The Need for Application-Aware Access Control Evaluation

William C. Garrison III
Dept. of Computer Science
University of Pittsburgh
bill@cs.pitt.edu

Adam J. Lee
Dept. of Computer Science
University of Pittsburgh
adamlee@cs.pitt.edu

Timothy L. Hinrichs
Dept. of Computer Science
University of Illinois at Chicago
hinrichs@uic.edu

ABSTRACT

Access control is an area where one size does *not* fit all. However, previous work in access control has focused solely on expressiveness as an absolute measure. Thus, we discuss and justify the need for a new type of evaluation framework for access control, one that is application-aware. To this end, we apply previous work in access control evaluation, as well as lessons learned from evaluation frameworks used in other domains. We describe the analysis components required by such a framework, the challenges involved in building it, and our preliminary work in realizing this ambitious goal. We then theorize about other areas within the security domain that display a similar absence of such evaluation tools, and consider ways in which we can adapt our framework to analyze these broader types of security workloads.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; I.6.5 [**Simulation and Modeling**]: Model Development; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Theory, Verification

Keywords

Access Control Scheme, Application-Aware, Expressiveness, Suitability Analysis

1. INTRODUCTION

Access control is one of the most fundamental aspects of computer security, and has been the subject of much formal study. However, existing work on the formal analysis of access control schemes has focused largely on comparing the *relative expressive power* of two or more access control schemes (e.g., [1, 7, 11, 17, 20, 21, 23, 25]). Although expressive power is an interesting and meaningful basis for comparing access control schemes, in practice it is not a sufficient indicator of suitability to any particular application. That is, the knowledge that a scheme $\mathcal{T}$ is more expressive than another scheme $\mathcal{S}$ provides no assurance that $\mathcal{T}$ is the best access control scheme for use within a particular real-world application context. It could be the case, for instance, that $\mathcal{S}$ is *expressive enough* for a particular application and also has lower administrative overheads than $\mathcal{T}$ would in the same situation. Furthermore, as was noted in a recent NIST report, access control is not an area with "one size fits all" solutions and, as such, systems should be evaluated and compared relative to application-aware metrics [14]. This report notes a variety of possible access control quality metrics, but provides little guidance for actually applying these metrics and carrying out *practical* analyses of access control schemes.

For these reasons, we advocate the development of an *application-aware* evaluation paradigm for access control schemes. Informally, this problem can be stated as follows: *Given a description of a system's access control needs and a collection of access control schemes, which scheme best meets the needs of the system?* Instances of this question can arise in many different scenarios, encompassing both the deployment of new applications and the reexamination of existing applications as assumptions and requirements evolve. Modern software applications are complex entities that may control access to both digital (e.g., files) and physical (e.g., doors) resources. Given that organizations are typically afforded little guidance in choosing appropriate security solutions, application-aware analysis of access control could help developers sort through the myriad available security frameworks (e.g., WPL [18], Spring [24], Shiro [3], etc.) and the multiple access control schemes embedded in each.

Despite the vast differences in approach between existing access control expressiveness evaluation and the application-aware evaluation process that we propose, we believe that expressiveness will remain a key component in the process. To this end, we investigate the use of expressiveness-based techniques for ensuring that the access control schemes considered for an application possess the necessary functionality. However, unlike prior work, we place application-aware constraints on the types of safety and security properties that must be preserved by an access control scheme while servicing the needs of a particular application. Furthermore, we study techniques used for cost analysis in other domains to present the first formal notion of an access control workload and offer guidelines for workload construction and access control cost analysis.

NSPW'12, September 18–21, 2012, Bertinoro, Italy.
Copyright 2012 ACM 978-1-4503-1794-8/12/09 ...$15.00.

Thus, we proceed in developing an application-aware evaluation framework in two parts: the structures used to formally specify the inputs to the analysis process (e.g., the description of the application in question, the access control schemes that are candidates for the application), and the analysis process itself (i.e., the general workflow used to analyze the access control schemes with respect to the application). We believe that the process is fairly mature. Indeed, we choose machine-like input structures whose inherent execution semantics make them natural to evaluate. This intuitiveness is one strength of our approach, and lends to our goal of creating a more formal representation of an informal process that is already carried out by administrators in the real world. Its familiarity in this sense means that it will be natural to analysts, while its formal rigor will allow administrators to have confidence in its results. However, at this point, the structures for specifying of inputs to the framework are much less mature, and each presents its own challenges. Thus, the "right" way of representing each component is a subject of ongoing work.

In this paper, we describe the analysis process and our preliminary progress in properly specifying the process's inputs. Although the ideas we present are somewhat nascent, they reflect significant progress towards realizing the ambitious goal of application-aware security analysis. We then present shortcomings of the basic forms of the various input structures we use, and discuss ways in which this framework must be improved to realize the full goals of application-aware evaluation.

Finally, we discuss other areas within the security domain that display a similar absence of application-aware evaluation tools. We believe that a framework with the properties described in this paper can be adapted toward formalizing other types of broader *security workloads* and analyzing candidate implementations with respect to those workloads. That is, given a more general security goal, we may be able to utilize a similar approach to that which we advocate for the analysis of an application's access control needs: first, identify the candidate solutions which are *capable* of servicing the security workload while preserving its requisite safety and security properties, then determine which solution works *best* relative to the application's specific cost valuations. In this way, we make progress not only in a more thorough evaluation of access control, but also a more complete evaluation of a range of security problems. Thus, we make the following contributions:

- We present the first discussion of the shortcomings of current, application-agnostic access control methods.

- We discuss requirements of a formal structure for specifying *access control workloads*, a novel concept that we introduce to capture an application's specific access control needs. A workload formalizes both the functional requirements of an application, as well as expected patterns of invocation and use of this functionality. We also describe our preliminary efforts in creating such a structure.

- We describe a two-phase, application-aware analysis process for access control schemes. In the first phase, we apply expressiveness-based access control evaluation techniques toward proving that candidate access control schemes are *capable* of executing the workload while maintaining security and safety properties deemed important to the application. In the second phase, we apply novel techniques inspired by cost evaluation in other domains toward calculating the *cost* of executing the workload within each candidate scheme.

- We describe ways in which an application-aware evaluation framework may lend itself to the development of similar techniques and tools in the broader security domain.

The remainder of this paper will be structured as follows. In Section 2, we present several scenarios that motivate the development of an application-aware access control evaluation framework, and discuss why past work in expressiveness evaluation in access control is insufficient for these scenarios. In addition, we highlight select work in cost analysis in other domains. In Section 3, we combine these ideas to form an overview of the design of an application-aware access control evaluation framework. In Section 4, we discuss methods for formally specifying the inputs to the analysis framework, including access control schemes, workloads, implementations and costs. In Section 5, we describe simulation-based methods for utilizing implementations to calculate the cost each scheme incurs in implementing a workload. We present some of our progress toward realizing an early version of such a framework in Section 6, and discuss shortcomings and open problems in Section 7, including the adaptation of such a framework to more general security problems. Finally, we conclude in Section 8.

2. BACKGROUND

In this section, we present several scenarios to motivate an application-aware analysis of access control, and discuss why past work is insufficient in addressing these scenarios.

2.1 Motivating Scenarios

In this section, we discuss several realistic evaluation scenarios for which previous work is ill-suited. One such scenario is the establishment of new systems with access control components. For example, suppose a new resort is opening, and to choose an appropriate security package, the systems managers consider a number of access control use cases. Guests should be allowed into the resort at all hours, but in certain areas during business hours only, and of course into no guest rooms but their own. Office staff should have access to the necessary portions of guest records during their shift. Maintenance staff must have access to supply areas and guest rooms, as well as records of which guest rooms are vacant. In modern facilities, we expect access to both physical and electronic resources to be controlled by computer systems. An application-aware access control analysis framework would enable the resort managers to formalize their workload's requirements and their candidate security packages' mechanisms, and use these formalizations to choose the package that best meets their needs with minimal overheads.

Another scenario that highlights the need for application-aware evaluation for access control is re-evaluating the access control components of existing systems whose requirements or operating assumptions have evolved. For example, two MITRE technical reports [13, 26] highlight the fact that the lattice-based access control scheme relied upon by the U.S. Armed Forces is beginning to show signs of age. In particular, [13] cites several examples of improper and unauthorized out-of-band data sharing that have occurred because it is "easier to ask for forgiveness than for permission," given the high delays and human costs associated with utilizing the proper channels. The second report [26] posits that this phenomenon is a byproduct of an increasingly dynamic military that relies on a dizzying array of data sources and ever-changing coalitions, but bases access decisions on a static classification system. In short, the changing military workload has led to confidentiality breaches due to an ill-fitting access control scheme. An application-aware analysis could help identify the root causes of these failures and assess the utility of alternate access control approaches.

Example 1. In the military application just described, data is generated simultaneously by a large number of independent processes. Coalitions are created, joined, and disbanded with high frequency. Thus, large numbers of accesses need to be granted or revoked at once, often within a very short time.♦

2.2 Related Work

The formal study of access control schemes began with the seminal paper by Harrison, Ruzzo, and Ullman that investigated the rights leakage problem [11]. This paper formalized a general access control model and proved that determining whether a particular access right could ever be granted to a specific individual—the so-called "safety problem"—was an undecidable problem. Shortly thereafter, Lipton and Snyder showed that in a more restricted access control system, this problem was not only decidable, but decidable in linear time [17]. These two results introduced the notion that the most capable system is not always the right choice—that restricting our system can yield higher efficiency and greater ease in solving relevant security problems. This led to many results investigating the relative expressive power of various access control schemes, often leveraging some notion of (bi)simulation (e.g., [1,7,20,21,23]).

Further work by Ammann et al. [1], Chander et al. [7], and Li et al. [16] developed simulation-based frameworks for comparing the expressive power of various access control schemes. These simulation frameworks proved to be too relaxed, allowing almost any reasonable scheme to be shown equivalent to all others. To address this, Tripunitara and Li [25] developed a more restrictive notion of expressive power. Their framework supersedes the more informal notions of simulation developed in prior works by requiring the use of specific types of mappings between systems that guarantee relevant security properties are preserved under simulation; this provides a greater level of precision when ranking access control schemes in terms of their expressiveness. Unfortunately, none of these frameworks support the comparison of access control schemes with regards to their ability to perform *well* within a particular environment.

As such, the most we can learn from these simulation-based frameworks is an absolute ranking in expressiveness, irrespective of the requirements of the application. As stated above, it is unclear that there is any benefit to expressiveness beyond "expressive enough," but past work does not even provide us with a method for ensuring that a scheme is expressive enough for an application. Thus, it seems clear that a new, application-aware access control evaluation paradigm is needed.

The need for application-aware evaluation of access control systems was reinforced by a recent NIST report, which states that "when it comes to access control mechanisms, one size does not fit all" [14]. The report bemoans the lack of established quality metrics for access control systems, going so far as to list numerous possibilities, but stopping short of explaining how one might choose between them or evaluate systems with respect to one's specific requirements. In this paper, we explore exactly this problem.

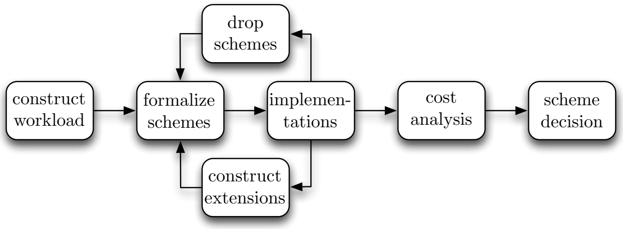


Figure 1: Workflow of an application-aware analysis framework for access control

Recent work by Wang et al. [27] describes methods to safely extend role-based access control schemes with delegation primitives. However, role-based access control is only one particular scheme, and delegation is only one particular capability that may be desirable in an access control environment. Thus, this work provides no guideline for extending other access control schemes, or using extensions to allow different classes of abilities. In our work, we discuss the need to extend arbitrary access control schemes with a variety of added capabilities (see Section 4.3.2).

As a result of the lack of access control evaluation tools that are application-aware, there is little work in the field for generating synthetic traces that are representative of an application. Thus, for inspiration in designing the first access control workloads, we turn to work in other domains. In the field of disk benchmarking, Ganger [10] observes that interleaved workloads provided the most accurate approximation of recorded traces. Thus, mechanisms for representing access control workloads must be capable of simulating the interleaved actions of multiple actors. This view is reenforced by the design of IBM's SWORD workload generator for stream processing systems [2]. This work also points out that synthetic workloads need to replicate both volumetric and contextual properties of an execution environment in order to provide an accurate indication of a system's performance within that environment. Thus, we conjecture that access control workloads as well may need to be capable of expressing not only volumetric statistics such as number of documents created, but also contextual statistics such as the type of content in created documents.

3. FRAMEWORK OVERVIEW

In this section, we present an overview of the components of an application-aware access control evaluation framework. Figure 1 depicts the process of utilizing such a framework to evaluate a series of access control schemes with respect to a specific application. First, the application's access-control-relevant properties and requirements are formalized as an access control workload. This workload must describe two components. The *operational component* specifies an abstract model of the capabilities the application requires of its access control solution, including the properties of this model that acceptable implementations must preserve. The *invocational* *component*, then, describes the usage pattern in which the final access control system will be expected to perform.

Next, in the capability analysis procedure, the access control schemes to be evaluated are formalized as state machines. Each state machine must be shown to be expressive enough to implement the operational component of the workload or extended to enable an implementation. An implementation is a formal mapping from the workload's operational component to an access control scheme that shows the access control scheme is expressive enough to simulate the workload. The notion of an implementation is application-aware in that an application can require that all access control scheme implementations satisfy several additional, high-level properties.

Finally, in cost analysis, traces of access control actions are generated to match the workload's usage pattern as specified in the invocational component. These traces are translated into concrete actions in each candidate scheme using the implementation, and the cost of executing the translated traces is evaluated, yielding a total cost for each scheme. The notion of "cost" that is used, in the application-aware spirit, can vary between applications, from operational costs such as data management overheads, to human-centric costs such as administrative overhead.

4. MODELING ACCESS CONTROL

In this section, we discuss the structures necessary to specify the inputs to our access control analysis framework. We describe minimal requirements of each structure, which lead us to the basic forms we use in our preliminary investigation (Section 6). We also discuss areas of improvement in our structures to increase expressiveness and ease of specification.

4.1 Models, Schemes, and Systems

At the heart of an access control system is the access control *model*. Intuitively, an access control model is a collection of data structures used to store the information needed to make access control decisions. Each configuration of these data structures (each access control *state*) defines an access control policy, and thus changing the contents of these data structures changes the policy. An access control model is formalized as the set of all possible configurations of its data structures, that is, the set of all possible *states*.

An access control *scheme*, then, refines a model by defining sets of commands and queries that can be used to interact with these states. Commands formalize the set of transformations that can be made to the state, and thus define the reachability relation between states. Queries define which questions users can ask of the system and how the system will interpret the state to construct its responses to such questions. Lastly, an access control *system* is an instantiation of a scheme, defining the subset of the scheme's commands that are immediately available, as well as an initial state.

Reasoning about models, then, allows conclusions to be drawn only about access control systems' data structures. This has proven to be too abstract, as previous work shows distinctions between schemes with identical models but different commands [7, 25] or queries [23, 25]. On the other hand, little is to be gained by including a system's initial state in an analysis; generalizing over *all* states in a scheme allows us to make stronger claims about its properties. Thus, in this paper, our discussions mainly consider access control *schemes*. This allows us to make claims that are general enough to affect more than just a single instantiation (claims that could be invalidated even by minor changes to the state), while being specific enough to enable a more powerful analysis than can be performed at the model level. Furthermore, the inclusion of the commands and queries enables reasoning about traces of actions, an integral part of cost analysis.

Thus, the main structure that we will study is the access control scheme, containing the following components:

- The *set of states*, which specifies the data structures the system will use to store access control information
- The *set of commands*, which specifies the procedures available for transforming the state
- The *set of queries*, which specifies the access control questions that can be asked of the system and answered from the information in the state

As will be seen later, the ability to represent both the state of the system, as well as its means of usage and modification is central towards analyzing a particular scheme's ability to service some access control workload.

4.2 Workloads

An access control workload needs to formalize all of the access control requirements of an application. Thus, we believe it should contain two components: an operational component that describes the capabilities the application requires of an access control scheme, and an invocational component that describes how these capabilities will be used. In this section, we present the first discussion of the structure of an access control workload.

4.2.1 Operational Component

The *operational component* of an access control workload must describe the *minimum* set of capabilities that a suitable access control scheme needs to support in order to properly operate within the application of interest. Thus, it should describe, at a high level, the following components.

- The *set of states*, which describes the access control relevant information that the system must maintain
- The *set of commands*, which specifies the procedures that the system requires for modifying the state
- The *set of queries*, which specifies the access control questions that the system needs to answer from the state
- The *implementation restrictions*, which specify what properties an implementation of this workload must possess

Describing the first three components can thus be accomplished by describing an access control scheme that meets the applications' requirements. We note that, while access control workloads and schemes can be formalized the same way, they differ in their intention: a scheme represents a functioning piece of software, while a workload's operational component is built by the analyst to represent the higher-level *desired* functionality of a system, without necessarily being appropriate for direct implementation.

While the first three components listed above represent a specification of the system's capabilities, the fourth describes high-level properties that any implementation of this workload must have. Intuitively, these properties dictate how each state and command of the workload is represented in the access control scheme. For example, the access control policy of any given workload state should be represented by an access control scheme state with exactly the same access control policy, or else the implementation would fail to enforce the same access control policy as the workload. But depending on the application, additional properties may be useful as well, such as requiring that a single command in the workload be represented as a single command in the access control scheme (therefore ensuring the atomicity of workload commands). We discuss such properties in Section 4.3.1.

Example 2. Recall the application described in Example 1. The operational component of the workload that represents this application should describe the components it uses to store the sets of users in coalitions, the documents that members have access to, based on their country of origin and need-to-know information, and other state information that naturally encodes the access control policy of the application. Commands should be available that add groups of people, perhaps previously unknown to the system, to new or existing coalitions, granting them some accesses in the process. Coalitions should also be able to be disbanded, removing large numbers of accesses from various classes of users. The set of queries may allow asking whether a user has a particular access, whether a user belongs to a particular coalition, whether a user's home country is a particular country, and any other data that is used to make decisions within the system. As a military (confidentiality) system, the type of implementation required will likely be relatively strict. ♦

4.2.2 *Invocational Component*

As we discussed in the previous section, describing the capabilities that an application requires of its access control scheme allows us to decide which schemes are *capable* of operating within a particular application. However, it offers no insight into which is most *suitable* for the application. Towards this goal, we believe that an access control workload should also contain an *invocational component* describing the ways in which the capabilities of the workload are to be used in practice.

The invocational component of an access control workload should describe the ways in which the system is expected to be used. At a minimum, the invocation component should be able to dictate the order in which commands are executed, and which queries are asked during which paths of execution. We identify the set of *actions* as the set of commands combined with the set of queries (i.e., the operations that can be executed within the access control system). A simple invocation structure, then, might simply describe the probability distribution among actions.

Example 3. The invocational component to accompany the operational component described in Example 2 should describe a number of correlations between actions. For example, when a coalition is formed, large numbers of documents that each own should be shared with the others. When a coalition is disbanded, these accesses should be revoked. Additionally, disbanding often indicates the end of a mission, which can cause the disbanding of other coalitions as well.♦

Unlike the operational component of a workload, for which there seems to be a single, obvious choice, in our preliminary investigations, we considered several different structures for specifying the invocational component of a workload and found them all to be plausible candidates. For example, while a probability distribution over actions is a conceptually simple description of possible invocations, it can not express any dependency between actions, making it unsuitable for an application in which, e.g., adding a user is necessarily followed by granting this user access to a shared resource. Below we describe three of the structures we considered.

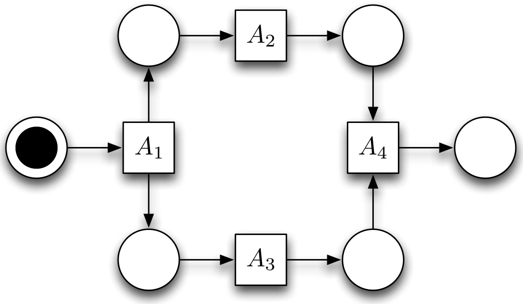


Figure 2: A petri net, demonstrating action dependencies

Petri nets are one possible candidate structure for an access control workload's invocational component. A Petri net is a type of state machine in which paths alternate between reaching *places* and *transitions*. This type of structure can be applied to access control workload invocations by labeling transitions with actions from the workload scheme. Whenever a transition is reached, the action it is labeled with is executed. Thus, as shown in Fig. 2, it is simple to represent action dependencies as described above. In this Petri net, action A_1 will be executed first, followed by A_2 and A_3 (in either order), and finally followed by A_4. In addition, Petri nets are a natural representation of concurrent and distributed processes, making them powerful for expressing applications in which, e.g., there are many users acting on the access control system simultaneously.

Another structure that may be useful in specifying access control invocations is the agent-based "data factory," as in SWORD [2]. As it was built for high-volume data processing systems, an invocation based on the techniques in SWORD would allow complex, interleaving traces of actions caused by a large number of communicating and cooperating entities. This type of high-volume, multi-agent scenario is likely to occur in access control systems with automatically-generated data from multiple sources that requires immediate processing.

Finally, we mention the use of constrained workflow systems [6, 9, 27]. These structures allow for specification of execution constraints for actions, such as those commonly used to enforce separation of duty. Thus, execution constraints are likely to be useful in specifying invocations for business-oriented applications.

As will be discussed in Section 6, our preliminary work on analyzing the invocation component of a workload treats it almost as a black box, and therefore we are free to defer the task of identifying the best way of using these structures to construct a flexible, expressive, easy-to-specify invocation to future work. Another reason to postpone its formal development is that, despite having already identified several desirable features of a structure for specifying invocations (e.g., the Petri net's ability to express concurrent execution), we expect to encounter additional features as we analyze new access control applications.

4.3 Implementations

An access control *implementation* is a mapping between a workload and a candidate scheme. The existence of such a mapping (and proof that it satisfies the required implementation properties) shows that the scheme is capable of executing the application described by the workload. Furthermore, an implementation provides a "recipe" that describes *how* to use the scheme to satisfy the requirements of the workload. We refer to the implementation that describes how to use an access control scheme $\mathcal{S}$ to satisfy the requirements of the application described by an access control workload W as an *implementation* of W in $\mathcal{S}$.

Intuitively, an implemenation dictates how the actions of the workload can be carried out in the access control scheme. An implementation in this sense is akin to a simulation in previous work [1, 7, 20, 21, 23]. Although some previous work [5] attempts to prove expressiveness by describing only the mapping from states to states, other work shows that commands [7] and queries [25] can also affect the ability of a scheme to successfully simulate another. Thus, an implementation must include the following:

- The *state mapping*, which translates an abstract workload state to an equivalent state in the implementing scheme
- The *command mapping*, which translates a workload command to a list of commands that perform the same action in the implementing scheme
- The *query mapping*, which maps a workload query to an equivalent query in the implementing scheme

Example 4. To specify a scheme's implementation of the workload detailed in Examples 2 and 3, one must describe several mappings. The coalition membership information and other workload state elements must be stored within the scheme's state components. Each of the workload's commands should be translated into a series of functionally equivalent commands in the scheme. Finally, the workload's queries should be mapped to equivalent scheme queries. ♦

4.3.1 Implementation Properties

An access control scheme can admit many different implementations of a workload, and those implementations are sometimes qualitatively different from one another. Two qualitatively different implementations (two different implementation types) may provide different security guarantees, and hence as mentioned in Section 4.2.1, a workload's operational component should specify the set of qualitative properties that an implementation must satisfy.

A fairly minimal property for implementations, *access-safety*, requires the set of accesses be preserved across the implementation. That is, for any given state in the workload, its equivalent state in an implementing scheme should respond in the same way to any queries of the form "Does subject s have access right r to object o?" Indeed, some past work in expressiveness analysis uses only this restriction in their simulations [1, 23]. A similar implementation constraint is *simple safety* (after the simple safety question [11, 25]), which asks, "Can subject s *ever obtain* access r to object o?" An implementation with simple safety preserves the answers to all instances of the simple safety question. Generalizing even farther, the *temporal-safety* property requires that all propositional formulas over all queries be preserved when either existentially or universally temporally quantified. (A universal quantifier requires a propositional formula to be true at *all* subsequent states, and an existential quantifier requires it to be true at *some* subsequent state.)

Grant-safety and revocation-safety relate to the execution of multiple commands in an implementing scheme to simulate a single command in the workload scheme. *Grant-safety* requires that, if a permission is granted to a subject during the chain of commands, it be required by the action. That is, no permissions are granted and later revoked. *Revocation-safety*, similarly, requires that no permissions be removed and later granted again. For example, if a workload command grants one permission, an implementation that is not required to be grant-safe could grant ten permissions and then revoke the nine that were unnecessary. Requiring these properties can thus eliminate implementations that are unfavorable, but can also prohibit certain acceptable implementations.

Another property that may be desirable for implementations is one we call the *homomorphic property*. An implementation is homomorphic if any substitution of the state elements' identifiers yields another valid implementation. This prevents implementations that use the names of subjects, objects, or other elements of the state to encode additional data (we identify this as a type of "cheating," as it can extend the capabilities of an access control system to be nearly limitless.)

Finally, *exposure-robustness* enforces that the system be safe (i.e., preserve the other required implementation properties) even if the implementing scheme's commands are used arbitrarily (i.e., not only in the way that simulates the workload scheme's commands). For an implementation to be safe under this definition, it must not be possible to violate other properties of the workload scheme by interleaving workload commands with implementing scheme commands. Implementations that are not required to preserve this property may only be safe if the scheme's commands are considered strictly in the context of simulating the workload's commands.

In our preliminary investigation (see Section 6), we use a particularly strict notion of implementation, which allows us to demonstrate our analysis process. A subject of future work is developing a more thorough understanding of various implementation properties in an effort to ensure that implementations are neither too strict nor too loose in preserving the important characteristics of an application. An analysis that uses too strict a notion of implementation may yield an inefficient mapping in an effort to preserve features that are unneeded. On the other hand, using an implementation that lacks the features necessary for a workload can lead to violations of security assumptions. For example, if the application in question will not reveal the access control scheme's commands directly to untrusted users, requiring an exposure-robust implementation may result in a suboptimal mapping. Conversely, if the scheme's commands *are* made available, failing to preserve exposure-robustness can result in an implementation that enters states that the workload considers unsafe.

4.3.2 Extending Schemes

If there is an implementation of the workload in each of the schemes one is interested in analyzing, and each of these implementations satisfies the required properties, then the schemes in question are all capable of implementing the workload. However, this is not always the case. It is possible, for example, that one of the candidate implementations satisfies access-safety, but accomplishes this by storing additional metadata in the names of files, and thus violates the homomorphic property. If this is a required property for implementations of the workload in question, but such an implementation can not be constructed easily within the scheme, then this scheme must be dropped from the analysis or enhanced in some way to expand its expressiveness and enable the stronger implementation. It is also possible that the scheme is not capable of implementing the workload at all, even under the most relaxed definition of implementation.

In either case, it is possible that this under-expressive scheme nonetheless has the lowest execution cost. Thus, if the additional required expressiveness could be added to the scheme in a similarly low-cost way, the resulting extended scheme may yield the most efficient implementation. For this reason, we claim that it is desirable to be able to *extend* the access control scheme in such a way that newly enables it to implement the workload. An extension to an access control scheme specifies one or more of the following:

- Additional state components to be added to the state
- Additional commands to be added to the command set
- Additional queries to be added to the query set

One must use care, however, when extending schemes. Although virtually any changes to an access control scheme will yield another valid scheme, not all changes will yield a scheme that preserves the security properties of the original. For example, almost any scheme will be "broken" if we add a "grant-all" command that grants all permissions to all subjects. If we allow an extension to arbitrarily change the specification of a scheme, we are then effectively writing an access control scheme from scratch, rather than extending an existing scheme. To maintain the intuition behind the term "extension," we enforce that the changes made to the scheme at most enable strictly greater expressiveness without affecting any of the scheme's possible existing implementations. Specifically, in order to safely extend a scheme, one must prove that the extension does not violate any of the security properties of the scheme. Which security properties are relevant can depend, as expected, on the type of implementation in question. Showing that there is an implementation of the original scheme within the extended scheme proves that the extended scheme can be used transparently in place of the original. The violation of even simple safety resulting from extending a scheme with the above "grant-all" command can be detected by attempting (and failing) to construct an implementation of the original scheme within this extended version while preserving simple safety.

Example 5. Consider the scenario in which, when constructing the implementation described in Example 4, it is discovered that the scheme does not have any state components that can be used to maintain a user's home country. However, the coalition-based workload, as described in Examples 2 and 3, requires this information for making policy decisions and answering certain queries. Thus, one may attempt to extend this scheme by adding to the state a relation that maps users to home countries and adding a query to ask whether a user is a citizen of a particular country. ♦

Thus, having justified the need for ensuring that an extension will not violate the scheme's security properties (with respect to the desired type of implementation), it is desirable to characterize the set of extensions that satisfy this requirement. This would enable us to more easily verify that an extension is safe, without constructing an implementation for every extension we want to use. A description of the structural properties of a safe extension (for a given type of implementation) would enable easier verification than checking the properties directly or building an implementation.

Note, though, that extending a scheme is not necessarily without penalty. Since, in practice, these extensions would be implemented as additional trusted code that communicates in a secure way with the original access control software, one may be concerned if a high proportion of the total state is stored within the extension, or if a large amount of communication needs to occur between the original state and the extension state. If desired, these concerns can be addressed within the definition of cost measure (see Section 4.4.1). For example, one can consider the measure, "the maximum proportion of total state contained within the extension."

4.4 Costs

In this section, we describe the input structures that specify the costs associated with each command and query in the candidate access control schemes.

4.4.1 Cost Measures

In order to calculate overall costs, one must first formalize the relevant measure (or measures) within which to evaluate the cost. We describe the minimal set of properties a cost measure must have. First, the measure must obviously include a set of elements, the *costs.* A binary operator is used to combine elements, i.e., add the costs of two actions into a single cost. Typically, we can assume that this operator is associative and commutative. Lastly, in the spirit of comparing costs, we enforce that the elements are (at least) partially ordered. (A specific application can require a total order over costs to avoid finding schemes incomparable.) Finally, we enforce that there are no "negative" elements (i.e., that the sum of two elements is greater than the both summands). In practice, we generally consider measures over the non-negative real numbers, using addition as the operator and magnitude as the (total) order.

This notion of cost measure can be used to encode a variety of interesting access control metrics, including several of those noted in a recent NIST report on the assessment of access control schemes [14]. For example, costs like "steps required for assigning and dis-assigning user capabilities" and "number of relationships required to create an access control policy" can be represented using the non-negative integers. Our notion of cost measure is general enough to represent many other types of costs as well. Metrics for human work such as "personnel-hours per operation" and "proportion of administrative work to data-entry work" can be represented using the non-negative integers and a vector of two integers, respectively. Maximum memory usage can be represented using the integers (using `max` rather than addition). For applications in which multiple metrics are relevant, vectors of cost measures can be used.

Example 6. Recall the coalition-based workload described in Examples 2 and 3. When analyzing the cost of an implementation of this workload, perhaps the most important metric is human-based, since the steps that must be performed by security experts seem to be the bottleneck in the military's current system. Thus, perhaps the measure for such an evaluation is, “average administrative overhead per action.” ♦

4.4.2 *Cost Functions*

In order to calculate the total cost of a particular implementation, costs of executing the implementing scheme's individual commands and queries must be determined. Sometimes, we can generalize over entire commands or queries (e.g., creating a document requires a constant amount of I/O). In other cases, the parameters of the command or query affect the cost (e.g., adding a user to the system is more expensive for users with greater capabilities). In addition, some costs depend on elements of the state (e.g., granting access to all documents with a certain property may require checking each document, a procedure that grows in cost with the number of documents in the system). In general, the required function maps each (command, state) or (query, state) pair to an element of the relevant cost measure.

In some systems, the number of commands and queries can be quite large, with costs that are hard to generalize (e.g., commands whose costs vary depending on both the executing user and the command's parameters). This makes an access control cost function difficult to specify concisely. A continuing thread of investigation in cost functions is in making them easier to specify, and finding special cases that can be expressed more compactly. One option we are investigating is to aggregate costs over sequences of operations that are common in a particular invocation, thus specifying costs as a coarser granularity. For example, rather than assigning a cost to each command in the scheme, it may be easier to assign a single cost to the administrative task of creating a new user, assigning this user default roles, and granting this user access to shared resources.

5. SIMULATION-BASED COST ANALYSIS

Once an analyst has constructed a series of implementations of the workload in various candidate schemes, she has a “recipe” for using each candidate scheme to execute the actions needed by the application of interest. In this section, we describe a method for combining this information with each scheme's cost function to derive the cost of using each scheme in the scenario described by the workload. We note that many of the invocation mechanisms described in Section 4.2.2 are machine-like. This makes them particularly well-suited to evaluation by simulation. Thus, the techniques we describe in this section are simulation-based.

Because many of the natural ways of expressing a workload's invocational component are structurally machine-like, they can often be executed almost directly. For example, the procedure to simulate the execution of a Petri net is inherent in its description, and probability distributions are easily sampled from, leading to natural simulation algorithms. Executing a workload's invocation in this way yields a list of actions to simulate within the workload scheme. These actions must then be translated into actions in each of the implementing schemes, using the specification of each scheme's implementation of the workload. The resulting actions are executed in each scheme, accruing cost and possibly altering components of the scheme's state. The general structure of an application-aware cost evaluation simulator appears in Fig. 3.

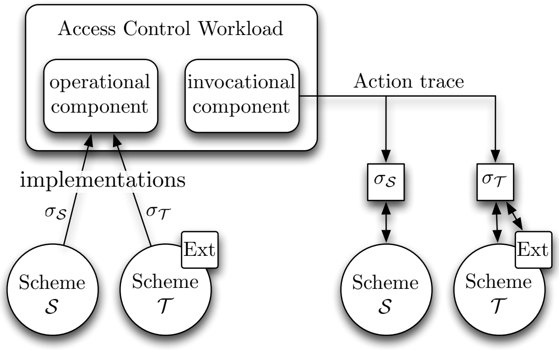


Figure 3: The structure of a modular, application-aware cost evaluation, using simulation

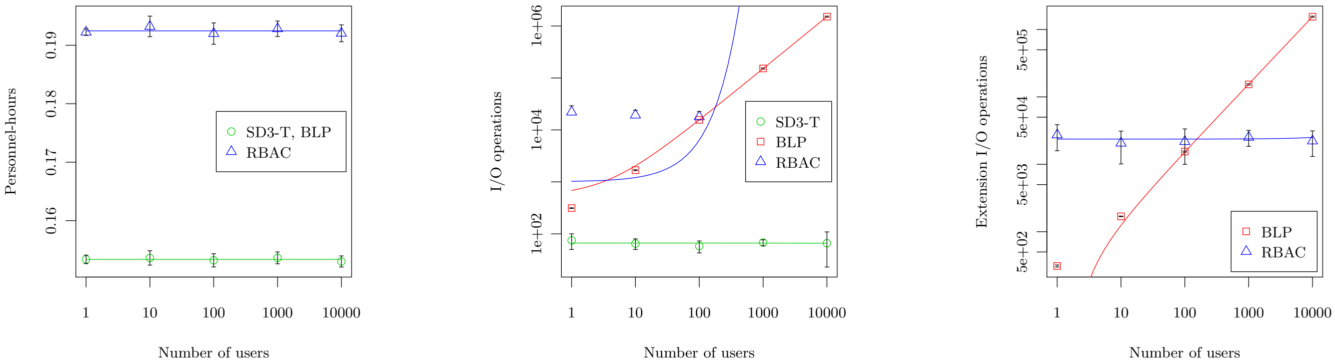
Proper methods for determining appropriate stopping conditions for a simulation is a subject of future work. In our preliminary study (Section 6), we observe the size of state elements and ensure that these sets reach a steady state. While this method results in relatively consistent simulation results in our study, more complex workloads (specifically, invocational components) will likely require more robust techniques for calculating stopping conditions.

Finally, we reiterate that the execution of the invocation component is independent of the remaining elements of the simulation, enabling the other components to view it as a black box that simply generates lists of actions to simulate. For example, although a Petri net and a multi-agent interleaved workflow system have very different specifications and meanings, they can each be used by the general analysis process without the remaining components requiring knowledge of how the list of actions is generated.

6. PRELIMINARY RESULTS

For the coalition-based application discussed in examples in previous sections, we have constructed a simplified workload. We wrote and used a simulator to evaluate the cost of implementing that workload on BLP (Bell-LaPadula scheme [4], the military's current access control solution), RBAC (role-based access control [22]), and SD3-T (an instantiation of SD3 [15] used to represent TBA, tag-based authorization [12]). We fixed the required implementation properties to coincide with the state-matching reduction, a type of implementation defined by Tripunitara and Li [25]. We found that both BLP and RBAC required extensions to implement the coalition-based workload, since neither of them is capable of storing the necessary metadata to represent the workload state. RBAC required a smaller extension because we were able to use the user-role relation to store workload data, but BLP required adding nearly a complete, separate access control scheme to its original state.

In our examination of this workload, we considered three measures: administrative personnel-hours, number of total I/O operations, and number of extenison I/O operations. We use administrative personnel-hours because it is relatively easy to quantify and reflects the amount of work that must be done by well-paid individuals to support the functional-



(a) Administrative cost (b) I/O cost (c) Extension I/O cost

Figure 4: Change in average cost per action as number of users is varied. Averages over 1,000,000 actions from initial state $|P| = 150$, $|D| = 10,000$.

ity of the access control scheme. Tasks such as managing system-wide policies, granting clearances, and creating and deleting users are all tasks that require work of administrator users. We use total I/O operations as a means of measuring the computational performance as systems get very large. This measure becomes relevant, e.g., in the context of implementations that must manipulate complex state when during execution. Lastly, we investigate extension I/O operations to capture the effect on expanding the trusted computing base of an access control scheme. Implementing an extension atop an existing access control system would require a trusted channel for communication between the original system and the extension, a cost our final measure attempts to capture.

Our simulator allows various parameters of the initial state—e.g., number of users and documents, size of policy—to be varied prior to the start of a given run. We chose a variety of start states, then ran our simulator on each to a point significantly beyond stabilization of the system. Figure 4 show charts for each of our cost measures as we vary the number of users in the simulation's initial state. These charts show averages over five runs. Error bars, where shown, indicate one standard deviation in each direction.

From the point of view of the administrative personnel-hours measure, SD3-T and the augmented BLP implementation are equally efficient, while the augmented RBAC implementation was consistently around 25% more expensive, as shown in Fig. 4a. These values do not vary with the number of users. SD3-T and BLP are equal in this measure due to the similarity in their implementations: all of the expensive actions in the augmented BLP scheme are within the extension, and its extension is very similar to the SD3-T system.

In terms of I/O operations, SD3-T is nearly always the most efficient. The cost of executing our workload remained almost constant with respect to the number of users, as seen in Fig. 4b. In contrast, BLP's I/O costs rose linearly with the number of users. RBAC's extension proved to be an expensive tradeoff. Although less information was kept outside of the original scheme, the information that was stored within the user-role relation was roughly exponential in the number of users. This resulted in much higher costs in I/O. Thus, as seen in Fig. 4b, RBAC's I/O cost grew exponentially with number of users, quickly growing in many orders of magnitude once more than 100 users were created.

Our final measure considers I/O costs associated with managing and querying extension state. Since SD3-T is sufficient to execute our workload without an extension, its costs in this category are always zero. However, an interesting relationship is seen between the augmented implementations of BLP and RBAC. Previously, BLP seemed much more feasible than RBAC due to the latter's managing a very large number of roles. However, considering extension state reveals a different scenario. As the number of users increased, Fig. 4c shows BLP's costs growing while RBAC's remain nearly constant. Thus, for systems with many users, RBAC's extension I/O costs were lower than BLP's.

Although this preliminary experiment allows us to demonstrate the usage of an application-aware evaluation framework, we identified several shortcomings in some of our simple input structures. For example, while constructing our example workload, we found ourselves wishing our invocation structure could express interleaved execution of actions by different entities. Furthermore, it is possible that using an overly strict notion of implementation resulted in our crafting suboptimal implementations. Thus, one direction of future work in developing an application-aware evaluation framework for access control is improving our techniques and structures for specifying the inputs to the analysis process. We discuss several ideas toward this end in the following section.

7. DISCUSSION

In this section, we discuss the future of application-aware analysis, including obstacles in realizing the framework as envisioned and ways in which it can be extended to the formalization of more general security workloads.

7.1 Future Obstacles

Refining Input Structures As described in Section 4, the various structures used to specify the inputs to our analysis process (e.g., invocations, implementations, cost functions) remain somewhat underdeveloped. We present the required features of each component, and in Section 6 use simple versions of each that satisfy these working requirements.

Determining the desired characteristics of more expressive, easier to specify components is an area of continuing work.

Abstraction Levels A recent NIST report [14] distinguishes three abstractions within access control: access control policies, models, and mechanisms. A policy describes the high-level requirements of a system, a model is the mathematical representation of the system, and a mechanism is the implementation that is deployed to enforce the policy. Since most of our work is done at the *model* abstraction, it would be preferable to have a deterministic way to transform a *policy* into the more concrete representation that we use for our workloads. This would make it easier for administrators to represent the system requirements appropriately, without relying on a "correct" translation from policy to model.

Implementation Non-Existence A proof that a particular implementation does not exist is typically harder to produce than a constructive existence proof. Thus, in our work so far, when discussing a lack of an implementation, we often resort to informal arguments for justification. Ideally, it would be possible to more easily prove the non-existence of of an implementation, since such proofs give higher confidence in the necessity of extending access control schemes.

Implementation Optimality The constructive nature of an implementation of a workload in a scheme leads quite naturally to the cost analysis of this scheme, as workload actions can be translated into scheme actions by the implementation. Given an access control scheme $\mathcal{S}$ and a workload W, we therefore carry out the cost analysis of a *particular* implementation of W in $\mathcal{S}$, rather than the *best* implementation of W in $\mathcal{S}$. It would be useful to develop techniques for proving the optimality of an implementation. This would enable analysts to make strong claims about the (sub-)optimality of an access control scheme for a given workload without needing to justify or defend the implementations used during their analysis. Furthermore, methods for automatically constructing implementations could enable the use of an application-aware framework by analysts without a strong understanding of constructing or proving implementations.

7.2 General Security Applications

Web security is built on public key infrastructures (PKIs) in which trusted certification authorities (CAs) provide certificates authorizing web services to assert a DNS name. This infrastructure, while typically thought of as an authentication system, can be modeled as an access control system that authorizes cryptographic keys held by web services to be bound to certain DNS names. Recently, there have been high profile compromises of CAs in the web PKI domain [8, 19]. These failures have made clear the fragility of the trust model and revocation mechanisms in the web space, and have inspired the community to examine methods both for reinforcing the system's mechanisms to prevent fraudulent certificate issuances and improving the robustness of the revocation infrastructure. However, there is considerable debate in the community regarding what the appropriate metrics for judging replacement systems should be, and how the different proposals compare under realistic conditions.

We believe that problems such as these can be represented as application-aware evaluation problems. For example, web services and their certificates can be represented within the operational component of a workload. This workload's commands could represent such operations as issuing and revoking certificates, and its queries could formalize, for example, a web user verifying the identity of a server. Candidate schemes, then, could be constructed to describe how each of these actions would be implemented in various proposed replacement architectures. The required type of implementation would need to enforce, e.g., that identities are not accepted without a certificate, and certificates are only be issued to the correct owner. The costs of individual actions could then be combined with an invocational workload component that describes the workflow of the typical web deployment, yielding a cost for each scheme. In a similar fashion, we believe that such techniques can be applied to a wide range of security problems, allowing analysts to model the application as a workload, candidate solutions as schemes, and the execution of the former using the latter as an implementation.

8. CONCLUSION

In this paper, we have motivated the need for new evaluation techniques for access control that transcend expressiveness and account for the differences between applications. We described the design goals of an application-aware analysis framework and discussed how it can be utilized to determine which access control scheme is best suited to an application. In such a framework, an application's requirements are formalized as a workload, a novel structure that enables evaluation of schemes' ability and cost to operate within the application. We presented the results of preliminary work in realizing our goal, and detail future plans for continued progress. Finally, we hypothesize that access control is not alone in its need for (or lack of) application-aware evaluation tools, and propose ways to apply these techniques to other problems in the security domain.

9. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under awards CNS–0964295 and CNS–1228697.

10. REFERENCES

[1] P. Ammann, R. J. Lipton, and R. S. Sandhu. The expressive power of multi-parent creation in monotonic access control models. *JCS*, 4(2/3):149–166, 1996.

[2] K. S. Anderson, J. P. Bigus, E. Bouillet, P. Dube, N. Halim, Z. Liu, and D. E. Pendarakis. Sword: scalable and flexible workload generator for distributed data processing systems. In *WinterSim*, pages 2109–2116, Dec 2006.

[3] Apache Shiro. `http://shiro.apache.org`.

[4] D. Bell and L. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corporation, 1975.

[5] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *TISSEC*, 6(1):71–127, 2003.

[6] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2(1):65–104, Feb 1999.

[7] A. Chander, J. C. Mitchell, and D. Dean. A state-transition model of trust management and access control. In *CSFW*, pages 27–43, 2001.

[8] I. Comodo Group. Comodo report of incident, Mar 2011. `https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`.

[9] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *SACMAT*, pages 38–47, 2005.

[10] G. R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *International CMG Conference*, pages 1263–1269, Dec 1995.

[11] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Comm. of the ACM*, Aug 1976.

[12] T. Hinrichs, W. C. Garrison III, A. J. Lee, S. Saunders, and J. C. Mitchell. Tba: A hybrid of logic and extensional access control systems. In *International Workshop on Formal Aspects of Security & Trust*, 2011.

[13] Horizontal integration: Broader access models for realizing information dominance. Technical Report JSR-04-13, MITRE Corporation JASON Program Office, 2004.

[14] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn. *Assessment of Access Control Systems*. NIST, 2006.

[15] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE S&P*, pages 106–115, 2001.

[16] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *J. ACM*, 52(3):474–514, May 2005.

[17] R. J. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *J. ACM*, 24(3):455–464, 1977.

[18] Microsoft Web Protection Library. `http://wpl.codeplex.com`.

[19] B. News. Iranians hit in email hack attack, Sep 2011. `http://www.bbc.co.uk/news/technology-14802673`.

[20] S. Osborne, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *TISSEC*, May 2000.

[21] R. Sandhu. Expressive power of the schematic protection model. *JCS*, 1(1):59–98, 1992.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, Feb 1996.

[23] R. S. Sandhu and S. Ganta. On testing for absence of rights in access control models. In *CSFW*, pages 109–118, 1993.

[24] Spring Security. `http://static.springsource.org/spring-security/site/`.

[25] M. V. Tripunitara and N. Li. A theory for comparing the expressive power of access control models. *JCS*, 15(2):231–272, 2007.

[26] U.S. Air Force Scientific Advisory Board. Networking to enable coalition operations. 2004.

[27] Q. Wang, N. Li, and H. Chen. On the security of delegation in access control systems. In *ESORICS*, pages 317–332, 2008.