A Pre-Kernel Agent Platform for Security Assurance

Yung-Chuan Lee
Department of Computer Science
Southern Illinois University
Carbondale, Illinois 62901
Email: ylee@cs.siu.edu

Shahram Rahimi
Department of Computer Science
Southern Illinois University
Carbondale, Illinois 62901
Email: rahimi@cs.siu.edu

Sarah Harvey
Department of Computer Science
Southern Illinois University
Carbondale, Illinois 62901
Email: shharvey@cs.siu.edu

***Abstract*—Without the security assurance from the underlining operating system, software agents and agencies are constantly under security threats through the operating system. Although the security issues between software agents and agencies have been studied intensively, any formulated counter-measurement of these issues are subject to attacks that are able to exploit the vulnerabilities of the operating system kernel (OS kernel). To protect agents and agencies against such risks, we proposed a novel approach that isolates the OS kernel and limit modifications to the agency system in the physical memory space. Our approach utilizes a hypervisor to create a virtualized environment for the operating system. The agency is then loaded into physical memory and injected into the operating system of the host without any interactions from the OS kernel. All memory access requests are monitored and managed by our approach to prevent unauthorized modification from the OS kernel and maintain the integrity of the agency system. Furthermore, any agent migrated to the host can be encrypted by our approach to prevent any alterations from the OS kernel. The encryption key is safely shared between the agency system and the hypervisor by utilizing part of the memory space of the agency as a communication interface. Consequently, our approach is able to limit the security threats that exploit the vulnerability of the operating system to the agency systems as well as ensure the integrity of both agents and agencies. This study describes the formulation and implementation of the proposed platform that can securely deploy agent agent systems to the host and effectively control access to host devices.**

***Index Terms*—software agent, security, hypervisor, virtualization.**

I. Introduction

The software agent provides a high dimension of flexibility that allows hierarchical structure formulation, decentralized implementation, atomic modularization, mobility and autonomous, to name a few [8], [9]. With the help of technological advances in mobile devices and well-populated wireless and cellular networks, users can virtually have constant network access. The software agent is no longer restricted to private enterprise environments but is available for broader applications such as e-commence and living assistants.

In addition to the fundamental security problems, the software agent introduces new security issues between agents and agencies. When a user dispatches an agent to another host, the agent carries sensitive user information and preference that are used to guide the agent's behavior and decision process. If such information and preference of an agent were able to be extracted, malicious users or hosts can exploit this to anticipate the agent's actions and gain maximum profits. For instance, in an auction system, a user will make sure its buying agent knows the maximum bid of each interested item and the bidding strategy. When the naive buying agent gets captured and that information is extracted, a malicious seller can craft a fake buying agent to push the naive buying agent to its maximum bid. Therefore, securing agents from malicious agencies and vice versa has became a major obstacle preventing the security agent from fully realizing its potential.

The security issues of software agents can be generalized in four categories: agent-to-agent, agent-to-agency, agency-to-agent and other-to-agency [10]. Agent-to-agent security threats concentrate on exploiting the vulnerabilities of other agents and launching attacks against them. Agent-to-agency security focuses on exploiting and attacking an agency platform. Agency-to-agent security refers to the agencies' compromise of the security of agents. Finally, other-to-agency pertains to attacks from external agents or agencies to an agency [10]. These threats are mainly addressed within agent and agency formulation, and the approaches are briefly reviewed in the next section.

It is important to recognize that an agency is nothing more than a regular program that provides resources to agents, and therefore is also extremely vulnerable to conventional attacks from the host. For instance, when a malicious user obtains kernel access to an operating system, the user can easily duplicate an agency's physical memory contents to extract sensitive information of the agents or capture agents at the network adapter during migration to examine their behaviors. This further renders any counter-measurements of the aforementioned software agent security threats useless as the malicious user can gain ultimate control of all devices (i.e. network adapter, memory and others) once the OS kernel is breached.

One way to warrant software agent security is to perform exhaustive checking of all instructions for each process running on the operating system to block unauthorized accesses to memory addresses, network adapter and other devices. This approach, however, can still suffer from root-kit attacks that target the vulnerability of the operation system kernel. To

978-1-61284-060-4/11/$26.00 ©2011 IEEE

remedy this, we propose a *Pre-Kernel Agent Platform* that operates below the operating system on a host and protects agent systems against threats from the operating system. The platform utilizes a hypervisor that is initialized and loaded independently from the operating system to ensure the integrity of the physical memory of the host to prevent any pre-emptive attacks. The hypervisor provides virtual devices to the host operating system and limits direct access to the host physical devices. Through the virtual devices, the hypervisor is able to intercept device access from the operating system and maintain the data integrity of monitored physical devices. Agent systems can be injected into the host operating system by the hypervisor to reduce security threats from the host operating system. Formulation details of the proposed platform are elaborated in Section III.

In this paper, first the state-of-the-art for software agent security is summarized in Section II to provide an overview of current developments. The proposed platform and its components are delineated in Section III. Implementation details are presented in Section IV. Finally, we conclude with a discussion of future directions in Section VI.

II. State-of-the-art Approaches

To the best of our knowledge, there are few existing studies focused on securing mobile agents and agencies against attacks from the operating system on a host. The difficulty lies with the fundamental design of the operating system: the kernel has ultimate control over both hardware devices and application processes. As a result, the software agent research community has often assumed that the operating system is trustworthy. The existing research in this area tends to utilize external hardware devices to create a trusted computing environment to address this problem [4], [12]–[14], [16], [18]. However, the special hardware requirements prevent the deployment of a mobile agent system on any existing regular computing machines and thus limits the adoption of a mobile agent system.

With the recent advancement of the central processing unit (CPU), virtualization techniques have been extensively studied and widely applied to security research. For instance, Chen and Noble argued that the virtual machine (VM) is superior to real machine, and discussed both the benefits and challenges of moving from real machine to VM [5]. Most importantly, the study identified semantic gap problems in a virtualized environment where service operating outside the guest VM cannot access the abstract information provided by the operating system in the guest VM. In sum, this position paper established the fundamental requirements for adapting virtualization techniques.

Jiang and colleagues proposed a stealthy malware detection to detect malware in guest virtual machine from the host operating system [11]. To overcome the semantic gap problem, the study constructed a semantic view of guest VM in a virtual machine monitor (VMM) and utilized VMWatcher in the host operating system to monitor malware in guest VM. The approach presents a compelling argument that a host operating system can ultimately detect and remove malware and security threats in the guest VMs. However, as the authors discussed in their study, the approach assumes a trustworthy VMM layer and does not provide any solution to prevent security threats from guest VMs to VMM.

Riley and colleagues introduced a kernel rootkit prevention system, NICKLE, to prevent unauthorized kernel code execution from rootkits [15]. NICKLE relies on a trusted VMM to perform physical memory shadowing of a guest VM kernel right after the decompression process during booting to ensure the integrity of a known clean kernel. Unauthorized instructions intended to be executed in the kernel space can be intercepted to guard any modification to the kernel of the guest VM. The study results demonstrated minimum performance overhead to the VMM platform.

Chiueh and colleagues proposed an approach to stealthily deploy and execute agents into other guest VMs in a virtualized environment [6]. The agent can be dynamically injected into a guest VM for execution in the kernel, and is invisible and difficult to intercept by the VM's kernel. The approach addressed the semantic gap problem with an in-guest agent, and the performance penalty of the injection process was found to be insignificant from the simulation.

Azab and colleagues proposed a hypervisor-based agent (HIMA) that lived inside a hypervisor to measure the integrity of the guest virtual machines running the host [1]. HIMA provides active monitoring to the important guest VM event, as well as memory protection of guest VMs to ensure the correctness of the integrity measurement. Lastly, Baiardi and Sgandurra proposed Psyco-Virt, which employs an introspection VM and agents to build trustworthy intrusion detection to monitor guest VMs [2].

Despite the fact that many studies utilize both virtualization and software agent techniques in the security field, none of these studies focus on preventing security threats from the operating system to mobile agents and agencies while maintaining the notion of agent mobility. The results of these studies indicate that the performance overhead caused by the virtualization is minimal, and we have similar performance expectations for our proposed platform.

III. The Platform

We believe that in order to provide fundamental security, it is necessary to separate hardware device controls from the operating system (OS) kernel. All hardware accesses from the OS kernel can then be managed and intercepted by our proposed *Pre-Kernel Agent Platform*. In this section, we will first provide a logical formulation, and then illustrate the design of the proposed platform in detail.

A. Formulation and Assumption

The fundamental design of an operating system (OS) grants the OS kernel ultimate access to the host hardware resources. Most security counter-measurements are built under the ideology that the integrity of the OS kernel is not altered. Once the OS kernel is breached, it renders all security counter-measurements useless. Countless evidence from threats like

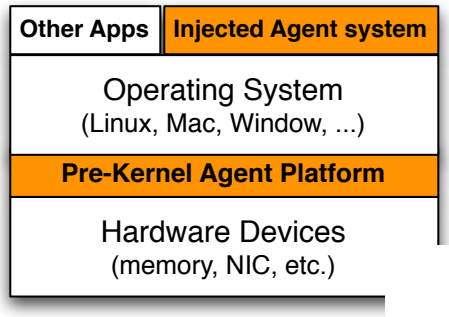


Fig. 1. Platform Overview

Before we move to the details of the desig
tions have been identified during our formu
security of the host and agent system secur
we assume the physical access to the host
tampering with devices of the host is not
assumption is necessary to eliminate the attack
hardware devices, as currently there are no
that can address this issue. Secondly, our ap
on security threats from the operating syste
system, including both the agent and agency, a
security problems between the agent and agen
by the agent system. Once we can guarantee
the agencies and agents during execution, the
an agent system can be preserved by enforc
mechanism of the agent system.

B. Pre-Kernel Agent Platform Design

Figure 2 shows a detailed view of the pro
which consists of three main components: ligh
visor, Device Access Monitoring (DAM) age
Loading and Injection (ALI) agent. The light-
sor is a thin and transparent layer between th
and the operating system, and provides virtua
the operating system. The main function of th
to manage and translate all virtualized hardw
the operating system to corresponding physic
DAM agent then hooks itself in the hypervisor
intercept the hardware access from the opera

ensure the initial integrity of the agency system, the ALI agent is responsible for loading the agency into a secure memory area and injecting the agency process to the operating system after the operating system is initialized.

The hardware access from the operating system kernel is now classified into three types: bypassed, monitored but allowed, and monitored but denied. When the operating system (OS) kernel accesses a virtualized device, the hypervisor checks with the DAM agent to determine whether the corresponding physical device of such an access is restricted. If the access is not restricted, the hypervisor bypasses the restriction from the DAM agent and grant direct access to the physical device to minimize the performance impact. On the other hand, for access to a restricted device, the DAM agent imposes access constraints and monitors the contents that are read or written by the OS kernel. Lastly, for the monitored but denied access (i.e., other programs attempt to read protected agency memory location), the DAM agent returns an access deny response back to OS kernel to gracefully terminate the request. By default, both the hypervisor and ALI agent are trusted and have direct and unrestricted access to the hardware devices to reduce the performance impact of the proposed

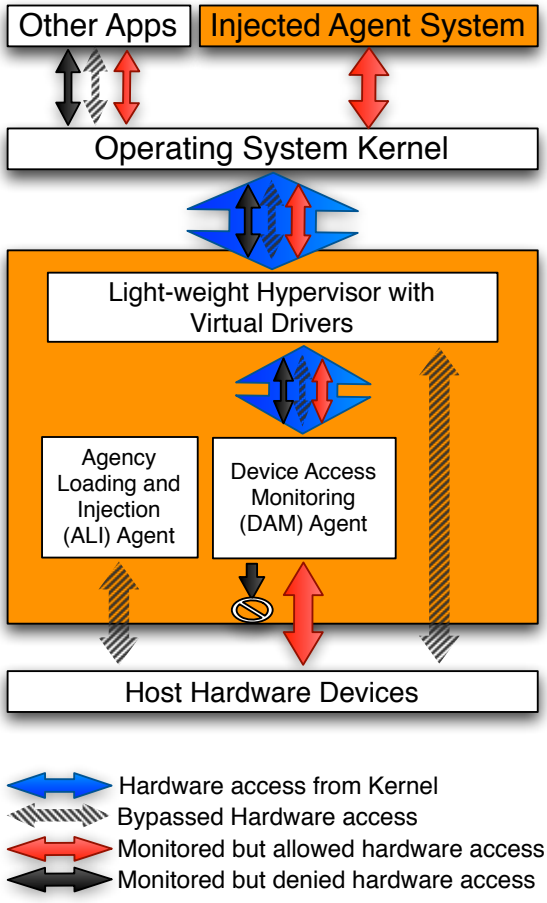


Fig. 2. Pre-kernel Agent Platform

C. Agency Loading and Injection Agent

The Agency Loading and Injection (ALI) agei sible for two main functions: loading binary ir an agency system into physical memory and into the OS kernel process pool. After the *Pre-K Platform* and the operating system initializes, th obtains a secure memory area and loads the ag instructions into it. The ALI agent then hijacl execution and injects the process by manipulati and EBP registers.

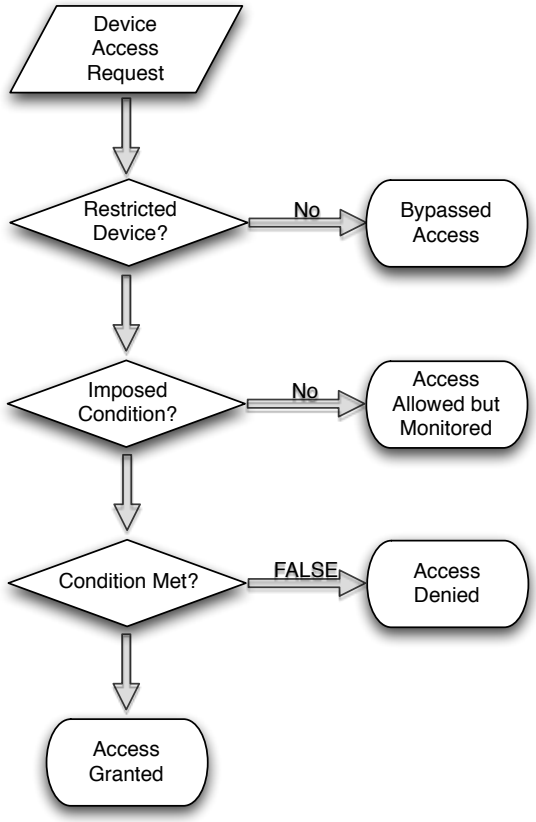
Once the injection process is completed, the inserts a new memory restriction rule to the dec the DAM agent and specifies that only the age can access and modify this secure memory are ALI agent has direct access to the physical memc executes once the host reboots, this process ha impact on the overall performance of the *Pre-K Platform*.

D. Device Access Monitoring Agent

The Device Access Monitoring (DAM) agent u nary decision tree to allow or deny device access r the kernel. Figure 3 illustrates this process. Th structure of the decision tree enables both con control of each device and the capability of fin performance of the platform. Each level adds an e control with the cost of the performance. Prior to all devices of the host can be configured as a resti with different rules and conditions.

Upon deployment, the ALI agent injects the ag into the operating system and returns the mem of the agency system to the DAM agent. The imposes maximum restriction on this location any memory access (both read and write) from a except the agency system itself. Furthermore, to modifications to each agent during migration, the monitors the network device to capture the agent p It then computes the digital digest of the migrated encrypts the agent with one-time key, and writes both digest and encryption key to a reserved block in the memory area. After the agency receives the migrated a reads the digest and key from the reserved block and the integrity of the agent.

Figure 4 shows the structure of the reserved block agency memory area. After the DAM agent writes the and key in the data area, it sets the *Ready Bit* to 1 a *ACK Bit* to 0 to signify that the data is ready to b Once the agency receives the agent from the operating kernel, it then reads the digest and key from the da and sets the *Ready Bit* to 0 and the *ACK Bit* to 1 to i that the data area is safe to be rewritten. The reserve is initialized to all 0s, and the mutual exclusion on th *Bit* write operations between the DAM agent and the is regulated by the *Ready Bit* and *ACK Bit*. Table I illu how this mutual exclusion is guaranteed.



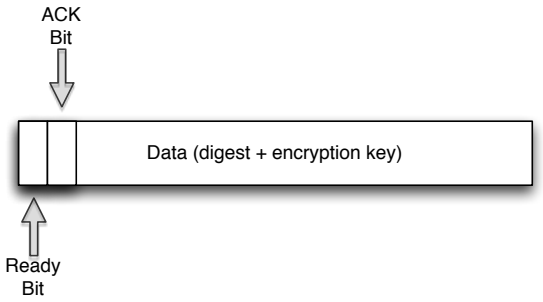


Fig. 4. Reserved block in the agency memory area

TABLE I
MUTUAL EXCLUSION ON WRITE OPERATION TO THE RESERVED BLOCK

Ready Bit	ACK Bit	Current State	Write Operation Allowed
0	0	Initial state	DAM agent
1	0	Ready state	The Agency
0	1	ACK state	DAM agent
1	1	Invalid state	

IV. Implementation Preparation

To provide a rapid proof-of-concept simulation, the initial implementation utilizes VirtualBox [17]. VirtualBox creates a virtual environment that is managed by a virtual machine manager that is rather similar to a hypervisor, and provides all the necessary hardware drivers to different operating systems. Instead of building a complete hypervisor with device drivers, the use of VirtualBox provides us with the leverage to concentrate on implementing the *Pre-Kernel Agent Platform* to examine its correctness and evaluate its performance. Furthermore, utilizing VirtualBox does not compromise the soundness of the experimental results since in VirtualBox the host operating system and the host hardware are treated as one virtual host with virtualized devices. In fact, the performance results will most likely be undermined because of this layer of "virtualization" between the host physical hardware and the hypervisor.

The implementation is still ongoing at the time of this writing and should be completed shortly since we have identified the function calls in the VirtualBox that are responsible for managing and manipulating memory and CPU execution. For the initial study, the *Pre-Kernel Agent Platform* will only monitor and restrict both memory and network adaptor access from the operating system kernel. Nonetheless, Algorithm 1 demonstrates an overview of the implementation.

We plan to adopt the Java Application Development Framework (JADE) [3] as the mobile agent system for this initial experiment. JADE has received great acknowledgement from both the software agent community and industry, and can provide comprehensive features such as FIPA ACL compliant [7] and mobility support. In addition to the main objective of proofing-of-concept in this experiment, we plan to show that only minimum modification is needed to extend an existing mobile agent system to support the agent migration integrity verification that is provided by our proposed platform. We have identified that in JADE, the Agent Management System (AMS) handles the migration process through Agent Communication Language (ACL). Algorithm 2 illustrates the steps needed to be added before the deserialization of the migrated agent.

V. Discussion

In addition to encrypting the migrated agents, another way to secure the agent when it arrives is to completely bypass the OS kernel and write the agent into the reserved block of the agency memory. In this way, the encryption can be omitted. However, since the size of an agent is not static and the data of the agent grows dynamically, it is difficult to determine the size of the reserved block beforehand. The cost of the dynamic memory allocation will most likely surpass the performance gain from not performing the encryption. In addition, this method will require further modifications to the existing agency system, which we prefer to prevent.

Since the agency runs inside the operating system, our approach eliminates the "semantic gap" problems mentioned earlier in Section II [5]. Even though the agency and agents are exposed to the operating system kernel, accessing agency

Algorithm 1 Pre-Kernel Agent Platform Pseudo Code

```
Let start be the start address of agency memory area
Let end be the end address of agency memory area
Let max be the maximum memory size to be used by the agency
Let agency be the binary instructions of the agency
#waiting for the OS to finish initialization#
while OS is not initialized do
  wait();
end while
#ALI agent loading and injecting the agency#
memoryAllocate(max, start, end);
loadAgency(agency, start, end);
hijackCPU(start, end);
decisionTreeInsert(start, end);
#DAM agent monitoring device access and securing migrated agent#
Let request be a device request
while OS is not in the process of shutdown or reboot do
  for every request in requestQueue do
    if request is Memory then
      if request is to agency memory area then
        if request is from agency then
          executeRequest(request);
        else
          denyRequest(request);
        end if
      end if
    else
      if request is Network Adaptor then
        Let packet be the current set of packets from network adaptor
        if packet is a migrated agent then
          Let key be a random generated key
          Let digest = computerDigest(packet);
          Let agent = encryptAgent(packet, key);
          Let ReadyBit be 1
          Let ACKBit be 0
          while getCurrentState() is Ready state do
            wait();
          end while
          insertDataToReservedBlock(ReadyBit, ACKBit, digest+key);
          transmitPacketsToKernel(agent);
        else
          transmitPacketsToKernel(packet);
        end if
      end if
    else
      bypassedAccess(request);
    end if
  end for
end while
```

Algorithm 2 Agent Migration Integrity Verification Pseudo Code

Let $readyBit$ be the Ready Bit content in the reserved block of agency memory
if $readyBit$ is set to 1 **then**
  Let $encrytedAgent$ be the migrated agent the agency received from the OS kernel
  Let $data$ be the contents in the Data in the reserved block of agency memory
  Let $digest$ = getDigest($data$)
  Let key = getKey($data$)
  Let $agent$ = decryptAgent($encryptedAgent$, key)
  Let $verified$ = verifyDigest($agent$, $digest$)
  if $verified$ is true **then**
    continue deserialization of the migrated agent
  else
    send FAIL ACL communication to the original agency report possible tampering from the OS kernel
  end if
  Set ACK Bit content in the reserved block of agency memory to 1
  Set Ready Bit content in the reserved block of agency memory to 0
end if

memory area is regulated by the *Pre-Kernel Agent Platform* to prevent unauthorized access from the kernel. The performance cost should be tolerable since the platform only performs two condition checks on each device access request.

By utilizing the memory area as a communication interface between the *Pre-Kernel Agent Platform* and the agency, our approach further enables customizable security assistance to the agency system without interference from the operating system kernel. For example, assuming the memory contents can be compromised by the operating system kernel in the worst case scenario, an agency system can be formulated to utilize the *Pre-Kernel Agent Platform* for encryption key management to enforce full content encryption and integrity check after initialization.

VI. Conclusion

The proposed *Pre-Kernel Agent Platform* inherits the operating system isolation property of the virtual machine and reduces the feasibility of many classical security problems, such as rootkit threats to the agent system. With minimum modification to the exiting agency systems, our proposed platform is able to provide sound counter-measurements against threats from the operating system that the agency system constantly runs on. Our presented platform further ensures the integrity of the migrated agents by computing digital digest and encrypting the agent when it arrives at the network adapter. This not only prevents the alternation from the OS kernel to the migrated agents, but also prohibit the OS kernel from extracting the content of the agent. By eliminating the possibility of security threats from the operating system to the agency and agents, our platform is able to create an environment in which the software agent security problem can be completely dealt with within the agency system.

Once the correctness of every detail of our platform is verified from the virtualBox simulation, we are planning to conduct a native customized hypervisor implementation. Because of the "virtualized" layer between the VirtualBox and host hardware, we anticipate that the performance resulted from the VirtualBox simulation will be adequate. We expect the performance of the native hypervisor implementation will be much higher. Since our approach heavily relies on the isolation property provided by the hypervisor, we will examine the possibility of security threats from the operating system kernel to the hypervisor in future studies.

References

[1] A.M. Azab, P. Ning, E.C. Sezer, and X. Zhang, "HIMA: A hypervisor-based integrity measurement agent", *In Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC 09)*, pages 193206, 2009.

[2] F. Baiardi and D. Sgandurra, "Building Trustworthy Intrusion Detection through VM Introspection", *The Third International Symposium on Information Assurance and Security*, pages 209-214, August 2007.

[3] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, "Jade Programmer's Guide", 2010. http://jade.tilab.com.

[4] S. Crane, "Privacy Preserving Trust Agents", Technical Report HPL-2004-197, HP Labs, Bristol, UK, November 2004.

[5] P. Chen and B. Noble, "When virtual is better than real", *In Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS 01)*, pages 133-138, Washington, DC, USA, May 2001.

[6] T. Chiueh, M. Conover, M. Lu, and B. Montague, "Stealthy Deployment and Execution of In-Guest Kernel Agents", *The Black Hat Technical Security Conference USA 2009*, Las Vegas, NV, July 2009.

[7] FIPA, "Foundation for Intelligent Physical Agents", http://www.fipa.org.

[8] ISTAG,"Scenarios for Ambient Intelligence in 2010", http://www.cordis.lu/istag.htm.

[9] ISTAG, "Ambient Intelligence: from vision to reality", http://www.cordis.lu/istag.htm.

[10] W. Jansen and T. Karygiannis, "Mobile Agent Security", Special Publication 800-19, National Institute of Standards and Technology, August 1999.

[11] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based öut- of-the-boxšemantic view reconstruction", *In Proceedings of the 14th ACM conference on Computer and communications security (CCS 07)*, pages 128138, New York, NY, USA, 2007.

[12] S. Pearson, "Trusted Agents that Enhance User Privacy by Self-Profiling", Technical Report HPL-2002-196, HP Labs, Bristol, UK, July 2002.

[13] S. Pearson, "How Trusted Computers can Enhance for Privacy Preserving Mobile Applications", *In Proceedings of the 1st International IEEE WoWMoM Workshop on Trust, Security and Privacy for Ubiquitous Computing (WOWMOM 2005)*, pages 609613, Taormina, Sicily, Italy, June 2005.

[14] A. Pridgen and C. Julien, "A Secure Modular Mobile Agent System", *In Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2006)*, pages 6774, Shanghai, China, May 2006.

[15] R. Riley, X. Jiang, and D. Xu,"Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing", *In Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*, pages 1-20, Boston, MA, September 2008.

[16] Z. Shen and X. Wu, "An Improved Security Method for Mobile Agent System by Using Trusted Computing Platform", *2010 International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 583-586, May 2010.

[17] VirtualBox. http://www.virtualbox.org

[18] U.G. Wilhelm, S. Staamann, and L. Butty, *Introducing Trusted Third Parties to the Mobile Agent Paradigm*, Secure Internet Programming, Lecture Notes in Computer Science, Springer, Heidelberg, vol. 1603, pages 469489, 1999.