# A Conceptual Model of Self-Monitoring Multi-core Systems

Dipankar Dasgupta
Dept. of Computer Science
University of Memphis
Memphis, TN 38138
dasgupta@memphis.edu

Harkeerat Bedi
Dept. of Computer Science
University of Memphis
Memphis, TN 38138
hsbedi@memphis.edu

Deon Garrett
Dept. of Computer Science
University of Memphis
Memphis, TN 38138
deong@acm.org

## Abstract

The paper discusses a conceptual model of building a self-monitoring mechanism in a multi-core system such that the health of the entire system can be monitored securely and in an implicit manner. Accordingly, a many-core system is vertically partitioned (from hardware to application level) so that a small set of processors (security core) is separated from other user data processing cores to build a security subsystem. Then the model delegates security handling tasks to this isolated subsystem and uses a set of privileged instructions to monitor the other part (user side) and interact using a secure message exchange protocol. Simulation of the proposed security mechanism using several off-the-shelf software tools will be performed in order to demonstrate the proof-of-concept.

## Keywords

Multi-core system, Resource partitioning, Self-monitoring system, Dedicated security core, Virtual machine monitors.

## 1. Introduction

Multi-core systems have been used in many ways, for concurrency to exploit parallelism for performance gains, secure execution, etc. For example, Green Hills Software's INTEGRITY Multivisor [7], a version of INTEGRITY Secure Virtualization (ISV) solutions, is a multi-core hypervisor that allows safe execution of trusted real-time critical software in parallel with untrusted applications. This enables virtual machines and applications to safely co-exist with guaranteed memory, CPU time resources, and strictly enforced device access control. Its secure execution environment is isolated from the primary user environment by a combination of TrustZone hardware and INTEGRITY software.

The Secure Execution Architecture (SEA) [8] was developed to run the security-sensitive code of an application (called Piece of Application Logic, PAL) in isolation from another software concurrently. The hardware resource isolation is achieved using AMD's Secure Virtual Machine (SVM) and Intel's Trusted Execution Technology (TXT), while Trusted Platform Module (TPM) was added for SEA's TCB to provide sealed storage and attestations of PALs while running in multiple CPUs.

In [9], a secure Extensible Firmware Interface (EFI) environment is paired with an insecure OS so that it supports secure and reliable bootstrapping, hardware partitioning, encryption services, as well as real-time security monitoring and inspection. With this architecture, the secure EFI environment provides users with a management console to authenticate, monitor and audit an insecure (off-the-shelf) OS. This architecture also has the unique capability to protect authentication rules and secure information such as encrypted data, even if the security aspect of an OS is compromised.

Baumann et al. [10] proposed a multi-kernel OS architecture, called Barrelfish, which treats the machine as a network of independent core running separate OS kernels. The goal is to build a distributed system of processes with separate OS kernel with no inter-core sharing, but communicate via explicit message-passing between cores, and split-phase request and response operations. This approach, however, uses shared memory among OS kernels.

In Configurable Isolation Architecture [11], the authors proposed a configurable method of partitioning the resources of multi-core processors for fault containment and to support redundancy. The primary focus of this work is to contain hardware faults by isolating faulty hardware components through dynamic reconfiguration in order to perform load balancing with gradual degradation of performance. Though the configurable isolation approach exhibits how to partition hardware resources to address the fault problem, this work does not consider security issues involving the OS, applications and user interactions. Furthermore, the isolation mechanism only triggers when a fault occurs not during the normal operation.

Check Point's VPN-1 Power built on CoreXL technology leverages the multi-core features to speed up network security scanning in real-time. It allows load balancing of security-related traffic among multiple cores while integrating firewall, intrusion prevention and VPN capabilities.

Garfinkel and Rosenblum [12] developed a virtual machine introspection (VMM or VMI) technique for inspecting a virtual machine from the outside providing strong isolation of IDS from the host it is monitoring. They built a prototype of MMI-based IDS, called *Livewire* and tested the same on different security scenarios. While this approach uses separate VM for monitoring, but the use of same underlying shared hardware resources for instantiating multiple VMs may take over the host by exploiting vulnerability in the hypervisor [13].

Virtual machines running on the same physical hardware could use covert channels to transfer illegitimate information, resources like virtual shared discs and virtual shared networks (features provided by the majority of hypervisors) can also be used to share information explicitly by malicious users. This is mainly because hardware resources are not as often monitored as network traffic and therefore, such types of attacks are advantageous to the attacker [14]. A hypervisor that maintains and runs a number of virtual machines (VM) becomes a major target for attackers. This is because a compromised hypervisor makes the VMs running over it open for attacks. This can include full control over the VMs and all their data. Therefore, it is in the best interest of the concerned to keep the hypervisor itself as secure as possible [15, 16].

VM Escape is a type of attack where a process running under a virtual machine can bypass the virtual layer and directly access hypervisor resources. This results in the process gaining access privileges as of the host machine which is the root and eventually take over the entire system. Such an attack results in a complete compromise of the virtual environment [15, 16]. Recent reports indicate that the Bluepill and Vitriol hacking tools try to install a malicious hypervisor by adding stealth backdoor functionality in a legal hypervisor that is already present. Vulnerabilities have been reported in Xen, which can be exploited by malicious, local users to bypass certain security restrictions or gain escalated privileges. A flaw in the VMware software that allows malicious code running in a virtual machine can take over the host operating system [13].

IBM developed the hypervisor security architecture (sHype) [6] to govern the control and mediated sharing of resources among virtual machines. sHype [6] extends existing resource-level isolation hypervisors by implementing mandatory access control on virtual machines (VMs). It focuses on strong isolation, mediated sharing of resources among VMs in a controlled and secure manner such that vulnerability in one VM does not affect others. However, this approach does not consider partitioning processor cores and provide non-sharable resources to security services. We observe that one-sided monitoring, i.e., where one partition can monitor the other for the purpose of security can lead to better system protection [12]. This is achieved as the security monitoring application running under the different partition is not under the scope of the monitored system and hence not subject to its vulnerabilities. Our approach focuses on enhancing the monitored system's security through implicit one-sided monitoring by employing the novel concept of vertical partitioning, where partitioning of resources is strictly mandated at both, the hardware and software level.

Kaspersky Labs [3] recently reported their patent, which comprises of a hardware-based antivirus system to combat malicious programs. This hardware device is installed between the system hard drive and the computing unit which is the system's CPU and main memory and is connected to the system bus or integrated into the disk controller. The hardware device works on a level above the operating system and therefore, is not dependent on the same. They claim that the device can therefore effectively combat malicious programs like rootkits that elevate their privileges in the system. We believe that our proposed research (described below) is along similar lines but more generic and flexible to secure cyber physical systems with high reliability, efficiency and self dependency.

## 2. Existing Resource Partitioning Techniques and Tools

Resource partitioning is considered as a vital step in realizing self-monitoring systems. This section discusses the currently available off-the-shelf tools that provide the ability to partition hardware resources for various purposes and can collectively assist in building such systems. Hardware resource partitioning can be primarily classified into the following categories: partitioning of CPU (or CPU affinity), main memory and secondary storage.

There exist several software tools for system resource partitioning. Here we discuss the capabilities and limitations of a few such tools. CPU partitioning or CPU affinity assignment can be performed on Linux using the cpuset [1] mechanism where one or more individual CPU processing units and memory nodes can be grouped together under a cpuset. Processes can be assigned to the newly created cpuset for partitioning such resources. Cpuset is OS dependent and affinity is set only for processes running within the OS. Therefore, to confine a complete OS under a given cpuset for purposes of security monitoring, additional tools like virtual machine monitors, described below, may be required. Tools like Dataram's Ramdisk [2] and VSuite Ramdisk has the ability to partition the main memory.

The newly created partition can then be used as a standalone storage device for storing data and instructions. Limitations for such tools include their dependability on the underlying operating system. The partitions created are volatile and erased during system reboot or shutdown. In general, the partitions created can only be used for storing data (as secondary memory) and not as a main memory as required for our purpose of security monitoring. Hard disk partitioning tools such as KDE partition manager, GParted, and Microsoft's fdisk can divide the secondary storage into several partitions that can be used for various purposes from storing data to loading separate operating systems. Even though disk partitions made by one operating system are identified by other operating systems, other systems have full access to the entire hard drive.

In addition to the above mentioned categories, software tools like Virtual Machine Monitors (VMM) [4] can be used for complete system hardware partitioning. VMMs are classified mainly into two types: type 1 (bare-metal)--Xen Hypervisor [5] and type 2 (hosted)--VMware Server. Type 1 VMMs partition the system hardware resources and provide resource isolation for guest operating systems running directly above it. While a type 2 VMM is similar to type 1, they run within an operating system and not directly as hardware utility. Both types, in general, do not provide one system or process the ability to uniquely monitor the other system or process and CPUs are not implicitly hard partitioned but shared.

## 3. Self-Monitoring Security Model

This Security Model will be designed for self-monitoring a multi-core system such that the health of the entire system can be monitored securely and in an implicit manner. Accordingly, a many-core system is vertically partitioned where a small set of processor cores is isolated to build a security subsystem which will not share resources with other data processing cores. This subsystem then uses a set of privileged instructions to monitor rest of the system. This isolated security subsystem acts as a command and control center) and uses secure message exchange protocol to interact with other operational cores (user part of the system). Self-monitoring in our case can be defined as the ability to partition system resources efficiently and effectively such that one partition can monitor the other for enhancing the system's overall security.

Figure 1 illustrates a layered architecture in implementing the proposed self-monitoring single-chip multi-core (with 4 processor cores) system. It shows an unequal vertical partitioning (from hardware to software to user levels) to build two separate systems (with no shared resources) where the smaller partition monitors the other side in an implicit manner.

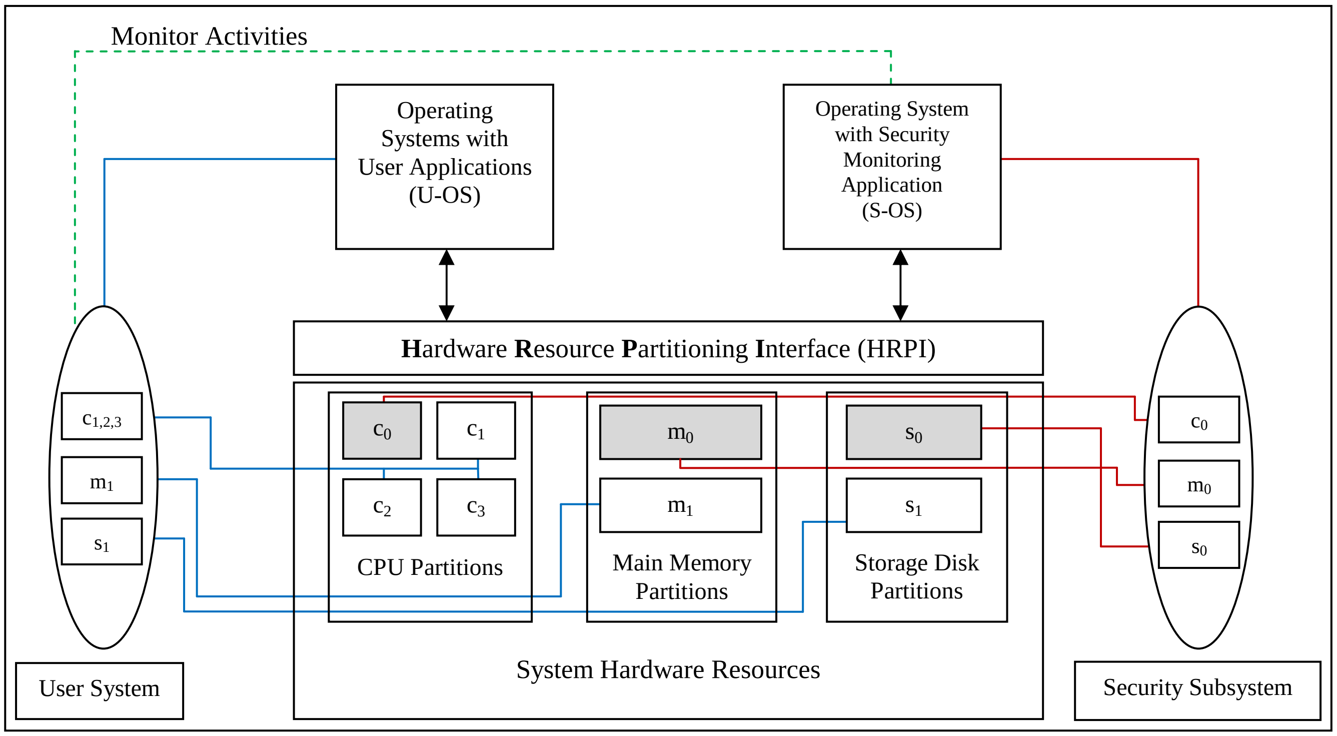
Monitor Activities
Operating Systems with User Applications (U-OS)
Operating System with Security Monitoring Application (S-OS)
Hardware Resource Partitioning Interface (HRPI)
c1,2,3
m1
s1
c0
c1
c2
c3
CPU Partitions
m0
m1
Main Memory Partitions
s0
s1
Storage Disk Partitions
c0
m0
s0
System Hardware Resources
User System
Security Subsystem

*Figure 1: shows the Hardware Resource Partitioning Interface (HRPI) to partition major system resources at the hardware level. Shaded units under system hardware resources are used by the Security OS (S-OS) running the Security Monitoring Application for its execution. S-OS also has the ability to monitor the resources used by the User OS for security purposes.*

The primary task revolves around complete isolation of some system resources by enforcing partition from hardware level to the application level. This is achieved by a Hardware Resource Partitioning Interface (HRPI) which creates two partitions of system hardware resources; one is the Security Subsystem with unique ability to monitor the user-side operating environment (other partition). A security console (not shown in the figure) with plugged-in administrative tools will assist the operator viewing suspicious events and alerts while monitoring the system activities.

As shown in figure 1, two operating systems run independently above HRPI layer; each uses non sharable hardware/software resources. Importantly, the security subsystem uses a different OS with a minimal privileged instruction set and independent of the user-side OS. Moreover, multi-core system-on-chip (SoC) can be augmented to handle security issues by monitoring low-level code execution in the user side.

The security HRPI will differ from currently available off-the-shelf hypervisors [5] by offering the additional ability to uniquely monitor the user operating system for security purposes and not being detected by the same. This ability is termed as one-sided monitoring. In above figure, resource units under System Hardware Resources represent the actual system hardware. HRPI partitions system resources at the hardware level. Shaded units, namely, $c_0$, $m_0$, $s_0$ are used by the operating system with security monitoring applications (S-OS) for its execution and are invisible to the user side operating system. Remaining isolated hardware resources, namely, $c_{1,\ 2,\ 3}$, $m_1$, $s_1$ on the other hand, are for use by the user operating system for its execution and are one-sidedly monitored by the S-OS for providing security. The security subsystems, however, should not be exposed to user applications.

Since the S-OS will be implemented on a set of resources isolated from the user operating system, attacks carried out on the user operating system will not affect the ability of S-OS to efficiently monitor the system. Software listed under the Resource Partitioning Tools section that offers processor and main memory partitioning achieve the same but in control of the underlying operating system.

## 4. Implementation of the security model using off-the-shelf software

The implementation of separate hardware for accomplishing such tasks, while effective, provides room for improvement. Our proposed model exercises a similar concept of hardware resource isolation for monitoring underlying operating systems and uses preinstalled system hardware for the same. There exist many system resource partitioning techniques; Section 2 discusses a list of these techniques and tools for hardware partition/allocation in a system in order to use shared hardware resources among multiple operating environment and processes.

We are simulating the proposed security mechanism using off-the-shelf software tools in order to demonstrate the proof-of-concept. As an example, processor and main memory node affinity can be set using the cpuset [1] functionality under Linux. A virtual machine can be installed under Linux and attached to the newly created cpuset, thereby enforcing the virtual machine to use only the allocated resources. Therefore, a guest (or User) operating system running under the virtual machine is also confined to the same set of resources for its execution.

Tools listed in Section 2 under the disk partitioning category have the ability to partition the physical hard disks, where disk partitions made by one operating system are identified by other operating systems. Thus by understanding their execution and implementing the same in combination with each other at a level below the user operating system, HRPI and S-OS can be realized to provide resource isolation at the hardware level and offer one-sided monitoring of the user operating system for protection.

The monitoring of user-side security will be carried out by the security subsystem through a security console with plugged-in administrative tools. The security console will gather various security statistics and periodically send them to the security Subsystem as Behavioral Messages. The Security Subsystem will compute averaged expected behavior and compares the currently observed behavior to the expected behavior information contained in the security check message. If an anomaly is detected (like Denial of Service, Buffer overflow), Security Subsystem is authorized to take protective actions. One way of handling a detected anomaly is to terminate the guilty process and refresh affected memory space.

The monitoring and protective functions will be emulated using various security alert messages in the presence of attacks, the proposed models be validated.  As a self-monitoring process, security subsystem (not assessable to users) will take various actions based on observed behavior of the system at multiple levels to keep the host/server operational. We are working to develop a proof-of-concept hardware-software test-bed to validate this innovative mechanism.

## 5. Summary

This security model proposes to vertically partition a multi-core system (from user level to hardware level) to build an isolated subsystem for monitoring the health of the entire system. This subsystem, called *security monitoring subsystem* consists of a small set of dedicated hardware resources, which include a processing core, RAM and secondary storage with separate OS will be use to monitor other parts of the system in an implicit manner. Since the security monitoring applications are under the different operating system, a compromise on the user operating system does not affect the monitoring capabilities of the security applications and thus help improve the overall system security. Another benefit of the security model is that it requires only minor modifications to existing hardware and software. Whereas other hardware-based monitoring solutions may require the addition of new hardware components, thereby incurring not only additional expense but also potentially drawing attention that the system features a known security augmentation. The proposed security model uses the same multi-core processor architectures present on essentially every consumer or professional level computers available today. While the software industry struggles to develop ways to take advantage of riches in available parallelism, the security subsystem model proposes a novel way of using processing cores that very often are idle with most software loads. The prototype will incorporate hardware level partitioning of resources with interfaces provided to facilitate secure communication across partition boundaries.

## 6. REFERENCES

[1] Cpuset Manual. Linux Programmer's Manual: http://www.kernel.org/doc/man-pages/online/pages/man7/cpuset.7.html

[2] Dataram's RAMDisk: http://memory.dataram.com/products-and-services/software/ramdisk

[3] Kaspersky Labs Business News: http://www.kaspersky.com/news?id=207576021 (accessed on March 13, 2010)

[4] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. In IEEE Computer Magazine, May 2005.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, and R. Neugebauer, I. Pratt and A. Warfield. Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003.

[6] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. V. Doorn, J. L. Griffin and S. Berger. SHype: Secure Hypervisor Approach to Trusted Virtualized Systems. IBM Research Report, RC23511 (W0502-006), February 2, 2005.

[7] INTEGRITY Multivisor, Green Hills Software at www.ghs.com.

[8] J. M. McCune, B. Parno, A. Perrig, M. Reiter, A. Seshadri. How Low Can You Go? Recommendations for Hardware-Supported Minimal TCB Code Execution. In ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), March 2008.

[9] X. Zhang, Y. Xie, X. Lai, S. Zhang and Z. Deng. A Multi-core Security Architecture Based on EFI, In Lecture Notes in Computer Science, Volume 4804, Springer Book Series, 2009.

[10] A. Baumann, P. Barham, P-E Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The Multikernel: A new OS architecture for scalable multicore systems. In the proceedings of 22nd Symposium on Operating Systems Principles (SOSP), October 11-14, 2009.

[11] N. Aggarwal, P. Ranganathan, N.  P. Jouppi, J.  E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In ACM Proceedings of the 34th annual international symposium on Computer architecture (ISCA), June 9-13, 2007

[12] T. Garfinkel and M. Rosenblum.  A virtual machine introspection based architecture for intrusion detection. Proc. In Network and Distributed Systems Security Symposium. Pages: 253-285. 2003.

[13] Vmware: Virtual-machine exploit lets attackers take over host.  http://news.zdnet.co.uk/security/0,1000000189,39661637,00.htm (accessed on March 13, 2010)

[14] V. D. Gligor.  A guide to understanding covert channel analysis of trusted systems. Technical report, National Computer Security Center, November 1993.

[15] J. Kirch.   Virtual Machine Security Guidelines.   The Center for Internet Security, September 2007.

[16] J. S.  Reuben.   A survey on virtual machine security. Technical report, Helsinki University of Technology, October 2007.