Transforming Commodity Security Policies to Enforce Clark-Wilson Integrity

Divya Muthukumaran
Pennsylvania State University
muthukum@cse.psu.edu

Sandra Rueda
Universidad de los Andes
sarueda@uniandes.edu.co

Nirupama Talele
Pennsylvania State University
nrt123@psu.edu

Hayawardh Vijayakumar
Pennsylvania State University
hvijay@cse.psu.edu

Jason Teutsch
Pennsylvania State University
teutsch@cse.psu.edu

Trent Jaeger
Pennsylvania State University
tjaeger@cse.psu.edu

ABSTRACT

Modern distributed systems are composed from several off-the-shelf components, including operating systems, virtualization infrastructure, and application packages, upon which some custom application software (e.g., web application) is often deployed. While several commodity systems now include mandatory access control (MAC) enforcement to protect the individual components, the complexity of such MAC policies and the myriad of possible interactions among individual hosts in distributed systems makes it difficult to identify the attack paths available to adversaries. As a result, security practitioners *react* to vulnerabilities as adversaries uncover them, rather than *proactively protecting* the system's data integrity. In this paper, we develop a mostly-automated method to transform a set of commodity MAC policies into a system-wide policy that proactively protects system integrity, approximating the Clark-Wilson integrity model. The method uses the insights from the Clark-Wilson model, which requires integrity verification of security-critical data and mediation at program entrypoints, to extend existing MAC policies with the proactive mediation necessary to protect system integrity. We demonstrate the practicality of producing Clark-Wilson policies for distributed systems on a web application running on virtualized Ubuntu SELinux hosts, where our method finds: (1) that only 27 additional entrypoint mediators are sufficient to mediate the threats of remote adversaries over the entire distributed system and (2) and only 20 additional local threats require mediation to approximate Clark-Wilson integrity comprehensively. As a result, available security policies can be used as a foundation for proactive integrity protection from both local and remote threats.

1. INTRODUCTION

A large fraction of modern computation is now deployed in distributed systems consisting of several, independently-developed software components. For example, web applications (e.g., a LAMP software bundle) consist of: (1) an operating system distribution and its system services (e.g., Linux); (2) a web server (e.g., Apache); (3) a database and other backend software (e.g., MySQL), and (4) custom server code (e.g., written in PHP) to which web clients connect to perform a variety of critical applications. Each of these components face their own threats and connecting them together into a distributed system only increases the avenues that adversaries can leverage to compromise the system.

Computing system compromises occur because *data integrity* is not managed effectively. Adversaries use the open accessibility to many distributed systems to attempt attacks ranging from malformed network packets to embedded executable content to imported files containing malware. One mechanism introduced into commodity systems to combat such attacks is *mandatory access control* (MAC) [33, 49, 51, 54, 27]. MAC enforcement limits processes to program-specific permissions to protect the kernel's integrity, even from some root processes[1]. MAC enforcement is now available in virtual machine monitors [9, 42, 22] (VMMs) and user-level programs [29], in addition to operating systems, enabling such control throughout the system. Further, such MAC enforcement is now integrated with network access control [26, 17, 32], presenting an opportunity for comprehensive access control in commercial deployments. However, the addition of all this enforcement does not seem to be changing the dynamics of security management. Preventing compromises is still a reactive task, fixing vulnerabilities as adversaries identify them.

We find that the current approach to securing systems using MAC enforcement forces administrators to be reactive. Commodity MAC policies often consist of many complex policy rules, so configuring MAC policies is now a task undertaken only by experts. As a result, administrators use the default MAC policies provided in OS distributions to protect their distributed systems blindly. However, commodity system MAC policies are designed based on the expected functionality required by processes, resulting in *least privilege* enforcement [43]. Unfortunately, almost every process in a commodity system is accessible to adversaries, even in its least privilege operation [53], so current commodity MAC

ACSAC '12 Dec. 3-7, 2012, Orlando, Florida USA
Copyright 2012 ACM 978-1-4503-1312-4/12/12 ...$15.00.

[1] A small set of programs are authorized to modify MAC policies in practice, but not all root processes as was the case previously.

policies do not protect process integrity comprehensively. While researchers have developed analysis tools to detect potential problems in commodity MAC policies [18, 52, 45, 55, 7], determining the resolutions to such problems is still a complex manual task, requiring MAC policy expertise. The result is that it is not practical for administrators to foresee all the possible attack paths available to adversaries, causing them to react to adversary exploits.

Instead, our goal is to generate MAC policies that protect system-wide integrity from the available security policies. Our insights are motivated by the Clark-Wilson integrity model [8]. The Clark-Wilson integrity model defines strict requirements for protecting integrity where *integrity verification procedures* (IVPs) validate high integrity data, only certified *transformation procedures* (TPs) modify high integrity data, and TPs protect themselves by upgrading or discarding any low integrity inputs they may receive. While MAC policies define the flows to and from processes, they fail to identify the data whose integrity is critical to those processes (e.g., require IVPs) or which process entrypoints[2] must protect themselves from adversary access. However, by using the available MAC policies, we find that we can construct an information flow problem that enables computation of placements of integrity verification and entrypoint mediation necessary to resolve all information flow integrity errors in the system.

In this paper, we develop a mostly-automated approach to compute the set of integrity verification and entrypoint mediation sufficient to protect information flow integrity in distributed systems. First, this method constructs system-wide data flow graphs from available MAC policies and "connection" policies (e.g., firewall policies) that describe how individual software components communicate in distributed systems. Second, we create information flow problems by adding integrity semantics to nodes in the data flow graph semi-automatically. Third, we compute a minimal placement of integrity verification and entrypoint mediation necessary to resolve all information flow errors automatically. Using this placement, we can produce a system-wide data integrity policy that approximates Clark-Wilson integrity as described above. We have found [41] that the resultant policies are equivalent to Decentralized Information Flow Control policies [20], possibly opening the way to leverage information flow-based enforcement mechanisms for commodity system deployments.

This paper makes the following contributions. First, we define an information flow problem whose data flow graph represents an hierarchical system of software components with independent entry and exitpoints. This enables us to model information flow system-wide and identify the need for mediation at program entrypoints. Second, we design a method to solve such problems using graph cuts, where the locations of possible cuts are constrained by the integrity of program packages. Third, we demonstrate our method on a custom web application running on virtualized Ubuntu SELinux hosts, which finds that only 27 additional entrypoint mediators and 20 integrity verification procedures are necessary to provide information flow integrity approximating Clark-Wilson integrity. This is the first approach to produce system-wide access control policies targeted to classical integrity in a mostly-automated fashion.

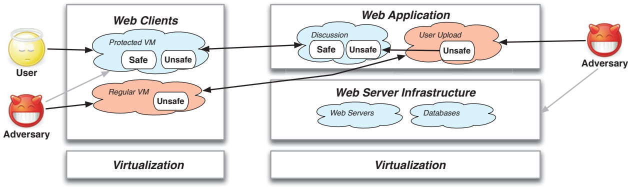

Figure 1: Example web application consisting of a web server with three web application components that handle data of different integrity levels on a standard web infrastructure. The web client is also divided into two VMs for regular and protected operation.

The remainder of the paper is structured as follows. In Section 2, we motivate the problem of deploying a secure web application using commodity MAC policies. In Section 3, we motivate the use of Clark-Wilson integrity as a guide for resolving information flow errors. In Section 4, we describe how to construct and solve the information flow problems necessary to compute an approximation of a Clark-Wilson integrity policy system-wide. In Section 5, we describe the tool implementation and evaluation results. In Sections 6 and 7, we detail related work and conclude the paper.

2. PROBLEM DEFINITION

2.1 Motivation

When administrators deploy distributed applications, one problem that they must address is whether that deployment protects the application's data integrity end-to-end. As an example, consider a web application that enables collaborative decision-support for groups of users shown in Figure 1. Such applications consist of two main tasks: data gathering (upload) and data analysis for decision-making (discussion). First, data is often gathered from external, perhaps untrustworthy, sources by some clients for others to evaluate. In this example, users use the regular web clients to gather data for upload to the *upload* web application component. Then, users use the protected client to examine this data and perform decision-making (with other clients) using the *discussion* web application component. To do this, the discussion web application component produces two streams of content, one (potentially) untrusted stream from external sources (from upload) and a second, high integrity stream from collaborative discussions of trusted sources, where both streams may be displayed in a manner suitable for users to distinguish one content stream from the other [13].

While there are multitude of threats that are possible against such applications, we identify three specific threats that represent the different types of problems we aim to address. First, the web infrastructure (e.g., LAMP software bundle including web server, database, etc.) may be threatened by remote adversaries. While the web infrastructure often consists of mature software that has been hardened against many threats as the result of years of penetrate-and-patch, new vulnerabilities continue to be discovered. Second, we have to be careful that a web application deployment does not introduce new attack paths for itself and the infrastructure. The web application components are gener-

[2] A program entrypoint is a program instruction that receives input from the operating system, such as the caller of the library function that invoke a system call.

ally custom software implemented for the specific web application, so they are not necessarily hardened to the threats that they may face. Remote threats that compromise one web application component may be used as a stepping to attack other web applications and the web infrastructure. Third, local threats may also be used to attack the web application. Some vulnerabilities are caused by trusting files that may have been supplied by adversaries by tricking users or compromised programs to import them. For example, a user may be tricked into importing malicious libaries to their home directory, which may lead to a untrusted search path vulnerability.

As a result, it is difficult to deploy web applications that protect data integrity end-to-end. Administrators currently focus their efforts on management of the web infrastructure, which consists of mature programs, runs on hosts they manage directly, and is well-supported by vulnerability reporting. Administrators assume that web infrastructure software prevents known threats, and they can track vulnerability reports to determine whether new vulnerabilities will require upgrades or software package changes. This is not the case for web application components and web client software, however. Such programs may be new, be managed at least partially by end users, and be sufficiently ad hoc that it is difficult to determine how a new vulnerability may affect them. That is, the reactive approach may not be effective because only their configuration may have the vulnerability and the method of reaction may differ between web application configurations. To address this limitation, we aim to develop a proactive approach to protecting data integrity.

2.2 Proactive Integrity

Designing for integrity traditionally requires finding solutions to an information flow problem that satisfies an integrity policy. For example, Biba integrity requires that no process receive any information flow (read or execute) containing data whose integrity level is lower than that of the process [5]. The information flow problem can be expressed using the model below:

Definition 2.1. *Let $\mathcal{I}$ be an* information flow problem, *$\mathcal{I} = (\mathcal{G}, \mathcal{L}, \mathcal{M})$, to find whether whether a data flow graph G with a level mapping function M for a lattice $\mathcal{L}$ contains any information flow errors, where:*

1. *A* data flow graph *$G = (V, E)$ consists of a set of nodes V connected by a set of directed edges E.*

2. *There is a* lattice *$\mathcal{L} = \{L, \preceq\}$. For any two levels $l_i, l_j \in L$, $l_i \preceq l_j$ means that l_i 'can flow to' l_j.*

3. *There is a* level mapping function *$M : V \to \mathcal{P}^L$ where $\mathcal{P}^L$ is the power set of L (i.e., each node is mapped either to a set of levels in L or to $\emptyset$).*

4. *The lattice imposes security constraints on the information flows enabled by the data flow graph. Each pair $u, v \in V s.t.$ $[u \hookrightarrow_G v \wedge (\exists l_u \in M(u), l_v \in M(v).\ l_u \not\preceq_{\mathcal{L}} l_v)]$, where $\hookrightarrow_G$ means there is a path from u to v in G, represents an* information flow error.

It has been shown that information flow errors in programs [28] and MAC policies [18, 45, 52] can be found automatically using such a model. For integrity, the lattice would represent an integrity policy, and the level mapping function would map levels in that integrity lattice to subjects and objects in the system.

Through the addition of a variety of security mechanisms over the last 10 years, administrators can now make a number of choices to protect the integrity of their deployments proactively, but the sum of these measures fall short of satisfying information flow integrity. For example, administrators can: (1) configure firewall policies, which define the possible data flows among hosts (including virtual hosts); (2) choose OS distributions with mandatory access control (MAC) policies (supported in several commodity systems [51, 33, 31, 27, 54]), which define the possible data flows among processes running on that OS; and (3) choose software packages to run on their hosts, which defines the possible data flows within programs. The problem is that many of these authorized data flows allow adversaries to access integrity-critical data and processing. The reason for this is that such policies are designed to enforce *least privilege* [43], where programs are only limited to the permissions required for them to function properly. However, in a runtime analysis study, we found that nearly every program is designed to receive some adversary-supplied data [53], which can lead to vulnerabilities when such untrusted data is accessed in unexpected ways or has unexpected values.

Researchers have long recognized this problem, but thusfar the solutions proposed involve significant, manual effort. In general, information flow errors may be resolved by *mediators*, which ensure that the runtime behavior of the system is consistent with its security requirements (i.e., integrity lattice). Mediators may be implemented as entire processes, such as guards, or as individual program statements, such as endorsers in security-typed languages [29]. In the information flow problem, a mediator associates an edge $(u, v) \in E$ with an integrity level to which data is raised $l \in L$ when transmitted on the edge by node u.

As system functionality often violates information flow integrity, the placement of mediators necessary to resolve such errors is an important and difficult problem. Recently, researchers have proposed a variety of approaches to enforce information flow integrity focusing on system abstractions [21, 47, 50, 20, 56, 57]. All of the above approaches require that administrators replace the commodity system policies with a new information flow policy that includes mediators. Only Practical Proactive Integrity [50] (PPI) provides some automated support for producing integrity policies from existing MAC policies, but it uses simple, two-level lattices and requires administrators to determine whether a process can handle all integrity threats, which is a difficult and error-prone task. To evaluate such risks more precisely, researchers have realized that it is important to identify and defend those program entrypoints accessible to adversaries[3], called the program's *attack surface* [16, 23]. Some of the information flow integrity enforcement mechanisms above reason about integrity based on the program entrypoints [47, 20]. See Section 6 for further details.

Rather than requiring administrators to specify a new information flow policies manually, we argue that such policies, including the mediators necessary to prevent information flow errors, can be computed automatically. Our goal is to use the system's available security policies to produce a system-wide policy with the minimal mediation necessary

[3] A program entrypoint is a program instruction that receives input from the operating system.

to resolve the system's information flow errors as defined in Definition 2.1. We are motivated by prior work that demonstrated that a placement of mediators that resolves all information flow errors is equivalent to a cut of error paths in the data flow graph [37, 19]. However, to make this idea practical, we explore how to produce an information flow policy for the example web application that resolves its local and remote threats. As a result, we find several additional challenges must be addressed, such as finding practical mediator options and producing cuts for general lattice policies. Thus, this work is the first to produce mediator placements necessary to resolve information flow errors system-wide. In a separate technical report [41], we prove that solutions to the information flow problem above are equivalent to legal Decentralized Information Flow Control policies in the Flume model [20], possibly opening the way to leverage information flow-based enforcement for commodity system deployments.

3. SOLUTION APPROACH

Our approach is motivated by two insights that enable the solution of information flow problems from the available security policies in commodity system deployments.

Clark-Wilson Mediators. As a guide for where to place mediators, we turn to the Clark-Wilson integrity model [8], which consists of rules that define the high integrity operation of a system. Of particular interest are the rules that define how high integrity data is processed securely, where: (1) high integrity data (CDIs) must satisfy *integrity verification procedures* (IVPs) (Clark-Wilson rule C1); (2) only approved programs called *transformation procedures* (TPs) may modify high integrity data (C2, E1); and (3) TPs may only receive low integrity data (UDIs) if that data is upgraded or discarded (C5). That is, TPs protect data integrity, but they require IVPs to validate that the data an application depends upon is high integrity and TPs must be capable of mediating low integrity inputs to upgrade or discard such data.

Clark-Wilson identifies two types of mediators: IVPs and the TP program entrypoints. In practice, we find that IVPs are useful for mediating the integrity of files imported into the system. In our example, the protected web client may include restrictions on the files introduced into the user's directory, which acts as a mediator to protect the processes that use such files. Examples of possible IVP mediation include signed package files, binding files to their values at installation [46], and policy-sealed data [44]. If the network inputs and/or local files cannot be validated by IVPs, then entrypoint mediation is placed to protect processes. In general, entrypoint mediation is application-specific, although researchers have proposed general methods to prevent some attacks, such as input handling libraries [39] and safe name resolution [6]. In this paper, we produce an information flow policy that places mediators sufficient to protect the system's data integrity, but not the mediation code. Providing effective mediation code remains an open research issue for all the current information flow enforcement approaches [20, 29], but we are the first to design a method for system-wide placement of mediation.

Choosing Practical Mediation. Computing mediation placements solely from the information flow problem alone ignores some practical considerations. First, several programs, particularly mature ones, have already been hardened by the reactive approach described earlier. We want to reuse those mediators in choosing placements. Second, mediator placement locations may be limited in their ability to solve information flow problems. For example, the web application may need to filter untrusted input data, but it cannot be trusted to protect the kernel's integrity. In past work, these considerations are left as manual tasks.

To address these practical problems, we compute: (1) the mediator placements for infrastructure components to enable their reuse in deployments and (2) the constraints to limit the maximum integrity level that each mediator may endorse. First, we break the task of computing mediation into two steps: we first compute the mediators required for a default install of the infrastructure VMs to estimate the mediation that is necessary for any deployment, and then we compute the additional mediators required for the deployed application at large on this infrastructure. As described above, the movement toward pre-configured VMs encourages their reuse, so we further encourage the reuse of what should be their fundamental mediation, required in all deployments. Second, we compute constraints on the maximum integrity level for each possible mediator location for input to the mediator placement method. Producing these constraints is based on prior work that creates mutual-integrity partitions of labels from MAC policies [53].

These insights can produce a variety of positive effects on computing mediator placements system-wide. Using Clark-Wilson integrity as a guide enables mediation placement for programs that do not enforce MAC policies explicitly, reduces the number of mediation locations to consider significantly, and provides defense for both local and remote threats. By estimating infrastructure mediation and limiting the scope of mediation, we distinguish between the mediation sufficient to protect infrastructure in general and that additional mediation required when those infrastructure components are combined with application software and data and connected into a distributed system. By constraining the integrity levels to which mediators may raise data, we limit the number of programs that can endorse data at each integrity level. We evaluate each of these possible effects in Section 5.

Assumptions. The key assumption in this work is that the programs, operating systems, and firewalls that enforce MAC policies do so correctly. This is a significant assumption given the size and complexity of such software, but it is the standard assumption in modern computing systems. Specifically, we assume that the programs, operating systems, and firewalls satisfy the *reference monitor concept* [2], which requires that a reference validation mechanism (i.e., MAC enforcement) must "must always be invoked" upon a security-sensitive operation, "must be tamperproof," and must be "small enough to be subject to analysis and tests, the completeness of which can be assured," which implies correctness. The reference monitor concept is certainly the goal of these MAC-enforcing commodity systems, even if they do not meet the letter of these requirements.

4. DESIGN

To produce a mediator placement system-wide, we need to construct a system-wide information flow problem (see Definition 2.1) and compute a placement that resolves all the information flow errors in that problem. Given available security policies, we claim that building system-wide infor-

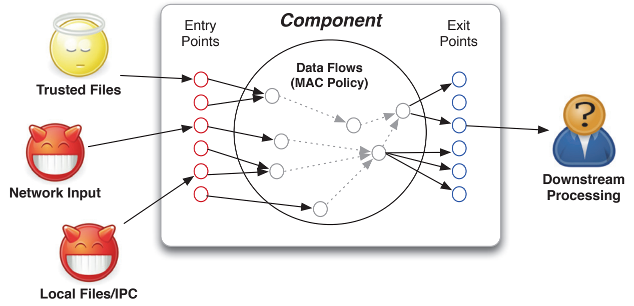

Figure 2: Each software component consists of entry and exit nodes. When the component enforces its own MAC policy then the entry and exit nodes are connected to nodes in the MAC policy.

mation flow problems can be largely automated, resulting a similar effort as configuring information flow problems for single entities (programs or MAC policies). We design an automated method to produce mediator placements from such problems that is an extension of the basic graph cut idea [37, 19] to address general lattices and constrained mediators.

4.1 Building Information Flow Problems

Constructing information flow problems cannot be completely automated because we cannot predict application-specific requirements. Nonetheless, we can greatly reduce the effort necessary to configure such problems, even in complex, distributed environments, such that the task of configuring a system-wide information flow problem is comparable to configuring information flow problems for one entity (e.g., as expected by prior analyses for programs [29] or systems [52]). In this section, we describe the two main insights that guide the construction of information flow problems.

Connection Policies. A modern system deployment now consists of several reference monitors (e.g., firewall, OS, program) independently enforcing access control policies. When enforcing mandatory access control, these policies represent the possible data flows among subjects and objects governed by their respective reference monitors. However, because security policies for each reference monitor are specified independently, the flows among subjects and objects belonging to different reference monitors are ambiguous. For example, while firewall policies limit how adversaries may access the host by port, the specific host processes using those ports are not identified explicitly. Researchers have addressed this particular ambiguity by introducing labeled networking [4], for which there are several implementations in Linux alone [17, 32, 26]. The problem is that administrators must then understand the policies (and implications therein) of each reference monitor necessary to connect the data flows to produce a system-wide data flow graph.

Instead, we find that such connections are either well-known or can be derived automatically, so administrator specification is unnecessary. In the web application, many subjects can be inferred by the use of privileged ports. Also, given the emergence of purpose-specific VMs, the subjects that can possibly use network resources can be easily identified (e.g., included with its specification). For the web application example, the use of unprivileged ports by the browser VMs must be limited to browser processing. A similar problem occurs when connecting program entrypoints to the subject and object labels in the MAC policy accessed by those entrypoints. In this case, we use runtime analysis to collect these data flows in a manner analogous to the construction of MAC policies from the permissions programs request [38, 31]. Thus, to construct a system-wide data flow graph for the web application, no administrator specification is necessary, if the distributors of purpose-specific VMs include connections between the network and VM processes (manual) and between program entrypoints and MAC labels (runtime analysis).

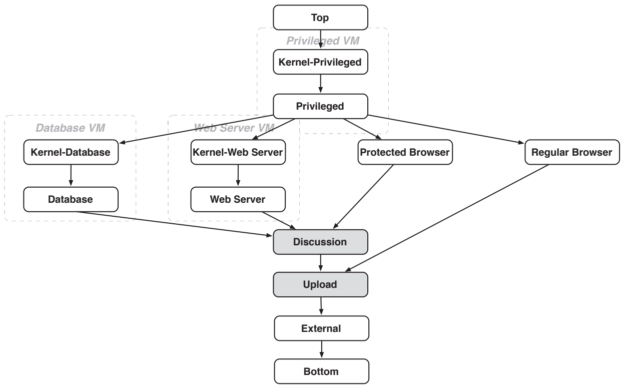

Figure 3: Integrity lattice for the web application. The infrastructure VM levels are highlighted and the application levels are shaded.

Using this information, we automatically construct a data flow graph consisting of components shown in Figure 2. Each component represents an operating system or program by its set of entrypoints, a module containing internal authorized data flows, and exitpoints. For operating systems, the MAC policy they enforce creates a subgraph within the module. For programs, we treat the program internals as a single node. If a program is information-flow aware, it can be represented in the same manner as a MAC enforcing operating system. The system is represented by a hierarchical graph of these components (we use the model defined by Alur [1]), where the network is the highest level, followed by hosts (operating systems) and programs.

Standard Infrastructure Mappings. In order to construct an information flow problem, administrators must produce an integrity lattice and level mapping function to the appropriate subjects and objects in each of the security policies in the distributed system. This task is difficult and error-prone. Researchers typically address this problem by limiting the scope of the information flow problem to a single reference monitor, where an expert can define the expected mapping for the system entities or program variables.

To simplify this problem, we reuse the integrity lattice and level mapping functions defined for each of the infrastructure components when building the system-wide information flow problem to limit the scope of effort required by administrators. We have defined an integrity lattice and level mappings for the web server, database, and privileged VM, and have been able to reuse them unchanged for various web application deployments. Since infrastructure aims to support a wide variety of application deployments essentially unchanged, we expect that reuse for other deployments will be likely. As a result, administrators will only have to specify integrity requirements for their application components.

For the web application, we only need to identify the two types of application data for discussion and external streams of data. Figure 3 shows the integrity lattice for the web application, distinguishing infrastructure levels from those in the application.

4.2 Computing Minimal Mediation

Researchers had the insight that placing a mediator to resolve information flow errors for a lattice policy containing two levels l_i and l_j is tantamount to generating an edge cut[4] of the data flow graph with the nodes mapped to l_i as the sources and the nodes mapped to l_j as the sinks [37, 19]. This property is called *Cut-Mediation Equivalence*. In practice, general lattices policies must be enforced, as shown in Figure 3, so we customize the solution to account for such policies, restricting components that have limited mediation abilities (i.e., only mediate for some, but not all, errors).

Given a directed graph G=(V, E) and a lattice $\mathcal{L}$=$\{L, \preceq\}$, there is a *cut problem* when there is an information flow error between two nodes with the level mapping function M. For general lattices, we must ensure that only information flows authorized by the lattice are possible in the information flow problem. In the web application, we must both protect the application from untrusted inputs, the operating systems in each VM from its applications, and virtualization infrastructure from the guest VMs. As a result, a general lattice creates a set of cut problems to solve, as many as one for each pair of integrity levels in the lattice.

Definition 4.1. *A* cut problem set*, C is defined as* $C = \{(l_i, l_j)|l_i, l_j \in L$, *where* $l_i \not\preceq l_j$, $[\exists\ u, v \in G\ .u \hookrightarrow_G v \wedge l_i \in M(u) \wedge l_j \in M(v)]\}$, *consists of such pairs of integrity levels* l_i *and* l_j. *The problem of finding the minimal cut for a cut problem set is called the* multiway cut problem.

Researchers have shown that the multiway cut problem is NP-Hard for directed graphs [12]. In the context of security lattices, researchers previously suggested a simple greedy solution to the problem that returns the *union* of the solutions for each individual cut problem [19].

To improve on the simple greedy approach, we use the insight that classical integrity encourages processes to upgrade input integrity to their level [8]. The semantics of mediation imply that a node that is picked as a mediator (a node in the graph cut) solution, will raise the level of the incoming data in order to resolve the information flow error. We also note that each node has a limit regarding how high it may upgrade any data; we call this the *maxraiselevel* of each node. Therefore, the graph-cut procedure produces *mediators*, $R \subseteq E \times L$, where each mediator upgrades the integrity of the output data on the edge (u, v) to an $l \in L$, which is at most the node u's *maxraiselevel*. A node's *maxraiselevel* is determined by its program and the levels mapped to it. Using prior work [53], we identify equivalence classes of labels in the MAC policy whose programs must mutually trust one another's integrity. Each node's *maxraiselevel* is the greatest lower bound of any mapping to a label in its integrity equivalence class.

We use this insight to define a property called *Mediation Dominance*. This property states that solving a cut problem in graph G for level l_1 may solve any overlapping problem in the same graph G for a level l_2 if $l_1 \preceq l_2$. The intuition behind this is that since l_1 is higher integrity than l_2, the semantics of mediation implies that any mediators that can mediate for l_1 can automatically mediate for l_2. Therefore, by solving the cut problems for l_1 before l_2, we get two advantages: (1) we may solve a smaller problem for l_2 (compared to solving the problem for l_2 independent of l_1) since *mediation dominance* enables us to remove the mediators computed for l_1 and any flows they fostered before solving the problem for l_2 and (2) if there is an overlap in the graph between the different cut problems of *comparable* lattice levels, then the size of ordered cut solution can be smaller than the naive solution, a *union* of the individual solutions.

```
1  MediationResolution($G, \mathcal{L}, M, MaxRaise$) {
2      $LS \leftarrow TopologicalSort(\mathcal{L})$
3      for ($l \in LS$)
4         do { $Sources \leftarrow \{l_i \in \mathcal{L} \mid l_i \not\preceq l\}$
5             $Mediators \leftarrow Mediators \cup$
6                 $MinimumCut(G, l, Sources, M, MaxRaise)$
7             }
```

Figure 4: Greedy Mediation Resolution Algorithm

The algorithm, Greedy Mediation Resolution, that solves a cut problem set of graph G for the lattice $\mathcal{L}$, is shown in Figure 4. The algorithm receives a data flow graph G, a lattice $\mathcal{L}$, the mapping M of nodes to levels, and the *maxraiselevels* for the mapped nodes $MaxRaise$. Line 2 first sorts the levels in the lattice to order the cut problem set based on mediation dominance as described above creating the ordered set of levels LS. The algorithm then chooses the cut problems from the set in order (Line 3), collecting the levels of source nodes that could cause information flow errors for the current cut problem's sink level (Line 4) and computing the minimum cut, given the nodes that map to those levels (from mapping M) and limited by the constraints on possible cuts ($MaxRaise$).

The running time of the algorithm is dominated by the time of computing the MinimumCut for each sink, (for every label in the lattice $\mathcal{L} = \{L, \preceq\}$). The running time of the MinimumCut algorithm is $O(n^3)$, thus, the running time of the MediationResolution algorithm is $O(|L|.n^3)$.

5. EVALUATION

In this section, we present the results of a prototype implementation of our approach on the web application system presented in Section 2.1. We first describe the prototype analysis tool and the configuration of the web application system in Section 5.1. We then evaluate the ability of our prototype analysis tool to compute mediation placements for the web application system in Section 5.2. We find that the remote threats of the web application deployment result in the need for 27 additional program entrypoints to be mediated to achieve Clark-Wilson integrity. We also explore the mediation of all possible local threats to the web browser and web server (including the web application), finding that 85 additional mediated entrypoints or 20 IVPs are sufficient to protect these processes.

5.1 Prototype and Experimental Systems

The prototype has an Eclipse front-end that provides access to: (1) parsers to build system-wide data flow graphs using XSM/Flask [9], SELinux [33], and iptables policies;

[4] In graph theory, given a graph G=(V,E), an edge cut of this graph with respect to a source and a sink is a set of edges whose removal will divide the graph into two components, one containing the source and the other containing the sink, such that the sink can no longer be reached from the source.

VMs	Default	Remote	Local
Database Server	281	15	-
Web Server	217	24	56
Privileged VM	335	0	-
Protected Browser	372	15	47
Regular Browser	371	15	-
Total	1576	69	103
Unique Mediators	525	27	85

Table 1: The number of mediators needed for: (1) infrastructure mediation per VM (default); (2) remote threats when application is deployed; and (3) local threats to deployment. For "unique mediators" we count each mediator only once even if it is used in a different VM.

(2) mapping rules to infer integrity level mappings and integrity level constraints for constructing information flow integrity models; (3) the Lemon graph library [34] to compute graph cuts for our information flow model; and (4) a module for generating DIFC-Flume [20] policies with integrity verification and entrypoint mediation from graph cuts. The code breakdown is as follows: (1) 4303 source lines of code (SLOC) for the Bison based parsers; (2) 405 SLOC in Prolog for mapping rules and some parsing; and (3) 4808 SLOC in C/C++ for the last two tasks.

The experimental system includes two hosts, for the web server and web client, which run Xen VM system 4.1. The web server host runs three VMs: a privileged VM (domain 0), a web server VM including an Apache web server and web application, and MySQL database VM. The client host runs two VMs, each configured to run a web browser, corresponding to the protected and regular VMs in Figure 1.

Each VM runs the Linux 2.6.31-23-generic kernel. The Xen hypervisor enforces XSM/Flask policies [9], and Each of the VMs enforce SELinux policies [33]. While all of the OS policies are SELinux, they are independent in the sense that each policy supports a distinct set of applications and these policies do not refer to the interactions among VMs. Also, each VM runs an iptables firewall to govern network communications. We assume that secure communication (e.g., IPsec or SSL) is used to protect any channel that carries application data between hosts. Unprotected channels are given the "External" level (i.e., system-low).

In addition to separating the functionality of different security levels using VMs, we also use features of the SELinux MAC system to protect the web application (on the server) and web browser (on the clients) processes further. For the web application, we use the `mod_selinux` module for Apache to generate separate web application processes to communicate with protected VMs, regular VMs, and other external parties. For clients, we use SELinux policy booleans to set more restrictive permissions. For web browsers, the base permissions (i.e., booleans are off) include access to system files (e.g., `/etc` and `/var`) and the user's home directory (e.g., for plugins).

5.2 Experimental Results

In this section, we show how to use our prototype tool to compute an information flow policy that approximates Clark-Wilson integrity for the experimental system.

Infrastructure Mediation. First, we compute the entrypoint mediation for each of the infrastructure components independently. Recall from Section 3 that the goal is to identify the mediation required for the deployment of any application on the infrastructure. We explore the ability to estimate infrastructure mediation by computing the mediation required of a VM configured to run the target package of the component, such as the Apache web server, MySQL database, or Firefox browser. For example, we configure all VMs with the 12 base modules of SELinux refpolicy 2.20120725, and the servers and web clients are configured with Ubuntu 11.10 server (28 modules) and Ubuntu 11.10 desktop (29 modules), respectively. Upon this, we install the application modules (e.g., four policy modules for Apache are added to the web server VM).

Table 1 (Default) shows our prototype's estimate of the minimal mediation provided by each infrastructure component. For example, the web server VM requires that at least 217 program entrypoints provide mediation to protect the web server process and kernel integrity from remote threats based on the default firewall and MAC policies and given the runtime trace. The estimates for other VMs are somewhat higher, indicating that the web server may be easier to defend than the others.

Note, however, that many of the mediators are common across VMs. There are 525 unique mediator entrypoints across all programs in these VMs (i.e., a program may appear in multiple VMs). In particular, 161 entrypoints are common across all VMs, which is approximately 50-75% of the mediator entrypoints in each of the infrastructure VMs.

One claim is that by examining entrypoint mediation rather than process-level mediation we greatly reduce the number of entrypoints that must be examined. The 122 programs that require mediation above have 2604 active entrypoints[5], meaning that about 20% of all these programs' entrypoints require mediation to prevent illegal information flows in default configurations. Thus, it is beneficial to focus on the individual entrypoints threatened by untrusted input.

Remote Threats to Applications. Table 1 (Remote) shows the entrypoint mediation required to block remote threats when the web application is deployed on this infrastructure. While new mediation is required in most VMs, only 27 new, unique entrypoints need to be protected due to the application deployment. 11 entrypoints are necessary for the new web application programs themselves to receive untrusted input. In addition, 10 new mediators are necessary for auditctl, a program for managing the kernel's audit system. The other six entrypoints occur in the dynamic linker in some programs (setfiles, modprobe, sh, etc.) that now have access to files that may be modified by remote adversaries. As a result, the linker may be prone to new untrusted search path vulnerabilities when the application is deployed. New attack paths for particular application deployments can be identified by this analysis.

The browser is not generally part of the web infrastructure, but we can leverage the idea of infrastructure to simplify mediator placement. First, like all the VMs, the browser has 161 mediators in common with the web infrastructure components. Table 1 (Default) shows the mediators required of our particular browser configurations. While such configurations may differ widely, once the administrator settles on

[5] Note that the number of entrypoints is based on the runtime analysis used to build the connection policies, see Section 4.1. Only the entrypoints actually used in this analysis are included.

a configuration, they can evaluate the default mediation expected. This may be useful should the users want to leverage new software for a particular task. For example, the IceCat browser is a relatively new GNU browser package, and our prototype identifies the entrypoints that require mediation for this new software. We recently found a vulnerability in IceCat because it did not protect an entrypoint [53], so identifying where mediation is required may be helpful in proactively preventing such bugs. Once vetted, the browser configurations themselves may be reused for many deployments, thus becoming part of the infrastructure.

Finally, we note that Xen's domain 0, the privileged VM, has no new mediation required due to the application deployment. This is to be expected as domain 0 is infrastructure for the VMs rather than applications.

Local Threat Mediation. Table 1 (Local) shows the amount of additional mediation sufficient to thwart local threats that may impact the web application. As the privileged and database VMs are deployed from the distribution, we only consider local threats to the web server and the web browser. We identify local threats as those objects (MAC object labels) that may be modified by subjects (MAC subject labels) that are not trusted by the target (web browser or web server). These untrusted subjects are mainly user subjects for running ad hoc programs on these VMs.

Of the 248 possible threats to the web browser and 217 threats to the web server based on the MAC policy[6], only 9 and 11 are unmediated local threats, respectively. That is, the other threats are already blocked completely by mediators placed for infrastructure or remote threats. For these unmediated local threats, we have a choice of mediating using integrity verification procedures (IVPs) or more entrypoint mediation. Recall that a Clark-Wilson IVP takes data as input and produces a certification that the data meets an integrity requirement (i.e., is a Clark-Wilson CDI). Since only TPs can modify high integrity data in Clark-Wilson, a problem in applying IVPs for local threats is that they are modifiable by untrusted subjects. One approach may be to apply IVPs to monitor such objects for unsafe values, at which point mediation is required. Another approach is to remove untrusted writers of such objects from the system. We show the administrator the set of untrusted subjects that can modify these threats, so that the administrator may choose to remove some of the associated subjects' or packages from the component.

For the web server, 56 more mediators are required, whereas for the browser 47 more mediators are required. As can be seen, most of these mediators are unique (85 out of 103), which differs from the mediators selected for infrastructure and remote threat mediation. Thus, local threats appear to present difficult challenges if they cannot be mediated at the source.

Finally, we note that all the objects only writable by TCB subjects or the target subjects may also present local threats if they are installed from untrusted sources. There are nearly 800 object labels for these files in each VM, but we expect that most would be derived initially from the signed package files. Further study is required to ensure that these files are installed from trusted sources, otherwise the threat they pose could be evaluated using the local threat analysis described above.

VM	Nodes	Edges	Parse Policy(s)	Build Graph(s)	Compute Cut(s)
Web Server	2028	6621	1.3	0.19	4.0
Privileged VM	2794	11383	4.3	0.38	6.2
Database Server	2577	10065	3.1	0.32	5.0
Protected Browser	2978	11674	3.8	0.34	5.8
Regular Browser	2978	11700	3.8	0.35	5.7
Remote Mediators	13427	51716	-	-	33.4
Total	13427	51716	16.4	1.58	60.1

Table 2: Performance of computing mediator placement shown per VM for the infrastructure mediation and for system for remote mediation.

Analysis Performance. Table 2 shows that mediation placement performance is practical for the example scenario. For individual VMs, infrastructure mediation can be computed in 5 to 11 seconds per VM. The VMs range in size from 2000-3000 nodes and 6000-12,000 edges. For the web application, the mediation placement can be computed in less than 80 seconds. Breaking the computation down into two steps for infrastructure and remote mediation does not save time, as a mediator placement for the entire information flow problem can be computed in 37 seconds (plus 18 seconds for parsing and graph build as in Table 2.

In terms of scaling the analysis to larger systems, we note two findings. First, there is no difference among the web client systems, so we are able to evaluate information flow integrity using just one such system. Second, we find that the infrastructure mediation required of the privileged VMs is the same as required when the application is deployed. This implies that the privileged VMs may be evaluated separately from the guest VMs. We will explore how to automate identification of such optimizations in future work.

6. RELATED WORK

Deploying web applications to enforce security requirements has been explored mainly by restricting the permissions available to web clients. For example, Tahoma [10] uses server-supplied manifests to describe the permissions authorized for web clients as part of that web application. Alternatively, FlowwolF provides information flow enforcement at each layer in software stack [14]. Also, browsers have been extended with the ability to enforce arbitrary policies, motivated by the OP browser design [13]. However, end-to-end configuration of web applications to satisfy information flow is not yet supported.

Researchers have long known that end-to-end integrity protection can be modeled as an information flow problem. However, classical integrity models [5, 8] rely on formal assurance for all high integrity processes that receive untrusted inputs, but formal assurance methods have not been developed that scale to the size of modern programs. As a result, least privilege [43] has been adopted as the security goal for integrity protection in commodity systems [33, 31, 27, 54]. However, different deployments imply different privileges, so commodity systems try to accommodate this through mechanisms to compute permissions requested by processes [38, 31, 3] and by providing runtime configuration options to specify permissions for options. Nonetheless, it is important that default configurations work in most cases, so researchers have found that the default MAC policies still permit several operations that would violate classical integrity [7]. Meth-

[6] It is just a coincidence the the number of untrusted object labels and the number of mediators required for the web server infrastructure in Table 1 (Default) is the same.

ods to simplify policies through the use of virtualization do not eliminate such information flows [14, 36].

Researchers have recently proposed practical approaches to integrity that approximate classical integrity models [47, 21, 50, 20, 40]. These practical integrity models are all strictly weaker than classical models, in that they do not require formal assurance for code with the authority to protect integrity while receiving untrusted inputs. However, they express restrictions on how such authority may be used by high-integrity programs. For example, the Decentralized Information Flow Control [20, 56] (DIFC) model provides processes with capabilities to make decisions about handling untrusted inputs.

With the emergence of MAC enforcement in commodity operating systems, researchers proposed various policy design tools to help system administrators and OS distributors configure policies [18, 52, 45, 7]. Broadly speaking, these tools enable a policy designer to evaluate compliance using *reachability* to identify whether an adversary can perform an unauthorized operation, even indirectly. Reachability analyses have also been performed for network policies in the form of *attack graphs* [35, 48, 30], but these represent attacker behavior rather than using the system policies. However, defining which operations are unauthorized and resolving any problems are manual tasks. Researchers have recently focused on defending an *attack surface* [16, 23], which is the set of program interfaces accessible to adversaries. The idea would be that if we can minimize the number of interfaces accessible to attackers, we could minimize our defensive effort.

The problem of verifying that a MAC policy satisfies a set of security requirements is a policy compliance problem. In a *policy compliance problem*, a policy is said to *comply* with a goal if all the operations authorized by the policy satisfy the constraints of the goal [25, 11, 24, 15]. The problem is that MAC policies often fail to comply with integrity requirements, as discussed above, so we must repair non-compliant cases. However, any MAC policy changes must also preserve necessary function, and balancing functional and security requirements is computationally complex in general.

7. CONCLUSION

In this paper, we developed a method for computing information flow policies that protect data integrity using mediation for distributed systems that are constructed from multiple, independent components. To protect the system's data integrity proactively, we have developed a mostly-automated method to transform a set of commodity MAC policies into a system-wide policy that provides classical integrity protection, in particular satisfying an approximation of the Clark-Wilson integrity model. To do this, we build an information flow problem and compute mediation by resolving any information flow errors by solving a type of graph-cut problem. We demonstrate our method on a custom web application running on virtualized SELinux hosts, which finds that only 27 additional entrypoint mediators and 20 integrity verification procedures are necessary to provide information flow integrity approximating Clark-Wilson integrity, showing the practicality of computing threats proactively for distributed systems. Solutions can be found in tens of seconds, making the proposed approach practical for many distributed systems.

8. ADDITIONAL AUTHORS

Additional authors: Nigel Edwards (Hewlett Packard Labs, Email: nigel.edwards@hp.com)

9. REFERENCES

[1] Alur, R., and Yannakakis, M. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst. 23*, 3 (2001), 273–303.

[2] Anderson, J. P. Computer Security Technology Planning Study, Volume II. Tech. Rep. ESD-TR-73-51, Deputy for Command and Management Systems, HQ Electronics Systems Division (AFSC), October 1972.

[3] http://fedoraproject.org/wiki/SELinux/audit2allow, 1996.

[4] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., and Haghighat, S. A. A domain and type enforcement unix prototype. In *Proceedings of the 5th conference on USENIX UNIX Security Symposium - Volume 5* (1995), SSYM'95, pp. 12–12.

[5] Biba, K. J. Integrity Considerations for Secure Computer Systems. Tech. Rep. MTR-3153, MITRE, April 1977.

[6] Chari *et al.*, S. Where do you want to go today? escalating privileges by pathname manipulation. In *NDSS '10* (2010).

[7] Chen, H., Li, N., and Mao, Z. Analyzing and Comparing the Protection Quality of Security Enhanced Operating Systems. In *NDSS* (2009).

[8] Clark, D. D., and Wilson, D. A Comparison of Military and Commercial Security Policies. In *IEEE Symposium on Security and Privacy* (1987).

[9] Coker, G. Xen Security Modules (XSM). http://www.xen.org/files/xensummit_4/xsm-summit-041707_Coker.pdf.

[10] Cox, R. S., Gribble, S. D., Levy, H. M., and Hansen, J. G. A Safety-Oriented Platform for Web Applications. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)* (2006), IEEE Computer Society, pp. 350–364.

[11] Dragoni, N., Massacci, F., Naliuka, K., and Siahaan, I. Security-by-contract: Towards a semantics for digital signatures on mobile code. In *EuroPKI* (2007).

[12] Garg, N., Vazirani, V. V., and Yannakakais, M. Multiway cuts in directed and node weighted graphs. In *in Proc. 21st ICALP, Lecture Notes in Computer Science 820* (1994), Springer-Verlag, pp. 487–498.

[13] Grier, C., Tang, S., and King, S. T. Secure web browsing with the op web browser. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (2008), IEEE Computer Society, pp. 402–416.

[14] Hicks, B., Rueda, S., King, D., Moyer, T., Schiffman, J., Sreenivasan, Y., Jaeger, T., and McDaniel, P. An Architecture for Enforcing End-to-End Access Control over Web Applications. In *Proceedings of (SACMAT)* (June 2010).

[15] Hicks, B., Rueda, S., St. Clair, L., Jaeger, T., and McDaniel, P. A logical specification and analysis for SELinux MLS policy. *ACM Transaction on Information and System Security 13*, 3 (2010).

[16] Howard, M., Pincus, J., and Wing, J. Measuring Relative Attack Surfaces. In *Proceedings of Workshop on Advanced Developments in Software and Systems Security* (2003).

[17] Jaeger, T., Butler, K., King, D. H., Hallyn, S., Latten, J., and Zhang, X. Leveraging IPsec for Mandatory Access Control Across Systems. In *Proc. 2nd Intl. Conf. on Security and Privacy in Communication Networks* (Aug. 2006).

[18] Jaeger, T., Sailer, R., and Zhang, X. Analyzing integrity protection in the SELinux example policy. In *USENIX Security Symposium* (Aug. 2003).

[19] King, D., Jha, S., Muthukumaran, D., Jaeger, T., Jha, S., and Seshia, S. A. Automating security mediation placement. In *ESOP* (2010).

[20] Krohn, M. N., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M. F., Kohler, E., and Morris, R. Information flow control for standard OS abstractions. In *Proceedings of the 21st ACM Symposium on Operating Systems Pr inciples* (Oct. 2007), pp. 321–334.

[21] Li, N., Mao, Z., and Chen, H. Usable Mandatory Integrity Protection For Operating Systems. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (May 2007).

[22] Linux KVM. Kernel based virtual machine. http://www.linux-kvm.org.

[23] Manadhata, P., Tan, K., Maxion, R., and Wing, J. M. An Approach to Measuring A System's Attack Surface. Tech. Rep. CMU-CS-07-146, School of Computer Science, Carnegie Mellon University, 2007.

[24] Massacci, F., and Siahaan, I. Matching Midlet's security claims with a platform security policy using automata modulo theory. In *In proceedings of NordSec* (2007).

[25] McDaniel, P., and Prakash, A. Methods and limitations of security policy reconciliation. *ACM Trans. Inf. Syst. Secur.* (2006).

[26] Morris, J. New secmark-based network controls for selinux. http://james-morris.livejournal.com/11010.html.

[27] MSDN. Mandatory Integrity Control (Windows). http://msdn.microsoft.com/en-us/library/bb648648%28VS.85%29.aspx.

[28] Myers, A. C. JFlow: Practical mostly-static information flow control. In *POPL '99* (1999), pp. 228–241.

[29] Myers, A. C., Zheng, L., Zdancewic, S., Chong, S., and Nystrom, N. Jif: Java information flow. http://www.cs.cornell.edu/jif,July2001-2003.

[30] Noel, S., Jajodia, S., O'Berry, B., and Jacobs, M. Efficient minimum-cost network hardening via exploit dependency graphs. In *ACSAC* (2003).

[31] Novell. AppArmor Linux Application Security. http://www.novell.com/linux/security/apparmor/.

[32] NetLabel - Explicit labeled networking for Linux. http://www.nsa.gov/selinux.

[33] Security-enhanced linux. http://www.nsa.gov/selinux.

[34] on Combinatorial Optimization, E. R. G. Lemon Graph Library. http://lemon.cs.elte.hu/trac/lemon.

[35] Ou, X., Boyer, W. F., and McQueen, M. A. A scalable approach to attack graph generation. In *CCS* (2006).

[36] Payne, B. D., Sailer, R., Caceres, R., Perez, R., and Lee, W. A layered approach to simplified access control in virtualized systems. *ACM SIGOPS Operating Systems Review 41*, 4 (July 2007), 12 – 19.

[37] Pike, L. Post-hoc separation policy analysis with graph algorithms. In *Workshop on Foundations of Computer Security (FCS'09). Affiliated with Logic in Computer Science (LICS)* (August 2009).

[38] Provos, N. Improving host security with system call policies. In *Proceedings of the 2003 USENIX Security Symposium* (August 2003).

[39] Provos, N., Friedl, M., and Honeyman, P. Preventing privilege escalation. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12* (Berkeley, CA, USA, 2003), USENIX Association, pp. 16–16.

[40] Roy, I., Porter, D. E., Bond, M. D., McKinley, K. S., and Witchel, E. Laminar: practical fine-grained decentralized information flow control. *SIGPLAN Not. 44*, 6 (June 2009), 63–74.

[41] Rueda, S., Muthukumaran, D., Vijayakumar, H., Jaeger, T., and Chaudhuri, S. Towards system-wide, deployment-specific mac policy generation for proactive integrity mediation. Tech. Rep. NAS-TR-0151-2011, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, Sept. 2011.

[42] Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J. L., and van Doorn, L. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *ACSAC* (Washington, DC, USA, 2005).

[43] Saltzer, J. H., and Schroeder, M. D. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (Sep. 1975).

[44] Santos, N., Rodrigues, R., Gummadi, K. P., and Saroiu, S. Policy-sealed data: a new abstraction for building trusted cloud services. In *Proceedings of the 21st USENIX Security Symposium* (2012), USENIX Association.

[45] Sarna-Starosta, B., and Stoller, S. D. Policy analysis for security-enhanced linux. In *WITS* (April 2004).

[46] Schiffman, J., Moyer, T., Jaeger, T., and McDaniel, P. Network-based root of trust for installation. *IEEE Security & Privacy* (Jan/Feb 2011).

[47] Shankar, U., Jaeger, T., and Sailer, R. Toward Automated Information-Flow Integrity Verification for Security-Critical Applications. In *Proceedings of the 2006 ISOC Networked and Distributed Systems Security Symposium* (February 2006).

[48] Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), IEEE Computer Society, p. 273.

[49] Smalley, S., Vance, C., and Salamon, W. Implementing SELinux as a Linux Security Module. Tech. Rep. 01-043, NAI Labs, 2001.

[50] Sun, W., Sekar, R., Poothia, G., and Karandikar, T. Practical Proactive Integrity Preservation: A Basis for Malware Defense. In *Proceedings of the IEEE Symposium on Security and Privacy* (2008), IEEE Computer Society, pp. 248–262.

[51] Sun Microsystems. Trusted solaris operating environment - a technical overview. http://www.sun.com.

[52] Tresys. SETools - Policy Analysis Tools for SELinux. Available at http://oss.tresys.com/projects/setools.

[53] Vijayakumar, H., Jakka, G., Rueda, S., Schiffman, J., and Jaeger, T. Integrity walls: Finding attack surfaces from mandatory access control policies. In *Proceedings of the 7th ACM Symposium on Information, Computer, and Communications Security (ASIACCS 2012)* (May 2012).

[54] Watson, R. N. M. TrustedBSD: Adding trusted operating system features to FreeBSD. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference* (2001), pp. 15–28.

[55] Zanin, G., and Mancini, L. V. Towards a formal model for security policies specification and validation in the selinux system. In *SACMAT* (2004).

[56] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. Making information flow explicit in HiStar. In *OSDI* (2006).

[57] Zeldovich, N., Boyd-Wickizer, S., and Mazières, D. Securing distributed systems with information flow control. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 293–308.