Practical Misconfiguration Identification in Access-Control Systems

Chad Spensky[*]

University of North Carolina
Chapel Hill, NC, USA `cspensky@cs.unc.edu`

August 29, 2010

Abstract

We develop an approach for identifying accesses that are not permitted by implemented policy but that share similarities with accesses that have been allowed. Such accesses are indicative of potential access-control policy misconfigurations; identifying the misconfigurations allows administrators to resolve them before they interfere with the use of the system. Our work improves on previous research by supporting a desired balance between the accuracy of its misconfiguration predictions and the amount of intended policy that it uncovers, and by removing the need to hand-tune various parameters. Our method introduces computational challenges that we address through the development of a novel algorithm. We revisit the methodology used to evaluate misconfiguration prediction algorithms in the past, and show that it substantially overestimates the benefits of such algorithms in real systems, owing to its tendency to reward predictions that can be deduced to be redundant. Finally, we go on to show that such deductions can be incorporated into the prediction algorithm to recover its benefits to a large extent.

1 Introduction

Access-control policy generally exhibits patterns across users and resources they access, owing to the use of groups and roles (perhaps only implicitly) in the creation of policy. These patterns are evidenced in the accesses that are allowed in the system. As such, if a user is permitted access to most of the same resources that other users access, then the few exceptions might represent *misconfigurations*, i.e., potential accesses that are consistent with *intended* policy but that are denied by the policy actually *implemented* in the system. With a few exceptions (e.g., [6, 10]), these kinds of misconfigurations have not been widely studied; yet in many environments, such as in the context of health information systems, eliminating misconfigurations that erroneously deny access is critical. Even when the cost of erroneously denying a single access is not as high, repeatedly denying access can severely inhibit the usability of an access-control system, and thus encourage users to circumvent it.

Bauer et al. [6] demonstrated that access patterns can be leveraged to predict misconfigurations and to infer what administrators should be consulted to resolve them, prior to these misconfigurations interfering with attempted accesses. In particular, they demonstrated a method for analyzing access logs and parameter settings for this method to yield a reasonable balance between *accuracy*, i.e., the fraction of predicted misconfigurations that were consistent with intended policy, and *benefit* (or coverage), i.e., the fraction of

[*]This research was conducted jointly with Lujo Bauer, Yuan Liang (Carnegie Mellon University) and Michael K. Reiter (UNC Chapel Hill).

intended policy that was uncovered through predicted misconfigurations. Due to the dearth of publicly available matching access and policy datasets, Bauer et al. demonstrated the success of their technique on only a single dataset and, more to the point, could quantify benefit and accuracy for various parameter settings only for that dataset. A manager who is considering employing their techniques in a different setting, however, has little assurance that the same parameter settings will yield similar results in her system.

The first contribution of this paper is to recast their method in a way that can be generalized to other settings and that, in particular, frees an administrator from needing to tune parameters to achieve a desired balance between accuracy and benefit in a new setting. Our method enables an administrator to chose a desired balance β, and ensures that $\frac{Benefit}{Accuracy} \approx \beta$ with no additional tuning. The difficulties of achieving this with the original Bauer et al. approach derive from its use of *association rule mining* [1], by which the access logs are analyzed to extract *association rules* of the form $x \Rightarrow y$, where x and y denote sets of resources. Informally, such rules mean that a user with access to x typically has access to y, as well, and so would be used to predict that y should be accessible to users who can access x. This approach is parameterized by the required *support* for, or fraction of all users with access to, x and y, and the required *confidence* of the rule, i.e., the fraction of users with access to x who also can access y. Rules are used as predictors of misconfigurations only when their confidence and support exceed these thresholds. While increasing the required confidence or support should generally increase accuracy and decrease benefit, how to incrementally adapt these two parameters to ensure $\frac{Benefit}{Accuracy} \approx \beta$ and, subject to that constraint, maximize the benefit and accuracy that result is an unresolved question.

To address this problem, in this paper we adopt an approach that combines confidence and support into a single parameter called *predictive accuracy* and uses a Bayesian framework to determine the contributions of support and confidence to the expected accuracy of a rule [23]. Collapsing support and confidence into a single parameter allows us to rank rules according to predictive accuracy and then to issue predictions in this order. As we will see, this provides a way to ensure $\frac{Benefit}{Accuracy} \approx \beta$ while providing good benefit and accuracy, but introduces a computational challenge, as computing predictive accuracy in a naive fashion is computationally intensive. To mitigate these overheads, we have adapted known association rule-mining data structures to incorporate the additional requirements for calculating predictive accuracy. In doing so, we have developed a novel algorithm for incrementally and efficiently updating our data structure after each new access. We expect that this algorithm may be useful in many different settings where the input to an association rule mining algorithm is made available only incrementally.

A final and, we believe, significant contribution of this paper is in revising the methodology for evaluating benefit, in particular, to provide a more realistic view of the benefits of misconfiguration prediction in a real system. Prior evaluations considered predictions that were made only on the basis of observed accesses. They permitted every prediction that matched intended policy to contribute to the measured benefit (and, to a lesser extent, the accuracy), irrespective of the fact that some of these predictions were also consistent with already implemented policy — and so were not misconfigurations at all. In this paper, we develop an evaluation framework that utilizes the implemented policy deducible from observed accesses and correct predictions to eliminate future predictions that are already implemented from contributing to benefit and accuracy. In doing so, we show that the actual benefits of misconfiguration prediction as cast in previous work are substantially overestimated. Fortunately, we also show that by making this deducible policy available to the prediction engine, we can recover much of the benefit and accuracy while retaining the target ratio β for $\frac{Benefit}{Accuracy}$.

To summarize, the contributions of our paper are the following. First, by utilizing a measure called *predictive accuracy*, we demonstrate an approach to mining association rules to ensure $\frac{Benefit}{Accuracy} \approx \beta$ for a provided parameter β, across a broad range of access patterns and underlying policies, and without requiring hand tuning of parameters for different datasets. Second, we devise a new, general-purpose algorithm for incrementally computing the best rules based on predictive accuracy, as new accesses arrive in the system. Third, we introduce a new evaluation methodology for misconfiguration prediction that provides a more realistic measure of the benefit that it offers. We show through this new evaluation framework that previous measurements of benefit will not be realized in practice, but that by incorporating policy that is deducible from past accesses into the prediction engine, benefit can largely be recovered. As such, we show that misconfiguration prediction can still provide substantial benefit, but primarily when it has more access to policy information than previous works considered.

2 Related Work

Several related works use data-mining or machine-learning techniques to analyze access-control policies. Firewalls were an early target for automated policy analysis, and a number of tools were developed for the empirical analysis of firewall policies (e.g., [5, 18, 27, 2, 29]). These tools use either machine-learning or static-analysis techniques, and typically enable an administrator to verify that a set of policies is consistent, or to verify that a policy obeys desired properties. Another approach incorporates such techniques in policy-specification tools, which use the output of the analysis directly to help an administrator specify policy that meets desired goals [14]. More closely related are works that use rule mining or Bayesian inference to analyze router policies with the intention of automatically finding misconfigurations (e.g., [17, 11, 16]). Similarly to firewall analysis, these approaches take as input configuration files and detect discrepancies between those configurations, e.g., user accounts without passwords, or router interfaces using private IP addresses. These works differ from ours in a number of ways, perhaps most notably in that they focus on finding inconsistencies in static policies. In contrast, we analyze policy as it is revealed in accesses over time, which gives rise to our analysis of policy in an incremental fashion, the basis for several of our innovations.

Our work is also similar to that of Das et al., who analyze access-control policy for file servers to detect inconsistencies between the permissions given to users who appear to be peers [10]. Das et al.'s system takes as input both low-level file-system policy (which user can access which file or directory) and metadata, such as group memberships information separate from the low-level policy, and identifies misconfigurations that either deny legitimate accesses or allow erroneous ones. In contrast to this work, we focus primarily on misconfigurations that prevent legitimate accesses, and our algorithm does not require access to policy or metadata sets other than what can be observed from a sequence of accesses. Similarly to Bauer et al.'s approach, the performance of Das et al.'s system depends on hand-tuning certain parameters; a main focus of our work is to render such tuning unnecessary.

Our approach to detecting misconfigurations has some similarity to role mining (e.g., [24, 15, 19, 20]), which seeks to distill from a low-level policy a collection of roles that could be used to represent the same policy more abstractly. Since the goal of role mining is to find a better representation of policy that exists, role mining algorithms take as input a whole policy, rather than processing a possibly partial policy incrementally, as we do. Also, the specific goal in role mining is to find commonalities between users that may be indicative of shared role membership, while in our approach we focus on the inconsistencies between users to detect misconfigurations and cause the policy to be amended.

A significant difference between our approach and related work is that we focus on a setting in which we learn policy incrementally, as accesses occur in a system, rather than having policy available to our tool at the beginning. This introduces more stringent performance requirements and led to our adaptation of existing algorithms for association-rule mining to perform well in our context. More specifically, we devise a new algorithm for predicting misconfigurations based on their predictive accuracy that efficiently assimilates new information on a per-access basis, and thus at a finer granularity than existing incremental rule-mining algorithms would permit (e.g., [9, 12, 4, 26]). As rule mining is a general inference technique, we expect that our adaptations to incremental rule mining will be useful in contexts beyond access control.

3 Association Rule Mining

Association rule mining is a method for finding relationships in databases that has been widely studied in the data-mining community. It involves using statistical measures to generate association rules of the form $x \Rightarrow y$, where x and y can be any resources. We utilize these rules to identify misconfiguration in an access-control environment. The rules that we consider are of the form "*Permission to access resources A, B, and C ⇒ Permission to access to resource D.*" For every user with permission to access *A*, *B*, and *C*, this rule would result in a prediction that that user should have access to resource *D*. An administrator could then either reject or grant this access. We consider a prediction to be a *correct* if the administrator grants a user Alice access to *D* and *incorrect* if the administrator denies access.

In the remainder of this section we discuss more formally how association rules are derived, setting the groundwork for describing our new algorithm in §4.

3.1 Confidence and Support

Many association rule mining techniques have been proposed, though the vast majority use the basic model that we describe here. Rule mining is conducted in the context of a "database" representing a binary relation $R \subseteq O \times A$ between *objects* O (e.g., users) and *attributes* A (e.g., resources that users access), in which case oRa means that user o accessed resource a. The *support* of $x \subseteq A$, denoted $S(x)$, is defined as $|\{o \in O : \forall a \in x, oRa\}|/|O|$, i.e., the fraction of objects related to all elements of x. The *confidence* of a rule $x \Rightarrow y$, denoted $C(x \Rightarrow y)$, is defined as $S(x \cup y)/S(x)$, or, intuitively, the fraction of objects related to x that are also related to y.

Rule mining algorithms seek to identify "high quality" rules, and to do so, typically prune rules using two parameters, min_sup and min_conf. Generally speaking, a rule mining algorithm will identify rules $x \Rightarrow y$ such that $S(x \cup y) \geq min\_sup$ and $C(x \Rightarrow y) \geq min\_conf$, and so higher values of min_sup and min_conf yield higher quality rules. For example, consider the sample database shown in Figure 1, where $O = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $A = \{a, b, c, d, e, f, g, h\}$. In this example, $S(\{b, d\}) = 3/8$ and $C(\{d\} \Rightarrow \{b\}) = 3/4$, and so the rule $\{d\} \Rightarrow \{b\}$ would be generated for any $min\_conf \leq 3/4$ and $min\_sup \leq 3/8$.

		A							
		a	b	c	d	e	f	g	h
	1	x	x	x	x	x	x	x	x
	2	x	x	x		x	x		
	3					x	x	x	x
O	4							x	
	5					x	x		x
	6	x	x	x	x				
	7		x	x	x				
	8				x				

Figure 1: Sample database

3.2 Predictive Accuracy

Both min_sup and min_conf have an impact on the quality of the rules that are generated. Unfortunately, it is often unclear how the two parameters together should be tuned to achieve desired performance. To simplify this issue, in this paper we adopt a method for combining confidence and support into a single measure called *predictive accuracy* [23].

Informally, the predictive accuracy of a rule is the expected value for its true accuracy given the confidence c of the rule and the support s that its precondition enjoys. This value is denoted

$$\mathbb{PA}(c, s) = \mathbb{E}(A(x \Rightarrow y) \mid C(x \Rightarrow y) = c, S(x) = s)$$

where the expectation is taken with respect to choice of association rule $x \Rightarrow y$ uniformly at random from among all possible rules. Scheffer derived a method for approximating it [23], as:

$$\mathbb{PA}(c,s) \approx \frac{\sum_a aB(cs;s,a)\mathbb{P}(C(x \Rightarrow y)=a)}{\sum_a B(cs;s,a)\mathbb{P}(C(x \Rightarrow y)=a)} \tag{1}$$

where $B(k;N,p)$ is the probability of exactly k successes in N independent trials, each with success probability p. An implication of this formulation is that to approximate the predictive accuracy of a rule, we need to compute $\mathbb{P}(C(x \Rightarrow y)=a)$, i.e., the probability with which a rule, drawn uniformly at random from all possible rules, will have confidence a. Scheffer [23] suggested estimating this probability through sampling. A contribution of our work, presented in §4, is an algorithm to incrementally track the confidence distribution across all possible rules and to efficiently compute $\mathbb{P}(C(x \Rightarrow y)=a)$ for any value a.

While support and confidence values are used to calculate the predictive accuracy, we need not choose predictions by placing conditions on confidence and support explicitly (e.g., the thresholds min_sup or min_conf). Rather, we can place conditions on the predictive accuracy of the rules we predict, thereby reducing the measures on which to place such conditions from two to one. One way to do so would be to define a threshold and prune all of the rules with predictive accuracy less than this value. Instead, and as we will discuss in §5, we make predictions in decreasing order of predictive accuracy and in a way that maintains $\frac{Benefit}{Accuracy} \approx \beta$.

4 Incremental Rule Mining

One of the major problems faced when using rule mining to identify policy misconfigurations is that data enters the system incrementally (e.g., one access at a time). To ensure that we capture misconfigurations as soon as possible, it is necessary to recalculate the predictive accuracy of candidate rules after each new item of information is received, a much higher frequency than would be typical in most data-mining domains. By blindly implementing rule-mining techniques, much of the work done at each increment would be repeated computations. As such, we require an *incremental rule mining* technique (c.f., [9, 8, 13]), though one tailored to computing predictive accuracy for rules.

In this section, we describe the data structure commonly used in rule mining (§4.1); our new approach for incrementally maintaining the data structure (§4.2) and the confidence values needed for calculating predictive accuracy (§4.3); and, finally, how to use the data structure to generate rules (§4.4).

4.1 Galois Lattice

Much previous work in rule mining utilizes a data structure known as a *Galois lattice* (e.g., [4, 22, 30]), which we found especially well suited for our needs. The Galois lattice provides a compact data structure for representing a database and an efficient way to parse the data structure to mine association rules. Recall that a partial order $\mathcal{G} = (G, <_{\mathcal{G}})$ is a lattice if any pair of elements u, v has a unique *greatest lower bound* $u \wedge_{\mathcal{G}} v$ and a unique *least upper bound* $u \vee_{\mathcal{G}} v$. Let $\mathcal{P}(\cdot)$ denote the powerset of its input set.

To define a Galois lattice for a database defined by relation R between objects O and attributes A, we define functions $f : \mathcal{P}(O) \rightarrow \mathcal{P}(A)$ and $g : \mathcal{P}(A) \rightarrow \mathcal{P}(O)$ by $f(O') = \{a \in A : \forall o \in O', oRa\}$ and $g(A') = \{o \in O : \forall a \in A', oRa\}$. In other words, f returns the attributes to which all input objects are related, and g returns the objects related to all attributes in its input. Let $G = \{n \in \mathcal{P}(O) \times \mathcal{P}(A) : n.o = g(n.a)$ and $n.a = f(n.o)\}$, where $n.o$ denotes the first component of n (i.e., a subset of O) and $n.a$ denotes its second component (i.e., a subset of A). We refer to each $n \in G$ as a *(closed) node*. Note that by this definition, for any two distinct nodes $n, n' \in G$, it is the case that $n.o \neq n'.o$ and $n.a \neq n'.a$. As such, $|G| \leq \min\{2^{|O|}, 2^{|A|}\}$.[1]

[1]A Galois lattice can, in the worst case, be of size exponential in the smaller of O and A. As such, our use of "efficient"

Let $<_{\mathcal{G}}$ be an irreflexive binary relation on G defined by set inclusion, specifically:

$$n_1 <_{\mathcal{G}} n_2 \Leftrightarrow n_1.o \subset n_2.o \text{ and } n_2.a \subset n_1.a$$

Then, $(G, <_{\mathcal{G}})$ is a (Galois) lattice, where

$$\begin{aligned} n_1 \vee_{\mathcal{G}} n_2 = n', \text{ such that } & n'.o = g(f(n_1.o \cup n_2.o)) \\ & n'.a = n_1.a \cap n_2.a \\ n_1 \wedge_{\mathcal{G}} n_2 = n', \text{ such that } & n'.o = n_1.o \cap n_2.o \\ & n'.a = f(g(n_1.a \cup n_2.a)) \end{aligned}$$

It will be convenient for presentation to define $\prec_{\mathcal{G}}$ to be the transitive reduction of $<_{\mathcal{G}}$. Figure 2 shows $\prec_{\mathcal{G}}$ of the Galois lattice for the database shown in Table 1, also known as a Hasse diagram.

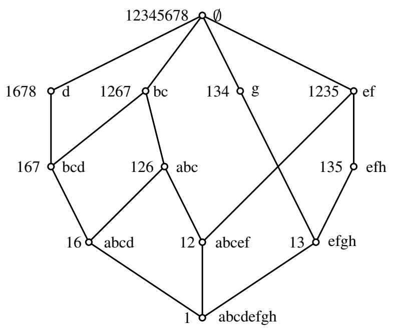


Figure 2: Hasse diagram of Galois lattice for the database in Figure 1

4.2 Incremental Updating

Valtchev et al. [25] showed that a Galois lattice for a database with objects O, attributes A, and relation R can be built using the following inductive algorithm, starting with $G = \emptyset$:

- **Base case:** For each $o \in O$, add $\langle g(f(\{o\})), f(\{o\})\rangle$ to G.
- **Induction step:** For any $n, n' \in G$, add $\langle g(n.a \cap n'.a), n.a \cap n'.a\rangle$ to G.

Once no more nodes can be added to G via the above rules, then $(G, <_{\mathcal{G}})$ is a Galois lattice.[2]

This inductive algorithm reveals one way to incrementally update an existing G with a new object o^* and new relation $R^* = R \cup R_{o^*}$ where $R_{o^*} \subseteq \{o^*\} \times A$ (and g^* and f^* defined accordingly). To do so, we can perform the following step for each $n \in G$: if there is a node $n' \in G$ with $n'.a = f(\{o^*\}) \cap n.a$, update $n'.o \leftarrow g(f(\{o^*\}) \cap n.a)$, and otherwise, add new node $\langle g(f(\{o^*\}) \cap n.a), f(\{o^*\}) \cap n.a\rangle$ to G. Finally, we add the node $\langle g(f(\{o^*\})), f(\{o^*\})\rangle$ to G if it does not already exist, as required by the base case. Several incremental algorithms [25, 26, 28] are optimized instances of this approach.

Unfortunately, this algorithm and others premised on updating the lattice with a whole new object o^* do not suffice for incrementally updating the lattice with a single new addition (o^*, a^*) to R. Since o^* may already exist, the preceding algorithm could result in nodes in G that are no longer closed (i.e., $n.o \neq g(n.a)$ or $n.a \neq f(n.o)$). This could be corrected by deleting o^* each time, and then re-inserting with the new attribute a^*. However, this would still require the implementation of a new method for object removal and is likely to be very inefficient.

So, here we propose an alternative algorithm for incrementally updating $\mathcal{G} = (G, <_{\mathcal{G}})$ with a new addition (o^*, a^*) to yield a new lattice $(G^*, <^*_{\mathcal{G}})$ reflecting the new relation $R^* = R \cup \{(o^*, a^*)\}$. Let f^* and g^* denote the updated versions of f and g that reflect R^*. Our algorithm for converting $(G, <_{\mathcal{G}})$ to $(G^*, <^*_{\mathcal{G}})$ takes advantage of the following property.

Lemma 4.1. *For any $A' \subseteq A$, if $A' \not\subseteq f^*(\{o^*\})$ then: $\langle g^*(A'), A'\rangle \in G$ iff $\langle g^*(A'), A'\rangle \in G^*$.*

Proof. Note that $f(O') = f^*(O')$ for any $O' \not\ni o^*$. Moreover, since $A' \not\subseteq f^*(\{o^*\})$, it is the case that

generally does not imply polynomial time in the size of O or A, but instead simply means "practical" for the datasets we consider. We will clarify the performance of our algorithm on our datasets in §6.

[2]To ensure a least node, it may also be necessary to add $\langle \emptyset, A\rangle$.

$g^*(A') = g(A')$. Therefore,

$$\begin{aligned}
\langle g^*(A'), A' \rangle \in G &\iff \langle g(A'), A' \rangle \in G && \text{since } g^*(A') = g(A') \\
&\iff A' = f(g(A')) && \text{by definition of } G \\
&\iff A' = f^*(g(A')) && \text{since } g(A') \not\ni o^* \\
&\iff A' = f^*(g^*(A')) && \text{since } g^*(A') = g(A') \\
&\iff \langle g^*(A'), A' \rangle \in G^* && \text{by definition of } G^*
\end{aligned}$$

□

By this lemma, $G' = G \setminus (\mathcal{P}(O) \times \mathcal{P}(f^*(\{o^*\})))$ is the set of nodes that will be carried over from $(G, <_{\mathcal{G}})$ to $(G^*, <^*_{\mathcal{G}})$. The inductive algorithm presented above indicates that G^* can then be built up from G' by inserting the nodes $\langle g^*(f^*(\{o^*\})), f^*(\{o^*\})\rangle$ (by the base case) and, for each $n \in G'$, $\langle g^*(n.a \cap f^*(\{o^*\})), n.a \cap f^*(\{o^*\})\rangle$ (by the induction step). Two further optimizations in our algorithm derive from judiciously determining which nodes in G' need not be visited in the application of this inductive step.

1. Consider any node $n \in G'$ such that $a^* \notin n.a$. In this case, $n.a \cap f^*(\{o^*\}) = n.a \cap f(\{o^*\})$, since $f^*(\{o^*\}) = f(\{o^*\}) \cup \{a^*\}$. Moreover, $g^*(n.a \cap f^*(\{o^*\})) = g^*(n.a \cap f(\{o^*\})) = g(n.a \cap f(\{o^*\}))$, since the objects related to $n.a \cap f(\{o^*\}) \not\ni a^*$ did not change by the introduction of (o^*, a^*). As such, if $a^* \notin n.a$ then $\langle g^*(n.a \cap f^*(\{o^*\})), n.a \cap f^*(\{o^*\})\rangle$ is already contained in G^*.
2. Consider any $n \in G'$ such that $f^*(\{o^*\}) \subseteq n.a$. Then, $n.a \cap f^*(\{o^*\}) = f^*(\{o^*\})$. As such, once $\langle g^*(f^*(\{o^*\})), f^*(\{o^*\})\rangle$ is added in the base case, no nodes $n \in G'$ such that $f^*(\{o^*\}) \subseteq n.a$ need to be considered.

Our incremental update algorithm thus works by traversing G', avoiding the nodes identified in (1) and skipping over the nodes identified in (2). In doing so, it must ensure that $\langle g^*(n.a \cap f^*(\{o^*\})), n.a \cap f^*(\{o^*\})\rangle$ is inserted into G^* if it was not present already and that all nodes in the resulting lattice are closed (by repairing or removing those that are not). The traversal starts from the least element in G and walks depth-first in reverse direction of $\prec_{\mathcal{G}}$. Any node n such that $f^*(\{o^*\}) \subseteq n.a$ is skipped over, and a branch is terminated if it reaches a node n such that $a^* \notin n.a$ or $n.a \subseteq f^*(\{o^*\})$ (meaning that the walk has left G').

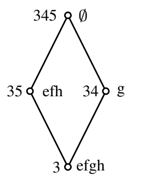
The effect of our optimizations is substantial. For example, for the datasets on which we test in §6, this algorithm traverses an average of less than 20% of the lattice per update and modifies about 4% of the nodes per update. We will provide more detail about the performance of this algorithm in §6.6.

4.3 Distribution of Confidence Values

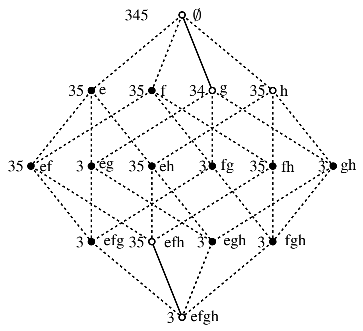
Previous work on incremental rule mining generally ranked rules according to their confidence and support, rather than predictive accuracy. Hence, we needed to devise a method that would enable efficient computation of predictive accuracy in the context of our incremental algorithm.

Recall that the calculations for predictive accuracy require a probability distribution of accuracies for all potential rules, which we approximate by using the current distribution of confidence values for those rules. Although association rules are defined as $x \Rightarrow y$, where $x, y \subseteq A$, for our algorithm we consider only the subset of potential rules for which $|y| = 1$. While this implies that each rule we generate through association rule-mining contains less information, it permits us to speed up the maintenance of the confidence distribution for such rules dramatically, as we describe in this section. Ignoring rules of the form $|y| > 1$, however, means that we must ensure that our distribution of confidences used to calculate $\mathbb{P}(C(x \Rightarrow y) = a)$ in (1) includes confidences only for rules with $|y| = 1$. We show in this section how we were able to exploit the incremental fashion of our system and our Galois lattice implementation to calculate the exact distribution of confidence values for all rules of the form $x \Rightarrow y, |x| \geq 1, |y| = 1$.

Figure 3 shows a Hasse diagram for a Galois lattice for objects O, attributes A, and relation R (Figure 3(a)) and a lattice $(L, <_{\mathcal{L}})$ with nodes $L = \{\langle g(A'), A'\rangle : A' \subseteq A\}$ and $<_{\mathcal{L}}$ defined by $n_1 <_{\mathcal{L}} n_2$ iff $n_2.a \subset n_1.a$ (Figure 3(b)). As before, let $\prec_{\mathcal{L}}$ denote the transitive reduction of $<_{\mathcal{L}}$. Note that in Figure 3(b), every (downward) edge (element of $\prec_{\mathcal{L}}$) corresponds to an association rule with $|y| = 1$. For example the edge $(\langle\{3,5\}, \{e,f,h\}\rangle, \langle\{3\}, \{e,f,g,h\}\rangle)$ would correspond to the rule $\{e,f,h\} \Rightarrow \{g\}$. As such, the desired distribution of confidence values is the distribution for the rules represented by these edges; e.g., the aforementioned rule has confidence $C(\{e,f,h\} \Rightarrow \{g\}) = \frac{|\{3\}|}{|\{3,5\}|} = 0.5$. In other words, computing the histogram of confidence values is equivalent to computing the confidences associated with the edges $\prec_{\mathcal{L}}$ depicted in Figure 3(b) from the Galois lattice in Figure 3(a).



(a) $\mathcal{G} = (G, \prec_{\mathcal{G}})$



(b) $\mathcal{L} = (L, \prec_{\mathcal{L}})$

Figure 3: Example Hasse diagram $\mathcal{G}$ and expanded lattice $\mathcal{L}$ where $O = \{3,4,5\}$ and $A = \{e,f,g,h\}$.

Two key observations enable this computation to be done efficiently.

1. Any $n \in G$ represents all nodes $\langle g(A'), A'\rangle \in L$ where $A' \in \mathcal{P}(n.a) \setminus \bigcup_{n' : n \prec_{\mathcal{G}} n'} \mathcal{P}(n'.a)$. In words, every n in G represents all nodes in the lattice $\mathcal{L}$ with attribute sets $A' \subseteq n.a$ but $A' \not\subseteq n'.a$ for any "parent" n' of n in $\mathcal{G}$ (i.e., $n \prec_{\mathcal{G}} n'$). Moreover, $g(A') = n.o$ for each such A', and so the support of A' is $S(A') = |n.o|$.
2. For any nodes $n_1, n_2 \in G$ such that $n_2 <_{\mathcal{G}} n_1$, there is an edge

$$\langle g(A' \cup \{a\}), A' \cup \{a\}\rangle \prec_{\mathcal{L}} \langle g(A'), A'\rangle$$

representing a rule $A' \Rightarrow \{a\}$ with confidence $\frac{|n_2.o|}{|n_1.o|}$, for each $A' \in \mathcal{P}(n_1.a) \setminus \bigcup_{n' : n_1 \prec_{\mathcal{G}} n'} \mathcal{P}(n'.a)$ and

$$a \in n_2.a \setminus \left(\bigcup_{n' : n_2 \prec_{\mathcal{G}} n' \leq_{\mathcal{G}} n_1} n'.a \right).$$

In other words, the pair n_1, n_2 represents rules $A' \Rightarrow \{a\}$ with confidence $\frac{|n_2.o|}{|n_1.o|}$ for which: $A' \subseteq n_1.a$ but $A' \not\subseteq n'.a$ for any "parent" n' of n_1; and $a \in n_2.a$ but $a \notin n'.a$ for any "parent" n' of n_2 on a path from n_1. The number of such rules can be computed efficiently by examining n_1, n_2, and their parents.

For example, consider nodes n_1, n_2 in Figure 2, where $n_1.a = \{a,b,c\}$ and $n_2.a = \{a,b,c,d,e,f,g,h\}$. n_1 has parents $\{\langle\{1,2,6,7\}, \{b,c\}\rangle\}$, and n_2 has parents on paths from n_1 of $\{\langle\{1,6\}, \{a,b,c,d\}\rangle, \langle\{1,2\}, \{a,b,c,e,f\}\rangle\}$. In the terminology of (2) above, A' is drawn from $\mathcal{P}(n_1) \setminus \bigcup_{n' : n_1 \prec_{\mathcal{G}} n'} \mathcal{P}(n'.a) = \{\{a,b,c\}, \{a,b\}, \{a,c\}, \{a\}\}$ and a is drawn from $n_2.a \setminus \left(\bigcup_{n' : n_2 \prec_{\mathcal{G}} n' \leq_{\mathcal{G}} n_1} n'.a\right) = \{g,h\}$. Thus, we can conclude that 8 rules are represented by n_1, n_2, namely

$$\begin{array}{llll} \{a,b,c\} \Rightarrow \{g\} & \{a,b\} \Rightarrow \{g\} & \{a,c\} \Rightarrow \{g\} & \{a\} \Rightarrow \{g\} \\ \{a,b,c\} \Rightarrow \{h\} & \{a,b\} \Rightarrow \{h\} & \{a,c\} \Rightarrow \{h\} & \{a\} \Rightarrow \{h\} \end{array}$$

Each has confidence $\frac{|n_2.o|}{|n_1.o|} = \frac{1}{3}$.

It is important to recognize that the number of rules represented by n_1, n_2 can be calculated without enumerating those rules. Moreover, this calculation need not be done for every pair n_1, n_2 after each update to $\mathcal{G}$. Rather, these calculations can be performed incrementally, in conjunction with incrementally updating

$\mathcal{G}$. Specifically, when a node n in $\mathcal{G}$ is modified or added, nodes n' such that $n' <_{\mathcal{G}} n$ are traversed to track the number of rules that the pair n, n' now represents and their corresponding confidences, and then nodes n' such that $n <_{\mathcal{G}} n'$ are similarly traversed. Node removals require similar operations.

As noted in §4.2, in our tests in §6, a lattice update modified roughly 4% of the lattice on average. As such, the above algorithm gives rise to traversals for about 4% of the nodes n in the lattice (i.e., traversals of nodes n' such that $n' <_{\mathcal{G}} n$ or $n <_{\mathcal{G}} n'$) after an average update. We will quantify the costs of these traversals in §6.6.

4.4 Generating Rules

The preceding sections summarized the approach by which we incrementally update $\mathcal{G}$ as new accesses arrive (§4.2) and how we maintain a distribution of confidence values across such updates without enumerating every association rule (§4.3). We now discuss how we actually generate rules from the lattice.

As discussed in §4.3, the single-target association rules represented by a pair n_1, n_2 of nodes can be counted and, if desired, enumerated. As such, it suffices to describe how to examine all pairs n_1, n_2 as quickly as possible and, more to the point, efficiently bypass nodes which require no consideration. Informally, our algorithm is structured as a downward traversal of the Hasse diagram of $\mathcal{G}$ starting from $\langle O, f(O)\rangle$, and then when visiting a node n_1 in this first traversal, conducting a nested, downward traversal of nodes $n_2 <_{\mathcal{G}} n_1$. For each n_1 and n_2, the algorithm uses the method discussed in §4.3 to enumerate the rules represented by these nodes that are not subsumed by other rules, where $x_1 \Rightarrow y$ *subsumes* $x_2 \Rightarrow y$ if $x_1 \subseteq x_2$ and $\mathbb{PA}(C(x_1 \Rightarrow y), S(x_1)) \geq \mathbb{PA}(C(x_2 \Rightarrow y), S(x_2))$.

To make rule generation efficient, we employ optimizations that permit us to bypass nested traversals, either in whole or in part. Specifically, to save space and time, we generate only a target number ℓ of rules at a time. For example, if $\ell = 100$, then we first generate rules 1–100 with the highest predictive accuracies and make predictions from them; we generate rules 101–200 with the next highest predictive accuracies only if additional predictions are needed. As such, when traversing $\mathcal{G}$, the algorithm maintains a list $(x_1 \Rightarrow y_1), \ldots, (x_\ell \Rightarrow y_\ell)$ of unique association rules, ordered in nonincreasing order of predictive accuracy. The algorithm additionally maintains values τ and γ_s in the course of updating $(x_1 \Rightarrow y_1), \ldots, (x_\ell \Rightarrow y_\ell)$, defined as follows:

$$\tau = \min\{s \geq 1 : \mathbb{PA}(1, s) \geq \mathbb{PA}(C(x_\ell \Rightarrow y_\ell), S(x_\ell))\}$$
$$\gamma_s = \min\{s' \geq 1 : \mathbb{PA}(s'/s, s) \geq \mathbb{PA}(C(x_\ell \Rightarrow y_\ell), S(x_\ell))\}$$

By the definition of τ, when visiting a node n_1 where $|n_1.o| < \tau$ in the aforementioned traversal of $\mathcal{G}$, the algorithm can recognize that for any $n_2 <_{\mathcal{G}} n_1$, every rule represented by n_1, n_2 has predictive accuracy less than that of $x_\ell \Rightarrow y_\ell$. (This inference derives from the fact that $s' < s$ and $c' < c$ implies that $\mathbb{PA}(c', s') < \mathbb{PA}(c, s)$.) As such, the nested traversal over nodes $n_2 <_{\mathcal{G}} n_1$ can be skipped. Similarly, when visiting a node n_2 in a nested traversal for node n_1, the algorithm can recognize that if $|n_2.o| < \gamma_{|n_1.o|}$, then the predictive accuracy of every rule represented by n_1, n_2 is less than that of $x_\ell \Rightarrow y_\ell$. As such, these rules need not be enumerated.

To help emphasize the significance of these optimizations, consider that, for the most difficult dataset discussed in §6.1 and for a reasonable β = .9, these optimizations allowed us to visit, in the worst case, 79% fewer nodes during the initial traversal, and in total enabled 95% fewer comparisons between nodes. We also show, in §6.6, that our algorithm substantially outperforms a more naive approach proposed by Scheffer, in some cases by almost three orders of magnitude.

5 Misconfiguration Prediction Framework

In this section we first detail our methodology for evaluating the efficacy of misconfiguration prediction (§5.1), which we argue is more encompassing than previous approaches. We then describe our method for guiding predictions to strike a desired balance between accuracy and benefit (§5.2).

5.1 Model and Definitions

Our approach for identifying misconfigurations operating uses records of actual accesses in the system. Let a *policy atom* be a user-resource pair (u, r). We denote the sequence of *unique* accesses in the system as $a_1, a_2, \ldots$ (i.e., $a_{\ell+1} \notin \{a_1, \ldots, a_\ell\}$) where each a_ℓ is a policy atom. Each access a_ℓ occurs at a distinct, integral logical time $\mathsf{time}(a_\ell) \in \mathbb{N}$. Logical times are totally ordered and are assigned so that $\mathsf{time}(a_\ell) < \mathsf{time}(a_{\ell+1})$. We refer to the *exercised policy* at time t to be $\mathit{Exercised}_t = \{a_\ell : \mathsf{time}(a_\ell) \leq t\}$.

In addition to actual accesses, additional policy might be *deduced* on the basis of information conveyed in accesses (or, as we will see below, in the results of misconfiguration predictions). For example, an access (u, r) might be accompanied by a credential that demonstrates that u has access to other resources besides r. For this reason, we define $\mathit{Deduced}_t$ to be the set of policy atoms that can be deduced from $\mathit{Exercised}_t$. In particular, $\mathit{Exercised}_t \subseteq \mathit{Deduced}_t$. We also assume that $\mathit{Deduced}_t \subseteq \mathit{Deduced}_{t+1}$, i.e., over time, more policy can be revealed, but previously existing policy is not invalidated. (Relaxing this assumption, e.g., to support revocation of policy, is possible and would not significantly impact our evaluation framework.) We stress that all contents of $\mathit{Deduced}_t$ might not be visible to our prediction engine, due to lack of integration between the prediction engine and the access-control system; e.g., the credentials accompanying an access might not be made available to the prediction engine. Hence, we define a set $\mathit{Visible}_t \subseteq \mathit{Deduced}_t$ that is the set of policy atoms visible to the prediction engine at time t. We do not generally require that $\mathit{Visible}_t = \mathit{Deduced}_t$ (we will discuss this more below), though we do presume that $\mathit{Visible}_t \subseteq \mathit{Visible}_{t+1}$, i.e., that the system never "forgets" information that it used in previous predictions, and that $\mathit{Exercised}_t \subseteq \mathit{Visible}_t$.

The job of our system is to issue *predictions* of what might be a misconfiguration, based on the accesses seen in the system so far. Like an access, each prediction is made at a logical time distinct from that of any access. However, predictions need not be issued at times distinct from each other, and generally they will not be. So, while we will still denote predictions by $p_1, p_2, \ldots$, they are only partially ordered by their logical times; specifically, $\mathsf{time}(p_\ell) \leq \mathsf{time}(p_{\ell+1})$. Let $\mathit{Predictions}_t = \{p_\ell : \mathsf{time}(p_\ell) \leq t\}$. Each prediction p_ℓ is a policy atom (u, r). A prediction p_ℓ is made by applying the algorithm of §4 to $\mathit{Visible}_t$ for $t = \mathsf{time}(p_\ell)$. When applying the algorithm in §4 to make a prediction at logical time t, the object set O is the users in $\mathit{Visible}_t$ and the attribute set is the resources in $\mathit{Visible}_t$. Each rule $x \Rightarrow \{r\}$ derived by that algorithm then yields a prediction p_ℓ of policy atom (u, r) at time $t = \mathsf{time}(p_\ell)$ if and only if (i) $\forall r' \in x : (u, r') \in \mathit{Visible}_t$; (ii) $(u, r) \notin \mathit{Visible}_t$; and (iii) $(u, r) \notin \mathit{Predictions}_{t-1}$. In other words, a rule will lead to a prediction for a user u if and only if user u has accessed all the resources in the precondition x of the rule but not resource r, and if the prediction has not already been made.

In a real system, each prediction would need to be judged, presumably by a human administrator (e.g., [6, 10]). For our evaluation, we determine the correctness of each prediction relative to an *intended policy* $\mathit{Intended}$, which is a set of policy atoms; intuitively, the intended policy is the ideal (though perhaps not implemented) policy in the system. We will discuss how we instantiate $\mathit{Intended}$ in our datasets in §6.1, but for our purposes here, we simply stipulate that $\mathit{Deduced}_t \subseteq \mathit{Intended}$ for all times t. We define the *correct* predictions inductively, as follows: $\mathit{Correct}_0 = \emptyset$ and $\mathit{Correct}_{t+1} = \mathit{Correct}_t \cup (\mathit{Predictions}_{t+1} \cap (\mathit{Intended} \setminus \mathit{Deduced}_t)\}$. As such, only predictions that are not already deducible are correct. The *incorrect* predictions can be defined more straightforwardly: $\mathit{Incorrect}_t = \mathit{Predictions}_t \setminus \mathit{Intended}$. We assume that our prediction system is informed of the result when it makes predictions, i.e., whether the prediction was correct, incorrect or already deducible. As such, $\mathit{Correct}_t \subseteq \mathit{Visible}_{t+1}$ and $\mathit{Predictions}_t \cap \mathit{Deduced}_t \subseteq$

$\mathit{Visible}_{t+1}$. This means that all predictions at time t are resolved prior to predictions made at time $t+1$ or later, though we stress this is a modeling simplification and not a requirement in practice.

In §6, we will evaluate the performance of our algorithm in three types of systems.

No deduction In a system with "no deduction", we define $\mathit{Deduced}_{t+1} = \mathit{Visible}_{t+1} = \mathit{Exercised}_{t+1} \cup \mathit{Correct}_t$. (Because $\mathit{Visible}_t = \mathit{Deduced}_t$ we know that $\mathit{Predictions}_t \cap \mathit{Deduced}_t = \mathit{Correct}_t$, and so $\mathit{Predictions}_t \cap \mathit{Deduced}_t$ need not be additionally counted in the expression of $\mathit{Deduced}_{t+1}$.) This is the setting in which previous proposals for misconfiguration prediction based on accesses have been evaluated [6]. A system permitting no deduction based on previous accesses might be, e.g., a system in which every access permission is demonstrated using a distinct per-resource capability. In such systems, it cannot be deduced that policy allows any accesses other than those that have already been exercised.

Eager deduction In a system with "eager deduction", we stipulate that $\mathit{Visible}_{t+1} = \mathit{Deduced}_{t+1}$, but generally $\mathit{Visible}_{t+1} \supseteq \mathit{Exercised}_{t+1} \cup \mathit{Correct}_t$. That is, we expect that it is possible to deduce more than just what has been observed or predicted, and all such deductions are "eagerly" exploited to improve prediction. An example of a system permitting eager deduction would be any system that reasons using the credentials presented in previous accesses and gathered from previous predictions (e.g., as would be possible in a proof-carrying authorization system [3]) and then imports these into the prediction engine.

Lazy deduction In a "lazy deduction" system, $\mathit{Visible}_{t+1} = \mathit{Exercised}_{t+1} \cup \mathit{Correct}_t \cup (\mathit{Predictions}_t \cap \mathit{Deduced}_t)$, but we permit $\mathit{Visible}_{t+1} \subseteq \mathit{Deduced}_{t+1}$. As such, we expect that there will be deductions that are not incorporated into the prediction algorithm (the meaning of "lazy"), but that are still relevant in measuring the success of a prediction algorithm as defined below. In other words, a lazy system is one in which policy that has not been observed or predicted can be consulted only after a prediction has been made. We expect most practical systems to be eager, lazy, or in between.

The measures of success that we produce for our system are intuitively the precision and recall of its predictions, which we call *accuracy* and *benefit*. Our definition of accuracy is natural:

$$\mathit{Acc}_t = \frac{|\mathit{Correct}_t|}{|\mathit{Correct}_t \cup \mathit{Incorrect}_t|}$$

Then, the accuracy Acc is simply Acc_t at the maximum value of t in the execution. Note that the denominator of Acc_t is the size of $\mathit{Correct}_t \cup \mathit{Incorrect}_t$ and not of $\mathit{Predictions}_t$; the difference is predictions that were already deducible by the time they were made. These predictions are not helpful (and thus are not counted as "correct"), but would presumably not be passed to a human, since the system can deduce their truth already (and so should not be counted as "incorrect"). Similarly, benefit is defined

$$\mathit{Ben}_t = \frac{|\mathit{Correct}_t|}{|\mathit{Intended}|}$$

and then the benefit Ben is simply Ben_t at the maximum value of t in the execution.

5.2 Algorithm for Enforcing Benefit vs. Accuracy Ratio

Intuitively, and as found by previous work [6], there is a tension between benefit and accuracy. Seeking to maximize accuracy typically involves making only those predictions that are very likely to be correct, which results in a lower benefit. On the other hand, maximizing benefit is achieved by making predictions more indiscriminately, and hence lowering accuracy. In previous approaches to misconfiguration identification, instantiating the prediction algorithm with different parameters led to results on different points of the spectrum from higher accuracy/lower benefit to lower accuracy/higher benefit. However, the relationship between different parameter sets and different points on this spectrum was both ad-hoc and varied across datasets, and so the parameters needed to be tuned by trial and error to achieve the desired tradeoff between benefit and accuracy.

As discussed in §1, a contribution of this paper is a method for ensuring $\frac{Benefit}{Accuracy} \approx \beta$ across a wide range of datasets, with no additional parameter tuning. A prediction engine can track Acc_t over time but, because it does not know $Intended$, it cannot track Ben_t precisely. So, instead, our method tracks

$$\textit{V-Ben}_t = \frac{|Correct_t|}{|Visible_{t+1}|}$$

since $Visible_{t+1}$ is (by the definition of prediction in §5.1) the engine's closest approximation to $Intended$ and one that should improve (get closer to $Intended$) over time. The prediction engine can thus compute

$$\frac{\textit{V-Ben}_t}{Acc_t} = \frac{|Correct_t \cup Incorrect_t|}{|Visible_{t+1}|} \tag{2}$$

and monitor for the event in which this ratio drops below the target value β. More specifically, in the absence of predictions, $Visible_{t+1}$ will continue to grow over time as new accesses are exercised (and t incremented), thus causing (2) to drop below β. Once that event occurs, the prediction engine can issue predictions until (2) climbs back above β, at which point it suspends making further predictions until (2) falls back below β. In particular, since an incorrect prediction at time t causes $Incorrect_{t+1} \supset Incorrect_t$ and has no effect on $Visible_{t+1}$, predictions may continue indefinitely only if predictions are always correct or already deducible. In either case, these predictions are either helpful or have no effect on the system.

Once predictions are solicited by (2) falling below β, all predictions are derived at the same logical time t using the rule generation algorithm described in §4.4 (i.e., based on the same visible policy $Visible_t$), until enough of these predictions are resolved to suspend predictions and allow logical time $t + 1$ to begin. The only exception is if rule generation exhausts the rules from the existing lattice $\mathcal{G}$, in which case, time $t+1$ is begun anyway — in particular, with $Visible_{t+1}$ incorporating the resolutions to predictions at time t before doing so — and predictions are continued until (2) again exceeds β or no new predictions are generated.

6 Results

In this section we evaluate the effectiveness of our misconfiguration prediction method described in §5.2, which we will refer to as "Ratio". The goals of our evaluations are fourfold.

- We seek to show that in a variety of types of systems — namely, No Deduction (ND), Lazy Deduction (LD), and Eager Deduction (ED) systems — the Ratio algorithm is competitive, in terms of the benefit and accuracy that it achieves, with traditional association-rule-mining algorithms that provide for fixed settings of min_conf and min_sup, even for the "best" settings of min_conf and min_sup (which will vary per system). To show this, we will present the benefit and accuracy achieved by Ratio alongside that achieved by traditional association-rule mining on graphs similar to receiver operating characteristic (ROC) curves. We emphasize that we would not expect Ratio to provide uniformly better benefit and accuracy than traditional approaches, as it was not designed to do so. (Rather, it was designed to ensure $\frac{Benefit}{Accuracy} \approx \beta$.) However, it is important that it remain competitive with traditional approaches. ND and LD systems will be discussed in §6.2–6.3, and ED systems in §6.4.

- The point of covering all of ND, LD and ED systems in our evaluation is to convey a lesson regarding the evaluation of misconfiguration prediction systems such as ours. Prior evaluations addressed only the ND case, but we view ND as representing fewer practical systems than LD and ED do: most systems that utilize group or role credentials would permit deducing other accesses from past ones that could be used to filter predictions before they reach a human (yielding an LD configuration) or that could be imported into the prediction engine wholesale (yielding an ED configuration). In §6.2 we show that when using an ND methodology to evaluate an LD system, one tends to significantly overestimate the benefit that is achieved by misconfiguration prediction. In §6.3, we examine the addition of *annotations* — extra group or role information that accompanies accesses — into the prediction engine to recover some of the benefit, but we find this offers only incremental improvement. However, we show in §6.4 that by moving to an ED configuration, where the prediction engine can leverage all deducible information from past accesses, benefit can be more substantially increased (though still not to the levels promised by an ND evaluation).
- We seek to confirm that Ratio does, in fact, succeed in ensuring that $\frac{Benefit}{Accuracy} \approx \beta$ and that traditional rule-mining approaches based on fixed values for min_conf and min_sup do not in our setting. We evaluate these conjectures in §6.5.
- We seek to establish that the Ratio algorithm, including both incremental lattice maintenance and prediction generation, is sufficiently efficient to perform well in many practical settings for this application domain. This will be the focus of §6.6.

Before beginning, in §6.1 we describe the datasets that we use for evaluation.

6.1 Datasets

In our evaluation, we use both a dataset generated by a real system and a range of synthetically generated datasets. In each dataset we are able to construct *exercised policy*, *deduced policy*, and *intended policy*, as described in §5.1.

Real dataset The dataset generated by a real system was provided by Bauer et al. and is a variant of the dataset used to evaluate their previous work on misconfiguration detection [6]. The system from which the data was drawn is a discretionary access-control system deployed in an office environment for controlling access to physical space. The system allows users to specify access policy both via roles and by directly delegating to individuals. The dataset encompasses a sequence of 26,383 accesses observed over 1,113 days of running the system, during which the system was used by 38 users and protected 35 resources. The *exercised policy* against which we test is the subsequence of this access log constructed by removing all duplicate accesses (i.e., for any principal and resource, only the first access by that principal to that resource is kept), and in this case comprise 247 unique accesses. Each access in the dataset is annotated with the policy information (e.g., role assignments or delegations) that made that access possible. We make use of these annotations to construct $Deduced_t$, i.e., the set of accesses we know to be possible at time t, for all times t covered by exercised policy. More specifically, $Deduced_t$ is produced via an algorithm that accumulates annotations into a knowledge base, and then attempts to infer all consequences of the facts present in this knowledge base (i.e., all accesses enabled by the knowledge base). Each annotation (or policy fragment) in this dataset was represented as a formula in an authorization logic, and the inference method was forward chaining; however, many other representations of policy would work equally well.

Finally, the corresponding *intended policy* was constructed by surveying the users of the system to learn what policy they had created or were willing to create that had not been observed during the operation of the system (e.g., because such policy was not required for any of the accesses that took place during the time period over which the access data was collected). For the rest of the paper we will informally refer to this dataset as the *real* dataset.

Synthetic datasets A practical algorithm for misconfiguration detection will have to perform well on a variety of datasets. Since we were able to obtain only a single real dataset for evaluation, we set out to create a range of synthetic datasets. Our goal was for these datasets to contain a mix of role- or group-based policy and direct person-to-person delegations, on the grounds that the former would be similar to real organizational access-control policies and the latter would inhibit prediction but typically occurs in practical systems. We also wanted the datasets to span a wider range in terms of the number of groups or roles, their sizes, and the depth of group or role hierarchies. As with the real dataset, we wanted each synthetic dataset to have exercised, deduced, and intended policy components.

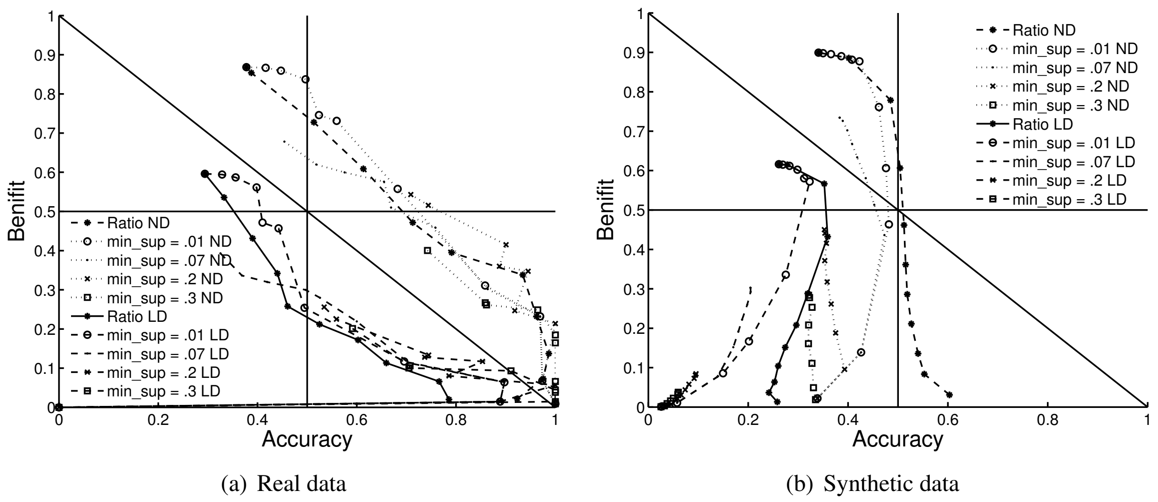
Roughly speaking, the intended policy component of each dataset was created via the following algorithm. First, we create a set of users and a set of resources, and allow some of those users direct access to some resources. With some prespecified probability, we then allow each user who has access to a resource to create a role, and probabilistically assign to that role some resources and some users. We iteratively repeat this process on all users who received access to a resource in the previous round of policy creation. At each iterative step, we probabilistically decide whether to continue to the next iteration or discontinue role creation. Role creation terminates either as a result of such a probabilistic choice, or because a target policy density has been reached. After creating role-based policy in this manner, we optionally augment it with direct delegations to achieve the desired mix of role-based and directly delegated policy. The direct delegations are created straightforwardly: we pick a target user and a resource to which she does not have access, and cause a user who does have access to that resource to delegate this access to the target. The algorithm is parameterized by probabilities that guide every step of the policy-creation process (whether to create another role, whether to add another user to a role, whether to iterate on role creation), but even repeatedly running the algorithm with the same set of parameters causes it to generate a wide range of policies. The algorithm guards against the creation of degenerate policies, or overly permissive ones; the synthetically generated datasets that we employ here had densities ranging between 30% and 45%, and averaging 35%, where density is the percent of possible policy atoms contained in intended policy.

Once intended policy has been created in this manner, we use it to randomly generate the sequence of accesses that comprises the exercised policy. The exercised policy is complete with respect to the intended policy, i.e., at the end of policy creation, $Exercised_t = Intended$. As with the real data, each access is annotated with the policy (e.g., group or role information) that enabled it. Once exercised policy is generated, we use it to compute $Deduced_t$, for every t within scope of the exercised policy. The process for computing $Deduced_t$ is the same as for the real dataset.

All datasets that we create in this manner we henceforth refer to as *synthetic* datasets. We generated synthetic datasets with 50 users and 50 resources, with 50 users and 70 resources, and with 70 users and 50 resources. For each of these three sets of parameters describing the number of users and resources, we generated 10 data sets; the results we show in the rest of this section are averages over the 30 datasets, 10 of each parameter set. For the synthetic datasets we use in our evaluation, 5% of the possible accesses is enabled by direct delegations, and the remainder is enabled by group- or role-based policy. (We evaluated increasing the fraction of direct delegations up to 60%, but we omit a detailed discussion due to space limitations; briefly, the accuracy of all prediction methods decreased somewhat as the policy became less structured, but the Ratio method appeared more resilient to this decrease than previous approaches.)

6.2 No Deduction Versus Lazy Deduction

We first examine the benefit and accuracy of our system using the No Deduction (ND) model described in §5.1, which is the model in which previous work was evaluated [6]. Figure 4 shows plots of benefit and accuracy achieved in an ND evaluation: Figure 4(a) shows results for the real dataset, and Figure 4(b) for



(a) Real data (b) Synthetic data

Figure 4: No Deduction vs. Lazy Deduction

synthetic datasets. Because there is only one real dataset, the points in Figure 4(a) are the actual results of each technique on that dataset, whereas each point in Figure 4(b) is an average over all synthetic datasets described in §6.1. In the Ratio curves, each point corresponds to a different value of β. Each other curve plots the benefit and accuracy of a traditional rule-mining algorithm with min_sup indicated in the legend and with min_conf varied; each point corresponds to a different setting of min_conf.[3]

An immediate observation from these figures is the impressive benefit and accuracy of misconfiguration identification in the ND model, particularly in the real data. As we have contended previously, however, the ND model, by setting $Deduced_{t+1}$ to be simply $Exercised_{t+1} \cup Correct_t$, precludes any deductions being made from the evidence of access-control policy that might have accompanied policy atoms. For example, if an access is accompanied by a credential illustrating a group membership of the user requesting access, then this credential can be used to deduce other aspects of intended policy. The ND model, however, does not permit such credentials to be taken into account.

Perhaps the easiest way to take this additional information into account is to use it to "filter out" predictions that can be proved to be correct before they reach a human administrator — the LD model. In doing so, the curves marked LD in Figure 4 result. A notable lesson from these new curves is that the ND model substantially overestimates the true effectiveness of misconfiguration prediction when deduction is possible.

More specifically, on the real dataset the highest benefit provided by any prediction algorithm evaluated in the ND case is 86.8%; the more realistic view represented by LD reveals the maximal benefit to be a much lower 59.6%. This maximal attained benefit of 59.6% indicates that many of the predictions credited as correct in the ND case were, in fact, already deducible by the time they were made, and hence were not indicative of misconfigurations. In fact, we can say for certain that this benefit of 59.6% is the maximum that could be achieved by any prediction algorithm operating in an LD case, because this is the highest value reached by a tuning of the naive algorithm that is so biased towards attaining high benefit that it makes every prediction for which there is *any* statistical evidence. This highest attainable benefit can be increased slightly by allowing the prediction algorithm access to more information, as we will show in §6.3–6.4.

As particularly evident in Figure 4(b), ND can also overstate accuracy, again because the highest-ranked association rules tend to be already deduced. In this case, the comparison with LD reveals that a large fraction of the predictions that are contributing to ND's high accuracy is redundant in light of what can be deduced, and that the non-redundant predictions, which are the only ones made by LD, are significantly less

[3]We used $\beta \in \{.05, .15, .25, .4, .55, .7, .9, 1.2, 1.6, 2.2, 20\}$ and $min\_conf \in \{.01, .1, .2, .3, .4, .5, .6, .7, .8, .9, .95\}$.

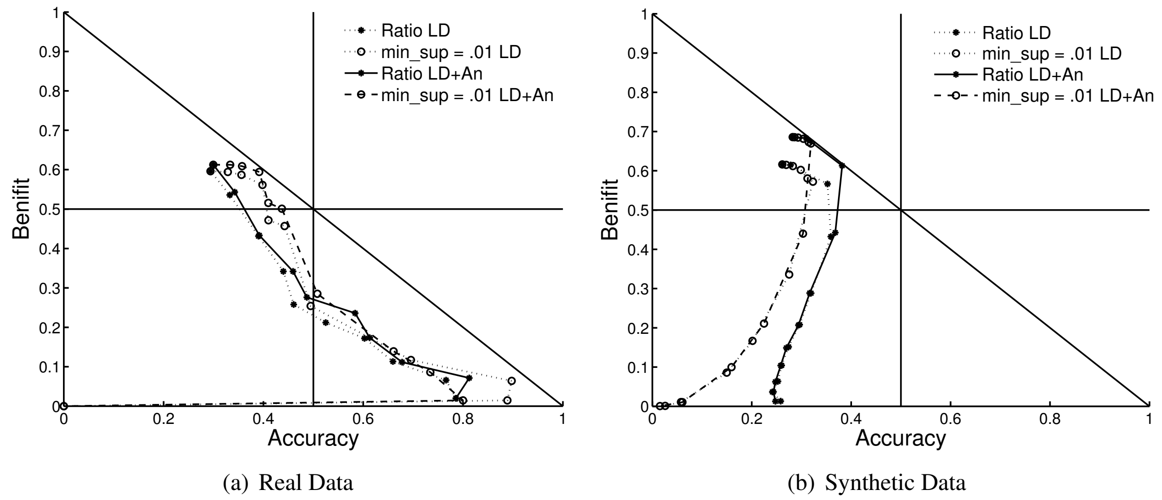
accurate. For example, using the Ratio method on the synthetic data with β = .7, accuracy falls from 51.2% in ND to 31.2% in LD (and benefit declines from 46.2% to 28.8%). We will show in §6.3–6.4 that much of this accuracy can also be recovered by giving the prediction algorithm access to more information.

Notice that the curves shown in Figure 4(a) are considerably different in shape than those in Figure 4(b). We hypothesize that a main reason for this is that in the synthetic datasets the sequence of exercised accesses was a random permutation of the possible accesses, whereas in the real dataset accesses had more locality both among users and among resources. This causes prefixes of the real exercised policy to be more indicative of patterns than prefixes of synthetic exercised policy, and hence a better predictor of misconfigurations.

Another take-away message from Figure 4 is that Ratio is indeed competitive with the various tunings of traditional rule-mining based on min_sup and min_conf, typically trailing the best such curve by only a few percentage points in each of benefit and accuracy (and, in some cases, beating the best such curve). One exception is that the distance between the best min_sup LD curve and "Ratio LD" curve grows when the parameters (min_conf or β, respectively) are configured to emphasize accuracy over benefit, in both Figures 4(a) and 4(b). However, these highest-accuracy configurations yield very few predictions, and so this larger gap in accuracy reflects only a small number of incorrect predictions by Ratio.

Figure 4 also motivates the move to using predictive accuracy in lieu of min_sup and min_conf, because it shows that different values of min_sup perform better in the different scenarios considered. Selecting the best min_sup and min_conf at a particular juncture is a challenge that predictive accuracy helps to resolve.

6.3 Utilizing Annotations



(a) Real Data (b) Synthetic Data

Figure 5: Lazy Deduction with annotations

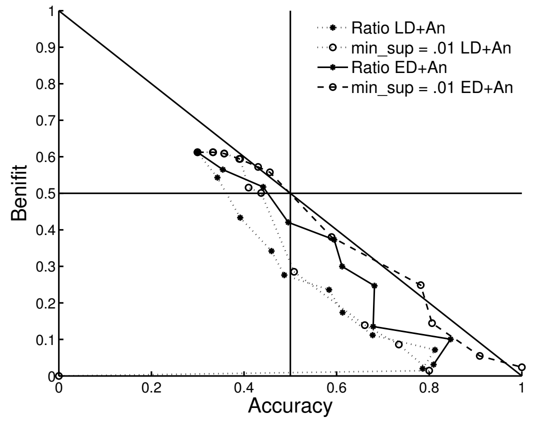
The LD curves in Figure 4 present a more sobering picture for the utility of misconfiguration prediction in systems permitting deduction of policy based on policy already seen. A middle ground between an LD system and fully incorporating all deductions into the prediction engine (an ED system, evaluated in §6.4) is importing *annotations* into the prediction engine (i.e., into $Visible_t$) in the form of group or role identifiers to which a user has been demonstrated to belong in the course of gaining access to other resources. For example, in our datasets we can extract group memberships from some of the credentials that accompany accesses. (We do not further reason about what other resources those group memberships might permit users to access, which is the additional power of deduction that ED systems use.)

We model annotations in our framework by introducing additional "resources" into the resource set $\mathcal{R}$; these resources describe groups and roles. When a new credential stating a user's membership in the group or role is observed, this is realized as a new policy atom — a new element of exercised policy. For the purposes of rule generation, these additional "resources" can appear only in x for a rule $x \Rightarrow y$. This permits rules like, e.g., "Membership in Students $\wedge$ access to Student Lounge $\Rightarrow$ access to Computer Lab".

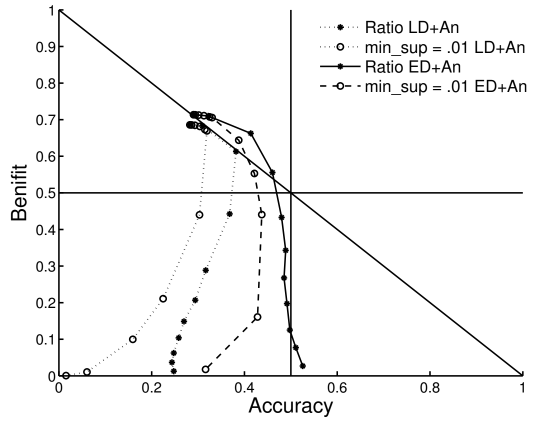
The intuition as to why including annotations could help is that it increases the potentially predictable misconfigurations. For example, while two users u, u' may have no actual resources that they have both accessed, knowing that they are both in a particular group (i.e., have "accessed" the corresponding group "resource") is information that can be leveraged to infer a misconfiguration. As such, a system employing annotations has a higher potential for uncovering misconfigurations than those that do not.

Figure 5 shows the gains that result from incorporating annotations in our framework. Each graph shows Ratio in an LD analysis, both with and without annotations, as well as traditional rule mining with $min\_sup = .01$ in comparable evaluations. (Other values of min_sup were comparable or worse.) The gains offered by the inclusion of annotations are noticeable but modest, for both Ratio and traditional rule mining. For example, when making predictions on the real dataset using Ratio with β = .7, accuracy improved from approximately 46.1% to 48.7%, and benefit from 25.7% to 27.6%. The addition of annotations did, however, increase benefit significantly in the synthetic datasets (Figure 5(b)) when parameters were tuned to maximize benefit, owing to the additional predictions that annotations allowed to be uncovered.

6.4 Eager Deduction



(a) Real Data



(b) Synthetic Data

Figure 6: Eager Deduction with annotations

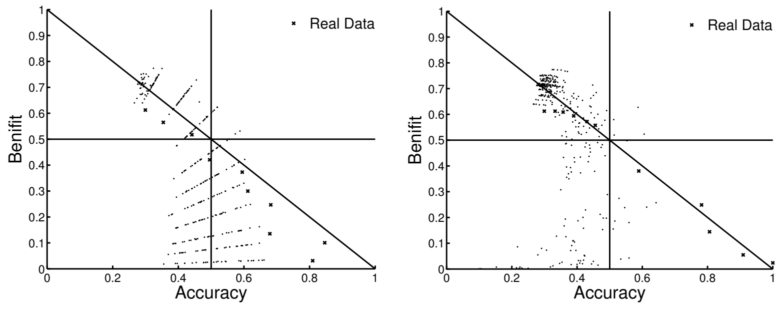
The previous section showed modest gains through introducing annotations that could be directly extracted from access credentials and added to exercised policy. In this section, we examine the power of incorporating all deducible policy atoms into $Visible_t$ as soon as those atoms can be deduced, i.e., an Eager Deduction (ED) system. The additional information this offers to the prediction engine results in substantially improved benefit and accuracy over that offered by LD and annotations alone, as shown in Figure 6. This benefit derives, we believe, simply from the engine using more complete information. Specifically, because more of the intended policy is known when making a prediction, similarities between users are more readily apparent and so the prediction is more likely to be an actual misconfiguration. More concretely, on the real dataset, using Ratio with β = .7, LD with annotations achieved a benefit of 27.6% and accuracy of 48.7%, while under ED with annotations this improved to 37.3% benefit and 59.5% accuracy. This represents a 35% increase in benefit and a 22% increase in accuracy. Note, as well, that the maximum benefit achievable on the real dataset by any algorithm in an ED configuration with annotations is 61.2%, and so the same point corresponding to $\beta = .7$ indicates that over 60% of the possibly identifiable misconfigurations were, in fact, correctly identified.

Two additional points are worth noting in Figure 6. First, Ratio again remains competitive with traditional rule mining (for which $min\_sup = 0.01$ is again shown as the best of the min_sup values we tried). As discussed in §6.2, the points at configurations emphasizing higher accuracy in Figure 6 represent very few predictions, and so while the gaps in accuracy are large in some cases — in favor of traditional rule mining in Figure 6(a), and in favor of Ratio in Figure 6(b) — they represent few actual predictions.

Second, in comparing Figures 6 and 4, we see that an ED system, despite its improvements over LD, does not fully regain the benefit and accuracy promised by the original ND analysis. For example, the $\beta = .7$ parameter mentioned previously exhibits a decrease in accuracy from 79% to 59% and a decrease in benefit from 39% to 37%. We nevertheless believe that the results in Figure 6 are compelling evidence of the utility of misconfiguration prediction in systems where misconfigurations need to be avoided.

6.5 Enforcing the Target Ratio

Recall that the primary motivation for the Ratio algorithm is to ensure that $\frac{Benefit}{Accuracy} \approx \beta$ for a given parameter β. While the justification for the way in which our algorithm accomplishes this is given in §5.2, the curves in the previous subsections provide little insight into the extent to which this is accomplished. To shed light on this, we show in Figure 7 the values of $\frac{Benefit}{Accuracy}$ at the end of each evaluation of misconfiguration prediction, for the different datasets and parameter values we considered. Figure 7(a) demonstrates that Ratio is able to provide very predictable ratio values for all of our datasets, while traditional rule-mining (Figure 7(b)) provides no such predictability. As a result, an administrator using Ratio can confidently set β at the birth of the system and achieve a long-term performance that will satisfy the chosen β. It is worth noting that in our real dataset, where $Exercised_t \subset Intended$ at the final time t, that the finishing points still exhibit the behavior sought by the Ratio algorithm.



(a) Ratio Method (b) $min\_sup = .01$

Figure 7: Scatter plot of $\frac{Benefit}{Accuracy}$ for different techniques

6.6 Performance

The focus of §4 was the development of an optimized incremental rule-mining algorithm that can be invoked as prescribed in §5.2 to generate misconfiguration predictions. In this section we report performance results for this algorithm, which we have implemented in Java.

Figure 8 shows boxplots that detail the time spent in lattice maintenance (§4.2), confidence distribution updating (§4.3) and in rule generation (§4.4) for various values of β on a 3GHz processor. Each point represented in a boxplot is the time spent in one invocation of the respective operation (lattice maintenance, confidence distribution update, or rule generation) in one 70-user, 50-resource synthetic dataset. Because each dataset includes annotations, the number of effective resources (actual resources plus groups and roles) escalated to between 120 and 164, depending on the dataset. Each boxplot represents all invocations in 10 different datasets. In each boxplot, the box shows the first, second and third quartile; the whiskers extend to cover all points within 1.5 times the interquartile range; and "+" denotes an outlier. These datasets (70 users, 50 resources, with annotations and eager deduction) represent the worst case for our algorithm of all the datasets we evaluated; our algorithm performs faster on all of the other datasets we used.

In order to gain an appreciation for the savings represented by the numbers presented in Figure 8, it is useful to compare them to certain non-incremental algorithm implementations that roughly provide alternatives to those steps represented in Figures 8(a)–8(c). For example, a very rough comparison point for the graph in Figure 8(a) is the operation of generating the *frequent itemsets* for a given database, which can be viewed as generating the lattice $(\mathcal{L}, <_{\mathcal{L}})$ described in §4.3. On the datasets represented in Figure 8, the median cost for doing so was 27ms, using a state-of-the-art C implementation due to Bodon [7]. This is more than $25\times$ the highest median shown in Figure 8(a) of 1.26ms.

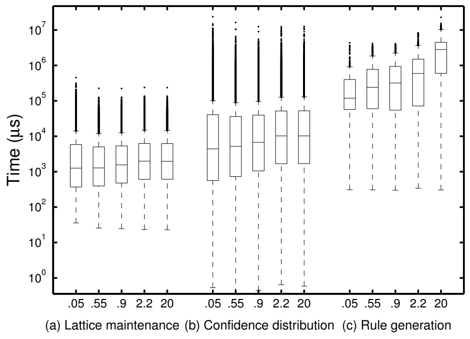


Figure 8: Runtime per operation (Eager Deduction with Annotations, 70 users, 50 resources) for $\beta \in \{.05, .55, .9, 2.2, 20\}$

A rough comparison for the cost of maintaining the confidence distribution incrementally in our algorithm, shown in Figure 8(b), is a non-incremental alternative suggested by Scheffer [23] that *approximates* the confidence distribution by sampling (in our use, single-consequent) association rules at random. For each size $s \in \{1 \ldots, |A| - 1\}$, where A is the set of attributes, this approach prescribes sampling a number of rules from among all rules $x \Rightarrow y$ with a precondition x of that size ($|x| = s$) and with a consequent y of size one ($|y| = 1$); Mutter et al. [21] attribute to Scheffer the suggestion of sampling 1000 such rules at random per value of s. For the datasets represented in the tests of Figure 8(b), the median cost of this approach in our implementation was 2.2 seconds, i.e., $220\times$ greater than the median shown in Figure 8(b).

A comparison for the cost of generating rules using our algorithm (Figure 8(c)) is a rule-generation algorithm proposed by Scheffer [23], conceptually a breadth-first walk down the lattice $(\mathcal{L}, <_{\mathcal{L}})$. Our implementation of Scheffer's approach on the dataset shown in Figure 8 induced a median cost for generating all rules ranked by predictive accuracy of 3 minutes, i.e., $122\times$ larger than the median for our algorithm (with $\beta = 20.0$, when our algorithm also generates all rules), shown in Figure 8(c). The maximum observed runtime of Schaffer's approach was approximately 3 hours, i.e., $800\times$ larger than the maximal point in Figure 8(c). Also note that smaller β values induce significantly less rule-generation time in our approach.

7 Conclusion

Policy misconfigurations that prevent legitimate accesses from succeeding are a significant impediment to the usability (and thus security) of access-control systems. Fortunately, accesses in a system often exhibit patterns that are indicative of intended policy, and access logs can be leveraged to identify policy misconfigurations before they cause harm.

In this paper, we improve the state of the art in identifying such misconfigurations in several ways. First, we provide a new, intuitive method for administrators to tune misconfiguration-detection systems to strike a desired balance between benefit (which measures how many misconfigurations are detected) and accuracy (which measures false positives in such detection), and we show empirically that this method is effective. Second, to detect misconfigurations we devise a new rule-mining algorithm that we show is significantly more efficient for scenarios such as ours; we expect this algorithm will also be useful in rule-mining applications unrelated to access control. Finally, we develop a new methodology for evaluating and deploying misconfiguration-detection systems, and we apply this methodology to several misconfiguration algorithms on both a real dataset and a collection of synthetic datasets. Our methodology allows previous results in misconfiguration detection to be interpreted more realistically, revealing some potential flaws in earlier analyses. Our methodology also shows that in most practical access-control systems more data is available that can be harnessed towards detecting misconfigurations than was previously used, and we show empirically that taking advantage of this data increases both the benefit and the accuracy of misconfiguration detection.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[2] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *23rd INFOCOM*, March 2004.

[3] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *6th ACM Conference on Computer and Communications Security*, 1999.

[4] J. Baixeries, L. Szathmary, P. Valtchev, and R. Godin. Yet a faster algorithm for building the Hasse diagram of a concept lattice. *Formal Concept Analysis*, pages 162–177, 2009.

[5] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *1999 IEEE Symposium on Security and Privacy*, May 1999.

[6] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *13th ACM Symposium on Access Control Models and Technologies*, pages 185–194, June 2008.

[7] F. Bodon. Surprising results of trie-based FIM algorithms. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, Nov. 2004.

[8] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proc. of the 17th Int. Conf. on Data Engineering*, 2001.

[9] D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the 12th International Conference on Data Engineering*, pages 106–114, 1996.

[10] T. Das, R. Bhagwan, and P. Naldurg. Baaz: A system for detecting access control misconfigurations. In *19th USENIX Security Symposium*, Aug. 2010.

[11] K. El-Arini and K. Killourhy. Bayesian detection of router configuration anomalies. In *2005 ACM SIGCOMM Workshop on Mining Network Data*, August 2005.

[12] C. Ezeife and Y. Su. Mining incremental association rules with generalized FP-tree. *Advances in Artificial Intelligence*, pages 147–160, 2002.

[13] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: mining and monitoring evolving data. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):50 –63, 2001.

[14] T. Jaeger, A. Edwards, and X. Zhang. Policy management using access control spaces. *ACM Transaction on Information and System Security*, 6(3):327–364, 2003.

[15] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining—revealing business roles for security administration using data mining technology. In *8th ACM Symposium on Access Control Models and Technologies*, June 2003.

[16] F. Le, S. Lee, T. Wong, H. Kim, and D. Newcomb. Detecting network-wide and router-specific misconfigurations through data mining. *IEEE/ACM Transactions on Networking (TON)*, 17(1):66–79, 2009.

[17] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb. Minerals: Using data mining to detect router misconfigurations. In *MineNet '06: 2006 SIGCOMM Workshop on Mining Network Data*, pages 293–298, 2006.

[18] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *2000 IEEE Symposium on Security and Privacy*, May 2000.

[19] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with semantic meanings. In *13th ACM Symposium on Access Control Models and Technologies*, pages 21–30, 2008.

[20] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In *14th ACM Symposium on Access Control Models and Technologies*, pages 95–104, 2009.

[21] S. Mutter, M. Hall, and E. Frank. Using classification to evaluate the output of confidence-based association rule mining. *AI 2004: Advances in Artificial Intelligence*, pages 133–148, 2005.

[22] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[23] T. Scheffer. Finding association rules that trade support optimally against confidence. *Intelligent Data Analysis*, 9(4):381–395, 2005.

[24] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *12th ACM Symposium on Access Control Models and Technologies*, 2007.

[25] P. Valtchev, R. Missaoui, R. Godin, and M. Meridji. Generating frequent itemsets incrementally: two novel approaches based on galois lattice theory. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2):115–142, 2002.

[26] D. Van Der Merwe, S. Obiedkov, and D. Kourie. AddIntent: A new incremental algorithm for constructing concept lattices. *Concept Lattices*, pages 205–206, 2004.

[27] A. Wool. Architecting the Lumeta firewall analyzer. In *10th USENIX Security Symposium*, 2001.

[28] Y. Yu, X. Qian, F. Zhong, and X. Li. An Improved Incremental Algorithm for Constructing Concept Lattices. In *World Congress on Software Engineering*, pages 401–405. IEEE, 2009.

[29] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. FIREMAN: A toolkit for FIREwall modeling and ANalysis. In *2006 IEEE Symposium on Security & Privacy*, 2006.

[30] M. Zaki and C. Hsiao. CHARM: An efficient algorithm for closed association rule mining. In *SIAM International Conference on Data Mining 2002*, 2002.