Virtualization Security Combining Mandatory Access Control and Virtual Machine Introspection

I. INTRODUCTION

As cloud computing increasingly takes the center stage in an organization's IT infrastructure, it provides a huge incentive for hackers to gain illegal access to virtualization environments. Therefore, new virtualization security solutions will need to be developed to counter these attacks.

Virtualization enables a layer of abstraction through device emulation, and facilitates resource sharing among multiple guests running different operating systems in the form of virtual machines (VMs).

A virtualization environment consists of multiple guest VMs and the virtual machine monitor (aka. hypervisor) running between VMs and the hardware to regulate their access to the underlying resources. With the wide use of virtualization technology, the underlying virtualization infrastructure is becoming a popular target for attacks. Attacks such as Distributed Denial of Service (DDoS) and privilege escalation are launched against the virtualization environment.

Existing virtualization security techniques to protect the virtualization infrastructure against attacks are limited in the protection scope and often incur high level overheads in a real-world deployment.

In this research we will investigate a memory-based virtualization security approach. Through combining protected in-VM monitoring with mandatory access control (MAC) using SELinux, our proposed approach aims to provide comprehensive protection of both the guest virtual machines and the underlying hypervisor against attacks, without incurring high level overheads.

The remainder of the paper is arranged as follows. A literature review is presented in Section 2, and the limitations of existing virtualization security techniques are discussed in Section 3. A new solution is proposed in Section 4. Section 5 presents the prototype while the future work is set in Section 6.

II. LITERATURE REVIEW

A. Mandatory Access Control (MAC)

In mandatory access control (MAC), access to resources is controlled and determined by the system administrator through access policies. This centralized approach to resource access control is different from discretionary access control (DAC), in which access to an object is determined by the object owner [1].

Based on Bell-LaPadula (BLP) access model [2], MAC assigns sensitivity levels to objects (resources) and clearance levels to subjects (users) which reflect their significance within the system. They are then used to define access policies which are enforced whenever a subject makes a resource access request. MAC can be implemented based on Linux Security Module (LSM), which is commonly existent in modern Linux kernels.

A MAC scheme typically consists of three components, namely, reference monitor, enforcement hooks and access control policies, as depicted in Figure 1. Reference monitor: is responsible for monitoring all resource access requests [3]. It uses the access enforcement hooks to intercept any resource access request and deliberates about the request according to a set of pre-defined access policies [4].

Access enforcement hooks: are invoked when a guest VM makes a resource access request. They intercept any resource access from the VMs and pass the intercepted information to the reference monitor. Access control policies (ACM): are a set of rules defined by the administrator to determine which resources a user is allowed access to as well as at what privilege levels. Access control policies are used by the reference monitor to determine whether a given resource access request should be granted.

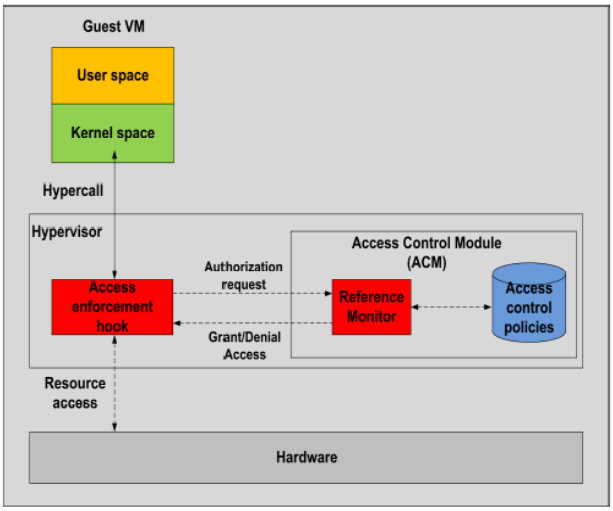

Figure 1. Architecture of Mandatory Access Control (MAC)

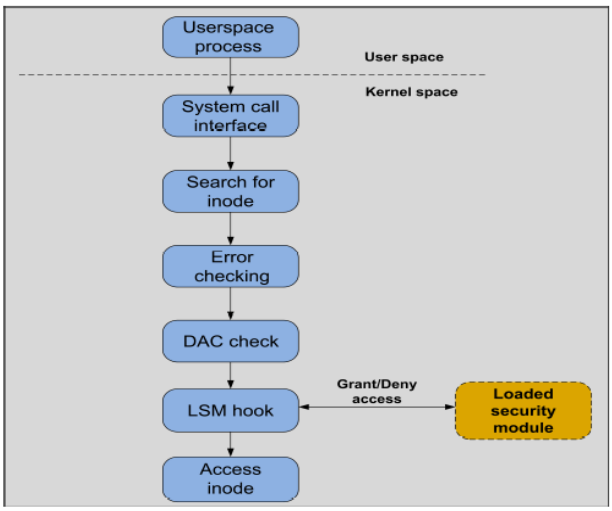

Figure 2. Linux Security Modules (LSM) architecture

A MAC-based virtualization security solution by [5] uses an access control module (ACM) to deliberate about resource access requests according to a set of access control policies. The policies are defined using Simple Type Enforcement (STE) and Chinese Wall Enforcement (CWE). While this enables fine-grained resource access control, the centralized nature of the ACM module means that it is vulnerable to hypervisor-targeted attacks.

This was extended in [6] to their sHype access control solution. It leverages the guest isolation property of the
hypervisor by enforcing STE and CWE access control policies for every resource access request. However, it does not provide mechanisms to maintain the MAC policies associated with a guest VM during VM migrations. sHype was extended in [7] to Shared Access Monitor (Shamon). It uses a physical TPM hardware for remote attestation, as well as MAC VMs to monitor inter-VM communication. Whenever two hosts want to communicate to enable inter-VM communication, a shared reference monitor is created which combines their access control policies before communicating via a IPSec-enforced communication channel. While this ensures the proper enforcement of MAC policies, it is impractical to be deployed on a real-world multi-host environment due to potential high-level overhead.

B. Linux Security Modules (LSM)

Linux Security Module (LSM) is designed to support different Linux security schemes and can be run as kernel modules without causing compatibility issues with the main Linux kernel [8]. LSM itself does not contain the logic to perform access control, but it provides the

software infrastructure required for security modules to access the internal kernel data structures.

LSM provides function hooks which are placed at critical points within the kernel. They are then connected to the internal kernel data structure objects by adding an additional security field which points to the functions [9]. Since both Xen Cloud Platform (XCP) and Kernel-based Virtual Machine (KVM) are Linux based, LSM can be used to implement MAC-based VM resource access control in platforms built upon XCP or KVM. Figure 2 illustrates how LSM works.

LSM is used in the Usable Mandatory Integrity Protection (UMIP) solution by [10]. It protects the system against network-based attacks by categorizing processes into lowand high-integrity levels. It uses the file information stored in the existing DAC access control to determine the binary programs and files which need to be protected. The information is then used to generate Mandatory Access Control (MAC) policies detailing a program's access privileges to a file.

LSM is incorporated with POSIX 1.e access control policy in [11] in their Lothlorien by using the Extended Attributes (EA) functionality to add security-specific information to the files' metadata. It uses the LSM hooks to intercept the user resource access request and extracts the request details, and then compares them with the entries in the policy database. While the separation of users and resources by zones helps to prevent privilege escalation attacks, it compromises usability since it does not allow a user to access files in different zones.

LSM is also used in MAC-based malware and rootkit detection solution in [12]. The process information is obtained by using LSM to intercept the system calls which are activated when a process is executed. They are then passed to the monitoring module to determine its legitimacy. The monitoring module is protected by using a MAC-based protection scheme to ensure that only whitelisted processes can access it.

C. Security-enhanced Linux (SELinux)

Developed by the National Security Agency (NSA), SELinux works alongside the existing Discretionary Access Control (DAC) to provide fine-grained resource access control. It combines the elements of Type Enforcement (TE), Role-based Access Control and Multi-level security (MLS) to provide flexible mandatory access control (MAC) [13]. SELinux assigns to subjects and objects security contexts, which are security attributes describing the access privileges associated with their access. A security context in SELinux is of the form user:role:type, each of which plays an important role in the specification of access control policies and can best be explained in reverse order. The type identifier is used by SELinux to enforce Type Enforcement (TE), which is the default access control in a SELinux-enabled Linux environment. It uses the types associated with subjects and objects to specify access control policies detailing objects [14]. The type enforcement helps ensure that no user has complete access to the objects even for a root user, preventing privilege escalation attacks.

The role identifier refers to the SELinux role which a given subject (process) is running under. While not used in a TE policy specification, it is used to limit the types which a process can be associated with in the system by associating users with the processes which they are allowed to run.

The user identifier refers to the SELinux user label of the process which created the object, and is used by SELinux to define policies. The user identifiers are then mapped to the users in the system, making it easier to perform access control among multiple users. It is used in SELinux together with the role identifier to specify the processes and files which a SELinux user can access, allowing for more finegrained access control.

The SELinux architecture consists of four components, namely, the SELinux hook, Access Vector Cache (AVC), the Security Server and the SELinux Security Policy. Figure 3 illustrates how SELinux works alongside the Linux Security Modules (LSM). SELinux is used in Virtual environment Secure File System (VSFS) [15] on the Xen virtualization platform. It uses SELinux to generate MAC policies by incorporating guest VM access privileges with their IP addresses in policy definition. The binding of IP addresses in the policy definition is not practical in a real-world environment, as it prohibits an owner from making changes to a file from another VM.

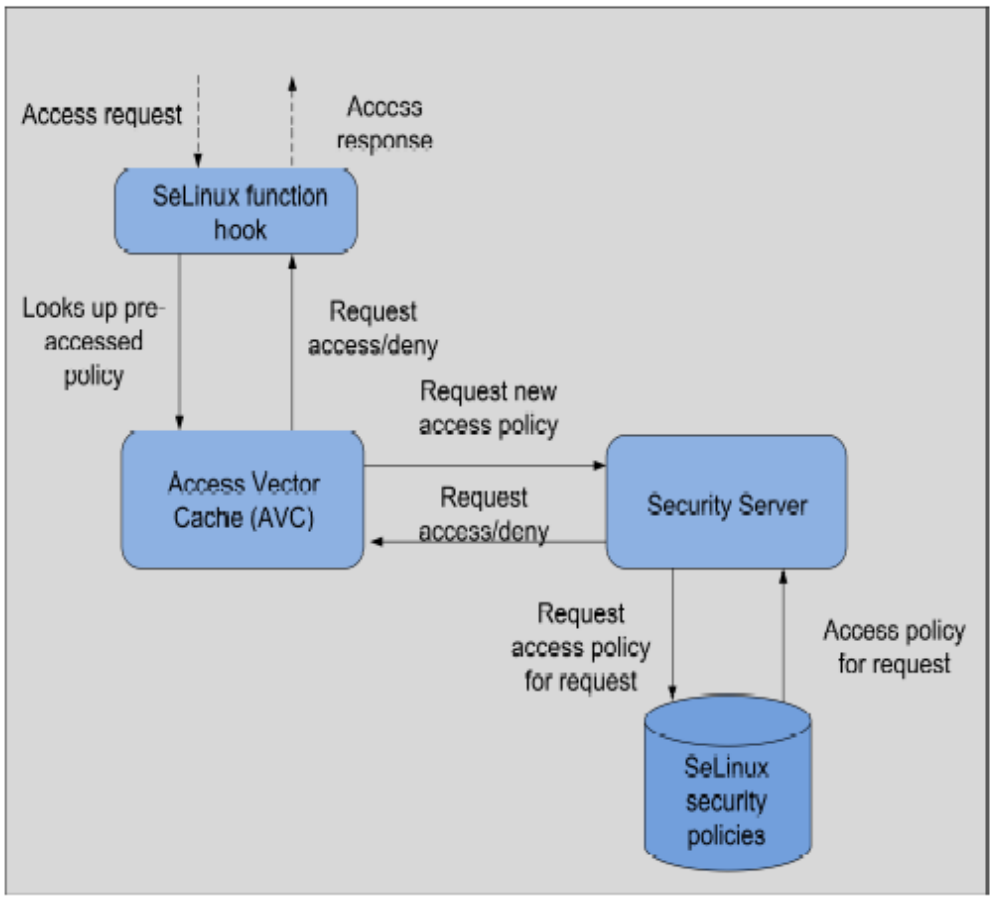

Figure 3. SELinux architecture

This limitation is addressed in [16] to protect the High Performance Computing (HPC) nodes against attacks. The user name and the group to which the user belongs is used instead of IP addresses in policy definition, improving the users' ability to access the files.

SELinux is incorporated with on-board Trusted Platform Module (TPM) in the MAC-based distributed usage control solution [17]. The SELinux policy associated with the file and the user is converted into an Extensible Access Control Markup Language (XACML) document and sent to the user client to ensure consistent enforcement of access control. While this approach can be adapted in a virtualization security context to ensure that the access rights associated with a virtual machine is maintained across host migrations, the need for TPM presents a drawback as not every server is equipped with it.

D. Virtual Machine Introspection (VMI)

Originally proposed in [18] to protect the monitoring tool from malware compromise, VMI monitors the guest VM activities by using information about the hardware on which the guest VM is running (e.g., physical memory) to estimate the state of the applications running within it. Figure 4 illustrates how VMI works.

The VMI-based solution [18] consists of an OS interface library and a policy engine. The monitoring module uses the former to access the guest OS while the latter is used for threat detection. While this protects the monitoring tool integrity by running it in a separate VM, it is limited in obtaining a comprehensive view of its internal state. An alternative rootkit detection approach [19] reconstructs the storage and memory contents of a guest VM based on external observations. While this approach achieves a certain degree of success in its attempt to bridge the semantic gap, the potential reconstruction overhead in a multi-VM environment will affect the overall virtualization environment performance.

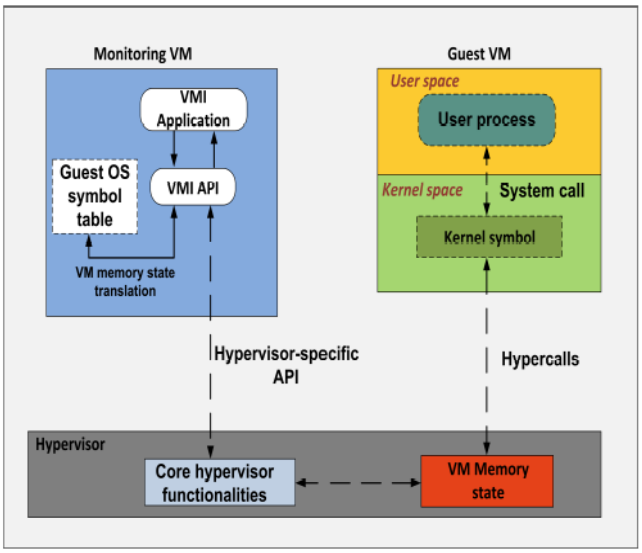

Figure 4. Virtual Machine Introspection

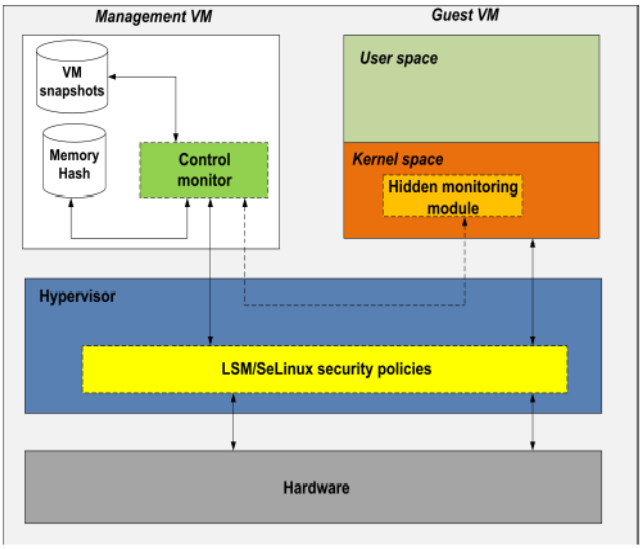

Figure 5. Proposed solution

VMI is incorporated in [20] with forensic memory analysis (FMA) and machine learning to implement a malware detection prototype. While this solution provides automatic malware detection using a machine learning algorithm, it fails to prevent hardware-based attacks.

III. LIMITATION OF EXISTING TECHNIQUES

Protecting the virtualization environment against attacks requires protecting both the guest virtual machines as well as the underlying hypervisor. Virtual Machine Introspection (VMI) enables the external monitoring of a guest VM's internal state by using the information obtained at the hypervisor level to infer its state.

While this protects the monitoring components (e.g., the memory scanner, signature database, etc) from compromise, it suffers from the semantic gap which exists between the actual VM internal state and the information obtained through external monitoring. In addition, it does not scale well in a real-world multi-guest environment where guests tend to migrate across hosts during their lifetime.

Protecting the hypervisor through Mandatory Access Control (MAC) allows for a centralized resource access control. The use of the Linux Security Module (LSM) together with SELinux enables the specification of fine-grained access control policies which can be assigned to

guest VMs based on their intended purpose, further enhancing the isolation among them. However, the complexity associated with defining SELinux access policies leaves the security administrator prone to making mistakes in policy specifications. This can result in creating additional security vulnerabilities within the virtualization environment for attackers to compromise it.

IV. PROPOSED SOLUTION

Our proposed solution is based on three ideas: protected in-VM monitoring; mandatory access control (MAC)-based hypervisor protection; and the leveraging of the functionalities of existing virtualization platforms and management solutions. Figure 5 illustrates the conceptual diagram of our proposed solution.

A. Protected monitoring module

Inserted into the guest VM's kernel space on the creation of a guest VM, the security monitor is responsible for monitoring the operations within the guest VM. It performs two functions, namely, to periodically calculate the hash values of the guest VM's process table and to examine the system call requests. When the VM is first created, the security monitor calculates the hash value of the memory contents and stores it. It periodically calculates the hash value of the guest VM's process table and compares it with the stored value. The hash values are then passed to the control monitor, which compares it with previously obtained values. The access logs within the guest VM will also be examined for signs of authorized access. The results from these two analyses will indicate the presence of a malware attack. When a process running in the guest VM's user makes a system call request, the monitoring module intercepts the request before passing it to the kernel for execution. The parameters of the system call are examined to determine if it is making a valid request, based on a pre-defined list of critical processes (such as the shell console, netcat, etc). Its security context is then used to consult the pre-defined SELinux policies to determine its access privileges before granting the request. SELinux will be used to assign a unique security context for the module, allowing it to run in a confined domain and protecting it against malware compromise. The assigned security context will be shared only between the module and the control monitor running on the management VM, thus preventing its access even if the attacker has obtained root access to the guest operating system.

B. MAC-based hypervisor protection

While the guest VMs will be protected through protected in-VM monitoring, the security of the hypervisor will be enforced through mandatory access control. When a VM is created, it will be assigned with the access policies which reflect its intended purpose. Requests made by the guest VM will be granted by looking up the SELinux security policies defined in the hypervisor protection module running in the hypervisor. Any attempts by the guest VM to violate the assigned access policies will be notified to the control monitor running within the control VM.

V. PROTOTYPE AND TEST SCENARIOS

A. Prototype platform

The HPC (High Performance Computing) servers from our VOTER (Virtualisation Open Technology Research) lab are used in the exploration of the proposed virtualization security solution. Xen Cloud Platform is installed on each of the HPC nodes, with one node acting as the controlling node. Using Apache CloudStack, all these nodes have been combined into a resource pool enabling the guest VMs to migrate from one host to another. Figure 6 shows how the infrastructure is set up.

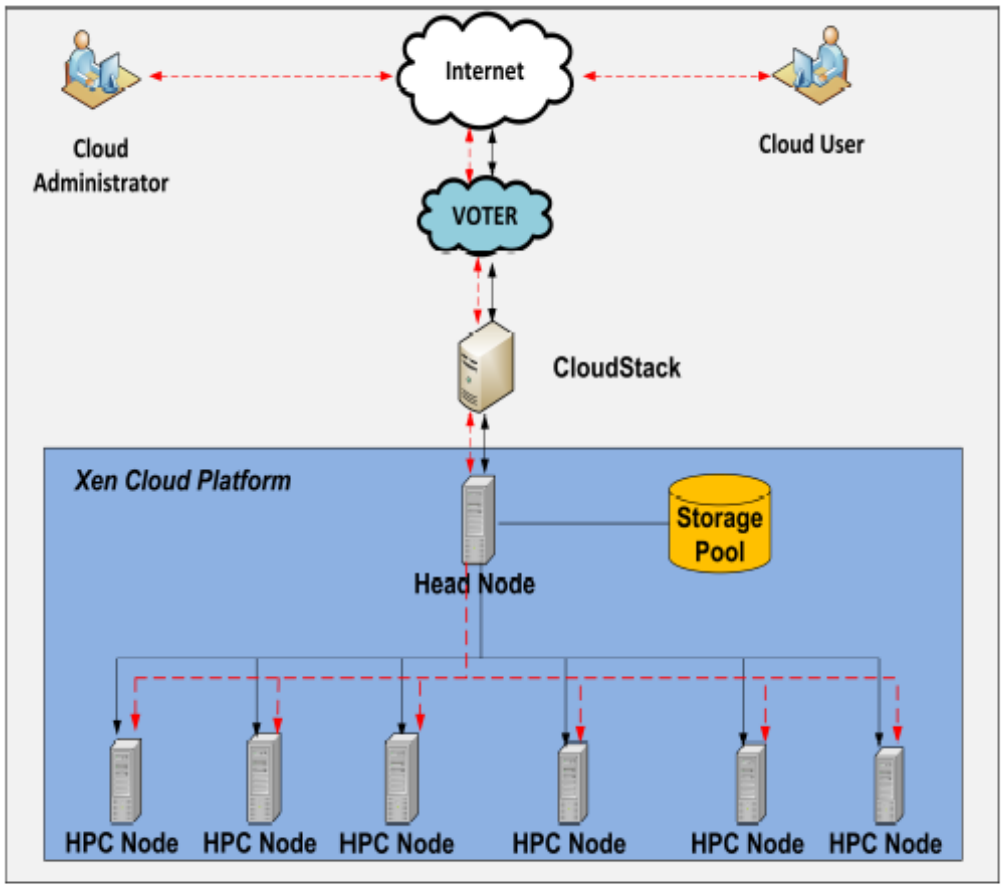

Figure 6. Prototype infrastructure setup

Monitoring of the guest VM internal state: In order to monitor the guest VM's internal state, a monitoring module is placed within its kernel space upon its creation. The module intercepts the system calls from processes and determining their security contexts.

The security context of the process making the request is determined by using the getcon function which is part of the Linux system programming application programming interface (API). The monitoring module is implemented using GNU C and the SELinux API.

Security context identification of guest VMs through SELinux: In order to protect the hypervisor, the security contexts of the guest VMs need to be assigned according to their usages within the system. This involves analyzing the kernel source code containing the Linux Security Module as well as security context assignment using SELinux together with LibVirt virtualization library.

B. Design of test scenarios

The proposed virtualization security solution will be evaluated in terms of its ability to detect abnormalities within the guest VM as well as its overall performance overhead incurred in the virtualization environment.
Measuring the effectiveness of proposed solution: The proposed solution will be evaluated in terms of its ability to protect both the virtual machines as well as the hypervisor against attacks. This will be done through subjecting it to penetration testing, as well as conducting a comparative study alongside alternative virtualization techniques.

Kali Linux is used to determine the ability of the proposed solution to detect attacks. The tool provides a collection of tools to simulate different forms of attacks and will be run against the guest VMs by remotely simulating attacks. The ability of the solution to detect the attacks as well as the time taken for the detection will be measured by analysing the timestamps in the guest VMs' log messages. The use of Kali Linux can be extended to determine the ability of the MAC-based hypervisor protection module to protect against attacks.This will help determine the ability of the propose solution to protect the virtualization environment against attacks.

Performance analysis using existing virtualization security techniques: In order to measure the additional performance overhead incurred by the proposed solution in a virtualization environment, a comparative performance analysis will be done by implementing two existing virtualization security techniques.

A virtual machine introspection (VMI)-based security solution based on the work done in [18] will be implemented, to compare the efficiency of the proposed solution in monitoring the guest VMs. Performance comparisons of these two approaches will give an insight into the performance overhead associated as well as their effectiveness.

The MAC-based virtualization technique proposed in [5] will be used to compare the effectiveness of the hypervisor protection module. While it is implemented on a proprietary IBM POWER hypervisor, the techniques mentioned in the work can be adapted to the XCP and KVM virtualization platforms. DBench and IPerf will be used for performance analysis.

VI. OUTLOOK

Our proposed solution has the strength of protected in-VM monitoring and at the same time leverages the Linux Security Module (LSM) using SELinux. Designed to overcome the limitations of existing techniques, our virtualization security solution proposed will protect guest VMs as well as hypervisors from attacks.

The future work will be using SELinux to define access control policies for guest VMs based on the *principle of least privilege*, and integrating it with the CloudStack management interface. In addition, the use of XML to convert guest VM access policies to enforce access control across VM migration will be explored as well.

REFERENCES

[1] M. Bishop, *Computer security: Art and Science*. Addison-Wesley, 2012, vol. 200.

[2] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," DTIC Document, Tech. Rep., 1973.

[3] J. P. Anderson, "Computer security technology planning study. volume 2," DTIC Document, Tech. Rep., 1972.

[4] ´U. Erlingsson, "The inlined reference monitor approach to security policy enforcement," Cornell University, Tech. Rep., 2003.

[5] E. Valdez, R. Sailer, and R. Perez, "Retrofitting the ibm power hypervisor to support mandatory access control," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 221–231.

[6] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. Van Doorn, J. L.Griffin, S. Berger, R. Sailer, E. Valdez, T. Jaeger *et al.*, "shype: Secure hypervisor approach to trusted virtualized systems," *Techn. Rep. RC23511*, 2005.

[7] J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer, "Shamon: A system for distributed mandatory access control," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 23–32.

[8] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Proceedings of the 11th USENIX Security Symposium*. USENIX Association, 2002, pp. 17–31.

[9] L. Tian, X. Rong, and T. Liu, “Design and implementation of linux file mandatory access control,” in *Network Computing and Information Security*. Springer, 2012, pp. 15–22.
[10] N. Li, Z. Mao, and H. Chen, “Usable mandatory integrity protection for operating systems,” in *Security and Privacy, 2007. SP’07. IEEE Symposium on*. IEEE, 2007, pp. 164–178.
[11] K. Harsha, B. M. Palavalli, S. Rao *et al.*, “Lothlorien: mandatory access control using linux security modules,” in *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*. IEEE, 2009, pp. 1–6.
[12] T. Isohara, K. Takemori, Y. Miyake, N. Qu, and A. Perrig, “Lsm-based secure system monitoring using kernel protection schemes,” in *Availability, Reliability, and Security, 2010. ARES’10 International Conference on*. IEEE, 2010, pp. 591–596.
[13] B. Hicks, S. Rueda, L. St Clair, T. Jaeger, and P. McDaniel, “A logical specification and analysis for selinux mls policy,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 3, p. 26, 2010.
[14] F. Mayer, D. Caplan, and K. MacMillan, *SELinux by example: using security enhanced Linux*. Pearson Education, 2006.
[15] G. Sala, D. Sgandurra, and F. Baiardi, “Security and integrity of a distributed file storage in a virtual environment,” in *Security in Storage Workshop, 2007. SISW’07. Fourth International IEEE*. IEEE, 2007, pp. 58–69.
[16] M. Blanc and J.-F. Lalande, “Mandatory access control for shared hpc clusters: Setup and performance evaluation,” in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE, 2010, pp. 291–298.
[17] X. Zhang, J.-P. Seifert, and R. Sandhu, “Security enforcement model for distributed usage control,” in *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC’08.IEEE International Conference on*. IEEE, 2008, pp. 10–18.
[18] T. Garfinkel, M. Rosenblum *et al.*, “A virtual machine introspection based architecture for intrusion detection.” in *NDSS*, vol. 3, 2003, pp. 191–206.
[19] X. Jiang, X. Wang, and D. Xu, “Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction,” in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 128–138.
[20] C. Harrison, D. Cook, R. McGraw, and J. Hamilton, “Constructing a cloud-based ids by merging vmi with fma,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 163–169.