。通过比较可以发现， ModSG 生成的外部进程信息与内部检查到的结果完全一致。

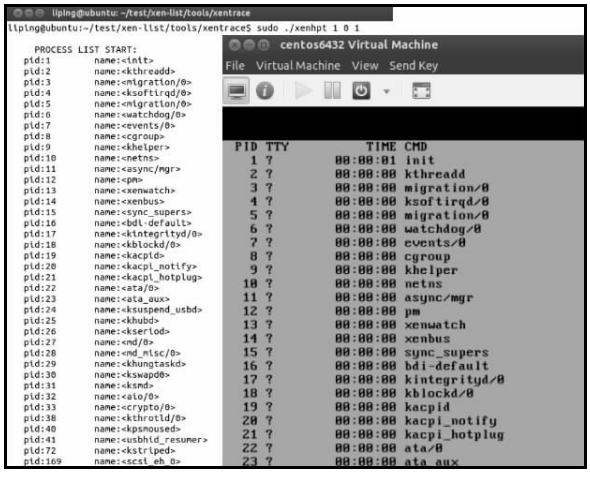


图 5　初始进程语义重构

当在 CentOS 内部启动一个新进程时，如图 6 右侧窗口所示，执行“vi &”命令后，左侧窗口中由 ModSG 重构的外部语义视图反映了该进程的基本信息。

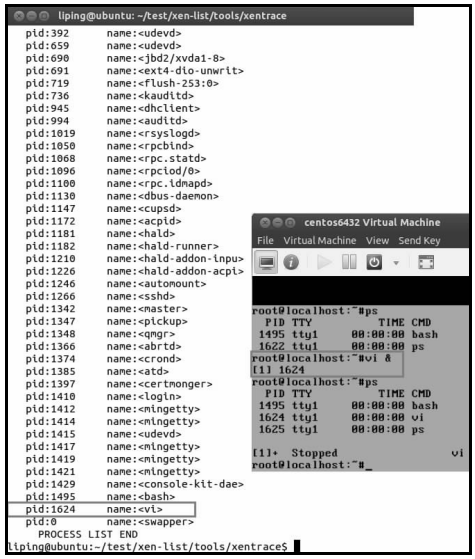


图 6　新建进程语义重构

ModSG 重构的语义视图也能够反映出隐藏进程的存在。例如，Suterusu[19]是一个功能强大的 Rootkit，可以隐藏进程、文件和网络端口等系统信息。在 CentOS 上编译安装 Suterusu，通过执行“./sock 1 1624”命令，隐藏进程号为 1 544 的进程，如图 7 右侧窗口所示。当用“ps”命令列举当前所有进程时，右侧窗口显示进程 vi 的信息已经被隐藏。左侧窗口展示了 ModSG 外部语义重构的界面，其中进程队列显示了被隐藏进程 vi。因此，ModSG 也可以有效发现隐藏进程。

图 7　隐藏进程语义重构

5.3　运行效率

本节考察 ModSG 的运行效率，分别对后端和前端模块的执行时间进行统计。

1）后端编译时间

图 8 是后端模块离线编译生成语义静态库的统计结果，横坐标是 4 种实验用操作系统环境，纵坐标是在对应环境下分别进行 10 次编译所需时间的平均值。由图可见，语义静态库所生成时间大约为 90~170 s。由于是离线运行，静态库的生成不会造成太大的系统开销，更不会影响前端语义视图重构的在线执行。

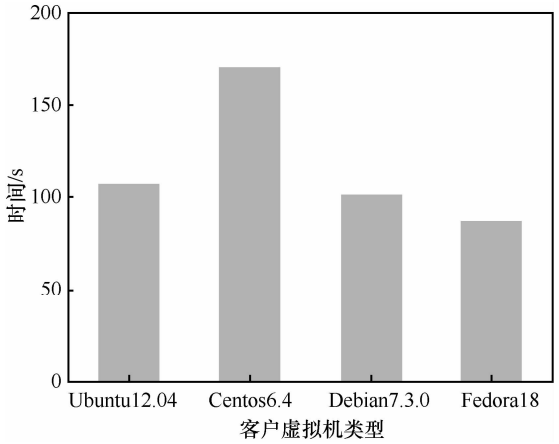


图 8　后端模块编译时间

2）前端执行时间

图 9 所示为在 4 种操作系统环境下，针对虚拟机内部不同数量进程进行外部高级语义视图重构的执行结果。其中，横坐标是进程数量，纵坐标是进行 10 次语义重构耗费时间的平均值。从图中可以看出，第一，语义重构所需时间基本上随进程数量线性变化，这主要是由于每出现一个进程都需要解析对应的硬件状态；第二，在线语义重构不影响用户体验，即使在进程数为 1 000 的情况下，在线执行所需时间仅为 450 ms 左右。这样的结果对 ModSG 而言是可控制的。

此外，当进程数量一定时(本实验中进程数为 800 个)，4 种内核版本虚拟机进行外部语义视图重构所需时间基本一致，图 10 反映了这一点。可见 ModSG 前端的执行不会因内核版本的变化出现较大的差异，这说明 ModSG 对内核版本是不敏感的，在运行效率上是稳定的。

综上所述，已有语义鸿沟修复方案中，AntFarm 侧重于统计虚拟机中实际进程数量，VMST 和 Virtuoso 侧重于生成自省工具，都不能准确获得虚拟机内部进程的高级语义信息。VIX 提供现成的库函数调用接口，但生成高级语义信息时依赖虚拟机系统内部文件。VMwatcher 和康华的方法通过解析内核数据结构重构语义视图，如果内核版本发生变化，需要调整各自的程序代码。

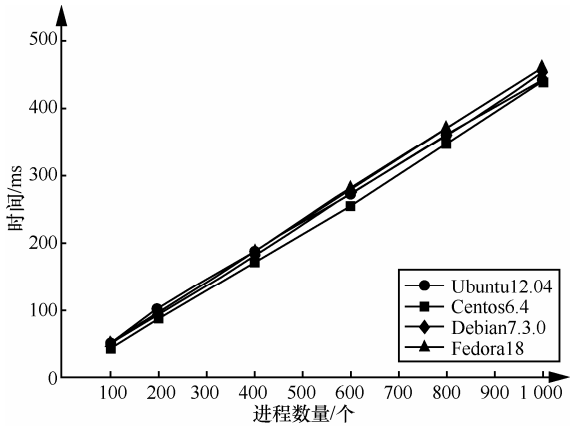


图 9　语义视图重构时间

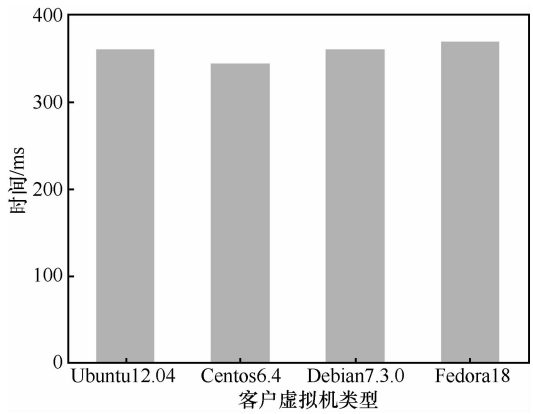


图 10　不同内核版本虚拟机的语义重构时间

本实验中使用的 4 种版本内核系统的源码不同，生成的内核映像也不同。例如，4 种内核的进程控制块 task_struct 结构体中，成员变量的顺序和偏移量都各不相同。以进程号 pid 的偏移量为例，Centos6.4 中为 0x4a8，Ubuntu12.04 中为 0x2ac，Debian7.3.0 中为 0x1e4，Fedora18 中为 0x2a4。任何修复语义鸿沟的方法都不能忽视这些差异。本文中的 ModSG 通过后端模块离线解析不同版本内核结构，为每种版本生成一个静态函数库供前端调用。用户进行语义鸿沟修复时只与前端交互即可获得所需内核信息，而不必受制于不同的内核结构。对用户而言，ModSG 的架构设计屏蔽了内核版本的差异，提高了通用性和可移植性。

6　结束语

本文提出了一种虚拟机自省中消除语义鸿沟的模块化方法，将语义修复分别以在线语义视图生成前端和离线内核语义解析后端 2 个模块实现。后端为前端提供语义重构时各种版本操作系统的语义信息，前端直接与用户交互。用户无需具备操作系统内核知识就可构建高级语义视图，增强了系统的可移植性。实验结果也证实了该方法是可行和高效的。在下一步的工作中，将丰富后端模块在获取文件列表、系统调用、网络连接等方面的功能，实现对虚拟机更细粒度的监视。

参考文献：

[1] GARFINKEL T, ROSENBLUM M. A virtual machine introspection based architecture for intrusion detection[A].Network and Distributed System Security Symposium [C].2003.

[2] JIANG X, WANG X, XU D. Stealthy malware detection through VMM-based “out-of-the-box” semantic view reconstruction[A]. Computer and Communication Security[C]. New York, USA, 2007. 128-138.

[3] JIANG X, WANG X. Out-of-the-box monitoring of VM-based high-interaction honeypots[A]. Recent Advances in Intrusion Detection[C]. Australia, 2007. 198-218.

[4] HAY B, NANCE K. Forensics examination of volatile system data using virtual introspection[J]. ACM Sigops OS Review, 2008, 42(3): 74-82.

[5] DOLAN-G B, PAYNE B, LEE W. Leveraging forensic tools for virtual machine introspection[R]. GT-CS-11-05, 2011.

[6] CHEN P M, NOBLE B. When virtual is better than real[J]. Hot Topics in Operating Systems (HOTOS '01), 2001, 8:133-138.

[7] JONES S T, ARPACI D, A C, ARPACI D, R H. Antfarm: tracking processes in a virtual machine environment[A]. Proc of the 2006 USENIX Annual Technical Conference[C]. 2006.

[8] LKCD. Linux Kernel Crash Dump[EB/OL]. http://lkcd.sourceforge.net/.

[9] 康华. 从 VMM 中识别 GUEST OS 中的用户进程[EB/OL]. http://blog.csdn.net/kanghua/article/details/1820785.
KANG H. Identify the user process in GUEST OS from VMM[EB/OL]. http://blog.csdn.net/kanghua/article/details/1820785.

[10] PFOH J, SCHNEIDER C, ECKERT C. A formal model for virtual machine introspection[A]. Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec’09) [C]. Chicago, Illinois, USA, 2009. 1-10.

[11] DOLAN G B, LEEK T, ZHIVICH M. Virtuoso: narrowing the semantic gap in virtual machine introspection[A]. Proceedings of the 33rd IEEE Symposium on Security and Privacy[C]. 2011, 32: 297-312.

[12] FU Y, LIN Z. Space traveling across VM: automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection[A]. Proceedings of the 33rd IEEE Symposium on Security and Privacy[C]. 2012.586-600.

[13] The Xen project power[EB/OL]. http://www. xenproject.org/.

[14] KVM[EB/OL]. http://www.linux-kvm.org/page/Main_Page.

[15] QEMU[EB/OL]. http://wiki.qemu.org/Main_ Page.

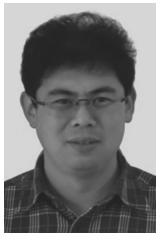
[16] 石磊, 邹德清, 金海. Xen 虚拟化技术[M]. 武汉: 华中科技大学出版社, 2009.
SHI L, ZOU D Q, JIN H. Xen Virtualization Technology[M]. Wuhan: Huazhong University of Science and Technology Press, 2009.

[17] 英特尔开源软件技术中心, 复旦大学并行技术处理研究所. 系统虚拟化:原理与实现[M]. 北京: 清华大学出版社, 2009.
Intel Open Source Software Technology Center, Parallel Processing Institute, Fudan University. System Virtualization: Principle and Implementation[M]. Beijing: Tsinghua University Press, 2009.

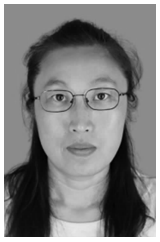
[18] ROBERT L. Linux Kernel Development[M]. New York: MacMillan Computer Publication, 2005.

[19] Suterusu[EB/OL].https://github.com/dschuermann/suterusu.

作者简介：



崔超远（1972-），男，内蒙古呼和浩特人，博士，中国科学院合肥智能机械研究所副研究员、硕士生导师，主要研究方向为虚拟化、信息安全。



乌云（1974-），女，内蒙古呼和浩特人，博士，中国科学院安徽循环经济技术工程院副研究员、硕士生导师，主要研究方向为人工智能、虚拟化。

李平（1989-），男，安徽安庆人，中国科学技术大学硕士生，主要研究方向为虚拟化技术。

张晓明（1979-），男，山东济南人，中国科学院合肥智能机械研究所副研究员、硕士生导师，主要研究方向为智能决策、智能计算、知识工程。