A Layered Approach to Simplified Access Control in Virtualized Systems

Bryan D. Payne
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332 USA
bdpayne@cc.gatech.edu

Reiner Sailer
Secure Systems Department
IBM T.J. Watson Research Center
Hawthorne, NY 10532 USA
sailer@us.ibm.com

Ramón Cáceres
Secure Systems Department
IBM T.J. Watson Research Center
Hawthorne, NY 10532 USA
caceres@us.ibm.com

Ron Perez
Secure Systems Department
IBM T.J. Watson Research Center
Hawthorne, NY 10532 USA
ronpz@us.ibm.com

Wenke Lee
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332 USA
wenke@cc.gatech.edu

ABSTRACT

In this work, we show how the abstraction layer created by a hypervisor, or virtual machine monitor, can be leveraged to reduce the complexity of mandatory access control policies throughout the system. Policies governing access control decisions in today's systems are complex and monolithic. Achieving strong security guarantees often means restricting usability across the entire system, which is a primary reason why mandatory access controls are rarely deployed. Our architecture uses a hypervisor and multiple virtual machines to decompose policies into multiple layers. This simplifies the policies and their enforcement, while minimizing the overall impact of security on the system. We show that the overhead of decomposing system policies into distinct policies for each layer can be negligible. Our initial implementation confirms that such layering leads to simpler security policies and enforcement mechanisms as well as a more robust layered trusted computing base. We hope that this work serves to start a dialog regarding the use of mandatory access controls within a hypervisor for both increasing security and improving manageability.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Security

Keywords

Virtualization, mandatory access control, layering, information flow, security, policy

1. INTRODUCTION

Virtualization is becoming ubiquitous. It is widely used in corporate data centers to reduce the total cost of ownership of computer systems, and has many useful applications in other areas as well. Virtualization promises to improve system security because it can provide strong confinement of virtual machines (VMs). Since strong confinement is not a default feature in most hypervisors, we believe that strong confinement is best achieved by adding mandatory access controls (MAC) to hypervisor systems in order to control information flow between virtual machines.

However, unconditional confinement of virtual machines introduces its own issues. One such issue arises because many meaningful applications require communication between multiple entities [8]. For example, consider a web browser and an email client. People expect to click on links in their email client to open web pages and to click on email addresses in their web browser to start new emails. This interaction could not take place if each application was run in completely confined virtual machines. Pure confinement prevents applications from doing their jobs. Therefore, there must be controlled sharing of resources between virtual machines that still enables flexible information flow [38].

A second issue is the complexity of MAC security policies. Complexity makes it hard, if not impossible, to create a secure policy while still having a usable system. This issue is apparent in current systems such as Fedora Core 6, where SELinux [31, 25] MAC security policies either cover the whole system (strict policies) and break some system applications, or cover only a small part of the system (targeted policies) providing partial system security. In the case of partial system security, some applications are not being fully protected with MAC, potentially making it easier for an attacker to break into the system. This vulnerability is a result of the complexity involved in making a strict policy that does not break the system functionality. In these types of monolithic systems, complexity arises from the need to control large numbers of relationships between fine-grained subjects and objects, such as between processes and files,

even if only simple guarantees are demanded (for example, do not leak social security numbers).

In this paper we propose a layered approach to MAC security policies that addresses both of the above issues. We follow the same layering inherent in virtualized systems: hypervisor, operating system kernel, and applications. The policy at each layer – along with the related enforcement hooks – controls information flow at the semantic level of that layer. We disaggregate applications such that each virtual machine runs one component of an application. For example, in a web server application we run the HTTP front-end and the database back-end in separate virtual machines. This organization allows us to benefit from the natural confinement and communication paths within a virtualized environment. We emphasize that the confinement enabled through virtualization addresses storage and legitimate channels, but does not address covert channels [23]. Techniques for properly addressing covert channels are known [21, 33], and are beyond the scope of this paper.

Our approach is designed to reduce the complexity of security policies by reducing the number of subject-object relationships. By running application components in separate virtual machines, MAC at the hypervisor level focuses on controlling information flows between virtual machines. At the same time, MAC at the operating system level focuses on controlling information flows within a component. We feel that the different levels of policy acting in combination yield the same expressiveness as previous monolithic policy approaches.

This paper describes our work on the architecture and the key challenges in this space. We believe that virtualization will quickly become an indispensable component in modern systems and that our architecture offers a great opportunity to include strong security into virtualized environments from the beginning. Our initial implementation works with the Xen [5] and PHYP [3] hypervisors and shows the practicality and viability of this architecture. Throughout this paper, we identify the open problems and implementation questions with the goal of starting a dialog regarding the best approaches to these problems.

The rest of this paper is organized as follows. Section 2 presents our layered security architecture. Section 3 describes the role of each layer in more detail, including some sample applications of our architecture. Section 4 discusses how our approach could be evaluated relative to previous approaches. Finally, Section 5 puts our proposal in the context of related work.

2. ARCHITECTURE

To motivate our architecture, consider the web application shown at the top of Figure 1. Two web servers (labeled A and B) share a common corporate database (labeled C). Imagine that each web server provides information at different sensitivity levels. For example, one could serve the Human Resources Department and one could serve the rest of a company. To ensure that data intended only for Human Resources (for example, social security numbers) is not leaked through the other server, a guard [18] is placed between the web servers and database. This guard is a simple application that, in this example, validates requests to the database along with the data passed back to the web server.

A variety of techniques could be used to enable this application scenario. The critical point is to ensure that information flow is restricted such that the web servers can only communicate with the database by going through the guard. Besides virtualization, some other techniques available include using a chroot environment combined with MAC to confine each process, using a microkernel with MAC to control interprocess communication (IPC), or using physically separate machines with some technique to authenticate inter-machine communication.

Chroot environments would not be ideal for this situation. While it is possible to enable communication between processes in a chroot environment [9], the resulting trusted computing base (TCB) would encompass the entire operating system and there would only be a minimal reduction in MAC policy complexity. Microkernels offer an interesting alternative. The microkernel and its supporting processes would all be part of the TCB [40], and analogous to the hypervisor in our architecture. In addition, the software components (for example, web servers, database, and guard) would need to communicate using IPC. This would require MAC within the IPC mechanism and strong confinement between the processes address space. Our architecture could be implemented using a microkernel, but we choose to use a hypervisor for the stronger native isolation features and the readily available MAC within the hypervisor.

The most commonly deployed option today are architectures built using physically separate systems: one for each web server, one for the database, and one for the guard. This configuration has several drawbacks when compared to using a virtualized approach. On the one hand, virtualization provides a mechanism for validating MAC labels associated with communication paths between the VMs. This can be done in a trusted fashion, at bind time, without requiring a MAC implementation in the OS kernel. Separate systems can only securely label their communication paths (that is, networking) using IPSec and MAC in each OS kernel [20]. Alternative techniques such as firewalls and IP filtering are specific to low-level network architectures, susceptible to compromise when a service is attacked, and difficult to manage in a dynamic environment. On the other hand, virtualization provides more traditional benefits including increased system utilization, server consolidation, and flexibility in terms of system provisioning and migration. Our approach uses virtualization to decompose the architecture into layers to achieve a considerable simplification in manageability and policy, as well as a cost effectiveness that enables such architectures to be deployed broadly in open business environments.

Our architecture is comprised of three layers as shown in the bottom of Figure 1: the hypervisor, the OS kernels, and the applications. Each layer can have its own policy and each layer is protected from all higher layers. Therefore each layer has its own security labels such that the labels of higher layers refine access control information delivered by lower layers. This enables finer-grained yet more abstract access control decisions in higher layers. In this context, we

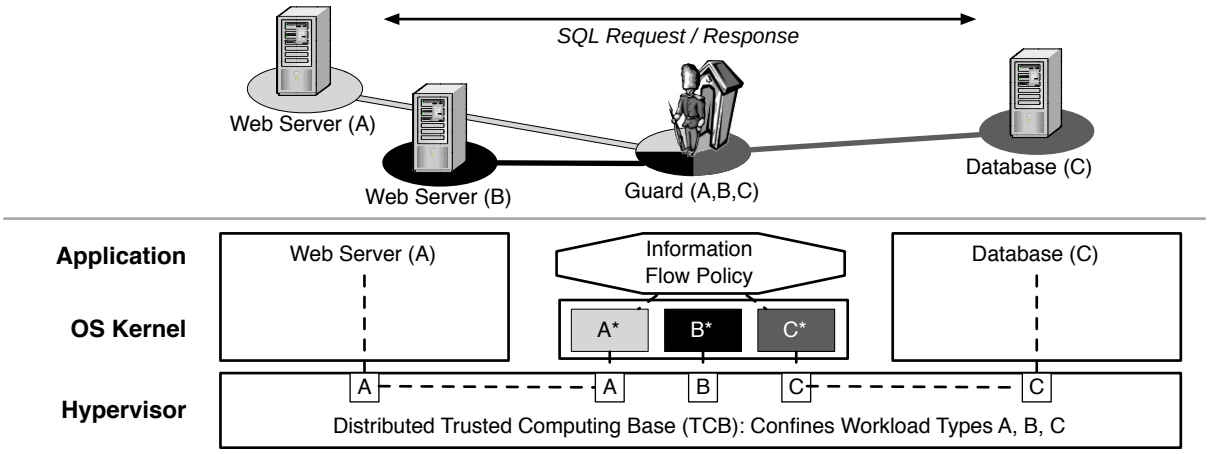

Figure 1: In this example, the guard controls SQL requests and responses passing between different web servers and a shared database. The architecture is divided into multiple layers. The lowest layer provides coarse-grained isolation of workload types A, B, and C, stored in sHype labels. Higher layers provide fine-grained filtering of requests and responses from A, B, and C inside the guard OS.

assign the following granularity to each layer:

- The **hypervisor** layer is responsible for securely multiplexing the hardware resources and controlling the information flow between VMs. Information flow is controlled by assigning labels to resources and VMs. The administrator can create simple type enforcement and Chinese Wall policies to express valid information flows and resource sharing [35]. The hypervisor is protected from higher layers using the hardware isolation capabilities in modern processors. In addition, multiple hypervisors can work together to enable distributed workload isolation [28].
- Any of the **OS kernels** may implement MAC (for example, SELinux). In the example above, the guard would require MAC to confine data received from the other virtual machines. The kernel is protected by the hypervisor layer from other OSs on the same platform and from its applications through hardware isolation capabilities in today's processors.
- When MAC is extended into an OS kernel, distinct processes at the **application** layer filter requests and responses. For example, the guard in Figure 1 provides filtering between processes handling OS labels A* and C* respectively. This filtering controls information flow on the granularity of SQL clauses in requests, and database records in responses.

These layers follow the traditional layers in a virtualized system. This allows for a natural one-to-one relationship between policies and layers. One policy is written for the hypervisor, one policy may be written for each OS kernel, and one policy may be written for each application. Each policy is able to stay with its respective layer during events that disrupt the static layered relationship (for example, VM migration). In this way, there is a significant benefit to maintaining separate policies for each layer. While the individual policies for each layer could be compiled into one large policy for the entire system, this would only increase the amount of code in the TCB (that is, the policy compiler) and would result in a single policy that is more difficult to audit and verify.

Leveraging layers of increasing abstraction is not a new concept and has proven beneficial in yielding manageable and efficient systems by dividing system tasks into different levels of abstraction (for example, the THE system [12] and the Venus system [24]). Here we use layering to manage the MAC policy complexity while limiting and compartmentalizing the code needed to ensure security.

Orthogonal to the layering, virtualization enables disaggregation of the applications by moving them into different virtual machines. This results in reduced policy complexity and maintains the ability to control the information flow using the hypervisor layer. Previous approaches would either put all the applications in one OS (resulting in higher policy complexity) or put them on different physical systems (limiting the ability to control information flow throughout the entire application between those systems).

Combining layering and disaggregation, our architecture can greatly simplify MAC. Furthermore, our architecture uses MAC to facilitate simple and meaningful information flow policies. This simplification comes from confining information flow between the applications where possible, and controlling the remaining information flows. Finally, a guard OS is used where fine-grained control is required.

3. MANAGING INFORMATION FLOW

Our approach is based on the realization that controlling inter-application communication within a single OS is different from controlling inter-VM communication. Consider the case of two applications running in a single OS. There are numerous ways these applications can communicate including IPC, networking, file access, shared memory, and more. Each of these communication paths must be controlled to protect the applications from each other and to control the

information flow between them. Looking at inter-VM communication, there are only two communication paths between VMs in Xen: event channels and shared memory. By working with a significant reduction in the number of communication paths, we can create an equivalent access control policy with greatly reduced complexity.

Our architecture could be implemented with any hypervisor and OS(s) that support MAC, but here we focus on the Xen hypervisor, with the Access Control Module (sHype) enabled to provide MAC in the hypervisor, and SELinux to provide MAC in the OS kernel. We are using these software components to build a prototype implementation of our architecture. While our implementation does not yet include all the features discussed in this paper, it is sufficient to highlight the challenges faced when implementing an application within our architecture. These challenges, along with our experiences from the implementation, are discussed throughout the remainder of this section.

For this discussion, we focus on the hypervisor layer and the OS kernel layer, and how they interact. We first define a few terms to simplify the following discussion. VMs providing confinement with OS-MAC are typically performing the role of a guard so we refer to these as *guard VMs*. We denote the labels assigned by the hypervisor as `hlabels` and the labels assigned by a guard VM as `glabels`.

Our implementation allows a hypervisor to assign an `hlabel` and one or more workload types to each VM and to each resource (for example, disk or network interface). VMs with more than one workload type in their `hlabel` are trusted by the hypervisor to provide data confinement between the workload types. This confinement may be provided using MAC or, when the VM is trusted, discretionary access control (DAC). These VMs may also be configured to allow controlled information flow between labels after comparing the information against a predefined policy. In this way, we use access control to define an information flow policy, as shown in Figure 2.

Using MAC with `hlabels`, our current implementation shows how resources can be safely shared between multiple workloads. Our configuration is similar to Figure 2, except the guard application provides disk access for multiple differently labeled VMs (for example, the web server and the database). The guard verifies that a given VM can access a disk resource at bind time by checking the VM and resource `hlabels` against the hypervisor policy. Once this access is permitted, no further checks are required within the guard. Our implementation uses similar techniques to control network communication between VMs.

Since guard VMs perform a single purpose, they can usually be designed to run from a small OS image (that is, no more than a few MB) such that the code can be evaluated, if desired [15]. Code evaluation [13, 1] in the guard VM would be required for high assurance applications, but is beyond the scope of this work. VMs with only one workload type in their `hlabel` do not require mandatory access control because that is only needed to provide confinement between different workload types. By reducing the necessity of OS-level confinement to a few, limited locations, this architecture reduces the administrative overhead required to properly set up and run these types of systems.

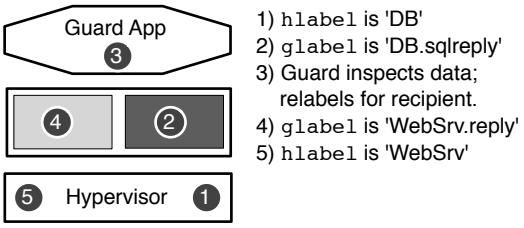

Figure 3: Labels are specific to a given policy, which is defined for each layer. Here we see the label transition as a SQL reply travels through a guard domain. The guard application can consider both the content in the message and its label in deciding whether to permit relabeling.

`Hlabels` and `glabels` are only meaningful in terms of the policy that defines them. Since we have different labels at each layer, we must also have different policies at each layer. For example, consider a system running sHype in the hypervisor and SELinux in a guard VM. Each of these mandatory access control systems has its own policy, its own labels, and its own subjects and objects. In other words, the semantics of the labels at each layer are different. Therefore, when data is passed between layers, special consideration must be given to ensure that the label is properly propagated across the layer boundaries.

When moving between layers, data can either go up (that is, hypervisor to VM) or down (that is, VM to hypervisor). When data moves up, it will need to be relabeled and the policy may require updating (for example, if a new label is created for the data). For example, our implementation converts a VM's `hlabel` to a `glabel` that is used to determine if the VM can connect to a given resource. In general, the data must be associated with a `glabel` in order for the OS-layer to properly protect it from higher layers. Since the OS layer trusts the hypervisor layer, it trusts the `hlabel` associated with the data. Therefore, the OS must maintain a mapping such that each `hlabel` can be mapped to exactly one `glabel`. When it receives the data, it changes the label according to the mapping so the data can be properly confined by the OS. The first time data associated with a particular `hlabel` moves up, no mapping will exist. In this case, a new `glabel` is created at run time. Figure 3 shows hows the label associated with a single message changes as it travels through a guard domain.

Dynamic label creation implies that the policy is also dynamic, which poses an interesting challenge in this space that we view as an open problem. Specifically, how does one provide the same security guarantees as today in a system such as SELinux, while allowing for dynamic label creation? We are developing a solution to this problem, which will allow our layered architecture to be deployed without requiring *a priori* coordination between the hypervisor-level and OS-level policies, therefore easing the administrative burden.

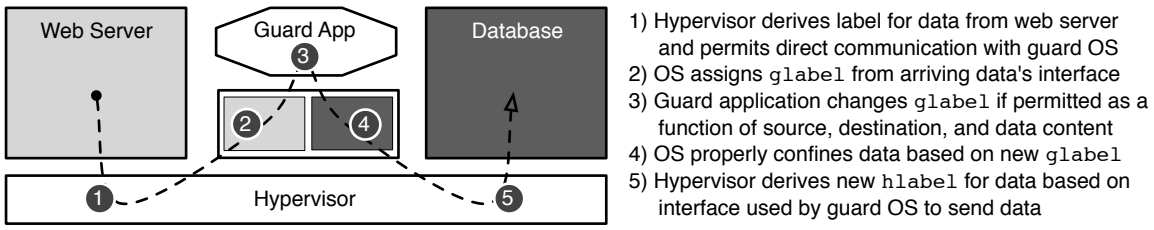

Figure 2: Information flow through the layered architecture for example communication from a web server to a database. Note that each step is very simple, but the end result is a fine-grained security policy.

Considering the other direction. When data passes from an OS into the hypervisor, it is the responsibility of the OS to associate the correct `hlabel` with this data. Using the same `hlabel` to `glabel` mapping described above, the OS will lookup the proper `hlabel` and pass the data to the hypervisor through a labeled path (e.g, a virtual network or virtual block device hypervisor interface that has the correct `hlabel` assigned). Since the VM will likely have finer-grained labels than the hypervisor, multiple `glabels` may map into a single `hlabel`. Since no new labels are created for data moving in this direction, there is no need to update the hypervisor policy as we did for the VM above.

This section described how MAC can be layered to reduce the overall policy complexity and manage information flows. We can use similar techniques to extend MAC into the application level to provide finer-grained control. Next, we describe four classes of applications that benefit from our architecture:

0-way Guard VM: A zero-way guard provides strict isolation of data. This is useful when you want to share a resource (for example, hard drive) between two or more VMs with different `hlabels`. While the hypervisor can securely share different disk partitions or disks between VMs, using a guard VM allows for safe sharing within a disk partition. In addition, a guard VM could be used to dynamically allocate disk space to each VM as needed, while properly scrubbing the disk when reallocating blocks between VMs.

1-way Guard VM: A 1-way guard only allows data to pass in one direction. This type of guard is useful for multi-level security (MLS) systems [7]. In MLS systems, data is implicitly allowed to pass from a lower level to a higher level without any checking. This could be setup directly with shared memory pages, ensuring that each VM has the appropriate privileges (that is, write-only for the low VM and read-only for the high VM). Alternatively, data passing from high to low must be inspected or sanitized to ensure that no sensitive data is leaked. This sanitization setup could take place within a guard VM.

2-way Guard VM: A 2-way guard is described as the example in Figure 1.

n-way Guard VM: An n-way guard contains groups of VMs with different `hlabels`. For example, consider a hosting service that manages computers from three competing companies. VMs for each company would be permitted unrestricted communication with other VMs from the same company (that is, with the same workload type in their `hlabel`). Communication with VMs from other companies could be restricted by the guard VM to a very narrow set of actions such as a subset of the HTTP protocol on TCP port 80.

4. DISCUSSION

The ultimate goal of our layered approach is to provide sound security protection. This, in turn, requires that our security architecture embodies sound security design decisions. While, in general, it is not possible to automatically generate a sound security architecture or mathematically verify the soundness of a security architecture, prior work has provided important design and evaluation criteria [36, 13, 1].

We argue that our layered approach measures very well with respect to the well-established design principles for protection mechanisms, which include economy of mechanism, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, and psychological acceptability. For example, our architecture facilitates open design by allowing one to add a layer or VM with its own security policy. And it facilitates separation of privilege by disaggregating the application components and controlling the inter-VM communication.

One key benefit of our architecture is reduced complexity. Consider an access control policy for n subjects and m objects. If the applications (subjects) run in the same VM, then the security mechanism must control $O(nm)$ potential interactions between subjects and objects. Figure 4(a) shows an example of this scenario, with three subjects (circles) and five objects (squares) in one operating system. By comparison, if we run each application in its own VM, and the hypervisor controls the information flow between the applications, then there are only $O(n+m)$ relationships that require control. This is derived by adding the connections from each VM to the hypervisor, $n+1$, plus the layering overhead, $\lambda(n+1)$, plus the connections to the OS objects, m. Note that λ is a constant that specifies the number of inter-VM communication mechanisms available within the hypervisor, which is 2 for Xen: event channels and shared memory. These connections are shown in Figure 4(b), where the VM to hypervisor connections are shown as vertical lines and the λ inter-VM communication mechanisms are shown as horizontal lines in the hypervisor. Reasonably sized applications would quickly benefit from the smaller number of overall connections even after accounting for the layering overhead. Complexity is reduced, in part, because the

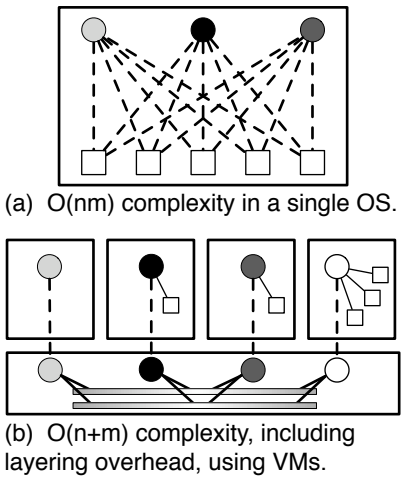

Figure 4: Our architecture (b) reduces the number of subject − object relationships, compared to a single OS (a).

application VMs leverage VM isolation and do not require MAC. Furthermore, we plan to formally verify that an access control model on our layered security architecture is at least as expressive as one on a single operating system [10, 41].

Reduced complexity leads to economy of mechanism, and more subtly but perhaps more importantly, better usability and, therefore, psychological acceptability. This follows from the fact that, when running an application on our layered architecture, an application programmer or a system administrator needs to specify a simpler security policy (for example, fewer number of entries in an access control matrix).

Another important factor in psychological acceptability is the overall system performance when a new security mechanism is incorporated. Any implementation of an application using our architecture should be evaluated for performance against the same type of system in a traditional single OS architecture. To this end, we need to first evaluate both setups without any security enabled. This would allow us to identify the overhead from security independently from the overhead from virtualization. While we do not have the full implementation and evaluation, our performance numbers for the Xen hypervisor are very encouraging. Our initial experience is that the overhead due to virtualization, while currently still varying widely, will be small (5-10%) when virtualization hardware support is fully leveraged. Using our architecture will introduce next to no (<1%) additional overhead on Xen. The security overhead performance numbers reported here are from sHype on Xen measuring inter-VM communication overhead using Xen-Oprofiling (Xenoprof [29]) and Flexible File System benchmarks (FFSB [2]).

Our initial experiences suggest that this architecture provides a viable technique for reducing the complexity of MAC policies with minimal performance impact. However, a complete performance and security evaluation is still needed to fully understand how this approach compares with existing techniques. As discussed above, the performance impact of virtualization is generally understood and adding MAC to the hypervisor has a minimal impact on performance. The open questions with regards to performance include measuring the overhead from disaggregation and the overhead from sending data through the guard. Each of these overheads should be compared against a system without any security and a comparable system that uses a single, monolithic MAC policy. For the security evaluation, the critical question is if our architecture is as expressive as an architecture that uses a monolithic MAC policy. We designed our architecture to maintain the MAC policy expressiveness by allowing MAC at each layer. We can still perform fine-grained control within a single VM while performing corse-grained control on information flow between VMs. Evaluating the expressiveness of this setup will require building a formal model of the types of control that are possible with both monolithic and layered MAC policies, and then showing that each model is equally expressive.

5. RELATED WORK

The primary motivation for this work is to reduce the complexity associated with controlling information flow between two or more software components. Existing techniques could implement such a system using multiple, physically separate machines. However, these architectures would rely heavily on the general use of MAC in each operating system (for example, SELinux). A side effect of providing this type of fine-grained access control entirely at the OS layer is that the associated policies are very complex [19]. Policy complexity has become an important topic within the MAC research community because this complexity has slowed the adoption of MAC-based systems [32, 4].

Our work represents a combination of related work from a variety of research areas including layered system architectures, the integration of virtualization and security, and the relationship between software complexity and security. Our architecture shares some goals with other efforts aimed at reducing overall system complexity and reducing the size of the TCB. The remainder of this section will provide details on these related areas of work.

Security and complexity are clearly at odds with each other. Saltzer and Schroeder identified several design principles for secure systems including economy of mechanism which they describe as "keep the design as simple and small as possible" [36]. Confirming the importance of this design principle, Basili and Perricone performed an empirical study showing that software has about 9.7 errors per 1000 lines of executable code [6]. Complex and large software architectures will lead to more errors, and each error is another possible security problem. More recently, Schneier has argued that "the future of computers is complexity, and complexity is anathema to security" [39]. Given the conflict between security and complexity, and given the complexity of MAC policies in systems such as SELinux, there is clearly a need to explore new architectures that will enable the benefits of MAC with reduced complexity.

Layering is a common technique used to reduce complexity. Layering allows one to divide a large and complex system into manageable components that each depend only on lower

level components. Layering was initially seen in systems such as THE [12] and Venus [24]. Today, it is a fundamental design principle for operating systems and any complex software architectures. Introducing layering to MAC policies is a logical next step so that the policies can enjoy the same reduced complexity seen by the system software.

The recent resurgence of virtualization can be attributed, in part, to the additional layer it provides and how this can be leveraged to reduce the complexity of systems management. However, one obstacle to the adoption of virtualization is the requirement to retain a level of isolation similar to what has been achieved via discrete physical systems. Achieving this level of isolation through logical separation in a virtualized environment has been studied and documented in the literature for over three decades. Madnick and Donovan showed in the early 1970's that virtualized platforms provide distinct security benefits [27]. Similarly, layered systems architectures and security models have an even longer lineage [12, 24, 26]. As virtualization has become more accessible in recent years, secure systems solutions have increasingly relied on these benefits [17, 16, 11, 30]. In addition, the protection offered by a virtualized environment has been used to protect malicious software [22, 34].

Significant research has been done in order to make virtualization platforms more suitable for security-oriented applications. Research efforts to integrate security into the hypervisor arguably reached a zenith with the KVM / 370 high assurance hypervisor project [37], and the VAX VMM Security Kernel [21], and is beginning to see a resurgence with efforts such as the sHype architecture [35], which is now integrated into the Xen open source hypervisor [5].

Virtualization is not the only construct available for reducing complexity and reducing the TCB size. Singaravelu and colleagues used the Nizza architecture as a TCB, which is based on the L4 microkernel [40]. In this system, secure applications are created by extracting a small component, referred to as an AppCore, that contains all the security-critical functionality. Legacy applications can communicate with an AppCore using L4's IPC mechanisms. The primary focus of the Nizza / AppCore architecture work is to isolate the trusted components in an effort to reduce the TCB size, and there is no discussion of how such a system could be made to work with MAC. The Asbestos Operating System by Efstathopoulos and colleagues takes a different approach by providing a new construct that controls information flow using dynamic labels and fine-grained isolation contexts [14]. While these projects do share similarities with our architecture, the goals and end results are different. Our focus is on reducing the MAC policy complexity using readily available virtualization software. Singaravelu and colleagues are focused on reducing the TCB size and Asbestos is an entirely new system with a much broader scope.

6. CONCLUSIONS

We introduced a layered access control architecture and showed how it simplifies the construction of mandatory access control policies. Our examples illustrate how these policies can be layered to provide information flow protections in diverse classes of applications. Our experiences with the initial implementation verified many of our claims. Continuing this research, the next steps are to extend our implementation to support strong confinement and fine-grained sharing for the multi-tier web-service application and perform a thorough evaluation of the architecture using this software.

7. REFERENCES

[1] Common criteria for information technology security evaluation version 2.1. `http://www.commoncriteria.org/docs/index.html`, 1999.

[2] Flexible file system benchmark (FFSB) version 5.1. `http://sourceforge.net/projects/ffsb`, 2006.

[3] W. J. Armstrong, R. L. Arndt, D. C. Boutcher, R. G. Kovacs, D. Larson, K. A. Lucke, N. Nayar, and R. C. Swanberg. Advanced virtualization capabilities of POWER5 systems. *IBM Journal of Research and Development*, 49(4/5), 2005.

[4] J. Athey, C. Ashworth, F. Mayer, and D. Miner. Towards intuitive tools for managing SELinux: Hiding the details but retaining the power. In *Proceedings of the 2007 Security Enhanced Linux Symposium*, March 2007.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Symposium on Operating System Principles*, October 2003.

[6] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42 – 52, January 1984.

[7] D. E. Bell and L. J. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, 1976.

[8] S. Bellovin. Virtual machines, virtual security. *Communications of the ACM*, 49(10), October 2006.

[9] A. Bennett. Hole-in-the-chroot. `http://clyde.concordia.ca/security/hole-in-the-chroot-v1/`.

[10] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 41–52, New York, NY, USA, 2001. ACM Press.

[11] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.

[12] E. W. Dijkstra. The structure of the "THE"-multiprogramming system. *Communications of the ACM*, 2(5):341 – 346, November 1968.

[13] DoD. Trusted computer system evaluation criteria. Technical Report DoD 5200.28-STD, Department of Defense, 1985.

[14] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. Labels and event processes in the asbestos operating system. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, October 2005.

[15] R. J. Feiertag and P. G. Neumann. The foundations of a provably secure operating system (PSOS). In *Proceedings of the National Computer Conference*, pages 329 – 334, Menlo Park, CA, 1979.

[16] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[17] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium*, February 2003.

[18] T. J. Gibson. An architecture for flexible, high assurance, multi-security domain networks. In *Network and Distributed System Security Symposium*, San Diego, CA, February 2001.

[19] T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the SELinux example policy. In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[20] T. R. Jaeger, S. Hallyn, and J. Latten. Leveraging IPsec for mandatory access control of linux network commmunications. In *Proceedings of ACSAC*, 2005.

[21] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A retrospective on the VAX VMM security kernel. *IEEE Transactions on Software Engineering*, 17(11):1147 – 1165, November 1991.

[22] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. Subvirt: Implementing malware with virtual machines. In *IEEE Symposium on Security and Privacy*, 2006.

[23] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613 – 615, October 1973.

[24] B. Liskov. The design of the venus operating system. *Communications of the ACM*, 15(3):144 – 149, March 1972.

[25] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)*, June 2001.

[26] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, June 1990.

[27] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the Workshop on Virtual Computer Systems*, pages 210 – 224, March 1973.

[28] J. M. McCune, S. Berger, R. Caceres, T. Jaeger, and R. Sailer. Shamon – a system for distributed mandatory access control. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[29] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *ACM/USENIX 1st International Conference on Virtual Execution Enviroinments*, pages 13–23, 2005.

[30] R. Meushaw and D. Simard. Nettop: A network on your desktop. *Tech Trend Notes (National Security Agency)*, 9(4):3 – 11, Fall 2000.

[31] National Security Agency. Security-Enhanced Linux. `http://www.nsa.gov/selinux/`.

[32] C. J. PeBenito, F. Mayer, and K. MacMillan. Reference policy for security enhanced linux. In *Proceedings of the 2006 Security Enhanced Linux Symposium*, March 2006.

[33] N. E. Proctor and P. G. Neumann. Architectural implications of covert channels. In *Proceedings of the 15th National Computer Security Conference*, pages 28 – 43, Baltimore, Maryland, 1992.

[34] J. Rutkowska. Subverting Vista kernel for fun and profit. In *Proceedings of Black Hat USA 2006*, 2006.

[35] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn. Building a MAC-based Security Architecture for the Xen Opensource Hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, December 2005.

[36] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Communications of the ACM*, 17(7), July 1974.

[37] M. Schaefer, B. Gold, R. Linde, and J. Scheid. Program confinement in KVM/370. In *Proceedings of the 1977 ACM Annual Conference*, pages 404 – 410, October 1977.

[38] G. Schellhorn, W. Reif, A. Schairer, P. A. Karger, V. Austel, and D. Toll. Verification of a formal security model for multiapplicative smart cards. In *ESORICS 2000*, 2000.

[39] B. Schneier. The process of security. *Information Security Magazine*, April, 2000.

[40] L. Singaravelu, C. Pu, H. Hartig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications: Three case studies. In *Proceedings of the 1st EuroSys*, April 2006.

[41] M. V. Tripunitara and N. Li. Comparing the expressive power of access control models. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 62 – 71, Washington, DC, 2004.