Discovering Access-Control Misconfigurations: New Approaches and Evaluation Methodologies

Lujo Bauer Yuan Liang
Carnegie Mellon University
{lbauer,yliang}@cmu.edu

Michael K. Reiter Chad Spensky
UNC Chapel Hill
{reiter,cspensky}@cs.unc.edu

ABSTRACT

Accesses that are not permitted by implemented policy but that share similarities with accesses that have been allowed, may be indicative of access-control policy misconfigurations. Identifying such misconfigurations allows administrators to resolve them before they interfere with the use of the system. We improve upon prior work in identifying such misconfigurations in two main ways. First, we develop a new methodology for evaluating misconfiguration prediction algorithms and applying them to real systems. We show that previous evaluations can substantially overestimate the benefits of using such algorithms in practice, owing to their tendency to reward predictions that can be deduced to be redundant. We also show, however, that these and other deductions can be harnessed to substantially recover the benefits of prediction. Second, we propose an approach that significantly simplifies the use of misconfiguration prediction algorithms. We remove the need to hand-tune (and empirically determine the effects of) various parameters, and instead replace them with a single, intuitive tuning parameter. We show empirically that this approach is generally competitive in terms of benefit and accuracy with algorithms that require hand-tuned parameters.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls; K.6.5 [**Security and Protection**]: Authentication; H.2.0 [**Information Systems**]: Security, integrity, and protection

General Terms

Security, Performance, Human Factors

Keywords

Access control, machine learning, misconfiguration

1. INTRODUCTION

Access-control policy often exhibits patterns across users and the resources they access, partly due to the use of groups and roles (perhaps only implicitly) in policy creation. These patterns are evidenced in the accesses allowed in the system. 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. Reactive access control, in which failed access attempts cause an administrator to be prompted to modify policy [6, 18], can help restore access after such failures. However, in many contexts (e.g., health information systems), the latency associated with such interventions may be unacceptable. Even when the cost of erroneously denying or delaying 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. As such, eliminating misconfigurations that erroneously deny access is essential in many contexts. Unfortunately, with few exceptions (e.g., [8, 7]), these kinds of misconfigurations have not been widely studied.

In this paper, we focus on identifying these access-control policy misconfigurations before they interfere with legitimate accesses. We improve upon previous work in several ways. We revisit the methodology for evaluating misconfiguration prediction algorithms and applying them to real systems. Prior work in which predictions were made on the basis of observed accesses evaluated these predictions by comparing them to intended policy. Any prediction that matched intended policy contributed to the measured *benefit*—the fraction of intended policy that was uncovered through predicted misconfigurations—and *accuracy*—the fraction of predicted misconfigurations that were consistent with intended policy. This included predictions that were consistent with already implemented policy and so were not indicative of misconfigurations at all.

We develop a richer framework that considers, in addition to exercised policy, the implemented policy that is deducible from observed accesses and correct predictions. This makes it possible to eliminate predictions that are already implemented from contributing to benefit and accuracy. Our framework allows us to more realistically evaluate misconfiguration prediction algorithms, and we show empirically that the actual benefits of misconfiguration prediction as cast in previous work are substantially overestimated. Fortunately, our framework makes it possible to improve misconfiguration prediction by making deducible policy available to the prediction engine. Our evaluation shows that through this we can recover much of the benefit and accuracy that previously appeared to be achievable.

We also recast the previous approaches to prediction so that they can be used effectively in different settings. Specifically, we remove the need for an administrator to tune multiple parameters until she achieves 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

CODASPY'12, February 7–9, 2012, San Antonio, Texas, USA.

tuning, while maximizing benefit and accuracy subject to this constraint. The difficulties of achieving this with some previous approaches, e.g., the approach of Bauer et al. [7], derive from their use of *association rule mining* [2], 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 a rule means 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 (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 generally increases accuracy and decreases benefit, how to incrementally adapt these two parameters to ensure $\frac{Benefit}{Accuracy} \approx \beta$, while maximizing benefit and accuracy, was unresolved.

To address this problem, we adopt an approach that combines confidence and support into a single parameter called *predictive accuracy* and that uses a Bayesian framework to determine the contributions of support and confidence to the expected accuracy of a rule [17]. 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. This provides a way to ensure $\frac{Benefit}{Accuracy} \approx \beta$ while providing good benefit and accuracy. We have scaled this approach to experiments involving tens of thousands of users or resources, making it potentially applicable, for example, for access control to physical or virtual resources in hospitals or to personal data in a social network.

To summarize, the contributions of our paper are: (1) A new framework for evaluating misconfiguration prediction algorithms and applying them to real systems, which reveals that previous evaluations overestimated the benefit of misconfiguration prediction but also makes it possible to largely recover the accuracy and benefit of prediction. (2) A new approach to configuring misconfiguration prediction algorithms that obviates the need for laborious hand-tuning of parameters, thus making it more feasible to apply misconfiguration prediction in different settings.

2. RELATED WORK

Several 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, 14, 20, 3, 21, 1]). These tools typically enable an administrator to verify that a set of policies is consistent, or that a policy obeys desired properties. Another approach incorporates similar techniques in policy-specification tools, which use the output of the analysis directly to help an administrator specify policy that meets desired goals [10]. More closely related are works that use rule mining or Bayesian inference to analyze router policies and automatically find misconfigurations (e.g., [13, 9, 12]). Similarly to firewall analysis, these approaches take as input configuration files and detect discrepancies between 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 [8]. Das et al.'s system takes as input both low-level file-system policy (which user can access which file) and metadata, such as group membership information separate from the low-level policy, and identifies misconfigurations that either deny legitimate accesses or allow erroneous ones. In contrast, we focus 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. Like Bauer et al.'s approach [7], 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.

		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: Relation R for a sample database

Our approach to detecting misconfigurations has some similarity to role mining (e.g., [19, 11, 15, 16]), which seeks to distill from a low-level policy a collection of roles that can 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.

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 sets of 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 *helpful* if the administrator grants a user Alice access to *d* and *incorrect* if the administrator denies access.

3.1 Confidence and Support

Rule mining is typically performed on a "database" representing a binary relation $R \subseteq O \times A$ between *objects* O and *attributes* A. (In our case, O is a set of users, A is a set of resources that users access, and oRa means that user o accessed resource a.) The *support* of $x \subseteq A$, denoted $\mathsf{S}(x)$, 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 $\mathsf{C}(x \Rightarrow y)$, is defined as $\mathsf{S}(x \cup y)/\mathsf{S}(x)$, 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 $\mathsf{S}(x \cup y) \geq \mathit{min\_sup}$ and $\mathsf{C}(x \Rightarrow y) \geq \mathit{min\_conf}$, and so higher values of $\mathit{min\_sup}$ and $\mathit{min\_conf}$ 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 $A = \{a, b, c, d, e, f, g, h\}$. Here, $\mathsf{S}(\{b, d\}) = 3/8$ and $\mathsf{C}(\{d\} \Rightarrow \{b\}) = 3/4$, and so the rule $\{d\} \Rightarrow \{b\}$ would be generated if $\mathit{min\_conf} \leq 3/4$ and $\mathit{min\_sup} \leq 3/8$.

3.2 Predictive Accuracy

Both $\mathit{min\_sup}$ and $\mathit{min\_conf}$ 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* [17].

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

$$\begin{aligned}&\mathsf{PA}(c, s)\\ &\quad = \mathbb{E}(\mathsf{A}(x \Rightarrow y) \mid \mathsf{C}(x \Rightarrow y) = c, \mathsf{S}(x) = s)\\ &\quad = \sum_a a\mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a \mid \mathsf{C}(x \Rightarrow y) = c, \mathsf{S}(x) = s\right) \qquad (1)\end{aligned}$$

where $\mathsf{A}(x \Rightarrow y)$ is the true accuracy of the rule $x \Rightarrow y$ and the expectation is taken with respect to choice of association rule $x \Rightarrow y$ uniformly at random from among all possible rules. Using Bayes' rule, (1) can be rewritten

$$\begin{aligned}&\sum_a a \frac{\begin{array}{r}\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{A}(x \Rightarrow y) = a, \mathsf{S}(x) = s\right)\\ \cdot\, \mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a \mid \mathsf{S}(x) = s\right)\end{array}}{\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{S}(x) = s\right)}\\ &\quad = \sum_a a \frac{\begin{array}{r}\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{A}(x \Rightarrow y) = a, \mathsf{S}(x) = s\right)\\ \cdot\, \mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a\right)\end{array}}{\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{S}(x) = s\right)} \qquad (2)\end{aligned}$$

where

$$\mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a \mid \mathsf{S}(x) = s\right) = \mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a\right)$$

since the underlying accuracy of a rule $x \Rightarrow y$ is independent of the support $\mathsf{S}(x)$ that its precondition x enjoys. The factor

$$\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{A}(x \Rightarrow y) = a, \mathsf{S}(x) = s\right)$$

is the probability of cs "successes" (yielding confidence c) out of s "trials" with a per-trial "success probability" of a, and so this value can be computed using the binomial distribution. Similar reasoning enables computing

$$\begin{aligned}&\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{S}(x) = s\right)\\ &\quad = \sum_a \begin{array}{r}\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = c \mid \mathsf{A}(x \Rightarrow y) = a, \mathsf{S}(x) = s\right)\\ \cdot\, \mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a\right)\end{array}\end{aligned}$$

provided that we can compute $\mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a\right)$, which is also needed for (2). Of course, since the only approximation we have for $\mathsf{A}(x \Rightarrow y)$ is $\mathsf{C}(x \Rightarrow y)$, we take

$$\mathbb{P}\left(\mathsf{A}(x \Rightarrow y) = a\right) \approx \mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = a\right) \qquad (3)$$

This yields the the following calculation of (1):

$$\mathsf{PA}(c, s) \approx \frac{\sum_a aB(cs; s, a)\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = a\right)}{\sum_a B(cs; s, a)\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = a\right)} \qquad (4)$$

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 $\mathbb{P}\left(\mathsf{C}(x \Rightarrow y) = a\right)$, i.e., the probability with which a rule, drawn uniformly at random from all possible rules, has confidence a. Scheffer [17] suggested estimating this probability through sampling.

While support and confidence are used to calculate the predictive accuracy, we need not choose predictions by placing conditions on confidence and support explicitly (e.g., the thresholds $\mathit{min\_sup}$ or $\mathit{min\_conf}$). As we will discuss in §4, by ranking rules in decreasing order of their predictive accuracy, we are able to make predictions in a way that maintains $\frac{\mathit{Benefit}}{\mathit{Accuracy}} \approx \beta$.

4. MISCONFIGURATION PREDICTION FRAMEWORK

In this section we detail our framework for evaluating misconfiguration prediction (§4.1), 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 (§4.2).

4.1 Model and Definitions

Our method for finding misconfigurations uses records of actual system accesses. 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., $a_{\ell+1} \not\in \{a_1, \ldots, a_\ell\}$) 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 $\mathsf{time}(a_\ell) < \mathsf{time}(a_{\ell+1})$. The *exercised policy* at time t is $\mathit{Exercised}_t = \{a_\ell : \mathsf{time}(a_\ell) \leq t\}$.

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 example, this credential might show that u is in a group that has access to other resources.) 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 some 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 a policy atom (u, r) and is made at a logical time $t \in \mathbb{N}$. However, predictions need not be issued at times distinct from each other, and generally they will not be. So, predictions $p_1, p_2, \ldots$ are only partially ordered by their logical times; specifically, $\mathsf{time}(p_\ell) \leq \mathsf{time}(p_{\ell+1})$. Let $\mathit{Predictions}_t = \{p_\ell : \mathsf{time}(p_\ell) \leq t\}$. A prediction p_ℓ is made by enumerating some number of association rules (with confidence less than one) in decreasing order of their predictive accuracies (see §4.2), computed

			No Deduction		Lazy Deduction		Eager Deduction	
t	Exercised	Deducible	New Rule(s)	New Prediction(s)	New Rule(s)	New Prediction(s)	New Rule(s)	New Prediction(s)
9	(u3, r6)	(u3, r2)	r6 ⇒ r7 r6 ⇒ r4	$\underline{(u3, r7)}$, ~~(u5, r4)~~	r6 ⇒ r7 r6 ⇒ r4	$\underline{(u3, r7)}$, ~~(u5, r4)~~	r2 ⇒ r4 r2 ⇒ r6 r6 ⇒ r2 r6 ⇒ r4 r6 ⇒ r7	$\underline{(u5, r2)}$, $\underline{(u2, r6)}$, $\underline{(u2, r7)}$, ~~(u2, r4)~~, ~~(u5, r4)~~
10							r2, r6 ⇒ r7	$\underline{(u3, r7)}$
11	(u3, r9)	(u5, r9)	r6, r7 ⇒ r4 r6, r7 ⇒ r9	$\underline{(u5, r9)}$	r6, r7 ⇒ r4 r6, r7 ⇒ r9	(u5, r9)	r2, r6, r7 ⇒ r9	$\underline{(u2, r9)}$
12	(u3, r10)	(u5, r10)	r6, r7, r9 ⇒ r4 r6, r7, r9 ⇒ r10	$\underline{(u5, r10)}$	r6, r7, r9 ⇒ r4 r6, r7, r9 ⇒ r10	(u5, r10)	r2, r6, r7, r9 ⇒ r10	$\underline{(u2, r10)}$
13	(u2, r6)	(u2, r7) (u2, r9) (u2, r10)	r6 ⇒ r7 r6 ⇒ r9 r6 ⇒ r10 r6 ⇒ r4	$\underline{(u2, r7)}$, $\underline{(u2, r9)}$, $\underline{(u2, r10)}$, $\underline{(u5, r2)}$, $\underline{(u3, r2)}$, ~~(u2, r4)~~	r6 ⇒ r7 r6 ⇒ r9 r6 ⇒ r10 r6 ⇒ r4	(u2, r7), (u2, r9), (u2, r10), (u3, r2), $\underline{(u5, r2)}$, ~~(u2, r4)~~		

Figure 2: A portion of the access log of the real dataset (described in §5.1) and resulting predictions (with $\beta = 20$, see §4.2), where users are denoted u1, u2, . . .; resources are denoted r1, r2, . . .; and the log is pictured beginning at $t = 9$ with $Exercised_8 = Visible_8$ $= \{$(u1, r1), (u2, r2), (u1, r3), (u3, r4), (u4, r5), (u5, r6), (u5, r7), (u6, r8)$\}$. Underlined atoms were added to $Helpful_t$ and canceled atoms were added to $Incorrect_t$.

using $Visible_t$ for $t = \mathsf{time}(p_\ell)$; i.e., the object set O is the users in $Visible_t$ and the attribute set A is the resources in $Visible_t$. Each rule $x \Rightarrow \{r\}$ derived in this way yields a prediction p_ℓ of policy atom (u, r) at time $t = \mathsf{time}(p_\ell)$ if and only if (i) $\forall r' \in x : (u, r') \in Visible_t$; (ii) $(u, r) \notin Visible_t$; and (iii) $(u, r) \notin Predictions_{t-1}$. 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., [8, 7]). For our evaluation, we determine the correctness of each prediction relative to an *intended policy*, $Intended$, 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 $Intended$ in our datasets in §5.1, but for now, we simply stipulate that $Deduced_t \subseteq Intended$ for all times t. Among the predictions that are consistent with intended policy, those that are not already deduced are *helpful*; we define these inductively as $Helpful_0 = \emptyset$ and $Helpful_{t+1} = Helpful_t \cup (Predictions_{t+1} \cap (Intended \setminus Deduced_t))$. The *incorrect* predictions can be defined more straightforwardly: $Incorrect_t = Predictions_t \setminus Intended$. 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, $Helpful_t \subseteq Visible_{t+1}$ and $Predictions_t \cap Deduced_t \subseteq Visible_{t+1}$. This means that all predictions at time t are resolved prior to predictions at time $t + 1$, though we stress this is a modeling simplification and is not necessary in practice.

In §5, we will evaluate the performance of this approach in three types of systems.

No deduction.

In a system with "no deduction" (ND), we define $Deduced_{t+1}$ $= Visible_{t+1} = Exercised_{t+1} \cup Helpful_t$. This is the setting in which previous proposals for misconfiguration prediction based on accesses have been evaluated [7]. A system permitting no deduction based on previous accesses might be, e.g., one 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 an "eager deduction" (ED) system, we stipulate that $Visible_{t+1}$ $= Deduced_{t+1}$, but generally $Visible_{t+1} \supseteq Exercised_{t+1} \cup Helpful_t$. 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 predictions. An example of an ED system would be one that reasons using credentials presented in previous accesses and gathered from previous predictions (e.g., as would be possible in a proof-carrying authorization system [4]) and then imports these into the prediction engine.

Lazy deduction.

In a "lazy deduction" (LD) system, $Visible_{t+1} = Exercised_{t+1}$ $\cup\, Helpful_t \cup (Predictions_t \cap Deduced_t)$, but we permit $Visible_{t+1} \subseteq Deduced_{t+1}$. As such, there are deductions that are not visible to the prediction algorithm (the meaning of "lazy"), but that are still relevant in measuring its success, as defined below. Specifically, a lazy system is one in which deducible policy that has not been exercised or predicted can nevertheless be consulted to "filter" predictions before they are posed to a human. We expect most practical systems to be eager, lazy, or in between.

Figure 2 illustrates a small portion of an access log from our real dataset described in §5.1, including the access that occurred in each time step ("Exercised") and the additional policy atoms that were deducible based on information accompanying that access ("Deducible"). As shown in this figure, ND counts all of (u5, r9) at $t = 11$, (u5, r10) at $t = 12$, and (u2, r7), (u2, r9), (u2, r10), and (u3, r2) at $t = 13$ as helpful, even though these can be deduced as already part of implemented policy at times $t = 11$, 12, 13, 13, 13, and 9, respectively. As such, LD does not count these as helpful. Because ED incorporates deducible facts when predicting misconfigurations, it uncovers policy more effectively, e.g., finding (u2, r6) at time $t = 9$ before it is exercised at $t = 13$ (and thus rectifying this potential misconfiguration before that access).

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:

$$Accuracy_t = \frac{|Helpful_t|}{|Helpful_t \cup Incorrect_t|}$$

Then, the accuracy *Accuracy* is simply $Accuracy_t$ at the maximum value of t in the execution. Note that the denominator of $Accuracy_t$

is the size of $Helpful_t \cup Incorrect_t$ and not of $Predictions_t$; the difference is predictions that were already deducible by the time they were made. These predictions are not helpful, but would presumably not be passed to a human, since their truth can be deduced already (and so are not "incorrect"). Similarly, benefit is defined

$$\textit{Benefit}_t = \frac{|Helpful_t|}{|Intended|}$$

and then the benefit *Benefit* is simply $\textit{Benefit}_t$ at the maximum value of t in the execution. When interpreting *Benefit* it is important to recognize that *Benefit* can never reach 1, since some policy atoms must be exercised before others can be predicted. Thus, it is important to interpret *Benefit* relative to the maximum value of *Benefit* that an algorithm could achieve in a given scenario. We show this in §5.

4.2 Algorithm for Enforcing Benefit vs. Accuracy Ratio

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 both was ad-hoc and varied across environments, and so the parameters needed to be tuned by trial and error, which is likely not possible in real-world applications, 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 scenarios, with no additional parameter tuning. A prediction engine can track $\textit{Accuracy}_t$ over time but, because it does not know $Intended$, it cannot track $\textit{Benefit}_t$ precisely. So, instead, our method tracks

$$\textit{Visible-Benefit}_t = \frac{|Helpful_t|}{|Visible_t|}$$

since $Visible_t$ is the engine's closest approximation to $Intended$ and one that should approach $Intended$ over time. The prediction engine can thus compute

$$\frac{\textit{Visible-Benefit}_t}{\textit{Accuracy}_t} = \frac{|Helpful_t \cup Incorrect_t|}{|Visible_t|} \qquad (5)$$

and monitor for the event in which this ratio drops below the target value β. That is, in the absence of predictions, $Visible_t$ will grow over time as new accesses are exercised (and t incremented), thus causing (5) to drop below β. Once that occurs, the prediction engine can issue predictions, adding each to $Helpful_t$, $Incorrect_t$ or $Visible_t$ once it is resolved, until (5) climbs above β. At this point, it suspends making further predictions until (5) falls below β. Since an incorrect prediction is added to $Incorrect_t$ and has no effect on $Visible_t$, and since a helpful prediction is added to both $Helpful_t$ and $Visible_t$, predictions can continue indefinitely only if they are always already deducible (and so have no effect) or helpful.

Once predictions are solicited as a result of (5) falling below β, they are derived at the same logical time t by enumerating rules (ignoring those with confidence 1) in decreasing order of their predictive accuracies based on 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 all rules with nonzero predictive accuracy, 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 continue until (5) again exceeds β or no new predictions are generated.

5. RESULTS

In this section we empirically evaluate our approach to applying misconfiguration prediction algorithms to real systems. The goals of our evaluation are twofold.

First, we seek to show that prior evaluations tended to overestimate the benefit of misconfiguration prediction in real systems. These prior evaluations considered only "no deduction" (ND) systems, which we believe are less likely to occur in practice than "lazy deduction" (LD) and "eager deduction" (ED) systems (as described in §4.1). In §5.2 we show that using an ND methodology to evaluate an LD system tends to significantly overestimate the benefit that is achieved by misconfiguration prediction. In §5.3, 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 §5.4 that by moving to an ED configuration, where the prediction engine can leverage all information deducible from past accesses, benefit can be more substantially increased (though still not to the levels promised by an ND evaluation).

Second, we evaluate our method for guiding predictions to strike a desired balance between accuracy and benefit (§4.2), which we call Ratio. We show that Ratio performs competitively, in terms of the benefit and accuracy that it achieves, with traditional association-rule-mining algorithms that require hand-tuning of min_conf and min_sup settings in order to achieve good performance. We perform this comparison on ND, LD, and ED systems (§5.2–5.4), and in each case we compare against the "best" settings of min_conf and min_sup (which vary and must be determined independently for each system). We also show that Ratio succeeds in ensuring the desired tradeoff between benefit and accuracy (§5.6), whereas the results of tuning min_conf and min_sup are often unintuitive and can only be determined empirically.

5.1 Datasets

To evaluate the misconfiguration prediction technique of §4.2 in the ND, LD and ED models, we require datasets for which we can construct *exercised policy*, *deduced policy*, and *intended policy*, as described in §4.1. None of these can be derived from implemented policies as represented in, e.g., file access-control lists or firewall rules, since implemented policies do not reveal which parts are exercised or, thus, what deductions those accesses permit. Moreover, implemented policies might not accurately reflect intended policy; indeed, our thesis is that by observing exercised policy, we can predict misconfigurations in implemented policy to bring it closer to intended. Fortunately, we have gained access to a dataset of exercised, deduced, and intended policies for a deployed system, and we augment this with numerous synthetic datasets. We detail both types of datasets below.

Real dataset.

The dataset generated by a real system is a variant of the dataset used in a previous work on misconfiguration detection [7]. 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, 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 comprises 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 use 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 system operation (e.g., because it was not required for any of the accesses observed). For the rest of the paper we will refer to this dataset as the *real* dataset.

Synthetic datasets.

A practical algorithm for misconfiguration detection should perform well on a variety of datasets. The single real dataset that we obtained is unlikely to be representative of datasets that would be generated in other real settings. Beyond differences in scale and density, policies in different datasets could be organized very differently. More specifically, the real dataset we used describes an environment where a large fraction of the allowed accesses are to resources to which everyone has access, which we expected could overstate the benefit of our approach.

To evaluate our system more thoroughly than is possible with just this one dataset, we created a range of *synthetic* datasets. Our goal was for our synthetic 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 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 iteration, we probabilistically decide whether to continue to the next iteration or discontinue role creation. Role creation terminates either by 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 creating 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 maximum t, $Exercised_t = Intended$. 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, using the same process as for the real dataset.

In §5.2–5.4, we present results using the real dataset described above, deferring treatment of the synthetic datasets until §5.5. We do so both to simplify §5.2–5.4 and because unlike the real dataset, the synthetic datasets permit us to additionally measure the impact of varying amounts of *chaff* on misconfiguration prediction. Here, chaff refers to direct delegations that do not conform to the structure (groups and roles) used in the generation of the remainder of the policy; i.e., 5% chaff implies that 5% of the possible accesses is enabled by direct delegations, and the remainder is enabled by group- or role-based policy. Unless otherwise specified, results for our synthetic datasets were based on 5% chaff. Also unless otherwise stated, we use 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 §5.5 are averages over the 30 datasets, 10 of each parameter set. Though we obtained similar results for much larger synthetic datasets, here we focus on these smaller datasets, which made it easier to evaluate the relative performance of different facets of our approach, across many parameter sets and many datasets per parameter.

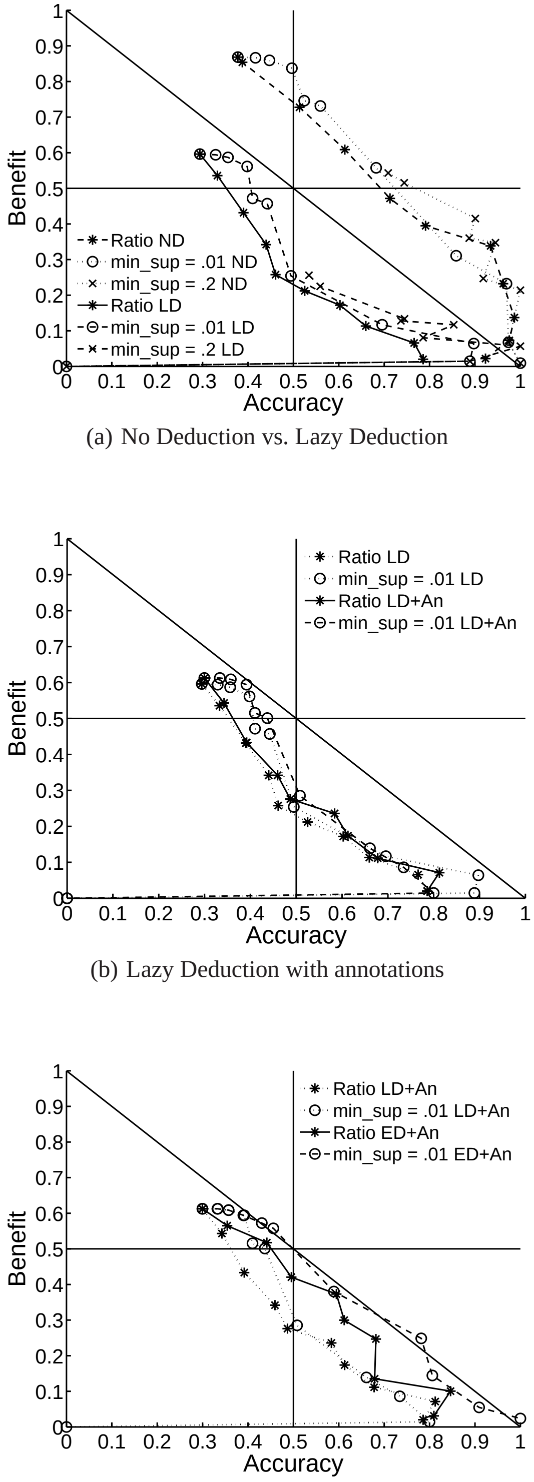
5.2 No Deduction Versus Lazy Deduction

We first examine the benefit and accuracy of our system using the ND model, which is the model in which previous work was evaluated [7]. Figure 3(a) shows plots of benefit (*Benefit*) and accuracy (*Accuracy*) achieved in an ND evaluation for the real dataset. 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.[1]

An immediate observation from this figure is the impressive benefit and accuracy of misconfiguration identification in the ND model. As we have contended previously, however, the ND model, by setting $Deduced_{t+1}$ to be simply $Exercised_{t+1} \cup Helpful_t$, precludes any deductions being made from the evidence of access-control policy that might have accompanied policy atoms. This is why so many predictions in Figure 2 (e.g., $(\mathtt{u3}, \mathtt{r2})$ at $t = 13$) are counted as helpful in the ND model even though they were already deducible (for $(\mathtt{u3}, \mathtt{r2})$, at time $t = 9$).

Perhaps the easiest way to take this additional information into

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



(a) No Deduction vs. Lazy Deduction

(b) Lazy Deduction with annotations

(c) Eager Deduction with annotations

Figure 3: Achieved benefit and accuracy on real dataset in ND, LD, and ED configurations

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 3(a) result. A notable lesson from these new curves is that the ND model substantially overestimates the 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 somewhat by allowing the prediction algorithm access to more information, as we will show in §5.3–§5.4.

Also evident in Figure 3(a) is that ND can overstate accuracy, again because the highest-ranked association rules tend to be already deduced. In this case, the comparison with LD reveals that many of the predictions that contribute to ND’s high accuracy are redundant with what can be deduced and that the non-redundant predictions, which are the only ones made by LD, are much less accurate. For example, using the Ratio method on the real data with $\beta = .7$, accuracy falls from 79.1% in ND to 46.1% in LD (and benefit declines from 39.5% to 25.8%). We will show in §5.3–§5.4 that much of this accuracy can also be recovered by giving the prediction algorithm access to more information.

Another take-away message from Figure 3(a) is that Ratio is competitive with the various tunings of traditional rule-mining based on min_sup and min_conf, typically trailing the best such curve by at most a few percentage points in each of benefit and accuracy. 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 strongly emphasize accuracy over benefit. However, these highest-accuracy configurations yield very few predictions, and so this gap in accuracy reflects only a small number of incorrect predictions by Ratio.

Figure 3(a) also shows the dramatic differences that can result from different values of min_sup as min_conf is varied. (For example, $min\_sup = .2$ attains a maximum benefit far below that of $min\_sup = .01$.) This motivates the move to our Ratio method to achieve a desired balance of $\frac{Benefit}{Accuracy}$. The extent to which the Ratio method accomplishes this is evaluated in §5.6.

5.3 Utilizing Annotations

A middle ground between an LD system and fully incorporating all deductions into the prediction engine (an ED system, evaluated in §5.4) is importing *annotations* into the prediction engine (i.e., into $Visible_t$) in the form of name of groups or roles to which a user has been demonstrated to belong in the course of gaining access to resources. For example, in our datasets we can extract group memberships from some of the credentials that accompany accesses.

We model annotations in our framework by adding new “resources” denoting groups and roles into the resource set. 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. During rule generation, these “resources” can appear only in x for a rule $x \Rightarrow y$. This permits rules like “Membership in Students $\wedge$ access to Student Lounge $\Rightarrow$ access to Computer Lab”. (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.)

The intuition behind including annotations 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") 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.

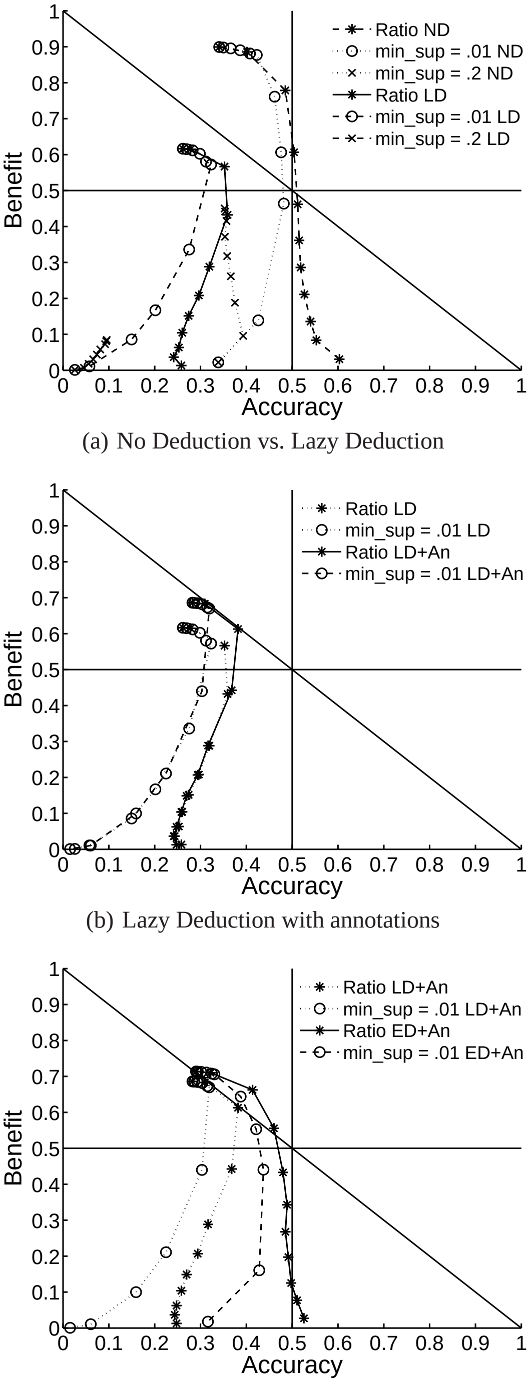
Figure 3(b) shows the gains that result from incorporating annotations in our framework. Each graph shows Ratio in an LD analysis, both with and without annotations, as well as traditional rule mining with $min\_sup = .01$ in comparable evaluations. (Other values of min_sup were comparable or worse.) 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 $\beta = .7$, accuracy improved from approximately 46.1% to 48.7%, and benefit from 25.7% to 27.6%.

5.4 Eager Deduction

The previous section showed modest gains through introducing annotations that could be directly extracted from access credentials and added to exercised policy. Next, we examine the power of incorporating all deducible policy atoms into $Visible_t$ as soon as those atoms can be deduced, i.e., an 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 3(c). This benefit derives, we believe, simply from the engine using more complete information. For example, in Figure 2, ED's prediction of $(\texttt{u2}, \texttt{r6})$ at $t = 9$, before it is exercised at $t = 13$, follows from the rule $\texttt{r2} \Rightarrow \texttt{r6}$, which is generated from the combination of the deducible atom $(\texttt{u3}, \texttt{r2})$ and the exercised atom $(\texttt{u3}, \texttt{r6})$. Because LD predicts $(\texttt{u3}, \texttt{r2})$ at $t = 13$, only after $(\texttt{u2}, \texttt{r6})$ is exercised, it fails to predict $(\texttt{u2}, \texttt{r6})$ in a helpful fashion. Due to such cases, using Ratio with $\beta = .7$, 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 that the maximum benefit achievable on the real dataset by any algorithm in an ED configuration with annotations is 61.2%, and so at $\beta = .7$ over 60% of the possibly identifiable misconfigurations were, in fact, correctly identified.

Two additional points are worth noting in Figure 3(c). First, Ratio again remains competitive with traditional rule mining (for which $min\_sup = 0.01$ is again the best of the min_sup values we tried). As discussed in §5.2, the points at configurations emphasizing higher accuracy represent very few predictions, and so while the gaps in accuracy at such configurations are large and favor traditional rule mining, they represent few actual predictions.

Second, in comparing Figures 3(c) and 3(a), 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 3(c) are compelling evidence of the utility of misconfiguration prediction in systems where misconfigurations need to be avoided.



(a) No Deduction vs. Lazy Deduction

(b) Lazy Deduction with annotations

(c) Eager Deduction with annotations

Figure 4: Achieved benefit and accuracy on synthetic datasets (averaged over 30 datasets) in ND, LD, and ED configurations

5.5 Results on Synthetic Datasets

Figures 4(a)–4(c) show plots analogous to those of Figures 3(a)–3(c). The primary difference between the methodologies for producing these figures is that each point in Figures 4(a)–4(c) is an average over all synthetic datasets described in §5.1. Recall that Figures 3(a)–3(c) are the results for a single dataset.

To avoid redundant discussion, we will not recount each experiment or the commonalities of results with those of §5.2–§5.4, except to note that these graphs support the conclusions drawn out in

§5.7. That said, the results of Figures 4(a)–4(c) appear to support our decision to evaluate on both real and synthetic datasets. As we conjectured, the real dataset yields results different from those obtained on the synthetic datasets. Evaluating on both yields a better understanding of how the different approaches perform in a range of settings that we believe to be realistic. For example, including annotations increased benefit more significantly in the synthetic datasets (Figure 4(b)) when parameters were tuned to maximize benefit, owing to additional predictions that annotations allowed to be uncovered. However, accuracy was worse on average in the synthetic datasets, particularly when tuned to maximize accuracy. Again, though, due to the few predictions made in these cases, the significance of this difference is unclear.

As previously mentioned, to cover a large range of parameter settings and enable us to report averages of many runs, we focused our evaluation on smaller datasets to produce Figures 4(a)–4(c). We also experimented with larger datasets to a degree and, interestingly, found the achieved benefit and accuracy tended to increase with the size of the dataset. For example, at $\beta = .9$ and 15% density, the benefit and accuracy increased by $\sim 23\%$ as the datasets grew from 50 users and resources to 250 users and resources. However, because these results were based on fewer datasets, we naturally must have less confidence in these results.

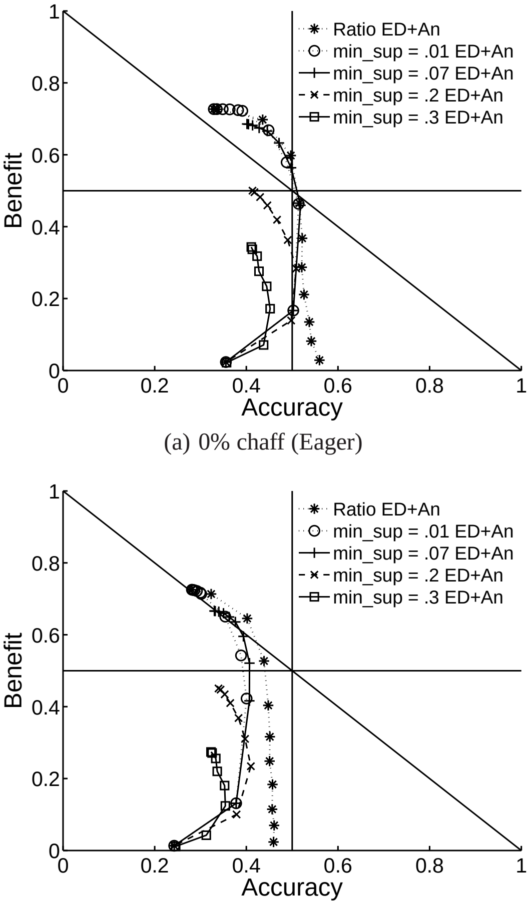
As discussed in §5.1, synthetic datasets also permit us to evaluate the impact of chaff (direct delegations) on misconfiguration prediction. Figure 5 shows the average benefit and accuracy achieved by different techniques on synthetic datasets with 0% and 10% chaff, versus the 5% chaff represented in the datasets in Figure 4. Each test was done using Eager Deduction with annotations. Generally these results suggest that Ratio is at least as robust to increasing chaff as traditional rule-mining methods, and perhaps is more so. Of course, by increasing the percentage of (random) chaff in the dataset, identifying any policy misconfigurations with association rule mining becomes increasingly difficult and, in the limit, untenable with any technique.

5.6 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 β. To examine the extent to which Ratio succeeds at accomplishing this, we show in Figure 6 the values of $(\mathit{Accuracy}, \mathit{Benefit})$ at the end of each evaluation of misconfiguration prediction, for both real and synthetic datasets and for the parameter values we considered. Figure 6(a) shows that Ratio is able to provide predictable ratio values for all of our datasets, while traditional rule-mining (Figure 6(b)) provides no such predictability. So, 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, the finishing points still exhibit the behavior sought by the Ratio algorithm.

5.7 Discussion

The evaluation in §5.2–§5.6 yields several high-level conclusions. First, our framework, which allows a more realistic evaluation of misconfiguration identification, reveals that previously reported results overestimated the effectiveness of misconfiguration identification for some realistic settings (§5.2). Second, much of this apparent loss can be regained through allowing the inference algorithm access to policy annotations and deducible policy (§5.3–§5.4). With these enhancements, the measured benefit of misconfiguration identification is substantial (Figure 3(c)). Third, our algorithm succeeds in maintaining the desired ratio between benefit and accuracy for all our datasets (§5.6). While evaluation on our synthetic datasets (§5.5) illustrates some differences in achieved benefit and accuracy versus the real dataset, both types of datasets support these three conclusions.



(a) 0% chaff (Eager)

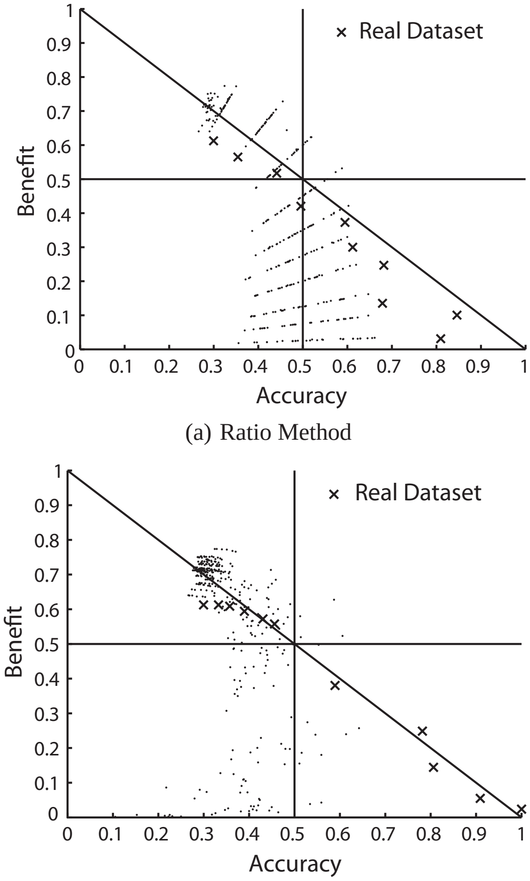
(b) 10% chaff (Eager)

Figure 5: Effects of increased chaff on misconfiguration prediction on synthetic datasets

6. CONCLUSION

Policy misconfigurations that interfere with legitimate accesses impede 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 two ways. First, we provide a new method by which administrators can 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, 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 by harnessing data available in most practical access-



(a) Ratio Method

(b) $min\_sup = .01$

Figure 6: Scatter plot of $($*Accuracy*$,$ *Benefit*$)$ for Ratio and traditional rule mining over 31 datasets, with control parameter (β in 6(a), min_conf in 6(b)) varied

control systems, the benefit and accuracy of misconfiguration detection can be largely recovered.

7. ACKNOWLEDGMENTS

This work was supported by NSF grant 0756998; by Carnegie Mellon CyLab under Army Research Office grant DAAD19-02-1-0389; and by ONR grants N000141010155 and N000141010343.

8. REFERENCES

[1] M. Abedin, S. Nessa, L. Khan, E. Al-Shaer, and M. Awad. Analysis of firewall policy rules using traffic mining techniques. *International Journal of Internet Protocol Technology*, 5:3–22, Apr. 2010.

[2] 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.

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

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

[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, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the Grey system. In *Information Security: 8th International Conference, ISC 2005*, volume 3650 of *Lecture Notes in Computer Science*, pages 431–445, Sept. 2005.

[7] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. *ACM Transactions on Information and System Security*, 14(1), May 2011.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] G. Stevens and V. Wulf. Computer-supported access control. *ACM Trans. Comput.-Hum. Interact.*, 16:12:1–12:26, September 2009.

[19] 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.

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

[21] 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.