An Actor-Based, Application-Aware Access Control Evaluation Framework

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
VMware, Inc.
thinrichs@vmware.com

ABSTRACT

To date, most work regarding the formal analysis of access control schemes has focused on quantifying and comparing the expressive power of a set of schemes. Although expressive power is important, it is a property that exists in an *absolute* sense, detached from the application context within which an access control scheme will ultimately be deployed. By contrast, we formalize the access control *suitability analysis problem*, which seeks to evaluate the degree to which a set of candidate access control schemes can meet the needs of an application-specific workload. This process involves both reductions to assess whether a scheme is *capable* of implementing a workload (qualitative analysis), as well as cost analysis using ordered measures to quantify the *overheads* of using each candidate scheme to service the workload (quantitative analysis). We formalize the two-facet suitability analysis problem, which formally describes this task. We then develop a mathematical framework for this type of analysis, and evaluate this framework both formally, by quantifying its efficiency and accuracy properties, and practically, by exploring an academic program committee workload.

Categories and Subject Descriptors

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

Keywords

Access control; Suitability analysis; Actor-based simulation

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 systems has focused largely on comparing the *relative expressive power* of two access control systems (e.g., [1,3,10,19,21,22,24,25]). Although expressive power is a meaningful basis for comparing access control systems, it exists only as a comparison made in absolute terms. That is, the knowledge that a system $\mathcal{Y}$ is more expressive than another system $\mathcal{Z}$ provides no assurance that $\mathcal{Y}$ is the best access control system for use within the context of a particular real-world application. It could be the case, for instance, that $\mathcal{Z}$ is *expressive enough* for a particular application and also has lower administrative overheads than $\mathcal{Y}$ would in the same situation. 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-specific metrics [13]. This position is supported by the recent introduction of application-specific access control analysis techniques [9,11].

Considering the wide availability of many diverse access control systems and the relative difficulty of designing and building new secure systems from the ground up, an interesting topic for exploration is that of *suitability analysis*. Informally, this problem can be stated as follows: *Given a description of an application's access control needs and a collection of candidate access control systems, which system best meets the needs of the application?* 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. A variant of this question was tackled by parameterized expressiveness [11], which evaluates an access control system's qualitative suitability to an application by assessing the strength of the security guarantees that it can satisfy while operating within that application. However, this type of suitability analysis provides no *quantitative* guidance in choosing the best access control solution for an applications.

In this paper, we identify and formalize the two-facet access control suitability problem, which considers both qualitative and quantitative metrics. We then propose techniques to facilitate this type of suitability analysis. As in parameterized expressiveness [11], we first formalize the notion of an access control workload to abstract the application's access control needs and the expected uses of these functionalities. Analysis then consists of two orthogonal tasks: (i) demonstrating that each candidate access control system can *safely* implement the workload, and (ii) quantifying the *costs* associated with using each candidate system. Toward carrying out such an evaluation, we develop techniques for representatively sampling from the workload's functionality, guidelines for formally specifying access control cost metrics, and a simu-

SACMAT '14, June 25–27 2014, London, Ontario, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2939-2/14/06 $15.00.
http://dx.doi.org/10.1145/2613087.2613099

lation framework for carrying out Monte Carlo-based cost analysis. In doing so, we make the following contributions:

- We formalize the *two-facet access control suitability analysis problem*, and articulate a set of requirements that should be satisfied by two-phase suitability analysis frameworks.
- We develop the first *two-phase suitability analysis framework*. We first establish whether candidate systems are expressive enough to safely implement the functionality of the workload via reduction. We then utilize a constrained, actor-based workload invocation structure to sample workload usage patterns and drive a simulation-based cost analysis that explores the expected costs of deployment.
- We evaluate our framework *formally* by proving that our simulation procedure is fixed-parameter tractable, and *practically* via a case study demonstrating how our framework can be effectively used to gain insight into a realistic scenario based on an academic conference management system.

Outline. In Section 2, we describe prior work in both access control expressiveness and simulation techniques, and summarize the tools that we use for the first phase of suitability analysis. In Section 3, we formalize the suitability analysis problem and articulate a set of requirements for suitability analysis frameworks. Sections 4 and 5 describe our approach to the second phase of suitability analysis, cost evaluation. We describe a case study investigating the use of our framework in Section 6. We then discuss the properties upheld by our framework and areas of future work (Section 7) and conclude (Section 8).

2. PRIOR WORK AND EXPRESSIVENESS

In this section, we describe related work in expressiveness analysis, including background details on the most relevant expressiveness framework, parameterized expressiveness [11]. We then discuss the shortcomings of even this expressiveness framework to motivate our approach. Finally, we discuss prior work on simulation techniques that have influenced our own cost analysis procedures.

2.1 Prior Work: Expressiveness

The formal study of access control systems began with a seminal paper by Harrison, Ruzzo, and Ullman [10]. This paper formalized a general access control model and proved that determining whether a particular access right could ever be granted to a specific subject—the so-called *safety problem*—was undecidable. Lipton and Snyder showed that in a more restricted system, safety was not only decidable, but decidable in linear time [19]. 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, in turn, led to many results investigating the relative expressive power of various access control systems (e.g., [1, 3, 17, 21, 22, 24]). Relative expressiveness analysis frameworks typically provide analysts with the tools to prove statements of the form, "System $\mathcal{Y}$ is at least as expressive as system $\mathcal{Z}$." Informally, this means that $\mathcal{Y}$ can simulate the behavior of $\mathcal{Z}$, and assures us that we can use $\mathcal{Y}$ in any scenario in which $\mathcal{Z}$ can be used. However, without formal justification for their simulation requirements, many such frameworks proved to be too relaxed, allowing almost any two reasonable systems to be shown to be equivalent: using such relaxed notions of simulation, systems in which safety is trivially decidable have been shown to simulate others in which it is undecidable [25]. Furthermore, due to various differences between notions of simulation, there have been conflicting relative expressiveness results that are difficult to reconcile [24, 25]. Thus, none of this prior work supports the comparison of access control systems with regards to their ability to perform *well* within a particular environment, a lack which was pointed out by a recent NIST technical report [13]. This has hindered the adoption of relative expressiveness analysis as a practical technique that enables analysts to choose the access control system that best meets their needs.

Parameterized expressiveness (PE) [11], by contrast, makes use of an access control *workload* to capture the requirements of the application. Expressiveness mappings (*implementations*) are then constructed between the workload and the access control systems that are candidates for accomplishing these requirements. An implementation is evaluated through the strength of the security requirements that it can preserve while satisfying the workload. In this way, systems can be compared by how well they can perform within the specific environment in which they are to be deployed.

2.2 Parameterized Expressiveness

Conducting expressiveness evaluation within PE begins by formalizing the requirements of the application and the access control systems that are candidates for accomplishing these requirements. Candidate systems are specified as state machines belonging to a particular access control *model*. Intuitively, an access control model is (i) a collection of data structures that store information pertinent to access control and (ii) a collection of procedures that expose only certain kinds of information about those data structures to an external observer. Each snapshot of the data structures in the model is an *access control state*. Each method that exposes information about the state is a *query*. An access control model differs from an arbitrary data structure because every state supports a special set of queries, the authorization queries, that define the access control policy to be enforced in that state. The access control policy for a state dictates which of all possible access control *requests* are granted.The authorization queries are denoted $auth(r)$, where r is one of the access control requests, e.g., a subject-object-right triple.

Definition 1 (Access Control Model) *An access control model $\mathcal{M}$ has fields $\langle \mathcal{S}, \mathcal{R}, \mathcal{Q}, \vdash \rangle$, where:*

- *$\mathcal{S}$ is a set of states*
- *$\mathcal{R}$ is a set of access control requests*
- *$\mathcal{Q}$ is a set of queries including $auth(r)$ for every $r \in \mathcal{R}$*
- *$\vdash$ is a subset of $\mathcal{S} \times \mathcal{Q}$ (the entailment relation)*

If $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{Q}, \vdash \rangle$, we use States($\mathcal{M}$) to denote $\mathcal{S}$ and Queries($\mathcal{M}$) to denote $\mathcal{Q}$. We use the term theory *to denote a truth assignment for all the queries in $\mathcal{Q}$. For state $s \in \mathcal{S}$, we use Th(s) (a subset of $\mathcal{Q}$) to denote the set of all $q \in \mathcal{Q}$ such that $s \vdash q$ (a convenient representation of the theory that holds at s). We use Auth(s) (a subset of Th(s)) to denote the set of all $auth(r) \in \mathcal{Q}$ such that $s \vdash auth(r)$.* $\diamondsuit$

Definition 2 (Access Control System) *An access control system $\mathcal{Y}$ has fields $\langle \mathcal{M}, \mathcal{L}, next \rangle$, where:*

- *$\mathcal{M}$ is an access control model*
- *$\mathcal{L}$ is a set of labels (also called commands)*
- *$next : States(\mathcal{M}) \times \mathcal{L} \rightarrow States(\mathcal{M})$, the transition function*

If $\mathcal{Y} = \langle \mathcal{M}, \mathcal{L}, next\rangle$, we use Labels($\mathcal{Y}$) to denote Labels($\mathcal{M}$), States($\mathcal{Y}$) to denote States($\mathcal{M}$), and Queries($\mathcal{Y}$) to denote Queries($\mathcal{M}$). The theories *of $\mathcal{Y}$ are all the theories of $\mathcal{M}$. For a finite sequence of labels $l_1 \circ \cdots \circ l_n$, we use terminal$(s, l_1 \circ \cdots \circ l_n)$ to denote the final state produced by repeatedly applying next to the labels $l_1, \cdots, l_n$ starting from state s.* $\Diamond$

The access control demands of an application are captured in an *access control workload*. The workload includes a state machine similar to an access control system that formalizes the application's required protection state, commands and queries. In addition, the workload contains a specification of the valid utilization patterns of this functionality, encoded as a set of *traces* through the system. Each trace defines an initial state and a sequence of labels that are executed.

Definition 3 (Workload) *An* access control workload *is defined by $\langle \mathcal{A}, \mathcal{T}\rangle$, where:*

- *$\mathcal{A} = \langle \mathcal{M}, \mathcal{L}, next\rangle$ is the operational component: an abstract access control system*
- *$\mathcal{T}$ is the invocational component: a set of pairs $\langle s_0, \tau\rangle$ where $s_0 \in States(\mathcal{A})$ and $\tau = l_1 \circ l_2 \circ \ldots$ is a sequence where $\forall i. l_i \in Labels(\mathcal{A})$.* $\Diamond$

The representational similarity between the workload's operational description and access control systems enables us to construct *implementations* of the workload: mapping states, labels, and queries in the workload to states, (sequences of) labels, and (procedures over) queries in the systems.

Definition 4 (Implementation) *Given a workload $\mathcal{W}$ and a system $\mathcal{Y}$, an implementation of $\mathcal{W}$ in $\mathcal{Y}$ is defined by $\langle \alpha, \sigma, \pi\rangle$, where:*

- *$\sigma : States(\mathcal{W}) \to States(\mathcal{Y})$ is the state mapping*
- *$\alpha : States(\mathcal{Y}) \times Labels(\mathcal{W}) \to Labels(\mathcal{Y})^*$, the label mapping*
- *$\pi = \{\pi_q | q \in Queries(\mathcal{W})\}$, where π_q is a function from theories for $\mathcal{Y}$ to $\{\text{TRUE}, \text{FALSE}\}$, the query mapping* $\Diamond$

Security properties that an implementation must uphold can be expressed as constraints on these mappings, and proofs are manually constructed to ensure their preservation. While most prior works propose *fixed* sets of security properties defining their particular notion of expressive power, PE describes a set of properties that can be "mixed and matched," and implementations are evaluated qualitatively by which properties they satisfy. For example, implementations can be restricted to mapping each workload label to a single system label (label atomicity), or string manipulation can be restricted to prevent the implementation from packing arbitrary data into string constants such as usernames (homomorphism).

Unfortunately, although PE evaluates access control systems in a way that takes into account the specific application and *qualitatively* evaluates suitability to that application, it does not enable analysts to *quantitatively* evaluate suitability (i.e., by considering efficiency/costs). In this work we develop a cost analysis framework which complements PE to allow both qualitative and quantitative suitability analysis.

2.3 Prior Work: Simulation Techniques

Our cost analysis procedure will need to efficiently sample representative *traces* of actions from the full description of allowed behavior within an application. For inspiration in generating access control traces, we turn to trace generation work in other domains. In the field of disk benchmarking, Ganger [6] observed 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 reinforced 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.

When formalizing the ways in which users of a system work together, it is important that one user does not execute an action which will render another user's work impossible to complete. Thus, during simulation, we must solve instances of the *workflow satisfiability problem* (WSP), a problem whose runtime complexity has been studied in the past and found to be NP-complete [26] but fixed-parameter tractable [4].

Our work is not the first to utilize Monte Carlo analysis when evaluating access control systems. For instance, Molloy et al. use Monte Carlo analysis to explore cost/benefit tradeoffs in the context of risk-based access control systems [20]. Our framework, on the other hand, uses this analysis technique to explore an arbitrary array of analyst-specified costs associated with the use of particular access control systems in the context of a given workload.

3. SUITABILITY ANALYSIS

In this section, we identify the access control suitability analysis problem and develop a set of practical requirements that solutions to this problem must satisfy.

3.1 Problem Definition

Given an access control workload, i.e., a formalization of an application's access control requirements, we postulate that assessing the *suitability* of an access control system for that application will involve two classes of measures: **expressiveness** (*qualitative* measures of the system's ability to *securely* satisfy the requirements) and **cost** (*quantitative* measures of the system's ability to *efficiently* satisfy the requirements). As such, suitability analysis is done in two phases. In the first phase, the analyst must ensure that the candidate systems are expressive enough to safely meet the needs of their application. This phase includes formalizing the candidate access control systems and the capabilities required by the application (the workload), followed by constructing—and proving security properties of—implementations that describe how the systems can safely satisfy the workload. Upon completion of this phase, the analyst will have narrowed down the list of systems to those that are expressive enough to safely operate within the application and described each system's qualitative suitability to the application. One approach to this process is presented in [11].

The notion of costs, on the other hand, requires examining ordered measures of suitability such as administrative overheads, workflow throughput, and additional storage that result from the choice of a particular candidate access control system. In the cost analysis phase, the analyst formalizes the cost measures of interest, structures used for sampling from the expected usage of the access control system, and the expected costs of actions within each system. These are then

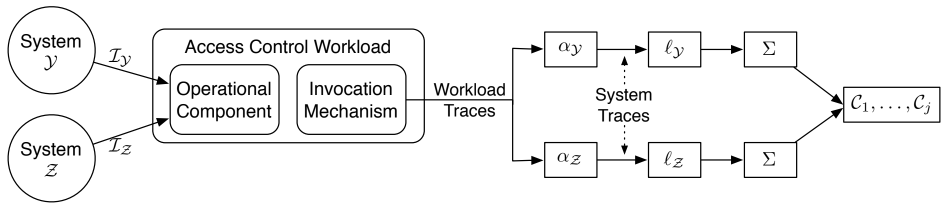


Figure 1: Overview of an application-aware analysis framework for access control

used to determine a partial order over the candidate systems describing their relative suitability to the application. More formally, these two phases describe the following problem:

Problem (Suitability Analysis) *Given an access control workload $\mathcal{W}$, a set of candidate access control systems $\mathfrak{Y}$, a set of security guarantees $\mathcal{G}$, and a set of ordered cost measures $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, determine:*

(i) the subset $\mathfrak{Y}' \subseteq \mathfrak{Y}$ of systems that admit implementations of $\mathcal{W}$ satisfying maximal subsets of $\mathcal{G}$

(ii) the subset $\mathfrak{Y}'' \subseteq \mathfrak{Y}'$ of systems that admit cost-optimal implementations of $\mathcal{W}$ relative to the lattice $\mathcal{C}_1 \times \cdots \times \mathcal{C}_m$

3.2 Solution Requirements

We now explore requirements for suitability analysis frameworks. First, we consider requirements in how representative traces through an access control workload ($\mathcal{W}$) are generated for exploration in cost analysis. Recall from Definition 3 that the workload specifies a (possibly infinite) set of traces. Exploring all possible traces during cost analysis will likely be impractical. Thus, in cost analysis, we must sample from this set in a way that selects traces that are representative of the expected behavior. Our first two requirements ensure that the framework can generate traces that accurately model the tasks carried out within an organization, and the interactions required to support and process these tasks.

Domain exploration Large applications are complex systems with subtle interactions, the emergent behaviors of which may not be captured during workload specification. It must be possible to efficiently explore many initial conditions (e.g., types of actors, operations supported, organization size, and operation distributions) to examine the effects of various levels of concurrency and resource limitation.

Cooperative interaction Tasks within large organizations typically require the interaction of many individuals. As such, suitability analysis frameworks should support operational workflows and constraints on their execution.

Next, we must ensure that the suitability analysis framework can be tuned to meet the specific needs of an application via choosing the metrics used to assess the suitability of an access control system for a given workload. This includes both the security guarantees used in expressiveness evaluation ($\mathcal{G}$) and the cost metrics used in cost evaluation ($\mathfrak{C}$).

Tunable safety There may be many different ways for some system to implement a given workload. Without enforcing structure on the implementation mapping, even the most under-expressive systems can appear to implement a workload [25]. It must be possible for an analyst to specify the security guarantees for implementations of their workload.

Tunable cost There is no single notion of cost that is sensible for use in every access control analysis [13]. A suitability analysis framework must be capable of representing many types of costs (e.g., computational, communication, and administrative), and examining multiple costs simultaneously.

Finally, we consider requirements that ensure that the suitability analysis framework remains practical to use, even for large-scale application workloads.

Tractability Steps of the analysis process that can be automated should be done so using tractable (e.g., polynomial time or fixed-parameter tractable) algorithms that remain feasible to use even for large systems.

Accuracy Since exploring all possible traces is impractical, it must be possible to approximate the expected error of costs obtained by exploring only a specific subset of traces.

These requirements guided the development of our suitability analysis framework; we will discuss our ability to achieve these requirements in Section 7.

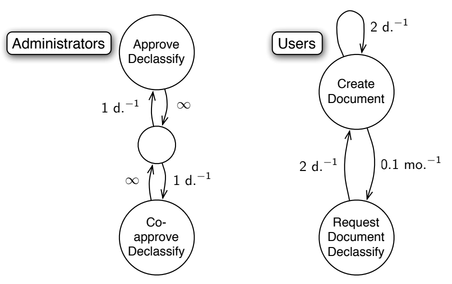
3.3 Approach Overview

Figure 1 depicts an overview of our approach to solving the suitability analysis problem. First, we carry out expressiveness analysis as described in Section 2. This consists of constructing a workload, systems, and implementations, and then proving that the implementations satisfy the desired security guarantees. Then, we augment the operational component of the workload (the set of all valid traces) with a mechanism for generating *representative* traces that satisfy desired properties. We then formalize the *cost measures*, which represent the quantitative metrics our application is sensitive to, and label the actions within each system with their respective costs. Finally, we use Monte Carlo simulation to repeatedly generate workload traces, translate them into equivalent system traces using the mappings constructed during expressiveness analysis, and execute the system traces while recording the costs of each action.

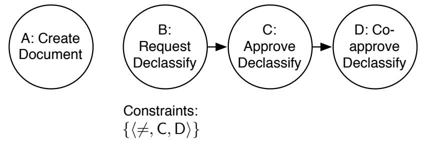
Cost analysis imposes minimal requirements on the expressiveness components that precede it in the analysis pipeline. Although we utilize PE for its flexibility, if an analyst desires some particular fixed set of security properties, other formalisms for expressiveness mappings can easily be used within our cost analysis framework. Although such an analysis will not have the benefit of PE's ability to *qualitatively* evaluate access control suitability, our cost analysis techniques for *quantitative* evaluation will work with any notion of expressiveness that satisfies several general properties:

State equivalence The expressiveness mapping includes a function that determines, given any workload state w, the equivalent system state s.

Action construction The expressiveness mapping includes a function that determines, given any system state s and workload action (label l or query q), a procedure in system



(a) Actor machines



(b) Workflow

Figure 2: Example invocational structures

labels and/or queries for effecting results equivalent to those caused by l or q.

Determinism Regardless of what labels or queries will be executed in the future, each workload state w always maps to the same system state s, and each (system state, workload action) pair maps to the same procedure in system actions.

Parameterized expressiveness clearly satisfies these properties by definition. Many other notions of expressiveness in the literature can also be rephrased to meet these requirements (i.e., by using them to construct mappings between a workload and a system, rather than between two systems). Notable examples include the state matching reduction [25], as well as the simulations defined by Chander et al. [3] and Ammann et al. [1].

4. TRACE GENERATION

The invocation mechanism of an access control workload describes valid usage of the access control system within the application being described. This is represented as a set of traces through the system's actions (labels and queries). In cost analysis, we need to sample from this set of traces in a way that is representative of the *expected* usage. We also need to do so in a way that satisfies the requirements set forth in Section 3.2. For example, *Domain Exploration* requires that we are able to alter input parameters. Implicitly in this requirement is the assumption that the trace reacts to these initial state parameters (e.g., more users typically means more frequent execution of labels and queries).

To this end, we define an extension of the invocation mechanism that utilizes the concepts of *actors* carrying out *actions* within the system. Actors are human users, daemons, and other entities that act on the access control system in ways that are described by *actor machines.* We express the various ways in which actors cooperate to complete a task using *constrained workflows.* This structure specifies dependencies between related actions, and utilizes constraints to restrict which user can execute each action. Together, these structures enable the modeling and simulation of complex and concurrent behaviors of the entities in a workload.

We now formalize *actions*, the basic units of work executed by an actor in the system. An action is a parameterized generalization of queries and labels. This allows us to specify the generic description of the action (e.g., check an access, assign a role) and separately assign the precise parameters (e.g., the specific users, documents, and roles involved). These can be assigned statically by the executing actor's behavior machine or dynamically during execution.

Definition 5 (Action) *Given an access control system, $\mathcal{Y}$, an access control* action *for $\mathcal{Y}$ is a function from a set of parameter spaces (derived from States($\mathcal{Y}$)) to the system's set of operations, and is defined as $\mathcal{A} : P_1 \times \cdots \times P_j \to$ Labels($\mathcal{Y}$) $\cup$ Queries($\mathcal{Y}$) $\cup \{\varnothing\}$, where:*

- *$P = \langle P_1, \ldots, P_j \rangle$ is the set of parameter spaces from which the actions's j parameters are drawn (e.g., the set of subjects, objects, roles).*[1] *We denote $P_1 \times \cdots \times P_j$ as P^*.*
- *$\mathcal{A} : P_1 \times \cdots \times P_j \to$ Labels($\mathcal{Y}$) $\cup$ Queries($\mathcal{Y}$) $\cup \{\varnothing\}$ maps each parameterization to a label or query in $\mathcal{Y}$, or to $\varnothing$, which designates no label or query is to be executed* ◊

To describe the behavior of actors, we employ state machines that we call *actor machines.* Each state is labeled with an action and a (possibly incomplete) parameterization. Transitions in this state machine are labeled with *rates* akin to those used in continuous-time Markov processes (e.g., [18]). We generate representative traces of actor behavior by probabilistically walking this machine, following transitions with probabilities proportional to their rates.

Definition 6 (Actor Machine) *Let $\mathcal{Y}$ be an access control system, A a set of actions from $\mathcal{Y}$, and $\mathfrak{V}$ a set of variable symbols. An* actor machine *for $\mathcal{Y}$ is the state machine $\langle S, \Phi, R \rangle$, where:*

- *S is the set of actor machine states*
- *$\Phi : S \to A \times (P_1 \cup \mathfrak{V}) \times \ldots \times (P_j \cup \mathfrak{V})$ labels each state with an action and a partial parameterization of that action (i.e., parameters can be assigned a static value or a variable to be assigned dynamically during execution)*
- *$R : S \times S \to \mathbb{R}$ is the set of transition rates* ◊

The semantics of the execution of an actor machine are as follows. R describes the rates of transitioning from one state to another. To achieve the Markov property, the time spent waiting to exit a state is exponentially distributed, with rate parameter proportional to the sum of the rates of all exiting transitions. An actor carries out a state's action upon entering the state (possibly after a pause, e.g., an action to submit a comment to a forum may pause for several minutes while the message is composed).

Example actor machines are demonstrated in Fig. 2a. In this example, we classify actors into administrators and users. Users generate documents and occasionally request a document be declassified for public consumption, while administrators approve declassification requests. Due to the labeled rates on this machine, an administrator is expected to approve a declassification request on average in one day, and roughly 10% of users request a declassification each month. Transitions labeled with ∞ occur immediately.

To describe dependencies between actions taken by one or more actors, we present the notion of a *constrained access control workflow*, which organizes the execution of actions. Formally, this structure specifies the partial order describing

[1]We typically use the first parameter of an action to represent the executing entity. For queries, this allows different responses for different queriers. For commands, this allows restrictions on the entities permitted to execute.

action dependence as well as constraints that restrict the set of users that can execute various actions.

Definition 7 (Constrained Workflow) *Let $\mathcal{Y}$ be an access control system and $\mathfrak{A}$ a set of access control actors within $\mathcal{Y}$. We say that $W = \langle A, \prec, C\rangle$ is an* constrained access control workflow *over the system $\mathcal{Y}$, where:*

- *A is the set of actions from $\mathcal{Y}$*
- *$\prec \subset A \times A$ is the partial order describing action dependencies. If $a_1 \prec a_2$, then a_2 depends on a_1, and a_2 cannot be executed until after an execution of a_1.*
- *C is the set of constraints, each of the form $\langle \rho, a_1, a_2\rangle$ (with $a_1, a_2 \in A$). Here, ρ is a binary operator of the form $\mathfrak{A} \times \mathfrak{A} \to \{\text{TRUE}, \text{FALSE}\}$. For example, $\langle \neq, a_1, a_2\rangle$ says that a_1 and a_2 must be executed by different actors.* $\Diamond$

Figure 2b displays a constrained workflow with two *tasks* (disjoint subsets of the workflow): document creation and declassification. The former is a degenerate task containing a single unconstrained action. Declassifying a document, on the other hand, requires the approval of two different administrators. The workflow allows administrators to approve declassification only after the request, and the approvals must be executed by distinct administrators.

Our *actor-based invocation mechanism* combines these components; it refines the set of traces included in an access control workload (Definition 3) using a constrained workflow, a set of actor machines, and a method for extracting the active actor machines from an access control state.

Definition 8 (Actor-Based Invocation) *Let $\mathcal{Y}$ be an access control system. We say that $I^{\mathcal{Y}} = \langle W, \mathfrak{A}, M\rangle$ is an* constrained, actor-based access control invocation mechanism *over the system $\mathcal{Y}$, where:*

- *W is a constrained workflow over $\mathcal{Y}$*
- *$\mathfrak{A}$ is the set of all actor machines*
- *$M : States(\mathcal{Y}) \to \wp(\mathfrak{A})$ is the actor machine liveness function (i.e., a function that maps access control states to the sets of actor machines active in states)* $\Diamond$

5. QUANTITATIVE COST ANALYSIS

5.1 Cost Measures

An important part of cost analysis is choosing relevant cost measures. These measures should be descriptive of what types of costs are important to the analysis, while also enabling the analyst to easily label actions with costs. For example, while "operational cost per day" may be representative of evaluation goals in industry, it is hard to assign costs in this measure to any access control action. A measure such as "average administrative personnel-hours," on the other hand, is more easily quantified and enables the same types of analyses. In this paper, we do not commit to a particular cost measure, but rather develop our framework to operate on any measure satisfying a number of simple properties.

Definition 9 (Cost Measure) *A* cost measure *is defined by the ordered abelian monoid $\mathcal{C} = \langle C, \bullet, \preceq\rangle$, where C is the set of costs, $\bullet$ is the closed, associative, commutative accrual operator over C with identity 0_C, and $\preceq$ is a partial order over C such that $\forall a, b \in C : a \preceq a \bullet b \wedge b \preceq a \bullet b$.* $\Diamond$

Definition 9 can be used to encode a variety of interesting access control measures, including several of those noted in a recent NIST report on the assessment of access control systems [13]. 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 cost measure $\langle \mathbb{N}, +, \leq\rangle$. Our notion of measure is general enough to represent many other types of costs as well. Measures for human work such as "personnel-hours per operation" and "proportion of administrative work to data-entry work" can be represented using the cost measures $\langle \mathbb{Z}^+, +, \leq\rangle$ and $\langle \mathbb{Z}^+ \times \mathbb{Z}^+, +, \leq\rangle$, respectively. Maximum memory usage can be represented using $\langle \mathbb{N}, \max, \leq\rangle$.

In order to calculate the total cost of a particular implementation, costs of executing the various actions within the implementing systems must be determined. Sometimes, the cost of any execution of an action is constant (e.g., creating a document requires a constant amount of I/O). In other cases, the parameters affect the cost (e.g., adding a user is more expensive for classes of users with greater capabilities). In addition, some costs depend on the current state (e.g., granting access to all documents with a certain property may require inspecting each document, a procedure that grows in cost with the number of documents in the system). Thus, a cost function is required to map each (action, parameterization, state) to an element of the relevant cost measure.

Definition 10 (Cost Function) *Let $\mathcal{Y}$ be an access control system, A a set of actions from $\mathcal{Y}$, and $\mathcal{C} = \langle C, \bullet, \preceq\rangle$ a cost measure. A* cost function *for $\mathcal{C}$ in $\mathcal{Y}$ is a function $\ell_{\mathcal{C}}^{\mathcal{Y}} : A \times States(\mathcal{Y}) \to C$, which maps each action and state to the member of the cost measure that best represents the costs associated with executing that action in that state.* $\Diamond$

In addition to the cost functions that are of specific interest to the analyst, our cost analysis simulation process also requires the specification of each system's *time function*. The time function is a cost function with measure $\langle \mathbb{R}, +, \leq\rangle$, describing the duration of time required to complete an access control action. This time corresponds to the duration that an actor pauses before completing an action when entering a state in the actor machine (e.g., the declassification example from Section 4).

5.2 Simulation Procedure

Once the analyst has defined the trace generation structures, a set of cost measures, and cost functions for each candidate system, she can conduct cost analysis via simulation. Our main simulation procedure, ACCostEvalSim (shown in Algorithm 1), conducts a single, randomized run of the system. First, each system's initial state is populated by sampling from a distribution provided by the analyst. An actor machine is then launched for each actor in these systems. At each time step, the clock is incremented and each actor machine is inspected for the next action, as per the execution semantics of the actor machine (Section 4). If the actor machine returns an action, an instance of the workflow satisfiability problem (WSP) [4, 26] is solved to ensure that the actor can execute the action without rendering the constrained workflow instance unsatisfiable. For independent actions (i.e., those in $\{a_1 \mid \nexists a_2.(a_2 \prec a_1)\}$), a new workflow instance is created and added to the pool of partially-executed workflows. Otherwise, the action is taken in the context of an existing workflow instance that is already in progress. After all actions for a time step are collected (and verified by WSat), their changes are effected in the state and their costs are accrued. Finally, the set of actors is adjusted according to changes in the state. Once the goal time is reached, the total costs are output.

Algorithm 1 Monte Carlo cost analysis simulation

```
Input: $\mathfrak{Y}$, set of candidate systems
Input: $\Sigma$, set of implementations ($\forall \mathcal{Y} \in \mathfrak{Y} : \sigma_{\mathcal{Y}} \in \Sigma$)
Input: $\mathfrak{C}$, set of cost measures ($\tau = \langle \mathbb{R}, +, \leq \rangle \in \mathfrak{C}$)
Input: $L$, set of cost functions ($\forall \mathcal{Y} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{Y}} \in L$)
Input: $I = \langle W, \mathfrak{A}, M \rangle$, invocation mechanism
Input: $s_0 \in States(\mathcal{W})$, start state
Input: $T_f$, goal time
Input: $t$, time step
Input: $\chi$, number of Monte Carlo runs

procedure ACCostEvalSim($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, s_0, T_f, t$)
    $\mathbf{S} \leftarrow \{\}$                         ▷ Initialize set of running AC systems
    $T \leftarrow 0$                                     ▷ Initialize master clock
    for all $\mathcal{Y} \in \mathfrak{Y}$ do            ▷ Initialize state
        $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathcal{Y}\}$
        $s_{\mathcal{Y}} \leftarrow \sigma_{\mathcal{Y}}(s_0)$          ▷ Current state of system $\mathcal{Y}$
        for all $\mathcal{C} \in \mathfrak{C}$ do
            $c_{\mathcal{C}}^{\mathcal{Y}} \leftarrow 0_{\mathcal{C}}$  ▷ Total cost of system $\mathcal{Y}$ in $\mathcal{C}$
        $\mathbf{A}_{\mathcal{Y}} \leftarrow M(s_{\mathcal{Y}})$       ▷ Set of running actor machines
        for all $\alpha \in \mathbf{A}_{\mathcal{Y}}$ do
            $T_\alpha \leftarrow 0$                      ▷ Per-actor clocks
    while $T \leq T_f$ do                                ▷ Main loop
        $T \leftarrow T + t$                             ▷ Increment clock
        for all $\mathcal{Y} \in \mathbf{S}$ do          ▷ Each AC system
            $K = \{\}$                                   ▷ Clear action list
            for all $\alpha \in \mathbf{A}_{\mathcal{Y}}$ do   ▷ Choose next actions
                if $T_\alpha < T$ then                   ▷ Check actor busy state
                    $\langle k, P_k \rangle \leftarrow$ nextAction($\alpha$)
                    if $k \neq \varnothing \wedge$ WSat($k, \alpha, P_k$) $\neq \varnothing$ then
                        $T_\alpha \leftarrow T + \ell_{\tau}^{\mathcal{Y}}(k)$          ▷ Busy state
                        $K \leftarrow K \cup \{\langle k, \alpha, P_k \rangle\}$       ▷ Save action
            for all $\langle k, \alpha, P_k \rangle \in K$ do    ▷ Compile costs
                for all $\mathcal{C} \in \mathfrak{C}$ do
                    $c_{\mathcal{C}}^{\mathcal{Y}} \leftarrow c_{\mathcal{C}}^{\mathcal{Y}} \bullet_{\mathcal{C}} \ell_{\mathcal{C}}^{\mathcal{Y}}(\sigma_{\mathcal{Y}}(\langle k, \alpha, P_k \rangle))$
                if $k$ is a command then
                    $s_{\mathcal{Y}} \leftarrow \sigma_{\mathcal{Y}}(next(s_{\mathcal{Y}}, k(P_k)))$   ▷ Update state
    for all $\mathcal{Y}' \in \mathfrak{Y}$ do
        Log $\left\langle \mathcal{Y}', c_{\mathcal{C}_1}^{\mathcal{Y}'}, \ldots, c_{\mathcal{C}_m}^{\mathcal{Y}'} \right\rangle$
```

To address the requirement of *Tractability*, we present the following theorem (proved in Appendix A), which states that ACCostEvalSim is fixed-parameter tractable (i.e., has polynomial runtime if a particular parameter is bounded). For an overview of parameterized complexity, see, e.g., [26].

Theorem 1 *Assuming that workflow constraints are restricted to* $\{=, \neq\}$ *(i.e., binding and separation of duty), the simulation procedure* ACCostEvalSim *is* fixed-parameter tractable *with the number of actions in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph).*

We also define two drivers for using this simulation procedure (see Appendix B). The first, ACCostEvalMC, randomly samples start states from a given distribution and uses the Monte Carlo technique to generate large numbers of data points, allowing the analyst to detect trends across a variety of start states. An alternative driver, ACCostEvalCI, decides how many simulation runs to conduct based on a desired confidence and the assumption of a particular distribution of costs across runs. This allows the analyst to fix a start state and repeatedly execute ACCostEvalSim until she can be confident that the results are accurate.

6. CASE STUDY

In this section, we demonstrate the suitability analysis process using our framework. This case study explores a workload based on an online management system for an academic conference, including paper submissions, reviews, and discussion. The specification of the workload is done in a group-centric secure information sharing model [14, 15], and our candidate systems include two variants of role-based access control [5, 23] and traditional UNIX user-group-other permissions [7]. The qualitative phase is conducted using parameterized expressiveness [11], and the quantitative phase is conducted using several cost measures that indicate how naturally the systems can implement the workload.

6.1 Workload and Candidate Systems

The workload's operational component (i.e., the abstract system that describes the application's requirements) is based on that used in [9] for a group-based program committee workload, as this system contains all the state, labels, and queries we need to *naturally* represent our conference workload. Users can join and leave groups, and objects can be added and removed from groups. The log of these events is used to decide whether a user can access an object. Users who perform a *strict join* to a group receive access only to objects added after they join, whereas a *liberal join* grants immediate access to all existing objects. A *strict leave* rescinds all of the user's accesses within the group; a *liberal leave* allows the user to retain access. All objects (i.e., papers, reviews, discussion messages) are added to groups via *liberal add*, and thus whether a user can access an object in a group is determined solely by the relative times the join and add took place and the variants of join/leave that was performed.

These capabilities naturally satisfy the requirements of the academic conference. When the program committee is formed, a *discussion group* is created, and each reviewer joins. Each paper is submitted to an *author group*, which holds the objects the author can see (initially, only the submitted paper). During the reviewing period, a *review group* is created for each paper, which the paper's reviewers join. Discussion about papers in contention takes place in the discussion group. When the group discusses a paper with which a reviewer has a conflict of interest, this reviewer will temporarily leave the discussion group (executing liberal leave to retain previous accesses and strict join to return without gaining access to the conflicted discussion).

We consider several role- and group-based candidate systems for implementing the conference workload. As such systems provide a level of indirection between subjects and objects, they are more likely to be effective at implementing the group-based conference workload than systems without this level of indirection (e.g., access control list systems). We choose widely-deployed candidate systems from both the industrial and consumer spaces, making them likely candidates for developing the type of system described by our conference workload. We evaluate the following candidate systems.

RBAC $RBAC_0$ is the most basic role-based access control system in the RBAC standard [23]. States contain the set of users U, set of roles R, and set of permissions P, as well as relations between them: $UR \subseteq U \times R$ describes users' membership in roles, and $PA \subseteq R \times P$ describes permissions' assignment to roles. A user u is authorized to permission p if $\exists r.(\langle u, r \rangle \in UR \wedge \langle r, p \rangle \in PA)$. Labels allow adding to and removing from all of U, R, P, UR, and PA.

Hierarchical RBAC While $RBAC_0$ grants a level of indirection between users and permissions, $RBAC_1$ includes a hierarchical structure over roles to further extend this abstraction. $RBAC_1$ includes all state elements of $RBAC_0$ as well as the role hierarchy $RH \subseteq R \times R$, a binary relation over R whose transitive closure is the *Senior* partial order (we sometimes designate the transitive, reflexive closure $\geq$). In hierarchical RBAC, a user inherits all permissions from roles junior to roles she is explicitly assigned. That is, a user u is authorized to permission p if $\exists r_1, r_2.(\langle u, r_1\rangle \in UR \wedge \langle r_2, p\rangle \in PA \wedge r_1 \geq r_2)$. Labels allow full manipulation of all state elements.

UNIX Permissions Finally, the *ugo* system [7,9] is based on the *user, group, other* system of access control in UNIX. Thus, if $RBAC_0$ and $RBAC_1$ fill the need for a commonly-used industrial standard system, *ugo* fills the role of a common consumer system. In *ugo*, objects can be associated with an owner user and group, and permissions are then granted to the user, the group, or everyone else.

6.2 Qualitative Analysis

In this section, we summarize the qualitative analysis we conducted to ensure that each of our candidate systems is capable of satisfying the conference workload. For brevity, and given the availability of other case studies discussing in-depth qualitative suitability analysis (e.g., [9,11]), in this case study we omit full proofs of expressiveness theorems (to be presented in an accompanying technical report), and consider a fixed set of implementation guarantees. For example expressiveness proofs which are sufficiently similar to our proofs of the theorems below, see [9]; for formal definitions of the implementation properties, see [9,11].

Correctness Correctness is a bare minimum requirement for an implementation in parameterized expressiveness. Intuitively, correctness says the following: a workload state's image in a system answers mapped queries exactly as the original state answers the original queries; and the same system state is reached by executing a workload action and mapping the result into the system or by mapping the initial state and executing the action's image in the system.

Weak AC-Preservation AC-preservation says that $\pi_{auth(r)}$ must map authorization request r from workload state w to system state $\sigma(w)$ directly, checking whether $\sigma(w) \vdash auth(r)$. This forces the workload and system to have identical sets of requests, which is not always the case (e.g., access matrix systems typically use subject-object-right triples, while role-based systems often use user-permission pairs). Weak AC-preservation captures the spirit of AC-preservation (ensures the use of the authorization procedure of the system) but allows us to define a request transformation function to map workload requests to system requests.

Safety Safety ensures that the intermediate states through which a system travels while implementing a single workload label do not add or remove granted requests except as determined to be necessary by the start and end states.

We implement the conference workload in $RBAC_1$ (role-based access control with role hierarchy) using techniques presented in [9]. Members who strict join a group are granted a subset of the permissions of older members, a pattern we can mirror naturally using a role hierarchy. In a way, older members "inherit" access to all added objects; new members only receive access to objects added after they joined. We thus create a chain in the role hierarchy for each group. When a g-SIS group is created, the top of the chain is created in $RBAC_1$. We name the top role of the chain after the group, and use this to correlate chains to groups.

When an object is added to a group, it is available to all members, and thus the corresponding permission in $RBAC_1$ is added to the bottom of the chain, where access to it

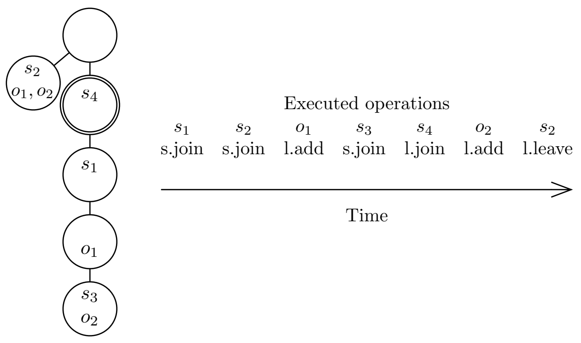


Figure 3: An example role hierarchy implementing the conference workload in $RBAC_1$

will be inherited upward (to older members). When a user *liberal* joins a group, they gain access to all existing objects, and thus we add this user in $RBAC_1$ to the top of the role chain, where she will inherit permission corresponding to each object. When a new user *strict* joins a group, they create a new "view" of the group, since they are not authorized to any existing objects. Thus, we create a new $RBAC_1$ role (named arbitrarily) and link it to the bottom of the group's chain.

When a user *strict* leaves a group, she is removed from any roles in the group's role chain, losing access to all objects in the group. When a user *liberal* leaves a group, on the other hand, she should retain access to the current set of objects. Thus, we create an *orphan role*, which does not inherit any permissions from other roles, and from which no users inherit permissions. Then, the leaving user is added to the orphan role, and the role is granted access to the group's objects to which the user currently has permission. Then, when the user is removed from the main role chain, she does not lose accesses. We give a demonstrative example of the role hierarchy structure in Fig. 3.

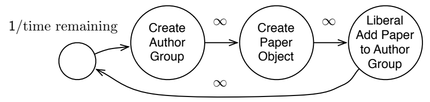
Using this technique, we can implement the conference workload in $RBAC_1$ while preserving correctness, weak AC-preservation, and safety.

Theorem 2 *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in* $RBAC_1$.

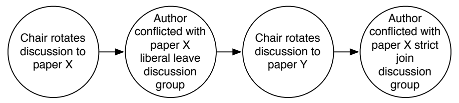
To implement the workload in $RBAC_0$ (role-based access control without role hierarchy), we follow the same procedure, but store the role hierarchy encoded in role names. This expands the set of roles to include a role named for every path through the logical hierarchy in the downward direction. Thus, if in $RBAC_1$ we would store a hierarchy that says $A \geq B$, $B \geq C$, and $A \geq D$, we represent this in $RBAC_0$ with roles $\{A, B, C, D, AB, ABC, AD, BC\}$. For every role r a user would be assigned to in $RBAC_1$, she will be assigned to each role starting with r in $RBAC_0$. In the previous example, if $\langle u, A\rangle \in UR$ in $RBAC_1$, then in $RBAC_0$ this maps to $\{\langle u, A\rangle, \langle u, AB\rangle, \langle u, ABC\rangle, \langle u, AD\rangle\} \subset UR$ in $RBAC_0$. This allows us to implement the conference workload in $RBAC_0$ while preserving our chosen implementation guarantees.

Theorem 3 *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in* $RBAC_0$.

Finally, although *ugo* has the inherent disadvantage that each object is owned by only a *single* user and group, it can implement the conference workload by mapping a workload object assigned to multiple groups to an object with a single group owner. This group then represents all groups with authorization and includes as members all users with access.



(a) Author's submit phase actor machine


Chair rotates discussion to paper X
Author conflicted with paper X liberal leave discussion group
Chair rotates discussion to paper Y
Author conflicted with paper X strict join discussion group

(b) Conflict-of-interest workflow

Figure 4: Case study invocational structures

Of course, this incurs a storage penalty; the magnitude of this overhead will be explored in quantitative analysis.

Theorem 4 *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in ugo.*

6.3 Quantitative Analysis

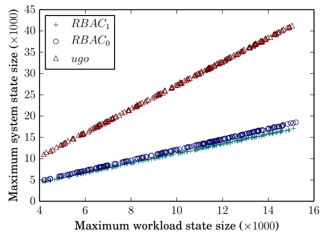
To perform simulation-based cost analysis, we formalized actor machines for the conference program chair, authors, and reviewers, as well as workflows that describe how these actors interact. These instantiations of the structures defined in Section 4 allow us to describe the usage of the workload system (which, recall, is essentially the system of the program committee workload in [9] with a different expected usage).

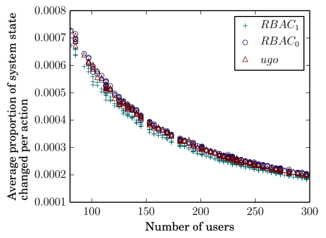
The actors in our system are a program chair, a set of reviewers, and a set of authors. Each paper is assigned three reviewers, and each reviewer is assigned nine reviews, and thus we have three times as many authors as reviewers (without loss of generality, we assume that one author is registered to submit each paper). The program chair is responsible for administrative tasks such as creating groups (the discussion group and each paper's review group), assigning reviews to reviewers, copying the submitted papers into their respective review groups, and rotating through the submitted papers during discussion. The program chair also transitions between the phases of the simulation, which determines the actions that the other actors can execute at any particular time. The phases proceed in the following order:

1. ***Create*** Chair creates the discussion group
2. ***Recruit*** Chair adds the reviewers to the discussion group
3. ***Submit*** Authors create author groups and submit papers
4. ***Review*** Chair creates review groups and adds assigned reviewers; reviewers add reviews to the review groups
5. ***Discuss*** Chair rotates discussion between various papers; reviewers add comments in the discussion group; conflicted reviewers leave during discussion
6. ***Notify*** Chair adds review summaries to author groups; authors read their summaries

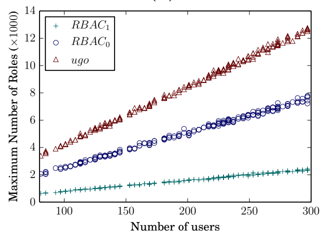
Formally, actor machines include the actions from all phases, and the chair uses workflows to ensure that only the current phase's actions are enabled. For simplicity, we often consider them as separate actor machines between which the actor transitions. Figure 4a demonstrates an example single-phase actor machine, and Fig. 4b demonstrates a particular workflow task, the conflict-of-interest workflow.

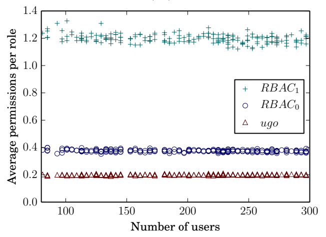
To conduct cost analysis, we built a Java implementation of ACCostEvalMC to simulate the conference workload. We repeated the simulation for 200 runs, randomly selecting the number of authors and reviewers (preserving the proportion of 3 times as many authors as reviewers).



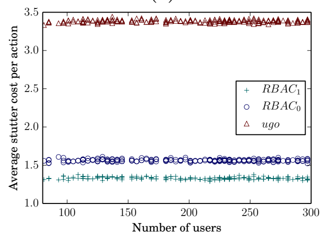


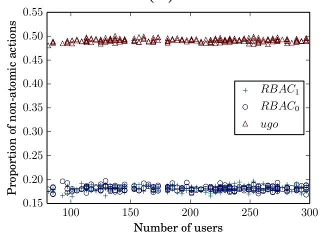
(a) (b)





(c) (d)





(e) (f)

Figure 5: Conference workload cost analysis using ACCostEvalMC and 200 runs

In Fig. 5a, we compare the maximum system state size to the size of the equivalent workload state, demonstrating the storage overhead needed to utilize each system. While the role-based schemes use a small amount of additional state, *ugo* requires several times the storage of the workload. This is due to *ugo*'s restriction that each object is owned by a single group; an object that should be accessed by multiple groups must be owned by a combined group which contains all the members of the originals. Figure 5b shows that a similar *proportion* of each system's state changes on average for each action executed, but given the larger storage required by *ugo*, this system will require much greater I/O as well.

We compare the number of users to the number of roles (groups in *ugo*) created in each system in Fig. 5c. $RBAC_0$ uses many extra roles to simulate hierarchy information, and *ugo* creates even more since each object is owned by only a single group. However, even $RBAC_1$ uses several times as many roles as there are users in the system, potentially indicating a poor fit from all three systems, as the administrative value of using roles is reduced when the the number of roles exceeds the number of users [27]. As indicated in Fig. 5d, roles in all three systems are particularly permission-sparse, averaging 1.2, 0.4, and 0.2 permissions per role in $RBAC_1$, $RBAC_0$, and *ugo*, respectively. In particular, $RBAC_0$ and *ugo* utilize many additional roles to store simulated hierarchy information, and many of these roles are never assigned permissions.

In Fig. 5e, we investigate the average number of *stutter steps* per action, or the average number of system commands that must be executed to simulate each workload command. $RBAC_1$, $RBAC_0$, and *ugo* must execute, on average, 1.3, 1.6, and 3.4 actions (respectively) for each workload action simulated. Furthermore, as shown in Fig. 5f, 18% of workload actions incur some stuttering in the role-based systems, and 49% incur stuttering in *ugo*. In scenarios where multiple users will be interacting with the system, this loss of atomicity necessitates the incorporation of an additional locking layer to ensure the system is not accessed in an inconsistent state.

6.4 Summary of Findings

The preceding case study shows that, under the lens of several cost measures, $RBAC_1$ is a better choice than $RBAC_0$ or *ugo* for implementing the conference workload, due to its native support for role hierarchies, a structure that can mimic the pattern of authorized requests common in the workload. However, even $RBAC_1$ utilizes a large number of permission-sparse roles, indicating that even it may be a tenuous fit for the workload. More importantly, though, our case study demonstrates that the concepts of our two-facet suitability analysis framework can be applied to a realistic workload and evaluate access control systems that are common in practice with respect to that workload. Our simulation procedure allows us to easily determine the overheads of using each system, and with an average runtime of around eight minutes per five-month simulation run, does so efficiently. The simulation procedure is also trivially parallelizable (since each run is independent), further enforcing its feasibility.

7. DISCUSSION AND FUTURE WORK

Requirements, Redux In Section 3.2, we outlined requirements to guide the development of our suitability analysis framework. We now discuss the degree to which each requirement was met. The *Domain Exploration* requirement is addressed by our workload formalism and our Monte Carlo simulation procedure: the former allows the analyst to specify a broad range of workloads, while the latter enables cost analysis over many workload instances. *Cooperative Interaction* is met by combining our invocation formalism with the WSP solver leveraged by ACCostEvalSim: constrained workflows articulate the ways in which cooperation must be carried out, while the use of actor graphs and the WSP solver ensures the generation of compliant traces. Although this paper makes use of a fixed set of implementation guarantees to define implementation safety, this is not mandatory. Proofs of safety are carried out manually, allowing any notion of safety to be used and providing *Tunable Safety*. Section 5 demonstrated that our notion of cost measure is capable of representing a wide range of system- and human-centric costs and thus provides *Tunable Costs*. Supporting multi-user workflows is seemingly at odds with the *Tractability* requirement, as the workflow satisfiability problem has been shown to be NP-complete [26]. However, the proof of Theorem 1 makes use of recent results [4] to show that ACCostEvalSim is fixed-parameter tractable in maximum task length (typically a small constant). More concretely, simulating each 5-month period in our case study took, on average, around 8 minutes. The *Accuracy* requirement is addressed in Appendix B, where we discuss how to calculate confidence intervals for point estimates of cost. In conclusion, the analysis framework developed in this paper meets each of the desiderata outlined in Section 3.2, and provides a flexible, efficient, and precise mechanism for analyzing instances of the access control suitability analysis problem.

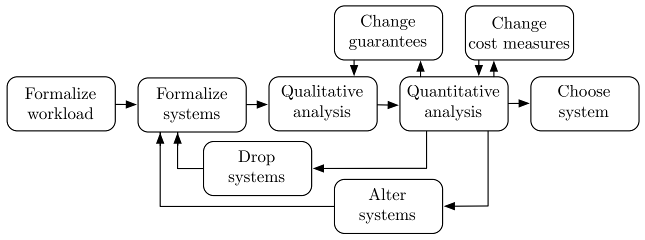


Figure 6: A possible two-facet, suitability analysis workflow supported by our framework

Unified Workflow This work explores a qualitative analysis in which we fix the security guarantees that implementations are required to preserve, and a single instance of quantitative analysis revolving around the implementations constructed during the fixed qualitative analysis. While this is an appropriate and demonstrative case study that supports the practical usefulness of our framework, in practice, analyses may be much more complex. For instance, qualitative analysis may yield very secure implementations (i.e., those that preserve very strict requirements), but quantitative analysis may reveal that all of these implementations are infeasibly inefficient. As a result, although a system may be *capable* of admitting an implementation satisfying stronger security guarantees, it may not necessarily be worth the additional cost to do so. The trade-off between strictness of security guarantees and efficiency became apparent in a more detailed case study that we conducted in prior work [9]. In this case, we chose to conduct cost analysis using the strongest expressiveness guarantees that allowed a feasible (i.e., quadratic or better) implementation using each candidate system. In general, this trade-off may inspire an analyst to revisit any number of the inputs to the analyses. For example, one may consider multiple implementations that preserve a range of weaker security guarantees and higher efficiencies. One might also consider introducing additional candidate systems. Thus, a practical analysis may be less linear than our case study and incorporate more backtracking, as in, e.g., Fig. 6. The techniques presented as part of our two-facet suitability analysis framework are general enough to naturally support such complex analysis workflows.

Future Work We have identified several important directions for future work in suitability analysis. Small changes in an access control system can be the difference between being able or unable to implement a workload safely [11] and/or efficiently [9]. To maximize reuse of existing, trusted systems, we are developing methods that allow an analyst to make small tweaks to a system that are *safe* (i.e., that preserve its desirable security properties and capabilities) while enabling greater expressiveness and/or efficiency. We have already shown that one particular method of tweaking systems is safe with respect to strong security guarantees, and have used this method to increase systems' expressiveness, allowing them to satisfy new workloads [8]. We are working to generalize this technique, formalize additional methods of tweaking, and prove safety for a range of security guarantees.

Encouraged by our success in using parameterized expressiveness, we are also working to extend the core concept of this expressiveness framework by defining granular sets of implementation guarantees that are more closely grounded in access control application. Rather than providing guarantees that prevent certain, somewhat arbitrary types of abuse, we aim to provide guarantees that reflect what an implementation mapping can do without violating the assumptions inherent in the application's usage of the chosen system.

Finally, this paper focuses on a form of the suitability analysis problem that is specific to access control. However, we believe that a more general formulation could enable better understanding of the trade-offs between the formal requirements and practical costs of solutions to a wider range of security problems.

8. CONCLUSION

Historically, most work regarding the formal analysis of access control systems has focused on expressive power, yielding a meaningful view of a system's capabilities but an incomplete view of its suitability for any particular application. In contrast, this work formalizes the *suitability analysis problem* to address both expressiveness and efficiency, and presents a methodology for application-specific evaluation of access control systems' suitability.

We show that this methodology satisfies a number of formal requirements for suitability analysis frameworks, covering flexibility, tractability, and accuracy. Furthermore, we present a case study that provides practical validation of the applicability of our methods to a realistic access control problem. Several access control systems (two role-based systems and traditional UNIX permission bits) are shown to be capable of implementing an academic conference workload with uniformly strong expressiveness guarantees but varying degrees of efficiency, enforcing the importance of measures aside from expressiveness and the significance of the suitability analysis problem.

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

9. 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] Kay S. Anderson et al. Sword: scalable and flexible workload generator for distributed data processing systems. In *Winter Simulation Conference (WSC)*, pages 2109–2116, Dec 2006.

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

[4] J. Crampton, G. Gutin, and A. Yeo. On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM TISSEC*, 16(1), 2013.

[5] David F. Ferraiolo et al. Proposed nist standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.

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

[7] S. Garfinkel, G. Spafford, and A. Schwartz. *Practical UNIX and Internet Security*. NIST special publication: Computer security. O'Reilly Media, 2003.

[8] W. C. Garrison III, A. J. Lee, and T. L. Hinrichs. The design and demonstration of an actor-based, application-aware access control evaluation framework. Technical Report arXiv:1302.1134, Feb 2013.

[9] W. C. Garrison III, Y. Qiao, and A. J. Lee. On the suitability of dissemination-centric access control systems for group-centric sharing. In *CODASPY*, 2014.

[10] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *CACM*, 19(8):461–471, Aug 1976.

[11] Timothy L. Hinrichs et al. Application-sensitive access control evaluation using parameterized expressiveness. In *IEEE CSF*, June 2013.

[12] J. Hromkovic. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer-Verlag, Berlin, Heidelberg, 2010.

[13] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn. *Assessment of Access Control Systems*. National Institute of Standards and Technology, 2006.

[14] Ram Krishnan et al. Group-centric secure information-sharing models for isolated groups. *ACM TISSEC*, 14(3):23, 2011.

[15] Ram Krishnan et al. Foundations for group-centric secure information sharing models. In *ACM SACMAT*, pages 115–124, 2009.

[16] A. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 2006.

[17] 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.

[18] T. M. Liggett. *Continuous Time Markov Processes: An Introduction*. Graduate Studies in Mathematics Series. American Mathematical Society, 2010.

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

[20] I. Molloy, P.-C. Cheng, and P. Rohatgi. Trading in risk: using markets to improve access control. In *NSPW*, 2008.

[21] S. Osborne, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM TISSEC*, 3(2):85–106, May 2000.

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

[23] Ravi S. Sandhu et al. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

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

[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] Q. Wang and N. Li. Satisfiability and resiliency in workflow authorization systems. *ACM TISSEC*, 13(4), 2010.

[27] Dana Zhang et al. RoleVAT: Visual assessment of practical need for role based access control. In *ACSAC*, pages 13–22, Dec 2009.

APPENDIX

A. PROOF OF THEOREM 1

Theorem 1 *Assuming that workflow constraints are restricted to $\{=, \neq\}$ (i.e., binding and separation of duty), the simulation procedure* ACCostEvalSim *is* fixed-parameter tractable *with the number of actions in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph).*

PROOF Our proof is by observation of Algorithm 1. The first loop (**for all** $\mathcal{Y} \in \mathfrak{Y}$) handles assignments and initializations. The final loop (**for all** $\mathcal{Y}' \in \mathfrak{Y}$) outputs results. The main loop, then, contains all of the computationally intensive code.

The expensive section of the algorithm starts after several nested loops, adding multiplicative factors for number of time steps (T_f/t), number of systems ($|\mathfrak{Y}|$), and number of actors. The steps with computational overhead are nextAction, which polls an actor machine for the next action, and WSat, which calculates whether a particular action can be taken by an actor without causing any workflow instances to become unsatisfiable (i.e., WSat solves an instance of the WSP problem).

By previous work [26], WSP can be solved in $\mathcal{O}(C \cdot A^{\alpha})$, where C is the number of constraints, A is the maximum number of actors, and α is the number of steps in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph). This greatly exceeds nextAction, which executes a single step in a continuous-time probabilistic machine (polynomial in actor machine size). Thus, the dominant factor in the complexity of Algorithm 1 is $\mathcal{O}\big(S \cdot C \cdot T \cdot A^{\alpha+1}\big)$, where $S = |\mathfrak{Y}|$ is the number of systems and $T = T_f/t$ is the number of time steps to simulate. Since T is an input, this means the algorithm is pseudo-polynomial in T and FPT in α. Since FPT is considered to be a generalization of pseudo-polynomial time [12], we refer to the complexity of Algorithm 1 as FPT, thus meeting our definition of tractable. □

B. SIMULATION DRIVERS

Algorithm 2 Monte Carlo driver for ACCostEvalSim

```
Input: $\mathfrak{Y}$, set of candidate systems
Input: $\Sigma$, set of implementations ($\forall \mathcal{Y} \in \mathfrak{Y} : \sigma_{\mathcal{Y}} \in \Sigma$)
Input: $\mathfrak{C}$, set of cost measures ($\tau = \langle \mathbb{R}, +, \leq \rangle \in \mathfrak{C}$)
Input: $L$, set of cost functions ($\forall \mathcal{Y} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{Y}} \in L$)
Input: $I = \langle W, \mathfrak{A}, M \rangle$, invocation mechanism
Input: $\Pr(s)$, probability distribution over start states
Input: $\chi$, number of Monte Carlo runs
Input: $T_f$, goal time
Input: $t$, time step

procedure ACCostEvalMC($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \Pr(s), \chi, T_f, t$)
    for all $[1, \chi]$ do                              ▷ Monte Carlo loop
        $s_0 \leftarrow$ random sample from $\Pr(s)$
        ACCostEvalSim($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, s_0, T_f, t$)
```

Since ACCostEvalSim executes only a single run of the simulation, in this appendix we present two drivers for using this simulation procedure for different analysis goals.

Algorithm 2 presents ACCostEvalMC. This driver utilizes the Monte Carlo technique; it calls ACCostEvalMC repeatedly, each time randomly sampling a start state from the given distribution. This allows the analyst to generate a large number of data points across a predefined pattern of start states, which makes it particularly effective in detecting trends across various start states. For example, in Section 6, we randomly choose a number of users in the system for each run, allowing us to see the effect this parameter has on the costs of using each system.

Because the repeated execution in ACCostEvalMC contributes to the complexity of the full analysis by only an additional pseudo-polynomial factor, ACCostEvalMC, like ACCostEvalSim is in FPT.

Corollary 5 *Under the same assumptions as Theorem 1, the simulation procedure* ACCostEvalMC *is in* FPT.

PROOF The driver ACCostEvalMC calls ACCostEvalSim χ times. Thus, the runtime complexity of ACCostEvalMC is a factor of χ greater than that of ACCostEvalSim. Since χ is an input, this contributes an additional pseudo-polynomial factor over the runtime complexity of ACCostEvalSim, and thus ACCostEvalMC is in FPT. □

Algorithm 3 Confidence-bounding driver for ACCostEvalSim

```
Input: $\mathfrak{Y}$, set of candidate systems
Input: $\Sigma$, set of implementations ($\forall \mathcal{Y} \in \mathfrak{Y} : \sigma_{\mathcal{Y}} \in \Sigma$)
Input: $\mathfrak{C}$, set of cost measures ($\tau = \langle \mathbb{R} \times \text{time}, +, \leq \rangle \in \mathfrak{C}$)
Input: $L$, set of cost functions ($\forall \mathcal{Y} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{Y}} \in L$)
Input: $I = \langle W, \mathfrak{A}, M \rangle$, invocation mechanism
Input: $s_0$, start state
Input: $T_f$, goal time
Input: $t$, time step
Input: $u \in (0, 1)$, desired confidence level
Input: $v \in (0, 1)$, desired tolerance

procedure ACCostEvalCI($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, s_0, T_f, t, u, v$)
    $n \leftarrow \emptyset$
    while $t_{|n|-1,1-u/2}\sqrt{\frac{S^2(n)}{|n|}} > v \cdot \bar{X}(n)$ do
        $n \leftarrow n \cup$ ACCostEvalSim($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, s_0, T_f, t$)
```

In the interest of satisfying the *Accuracy* requirement, we also present a second driver, which allows the analyst to achieve an intended confidence in the cost value generated for a particular start state. Using this approach, we can decide the number of simulation runs to conduct based on a desired confidence and the assumption of a particular distribution of costs across runs, terminating when a satisfactory confidence is reached. For example, assuming a normal distribution of costs across runs, we can use the fixed-sample-size procedure for point estimate of a mean [16], which says that the confidence interval for a mean is:

$$\bar{X}(n) \pm t_{|n|-1,1-\frac{\alpha}{2}}\sqrt{\frac{S^2(n)}{|n|}}$$

where $\bar{X}(n)$ is the sample mean, $\frac{S^2(n)}{|n|}$ is the sample variance, and $t_{\nu,\gamma}$ is the critical point for the t-distribution with ν degrees of freedom. The resulting range is an approximate $100(1-\alpha)$-percent confidence interval for the expected average cost of the system. During simulation, we repeatedly calculate the confidence interval for incrementing n, terminating when a satisfactory confidence is reached. For example, assuming we desire a 90-percent confidence interval of no more than 0.1 of the mean, we run the simulation repeatedly until:

$$t_{|n|-1,0.95}\sqrt{\frac{S^2(n)}{|n|}} \leq 0.1\bar{X}(n)$$

Algorithm 3 demonstrates ACCostEvalCI, which uses this approach to execute ACCostEvalSim until a desired confidence is reached, rather than executing for a fixed number of runs.