Detecting and Resolving Policy Misconfigurations in Access-Control Systems

LUJO BAUER, Carnegie Mellon University
SCOTT GARRISS, Google
MICHAEL K. REITER, University of North Carolina at Chapel Hill

Access-control policy misconfigurations that cause requests to be erroneously denied can result in wasted time, user frustration, and, in the context of particular applications (e.g., health care), very severe consequences. In this article we apply association rule mining to the history of accesses to predict changes to access-control policies that are likely to be consistent with users' intentions, so that these changes can be instituted in advance of misconfigurations interfering with legitimate accesses. Instituting these changes requires the consent of the appropriate administrator, of course, and so a primary contribution of our work is how to automatically determine from whom to seek consent and how to minimize the costs of doing so. We show using data from a deployed access-control system that our methods can reduce the number of accesses that would have incurred costly time-of-access delays by 43%, and can correctly predict 58% of the intended policy. These gains are achieved without impacting the total amount of time users spend interacting with the system.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; H.2.0 [**Database Management**]: General—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms: Experimentation, Human Factors, Security

Additional Key Words and Phrases: Access control, policy inference, machine learning

ACM Reference Format:
Bauer, L., Garriss, S., and Reiter, M. K. 2011. Detecting and resolving policy misconfigurations in access-control systems. ACM Trans. Info. Syst. Sec. 14, 1, Article 2 (May 2011), 28 pages.
DOI = 10.1145/1952982.1952984 http://doi.acm.org/10.1145/1952982.1952984

1. INTRODUCTION

At some point, each of us has had a request to access a resource be denied that should have been granted. Such events are typically the result of a misconfiguration of access-control policy. Resolving these misconfigurations usually involves a human user to confirm that policy should be modified to permit the requested access. As a consequence, the misconfigurations are often disruptive and time consuming, and can be especially

This work was supported in part by NSF grant 0756998; by the Carnegie Mellon CyLab under Army Research Office grant DAAD19-02-1-0389; by Air Force Research Laboratory grant FA87500720028; by the AFRL/IF Pollux project; by ONR grants N000141010155 and N000141010343; and by a gift from Intel.

Authors' addresses: L. Bauer, CyLab, Carnegie Mellon University, 4720 Forbes Ave., CIC Building, Pittsburgh, PA 15213; email: lbauer@cmu.edu; S. Garriss, Google, 1600 Ampitheatre Parkway, Mountain View, CA 94043; email: sgarriss@google.com; M. K. Reiter, Department of Computer Science, University of North Carolina at Chapel Hill, Campus Box 3175, Sitterson Hall, Chapel Hill, NC 27599-3175; email: reiter@cs.unc.edu.

DOI 10.1145/1952982.1952984 http://doi.acm.org/10.1145/1952982.1952984

frustrating if the person who can change the policy cannot be reached when access is needed.

As a result, identifying and correcting misconfigurations before they result in the denial of a legitimate access request is essential to improving the usability of any access-control system. Eliminating all such misconfigurations in advance of accesses is arguably an unattainable goal (unless we invent a technique for reading users' minds). In this article we set out to show, however, that *most* such misconfigurations *can* be eliminated in advance. Eliminating a misconfiguration is a two-step process: we first must identify a potential misconfiguration and then attempt to resolve it by contacting the most appropriate human. The contributions of this article are fourfold: (i) to develop techniques to identify potential misconfigurations (Section 2), (ii) to develop techniques to resolve misconfigurations once they have been identified (Section 3), (iii) to evaluate these techniques on a data set collected from a deployed system (Sections 2.3 and 3.3), and (iv) to evaluate the effectiveness of these techniques when only partial data is available (Section 4).

Identifying misconfigurations. Intuitively, identifying misconfigurations is possible because in most practical settings there is significant similarity in the policy that governs access to related resources. Consequently, the history of permitted accesses may shed light on which accesses that have not yet been attempted are likely to be consistent with policy.

The method we explore for identifying access-control misconfigurations is a data-mining technique called *association rule mining* [Agrawal and Srikant 1994]. This technique enables the inference of if-then rules from a collection of multiattribute records. Intuitively, rule mining identifies subsets of attributes that appear in multiple records. These subsets are used to construct rules that suggest that, if all but one of the attributes of a subset are present in a record, then the last attribute should also be present. We employ association rule mining to identify potential misconfigurations in access-control policy from a global view of access logs by representing each resource that is accessed as an attribute, and the set of resources accessed by an individual as a record. Records for which the mined rules do not hold represent potential misconfigurations. However, such rules can often repeatedly produce incorrect predictions. We address this through the use of a feedback mechanism described in Section 2.2.1.

Resolving misconfigurations. Once a potential misconfiguration has been identified, we have to resolve the misconfiguration, which entails determining which human is best able to correct the access-control policy. In systems where policy is governed by a single administrator, this process is straightforward. In a distributed access-control system, however, it may not be clear which of potentially many users would be willing or able to extend the policy to grant the access. Since user interaction has a cost (in time and user aggravation), our technique must balance the desire to proactively resolve a misconfiguration with the desire to avoid unnecessary user interaction.

Our proposed resolution technique again relies on past user behavior; specifically, we determine which users have in the past created policy with respect to the particular user or resource in question, and suggest to those users that they correct the identified misconfiguration. In this way the misconfigurations can be resolved before they inconvenience the users that they affect. Compared to more reactive methods that attempt to resolve a misconfiguration only after an access that should be allowed fails, our technique can drastically reduce time-of-access latency and even the total time users spend interacting with the system without impacting the number of interruptions.

Evaluation. We evaluate the effectiveness of our techniques on data collected from Grey, an experimental access-control system that has been deployed and actively used at one of our institutions to control access to offices [Bauer et al. 2005]. This deployment is one in which misconfigurations that prolong access to an office substantially diminish the perceived usability of the system [Bauer et al. 2007a]. The dataset used in this article involves accesses to 25 physical doors by a community of 29 users. Each user accesses a door using a Grey application on her smartphone, which communicates via Bluetooth with an embedded computer that unlocks the door if the user is authorized to do so. Data drawn from logs allows us to reconstruct in detail how policy was exercised over the course of 10,911 attempted accesses. This data shows that there is a high degree of resource sharing; for example, 22 of the 25 resources were accessed by more than one user. Additionally, Grey supports dynamic policy creation, in that a user can delegate her authority to open a door using her smartphone, either at her own initiative or in response to a request to do so—for example, because another user has requested she do so from his Grey-enabled phone. This action results in the creation of a credential that represents this delegation. Grey's support for creating credentials to resolve policy misconfigurations allows us to record how users were able to resolve misconfigurations in real life. We augment this data with information collected from a user survey that asked what policy users were willing to implement should the need arise. Results from the survey allow us to determine precisely how users would resolve misconfigurations in scenarios that did not occur over the course of our deployment. This data is essential for evaluating the effectiveness of our technique for proactively resolving misconfigurations.

As expected, the performance of the methods we explore can be tuned to achieve a desired tradeoff between success in detecting and guiding the repair of misconfigurations, and the inconvenience to users of suggesting incorrect modifications to policy. On our data set, for a particular, reasonable set of parameters, we correctly identify 58% of intended, but not yet implemented policy, that is, 58% of the misconfigurations in the implemented policy. Using these predictions, our technique for resolving misconfigurations is able to proactively implement the needed policy for 43% of accesses that would otherwise have incurred a costly time-of-access delay. Each such correction results in significant savings in time-of-access latency. These gains are achieved without impacting the total time users spend interacting with the system.

We believe our techniques can be applied to a variety of access-control systems beyond the one on which we evaluated them, including those in which policies are organized using groups or roles, as is the case for RBAC systems. However, evaluating the effectiveness of our approach in such systems remains future work.

Performance with partial data. In practice, it may be infeasible to have an unfettered view of *all* the accesses in a system. For example, the data available to our misconfiguration detection algorithm may not include accesses by individuals who prefer to withhold their data or accesses of specific resources, like bathrooms, that users may prefer not to be monitored. Withholding data from our algorithm in this way will necessarily prevent it from making predictions about the user or resource whose data is missing and could potentially have a detrimental effect on the algorithm's ability to predict the accesses of other users or resources.

To investigate our algorithm's robustness to withholding data, we evaluate its performance on incomplete datasets. We construct these by removing from the dataset derived from our deployment of Grey all the accesses by various subsets of users or to various subsets of resources. We show that our ability to detect misconfigurations in the worst case decreases only slightly when subsets of data up to size 20% are removed. This indicates that, for our dataset, subsets of users can be removed for privacy or scalability reasons without significantly impacting the experience of the remaining users.

2. TECHNIQUES FOR IDENTIFYING MISCONFIGURATIONS

To identify potential policy misconfigurations, we first use *association rule mining* to detect statistical patterns, or *rules*, from a central database of previously observed accesses (Section 2.1). We then analyze our data using these rules to predict potential misconfigurations, or instances of the data for which the rules do not hold (Section 2.2). Once we determine whether or not a prediction was correct, we incorporate the result into a feedback mechanism to promote the continued use of rules that accurately reflect policy and prune rules that do not.

To illustrate the usefulness of this technique, consider the following scenario. Bob is a new student who is advised by Alice, a professor. Bob and Alice both work in a building where the same system controls access to Alice's office, Bob's office (which is shared with some of Alice's other students), a shared lab, and a machine room. When the department assigns Bob an office, it configures access-control policy to allow Bob to gain access to his office (e.g., by giving Bob the appropriate key). Though Alice is willing in principle to allow Bob access to the lab and machine room, both she and the department neglect to enact this policy.

The first time Bob attempts to access the shared lab space, he is denied access as a result of the misconfiguration, at which point he must contact Alice or the department to correct it. This process is intrusive and could potentially take minutes or even hours. However, the past actions of Bob's office mates suggest that people who access Bob's office are very likely to also access the shared lab space and machine room. The techniques we describe here allow us to infer from Bob's access to his office that Bob is likely to need access to the lab space and machine room. Identifying this in advance allows for the misconfiguration to be corrected before it results a denied access and wasted time. Detecting the misconfiguration is accomplished using only the history of accesses in the system; the technique is independent of the underlying access-control mechanism, policy, and policy-specification language.

2.1. Association Rule Mining

The objective of association rule mining is to take a series of records that are characterized by a fixed number of attributes, for example, Boolean attributes A through D, and discover rules that model relationships between those attributes. Suppose that, for 75% of the records where both A and B are true, D is also true. This property would give rise to the rule $A \wedge B \rightarrow D$. One measure of the quality of this rule is *confidence*, which is defined as the percentage of time that the conclusion is true given that the premises of the rule are true (75% for this example). Confidence represents the overall quality of a rule.

We employ the Apriori algorithm [Agrawal and Srikant 1994] to mine association rules, although other methods also exist. In Apriori, an *itemset* is a group of attributes, and the *support* of an itemset is the fraction of records in which those attributes occur together. Apriori first builds all itemsets with support exceeding a specified minimum. For each of these itemsets (e.g., $A \wedge B \wedge D$), Apriori enumerates all subsets of the itemset (D, $B \wedge D$, etc.). Using each subset as the premise for a rule and the remainder of the itemset as the conclusion, Apriori calculates the confidence of the rule, and keeps only rules whose confidence exceeds a specified minimum. The support of the rule is the support of the itemset from which it was generated, that is, of the itemset consisting of the rule's premise and conclusion.

In our context, each resource is represented by a Boolean attribute. Each user in the system is represented by a record in which the attributes corresponding to the resources that the user has accessed are set to true. For example, if attributes A through D each represent a resource, the record $\langle$false, true, true, false$\rangle$ would represent a user who has accessed resources B and C.

In our scenario, a small number of high-quality rules may identify statistically significant patterns, but on too small of a subset of the overall policy to be of much use. Thus, in tuning the output produced by Apriori, our objective is to balance the quality of the rules produced with the quantity of those rules. We achieve this by varying the minimum allowable confidence and support that a rule may have. Requiring a higher confidence biases the output toward rules that are true with high likelihood, while requiring higher support biases the output toward rules that describe patterns that occur with higher frequency. Section 2.3 describes the extent to which these parameters affect our ability to detect misconfigurations.

2.2. Using Mined Rules to Make Predictions

We use the rules output by Apriori to identify potential policy misconfigurations. A potential misconfiguration is a record for which the premises of the rule hold, but the conclusion does not. If a rule has a confidence of one, it implies that for all records which the premises of the rule hold, the conclusion of the rule holds as well. This means that every user who has accessed the resources represented by the premises of the rule has already accessed the resource mentioned in the conclusion. Hence, these rules cannot be indicators of misconfigurations, and so we ignore them. For each remaining rule, we identify the records for which the premise holds, but the conclusion does not. Each such record represents a potential misconfiguration; we predict that the user represented by that record should have access to the resource identified by the conclusion of the rule.

2.2.1. Feedback. One limitation of using mined rules to predict policy is that a dataset may contain several patterns that are statistically significant (i.e., they produce rules whose confidence exceeds the minimum) that are nonetheless poor indicators of policy. For example, the rule (perimeter door $A \rightarrow$ office D) may have medium confidence because door A is physically close to office D, and is therefore used primarily by the professor who owns office D and by his students. However, should this rule be used for prediction, the professor will be asked to delegate authority to enter his office to everyone who accesses door A.

To prevent the system from making repeated predictions on the basis of poor rules, we introduce a feedback mechanism that scores rules according to the correctness of the predictions they produce. A correct prediction is one that identifies a misconfiguration that a human is willing to repair. The idea is to penalize a rule when it results in an incorrect prediction, and to reward it when it results in a correct prediction. Rules whose score drops below a threshold are no longer considered when making predictions.

To illustrate how scores are computed, consider the following example. Suppose that the four resources A through D are commonly accessed together. This implies that Apriori will construct, among others, the itemsets $A \wedge B \wedge D$ and $A \wedge C \wedge D$. Suppose that Apriori then constructs from these itemsets the rules $A \wedge B \rightarrow D$ and $A \wedge C \rightarrow D$. Our feedback mechanism keeps a pairwise score for each premise attribute and conclusion, that is, (A, D), (B, D), and (C, D). If the rule $A \wedge B \rightarrow D$ is used to make a correct prediction, the scores for (A, D) and (B, D) will be incremented. If the prediction was incorrect, those scores will be decremented. If $A \wedge B \rightarrow D$ produces four incorrect predictions and $A \wedge C \rightarrow D$ produces three correct predictions, the scores for (A, D), (B, D), and (C, D) will be -1, -4, and 3, respectively.

The score for a rule is the sum of the scores of the premise attribute, conclusion pairs. In the scenario described above, $A \wedge B \rightarrow D$ would have a score of -5, while $A \wedge C \rightarrow D$ would have a score of 2. If the threshold for pruning rules is zero, $A \wedge B \rightarrow D$ would be pruned from the set of rules used to make predictions.

The reason for employing this technique instead of a more straightforward system where each rule is scored independently is that an itemset will often result in many more rules than the example presented above. (In fact, Apriori may produce a rule with every combination of A, B, and C as the premise.) Our technique allows us to more quickly prune groups of similar rules that are poor indicators of policy.

This feedback strategy performs well in our evaluation. However, in a hypothetical scenario, a rule could acquire a high positive score, which would require that many failures occur before the rule is pruned. In situations where this is unacceptable, the system could employ alternative feedback strategies that consider only recent accesses or compute the feedback score as a percentage of successful predictions. In the unlikely event that a rule's feedback score is high but its continued use is problematic, an administrator can manually prune the rule.

2.3. Evaluation

In evaluating our prediction techniques, we distinguish between several different types of policy. *Implemented policy* is the policy explicitly configured in the system that grants authority to users. Data from logs is sufficient to learn the entire implemented policy for our deployment environment. *Intended policy* includes implemented policy and policy that is consistent with users intentions but has not yet been configured in the system. *Exercised policy* is the subset of implemented policy that allowed the accesses that were observed in the logs. This is the only kind of policy that is used for making predictions; the other policy sets are used purely to evaluate the effectiveness of our methods. Finally, *unexercised policy* is the intended policy without the component that has been exercised.

Since intended policy can only be partially gathered from system logs, we distributed a questionnaire to each user who acts as first-line administrator for a resource. The questionnaire listed all of the resources that the administrator had the authority to delegate. For each such resource, the administrator was asked to select (from a list) the users to whom she would be willing to delegate authority. This allowed us to determine, for situations that did not occur in actual use of the system, whether the administrators would be willing to grant authority. These responses were used solely to analyze the accuracy of our predictions.

Our evaluations took place in a simulated environment defined by the usage data that we collected from our deployment. The logs describe 10,911 access attempts of 29 users to 25 doors. For each access attempt we logged who attempted to access which resource, when the attempt was made, and whether it succeeded; we refer to each such record as an *access event*. Several subsets of the implemented policy were completely or partially preconfigured, for example, seven perimeter doors were preconfigured to be accessible to all users through a policy that used groups, and gaining access to these doors required no policy reconfiguration.

We replay the sequence of access events logged by our system and, after every event, attempt to predict which new policies are consistent with the accesses observed so far. Figure 1 shows an overview of the steps that take place during each simulation iteration. The performance numbers we report are aggregated over the entire run of the simulation. A prediction is considered *accurate* with respect to exercised policy if the predicted policy is exercised in the data set for the first time after the the prediction was made. We also evaluate accuracy with respect to intended policy; here we count a prediction as accurate if it is consistent with intended policy that has not yet been exercised (regardless of whether it is ever exercised). Although accuracy of prediction is an important metric, in practice it is important to achieve both good accuracy and good *coverage*. Coverage is the percentage of the target space predicted, that is, the

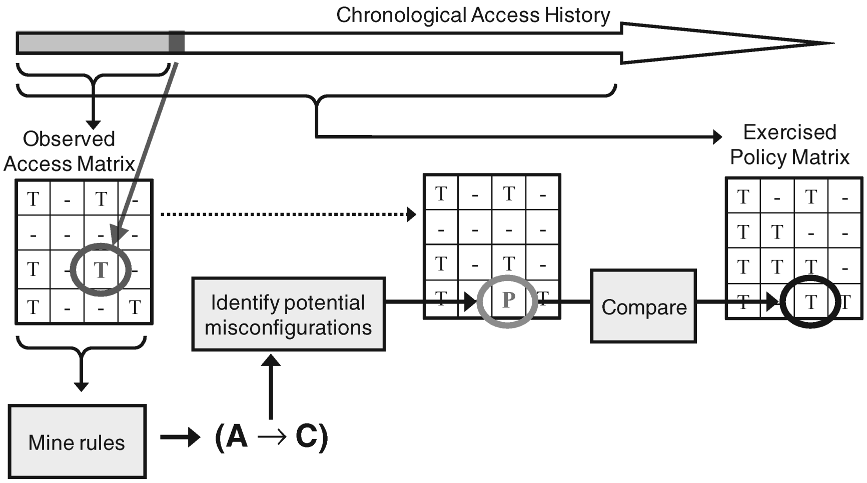

Fig. 1. Simulation steps at every iteration: (1) augment access matrix that represents policy exercised so far; (2) mine rules; (3) identify potential misconfigurations by applying rules to access matrix from step 1; (4) evaluate predictions by comparing against entries in the access matrix that represents all exercised policy (or intended policy; we report both sets of results).

percentage of accesses that would be allowed by the intended (or, alternatively, exercised) policy that we are able to predict.

2.3.1. Prediction Accuracy. We evaluate the accuracy of our predictions with respect to both exercised and intended policy. Accuracy is affected by tuning three parameters: the minimum confidence, *minconf*, and support, *minsup*, that a rule may have, and the feedback threshold, *fthresh*, that governs the minimum feedback score that a rule must have to be used in the prediction process (see Section 2.2.1). We evaluate three values for *minsup*: 0.03, 0.06, and 0.09, which represent supports of one, two, and three records. We evaluate confidence values ranging between 0.01 and 0.8. Values above 0.8 resulted in so few predictions that the accuracy was not meaningful.

In practice, the feedback score represents the result of previous attempts to resolve misconfigurations. However, the success rate of the resolution process depends on other factors, like our ability to efficiently detect whom to prompt to correct detected misconfigurations (see Section 3). To evaluate the accuracy of our predictions in isolation from the resolution process, we use *ideal feedback*, in which the scoring is based on what we know to be the intended policy in the system. We revert to the standard form of feedback when evaluating the resolution process in subsequent sections.

We evaluated *fthresh* values of −1, 0, and 1, using the technique described in Section 2.2.1 to score each rule. A rule is used to generate predictions only if it had no feedback score or if its feedback score was greater than or equal to the threshold value *fthresh*. For clarity, we present only the results obtained with *fthresh* set to 1, since that setting offered the greatest benefit.

Figures 2 and 3 show the prediction accuracy with respect to exercised and intended policy. As expected, including feedback in the prediction method improved accuracy. Using rules with higher confidence and support parameters uniformly improves the accuracy of predictions with respect to intended policy, but the benefit with respect to exercised policy peaks at a confidence of around 0.65. Intuitively, this shows that, while past behavior gives us more insight into intended policy, future accesses are not drawn uniformly from this space. We conjecture that a larger dataset, in which the exercised

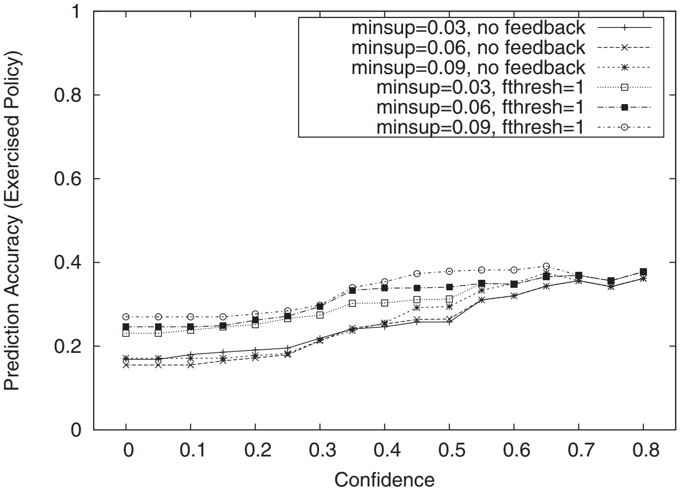

Fig. 2. Prediction accuracy (exercised policy).

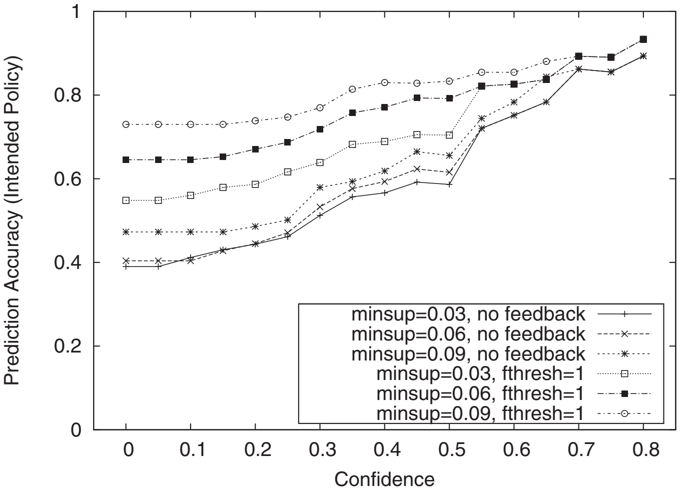

Fig. 3. Prediction accuracy (intended policy).

policy covered a greater part of the intended policy, would show improved performance with respect to exercised policy.

The increased accuracy achieved by using higher-quality rules lowers the total number of predictions, as we will discuss in Section 2.3.2.

2.3.2. Prediction Coverage. We evaluate prediction coverage for exercised and intended policy while varying the same parameters as when evaluating prediction accuracy. Our findings are shown in Figures 4 and 5. As expected, again, coverage decreases as we improve the accuracy of the rules. That is, the more accurate rules apply in a smaller number of cases, and so predictions that would be made by a less accurate rule are missed. Interestingly, a sharp dropoff in coverage doesn't occur until confidence values are raised above 0.5, suggesting that values in the range between 0.3 and 0.5 may produce rules that are both accurate and have good coverage. With reasonable

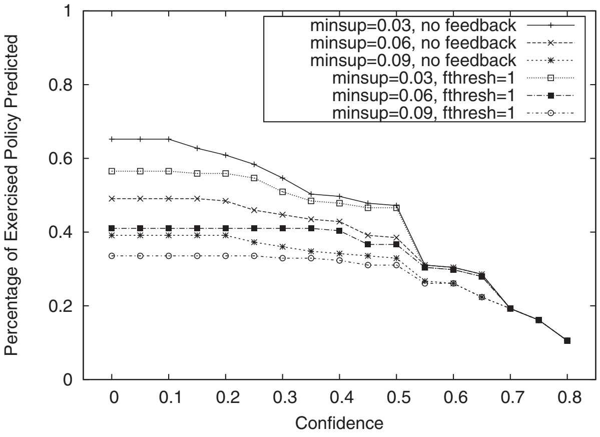

Fig. 4. Coverage of exercised policy.

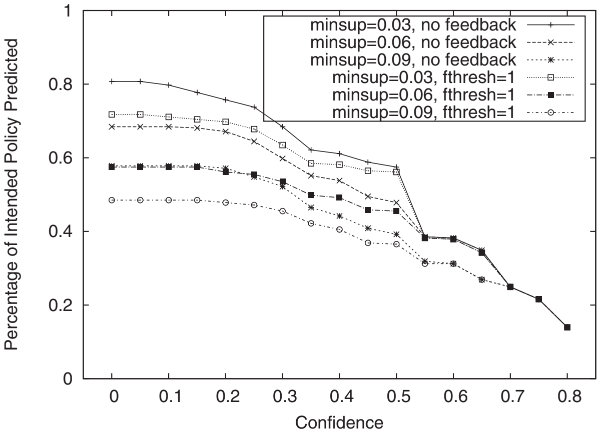

Fig. 5. Coverage of intended policy.

parameters (*minsup* = 0.03, *minconf* = 0.4, and *fthresh* = 1), our predictions cover 48% of the exercised policy and 58% of the intended policy.

2.3.3. Accuracy over Time. In addition to measuring the aggregate accuracy of predictions across an entire run of the simulator, it is interesting to know how prediction accuracy varies as a function of time. To measure this, we compute the accuracy of predictions over intervals that contain 50 predictions each. Figure 6 shows these results for different values of *minconf* with *minsup* fixed at 0.03. Rules with a higher minimum confidence make fewer predictions and so result in fewer data points. Different values of *minsup* exhibit similar trends.

Somewhat surprisingly, we found that the predictions made early in the simulation are, roughly speaking, as accurate as those made later when more history was available to the rule-mining algorithm. We conjecture that this is because the initial data points, though relatively few in quantity, were of high quality. In other words, the early accesses are made by a small number of people and to a small number of shared resources, and so were representative of a very small but often exercised subset of the intended policy.

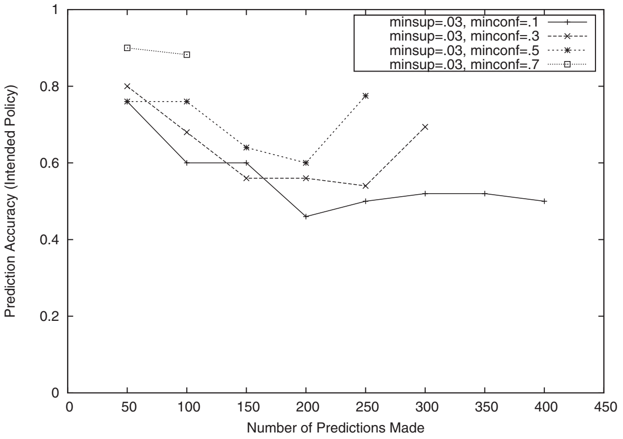

Fig. 6. Prediction accuracy versus time.

2.4. Discussion

We evaluated our ability to predict accesses that are consistent with policy with respect to both accuracy and coverage. Reasonably good performance was achieved using each metric, but, more importantly, we identified a combination of parameters for which the predictions were both reasonably accurate (i.e., not erroneous) and covered a large portion of the unexercised or eventually exercised policy. Specifically, minimum-confidence values between 0.3 and 0.5 achieved the best tradeoff. For these parameters, the increased accuracy resulting from our use of feedback to prune rules far outweighed the associated decrease in coverage.

Varying the minimum support that a rule must have did not impact the results as much as the other parameters. The higher coverage that resulted from a minimum-support value of 0.03 outweighed the increase in accuracy achieved by using higher values, and so for the evaluation in Section 3.3 we will fix the minimum support to 0.03.

Finally, we found that the predictions made with relatively few data points were roughly as accurate as predictions made with many more. Consequently, it appears that our methods would work well even in the early phases of adoption or deployment of a system that uses them.

One important question is to what extent will the success of our technique on our dataset carry over to systems with different access patterns and policies. Our technique exploits the observation that principals with similar access patterns are often granted access to those resources via similar policies. Thus, in any system where users' access patterns imply characteristics of access-control policy, our technique is likely to provide benefit, regardless of how policy is organized (e.g., only with roles, as in an RBAC system). Our technique will not be effective in a system where few shared resources exist or there is little overlap between the access patterns of individual users.

In systems in which permissions can be assigned directly to users, converting our predictions to policy is straightforward for the appropriate administrator. In systems where policy is more abstract (e.g., an RBAC system), acting on a suggestion may require adding a user to a new group or role, assigning permissions to a role, etc. To make our predictions easier to act on in such a system, it would be helpful to automatically generate suggestions as to what this step could be.

Our technique is effective as long as there is a discrepancy between implemented policy and intended policy. If a system is able to exactly implement the intended policy and the intended policy is fully known at the outset, then there are no misconfigurations to detect. However, regardless of the expressiveness of the policy language used by the system, neither of these conditions is likely to be met: the intended policy is often dynamic, that is, developed in response to new situations; and even when the intended policy is not dynamic, it is rarely fully specified at the outset. Therefore, it seems likely that most systems will have misconfigurations.

3. TECHNIQUES FOR REPAIRING MISCONFIGURATIONS

Once a potential misconfiguration has been identified, it is best if the policy is corrected as soon as possible to maximize the likelihood that the misconfiguration will not affect users. Since an identified misconfiguration might be erroneous, a user with the authority to modify policy must be queried to determine if the identified misconfiguration is consistent with the intended policy. If it is, then the user can repair the misconfiguration by altering or extending the policy. How the user elects to modify the policy is orthogonal to our work; it could entail changing an access-control list on a server or issuing digitally signed credentials that create a new group and delegate authority to that group. However, if there is only a single administrator in charge of policy, the question of which user to contact becomes trivial. Hence we look at the more difficult problem of repairing misconfigurations in distributed access-control systems in which multiple users may have the ability to modify policy.

Following the example in Section 2, when Bob discovers that he cannot access the shared lab space, he must determine whom to ask for assistance in resolving this misconfiguration. Since the lab space is shared, Alice may not be the only person with the authority to modify access-control policy. Professors Charlie and David may also be able to edit the policy governing the lab, but they may not be willing to grant authority to Bob. In this case, then, we would like the system to contact Alice and suggest to her that she amend policy to allow Bob access.

In this scenario, our previous work relied on Bob's intuition to direct queries to the most appropriate principals [Bauer et al. 2007b]. However, we would like to resolve such misconfigurations proactively, that is, at a time when Bob is not actively interacting with the system. Contacting the user to obtain his intuition at a time when he is not already interacting with the system would necessarily involve an additional user interruption. Instead, we attempt to determine which users are most likely to be able and willing to resolve the misconfiguration by analyzing past user behavior.

3.1. Users to Contact

The strategy that is most likely to succeed in repairing the misconfiguration is one that exhaustively queries all users who have the relevant authority, but this process is likely to annoy users who are unnecessarily interrupted. Therefore, when deciding which users to contact, we must balance the desire to repair the misconfiguration with the desire to avoid unnecessary user interaction.

To determine which users have the authority to resolve a misconfiguration, we could analyze the implemented access-control policy. However, the language for expressing policy varies widely between systems, and we wish to design a technique that is not specific to any particular language. Instead, we determine who has authority to repair a misconfiguration by analyzing the observed behavior of users when they resolved past misconfigurations. This is possible because, in addition to logging past access attempts, our system maintains data about whom users contacted when attempting to manually resolve misconfigurations. The intuition is that, because of similarities in the structure of access-control policy, principals who have rendered assistance in similar scenarios in the past are likely to provide assistance in the future, as well. For cases where this strategy is insufficient, we also consider the users who have previously accessed the resource, as they may be allowed to redelegate authority.

3.2. Directing Resolution Requests

We propose four different strategies for constructing a candidate list of these principals based on past user behavior. Once this candidate list has been assembled, it is sorted in descending order by the number of times the principal has rendered assistance in previous scenarios.

Strategy 1: OU The candidate list consists of the principals who previously rendered assistance to Other Users (OU) when they attempted to gain access to the resource mentioned in the prediction.

Strategy 2: OR The candidate list consists of the principals who previously rendered assistance to the user mentioned in the prediction when that user accessed Other Resources (OR).

Strategy 3: U The candidate list consists of the Union (U) of the lists produced by the OU and OR strategies.

Strategy 4: UPPA The candidate list contains the list U plus the Principals who Previously Accessed (UPPA) the resource mentioned in the prediction. Since these principals have not previously rendered aid to anyone, they will be sorted to the end of the candidate list.

The last strategy aims to cover the case where the structure of the policy that would authorize the predicted access is slightly different than the policy that authorized previous accesses. For example, should the system predict that Bob will access Alice's office, past observations may show that Alice's first access required the department to reconfigure policy. However, the department is unlikely to authorize Bob, whereas Alice (who has previously accessed the office) may be willing to provide the needed delegation.

3.3. Evaluation

Our objective is to evaluate the ability of our resolution techniques to resolve misconfigurations using the simulated environment described in Section 2.3, and to determine to what extent our techniques affect the usability of the system.

Proactively resolving misconfigurations decreases the number of *high-latency accesses*, or accesses where a misconfiguration must be corrected at the time of access. However, resolving misconfigurations does involve user input; this effect is measured by counting the number of *user interruptions*. The length of time from making a prediction to the instant when the predicted access will be attempted we call the *maximum resolution time*. For the attempted access not to be a high-latency access, the misconfiguration must be resolved within this interval after the prediction. Our *success rate* is the percentage of misconfigurations that we resolve within this interval. Finally, the *total user-interaction time* is a rough estimate of the time users spend interacting with the system, both with and without our techniques.

3.3.1. *Success Rate.*

The process for resolving a misconfiguration consists of constructing a candidate list of principals to query, determining how many of those candidates to query, and performing the queries. We evaluate the success rate using the four different strategies for constructing a candidate list presented in Section 3.2 with three different limits on the number of candidates to query.

Figure 7 shows the success rates obtained with these strategies when *minconf* and *minsup* are set to 0.4 and 0.03, respectively. The U and UPPA strategies are more likely to succeed than OU (which consults those who helped other users access the same resource) or OR (which consults those who helped the same user access other

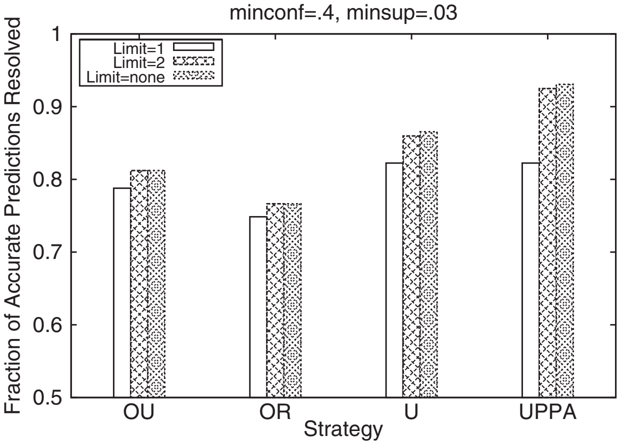

Fig. 7. Success rate of resolution strategies.

resources). This is not surprising, because U combines the results from OU and OR, and UPPA further extends U (by consulting users who previously accessed the same resource). In all cases, there was a noticeable benefit if the top two candidates were consulted instead of just the top candidate, but asking candidates beyond the top two offered little additional benefit. When only the most likely candidate was consulted, UPPA and U were equivalent, since UPPA's extension of the candidate list involved only appending to it. There is, of course, an increased overhead cost when consulting more than just the top candidate; we analyze this cost below in Sections 3.3.3 and 3.3.5.

3.3.2. High-Latency Accesses. If an access request is consistent with the intended access-control policy, but not with the implemented policy, then the user must attempt to resolve the misconfiguration prior to accessing the resource. The main cause of latency and inconvenience in this scenario is that human intervention is required to augment existing policy at the time of access. At best, this is inconvenient to both the person requesting access and the person who must repair the misconfiguration. At worst, the person who can resolve the misconfiguration may not be available, and the delay for repairing the misconfiguration may be very lengthy. The data we collected from our deployment encompasses 212 time-of-access policy corrections; in 39 cases a user's access was delayed by more than 10 min, and in 21 by over an hour.

Here we evaluate the extent to which our prediction and resolution techniques reduce the number of these high-latency accesses. The amount of reduction depends on the extent to which the predictions cover exercised policy (Figure 4) and the ability of the resolution process to succeed in the cases where exercised policy is correctly predicted (Figure 7).

Figure 8 shows the extent to which our prediction and resolution techniques reduce the number of high-latency accesses. Due to its higher success rate, consulting two users with UPPA reduces high-latency accesses more than consulting only one. Lower minimum-confidence thresholds yield greater reductions, because they produce greater coverage. More importantly, significant reductions can be achieved with confidence values that are likely to result in an acceptable amount of overhead: for example, a minimum-confidence value of 0.4 and a limit of two user consultations per misconfiguration reduces the number of high-latency accesses by 43%.

3.3.3. User Interruptions. Proactively resolving a misconfiguration may involve proactively querying users to determine the intended policy. We consider an interruption to

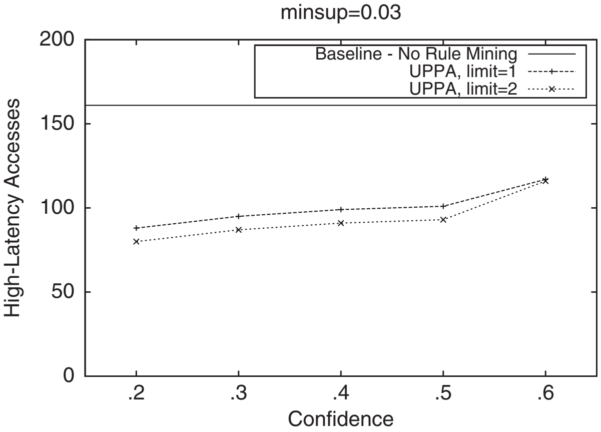

Fig. 8. Number of high-latency accesses.

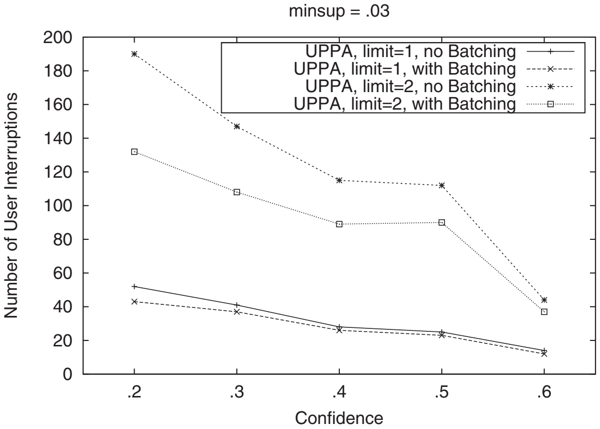

Fig. 9. Proactive user interruptions.

be any query directed to a user as part of the resolution process. This overstates the extent to which a user must be involved, because some queries directed to a user can be answered automatically by the user's phone in Grey, for example, if the misconfiguration can be resolved by policy that is implemented, but has not yet been exercised. Our survey did not ask users how they would implement their intended policies, so our simulations consider only policy that was exercised over the course of our deployment.

Often, several misconfigurations are detected at the same time, and their corresponding resolution processes query the same user, resulting in multiple interruptions. We introduce an optimization, *batching*, that groups these queries into a batch, allowing the user to answer them all as part of a single interactive session. This optimization is motivated by the observation that the incremental cost of additional user interaction is less than that of the initial interruption.

Figure 9 shows the number of user interruptions for our data when consulting either one or two principals chosen by the UPPA strategy and varying the minimum-confidence values for rules. Some principals, such as a department credential server, automatically respond to requests using implemented policy without requiring user interaction, and therefore these requests are not reflected in Figure 9. As expected,

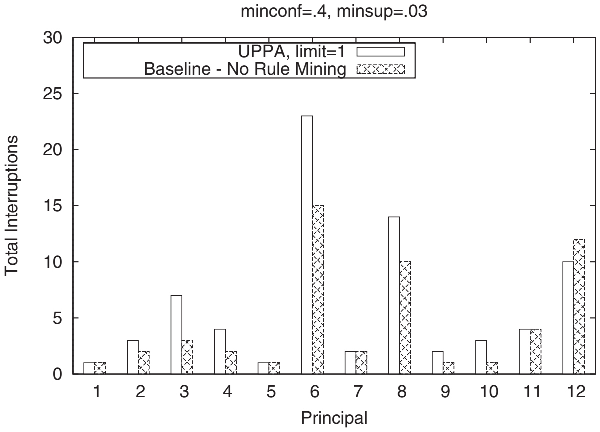

Fig. 10. Interruptions by principal.

consulting two principals instead of one increases the number of interruptions (and the success rate, as previously discussed). In some scenarios, the first principal consulted is one that responds automatically, but the second consultation requires user interaction. This explains how consulting two principals can produce more than twice as many interruptions as consulting only a single principal. Batching is more effective when consulting two principals and for low minimum-confidence values; these settings result in more attempts to consult users, and thus more opportunities to batch requests.

Interestingly, minimum-confidence values of 0.4 and 0.5 cause a marked decrease in the number of interruptions without significantly increasing the number of high-latency accesses (Figure 8).

Our techniques direct resolution requests to the principals who have provided assistance in similar scenarios in the past. If many principals administer a single resource, this approach could effectively penalize an administrator for providing assistance by directing more requests to that administrator. We discuss the limitations of our techniques and proposals for addressing them in Section 5.3. However, we find that this problem is not prevalent in our dataset. Figure 10 shows the distribution of requests (both proactive and reactive) between the principals who administer policy in the scenario where our techniques are used, and in the baseline scenario where rule mining is not used. We find that the distribution of requests in the scenario with our techniques is largely similar to the baseline scenario. When using our techniques, Principal 6 receives 23 requests over the duration of the simulation, which averages to one request every 21 days. Thus, in the context of our dataset, our techniques do not appear to unduly inconvenience even the most frequently interrupted users.

3.3.4. Maximum Resolution Time. One advantage of our approach is that it allows administrators to respond to requests to resolve misconfigurations when it is convenient for them to do so, rather than requiring them to resolve misconfigurations at the time of access. However, should an administrator tarry for too long, the user may attempt to access the resource before the administrator resolves the misconfiguration. Therefore, an administrator has a window of time in which to resolve the misconfiguration before it results in a high-latency access. We refer to this window as the *maximum resolution time*, which we define as the amount of time between the identification of a misconfiguration and the eventual accesses of the resource. The only predictions that are relevant for this metric are those whose resolution would prevent a high-latency access; that is, we are concerned only with predictions that are correct with respect to the exercised

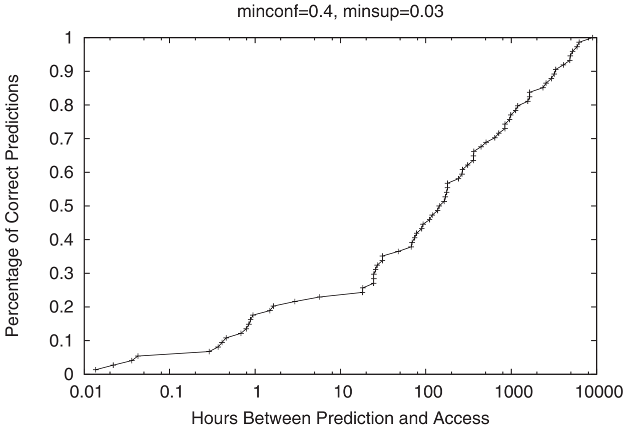

Fig. 11. Maximum resolution time.

policy. The maximum resolution time of all other predictions is effectively infinite. The maximum resolution time is independent of the resolution strategy employed, as it measures only the time in which the resolution must be completed.

Figure 11 depicts the maximum resolution times observed with *minconf* set to 0.4. Other *minconf* values yielded nearly identical results. The *x* axis shows the maximum resolution time, that is, the delay between the time the prediction is made and the time at which the access occurs. The *y* axis shows the percentage of correct predictions that must be resolved within that time.

Approximately 6% of the correctly predicted misconfigurations need to be resolved in under 10 min for high-latency access to be avoided; 18% need to be resolved in less than an hour; and 75% allow an administrator over a day to respond. The timeliness with which administrators will be able to respond will naturally vary according to the implementation of the system and, particularly, on the method of contacting the administrator. For some systems a response time of only a couple of minutes will be infeasible; for other systems, such as Grey, the system from which we drew the data for evaluating our approach, response latencies of around 1 min are common.

In any case, these results indicate that, in the vast majority of cases, misconfigurations are identified in a sufficiently timely fashion for an administrator to be able to repair these misconfigurations before they affect users.

Requiring administrators to resolve misconfigurations too frequently, for example, interrupting them once a minute several times in a row, could easily be a source of inconvenience. It would be straightforward to batch several requests for resolution into one interruption to an administrator. Because most requests do not need to be resolved immediately, the batched requests could be presented to the administrators less frequently. Reinterpreting the results depicted in Figure 11 tells us that limiting interruptions to at most one every 10 min will still allow over 94% of resolution requests to be serviced in a sufficiently timely fashion, to at most one every hour over 82% of requests, etc.

3.3.5. User-Interaction Time. The results described thus far demonstrate that increased proactive user interaction will reduce the number of high-latency accesses, but it is difficult to determine when the associated costs of interaction outweigh the benefit of avoiding high-latency accesses.

Any attempt to quantify this tradeoff is necessarily an approximation; there are many factors that influence a user's perception of the system that cannot be measured precisely. High-latency accesses, for example, typically annoy users far more than

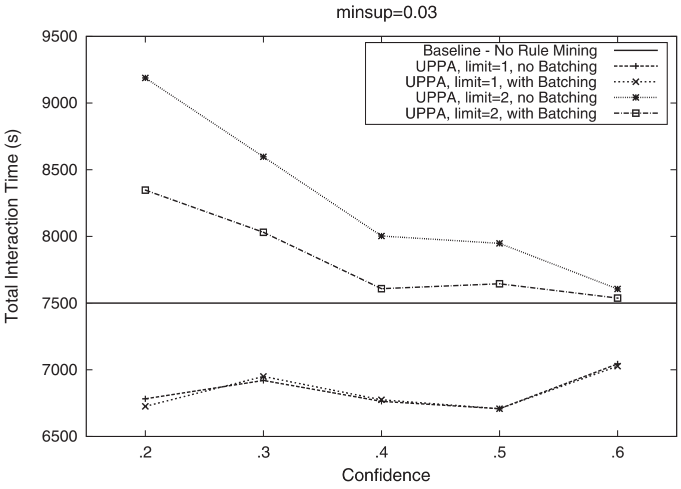

Fig. 12. Total user-interaction time.

interruptions of similar length that occur off the critical path to an access [Bauer et al. 2007a]. In this evaluation, we measure only (1) the delay observed by the user who is attempting to gain access while the policy is being reconfigured (how long Bob waits for access to Alice's office when Alice must reconfigure the policy); (2) the duration of the reconfiguration (how long Alice takes to reconfigure her policy to allow Bob access or to decide Bob should not have access); and (3) the length of time required to obtain credentials that describe implemented policy when no reconfiguration is necessary. In our system, the time-of-access latency caused by misconfigurations (1) ranged from 25 s to 18.6 h, with a median of 98 s (the average, heavily influenced by a few outliers, was 53 min). The median time observed for latency (2) was 23 s, only 5.8 s of which was spent selecting the appropriate correction (the rest was Alice finding her phone, etc.). Finally, the average duration of latency (3) was 6.9 s. These particular latencies are specific to our system; we anticipate that they would be much higher (on the order of hours) in systems that do not offer integrated support for resolving misconfigurations. However, our results will hold as long as there remains a similar relationship between the latencies (e.g., delay incurred by a user waiting for access is much greater than delay incurred by a user modifying policy), which we feel is likely for the vast majority of access-control systems.

With these caveats in mind, we use our simulation results and the timings above to approximate the total time that all users would spend interacting with the system (accessing resources or creating and correcting policy). If a misconfiguration must be resolved on the critical path of access, the total user-interaction time is the sum of latencies (1) and (2). If a misconfiguration is resolved prior to access, the total interaction time is simply latency (2), as the resulting credentials can be proactively distributed to the recipient of the delegation. Guided by data from our deployment, we weight the various latencies incurred by users as follows. Requests for credentials that occur on the critical path to an access being granted are assumed to take 6.9 s if the request can be completed without user interaction, and 98 s otherwise. Whenever a query requires user interaction, we assume that the user takes 23 s to respond to the query. We assume that each additional question posed to the user as part of a batch takes 5.8 s.

We calculate the total time all users (including administrators) would spend interacting with the system using the UPPA strategy and varying the minimum-confidence level required of the rules. The results are shown in Figure 12. Restricting UPPA to consult a single user results in a slight time savings for all minimum-confidence values tested. Allowing a second user to be consulted results in a small increase in total time for higher minimum-confidence values and larger increase in total time for lower values. Notably, with a minimum-confidence value of 0.4 and batching enabled, UPPA results in a only a slight overall time increase even if it is allowed to contact two principals.

In computing the total time spent interacting with the system, we assume that administrators react to (i.e., take notice of) requests to resolve misconfigurations in a sufficiently timely manner for the misconfigurations to be resolved before they affect users. An administrator not reacting quickly enough has the same effect as not detecting the misconfiguration. As discussed in Section 3.3.4, our analysis indicates that this assumption will hold in most cases.

3.4. Discussion

Each strategy for directing queries in our resolution mechanism is capable of resolving a majority of identified policy misconfigurations. In particular, the UPPA strategy, when allowed to consult two users, is able to resolve roughly 92% of such misconfigurations. The proactive resolution of these misconfigurations results in a drastic reduction (between 40% and 50%) in the number of high-latency accesses. Consulting two users in the UPPA strategy is more effective at resolving misconfigurations than consulting a single user, but this increase in effectiveness comes at a cost of significantly more user interruptions. Batching can reduce the number of user interruptions by approximately 15% when consulting more than one user.

To summarize: our results show that a significant reduction (43%) in the number of high-latency accesses can be achieved without impacting the total amount of time users spend interacting with the system.

Our technique for resolving misconfigurations is general in that it operates on the basis of observed behavior rather than inspection of access-control policy. It is therefore independent of both the underlying policy-specification language and the manner in which misconfigurations are repaired. It does, however, require that the system know who resolved previous misconfigurations. Since changes to policy are generally logged, we feel that this is a reasonable requirement.

The results describing the estimated user-interaction time are necessarily specific to our system. However, we believe them to be conservative, particularly for systems that do not provide integrated support for resolving misconfigurations (unlike ours). In such systems, the duration of each high-latency access is likely to be dramatically higher. Our techniques significantly reduce the number of high-latency accesses, and so the case for proactively resolving misconfigurations in such a scenario is likely to be even stronger than the one we present here.

4. RULE MINING WITH PARTIAL DATA

To this point, we have assumed that all access logs are available to the rule-mining algorithm. However, some scenarios may restrict the available data to a subset of the overall data, for example, to protect privacy or improve scalability. Specific users may prefer that their access logs be excluded from rule mining to prevent others from observing their behavior. Similarly, users may prefer that access to particular resources, such as a bathroom, not be logged. As the complexity of rule mining in our scenario grows with the number of resources, large organizations may find it necessary to mine rules using subsets of resources to improve scalability.

If data pertaining to a particular resource is withheld, the mined rules will not reference that resource, and therefore cannot predict accesses of that resource. Similarly, if a user's data is withheld, our technique will not predict any accesses for that user. However, removing large fragments of the input data can also have a detrimental effect on rule mining on the remaining data. In particular, consider two resources, A and B, that are governed by extremely similar policy (e.g., the rule $A \rightarrow B$ has high confidence), but are statistically unrelated to any other resources (e.g., $C \rightarrow B$ has low confidence for any $C \neq A$). If data pertaining to A is withheld, then it is unlikely that there will be many correct predictions regarding B, as the remaining rules will either not meet the minimum-confidence requirement, or will be pruned by our feedback mechanism after making several incorrect predictions.

Removing user data can also severely degrade the performance of rule mining on the remaining data. Following the above example, if a significant fraction of the users who have accessed A are withheld from rule mining, the support of the itemset (A, B) could drop below *minsup*. Alternatively, the confidence of the rule $A \rightarrow B$ could fall below *minconf*. Either situation would prevent predictions of B based on accesses of A.

These examples are worst-case scenarios. In our evaluation, we find that there is enough overlap between rules to compensate for the loss of subsets of the input data. In the context of our example, this implies that there is often a rule $C \rightarrow B$ that will predict many of the misconfigurations pertaining to B if the rule $A \rightarrow B$ is eliminated. In this section, we investigate the extent to which the removal of portions of the data impacts the performance of the remainder of the system. However, our dataset is drawn from a relatively small deployment, and it is therefore difficult to determine the extent to which these results imply properties about other policies.

4.1. Partitioning Strategy

With 25 users and 29 resources, our data can be divided into 2^{54} distinct subsets; it is therefore intractable to simulate all possibilities. Instead, we elect to partition the set of resources into blocks, and evaluate the performance when removing the data one block at a time. Our objective is to partition the data such that we can remove the data most and least likely to impact rule mining in order to estimate the maximum and minimum impact of withholding data. As patterns are formed when multiple users access the same resources, we would expect that removing data pertaining to the resources that were accessed by the most users to have a much greater impact than removing data pertaining to the resources that were accessed by the fewest users. Similarly, removing the data of the users who accessed the most resources should have a greater impact than removing the data of the users who accessed the fewest resources.

As the input data to the rule-mining algorithm does not include the total number of accesses, we elect to partition our data on the basis of unique accesses as follows. We first sort the resources by the number of unique users that have accessed each resource. If two resources have been accessed by the same number of unique users, the total number of accesses is used to break the tie. We partition the sorted list into quintiles (blocks), and assign each quintile a letter designation, A through E, with A representing the 20% of resources that have been accessed by the fewest unique users and E representing the 20% that have been accessed by the most unique users.

For users, we perform a similar process, instead sorting users by the number of unique resources that each user has accessed. Ties are broken using the total number of accesses made by a user. The users are also partitioned into quintiles, labeled F through J, with F representing the 20% of users who have accessed the fewest unique resources.

4.2. Evaluation

We present two metrics for each partitioning: the number of high-latency accesses and the total user-interaction time.

In Section 3.3, we showed that our techniques could significantly reduce the number of high-latency accesses incurred by users of the system. This is a positive result, as high-latency accesses are inconvenient and frustrating to users. Here, we analyze the

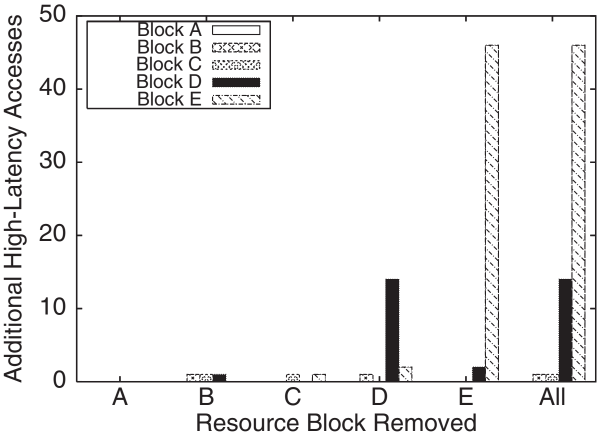

Fig. 13. Additional high-latency accesses incurred by the removal of each resource block.

number of high-latency accesses to determine the extent to which removing a block of data will diminish the benefit of our techniques. In Section 3.3.5, we also estimated that our techniques would not impose any net increase in the total time that users spend interacting with the system. Here, we investigate the extent to which this result holds when portions of the dataset are withheld from the rule-mining algorithm. We compute the user-interaction time using the method described in Section 3.3.5.

For these simulations, we fix all other parameters to values that yielded good results in the preceding sections. Namely, we fix *minconf* to 0.4, *minsup* to 0.03, *fthresh* to 1, the resolution strategy to UPPA with a limit of one request, and we disable batching as its benefit is minimal when limiting the resolution procedure to one request. The results obtained with a limit of two requests are similar.

4.2.1. Resource Partitioning. To measure the impact of removing a block, we count the number of high-latency accesses of resources in each resource block. For example, if block A contains door1 and Alice's access of door1 incurs a costly time-of-access delay to resolve a misconfiguration, this will be reflected only in the number of high-latency accesses for block A. Figure 13 depicts the net change in high-latency accesses incurred when a single block is removed. The tics on the x axis indicate which block of data was removed. Each x-axis tic has five bars, each representing the number of additional high-latency accesses that occur when that block of data is withheld. For example, tic D in Figure 13 shows that the removal of data for block D results in no additional high-latency accesses of resources in blocks A and C, one for block B, 14 for block D, and two for block E. The "all" tic on the x axis represents the scenario in which rule mining is not used at all. From this tic, we can see that rule mining with the given parameters can prevent 62 high-latency accesses across all blocks.

Interestingly, the number of high-latency accesses of resources in blocks A-C increases only slightly when rule mining is not used at all. This indicates that our techniques are primarily resolving misconfigurations pertaining to the resources in blocks D and E, i.e., the resources that are shared amongst the most users. Additionally, the removal of block D does not substantially increase the number of high-latency accesses to resources in block E, and, conversely, removing block E produces only minor effects on block D.

Figure 14 shows the estimated user-interaction time when no blocks are withheld (the "none" tic), when each resource block is withheld, and when rule mining is not used at all (the "all" tic). For each resource block, withholding that block's data still results

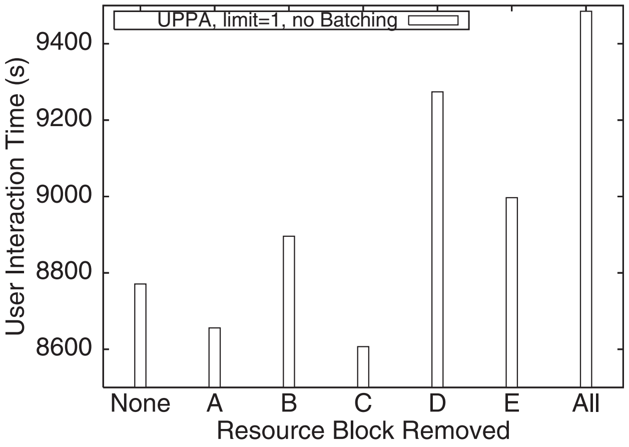

Fig. 14. Total user-interaction time with the removal of each resource block.

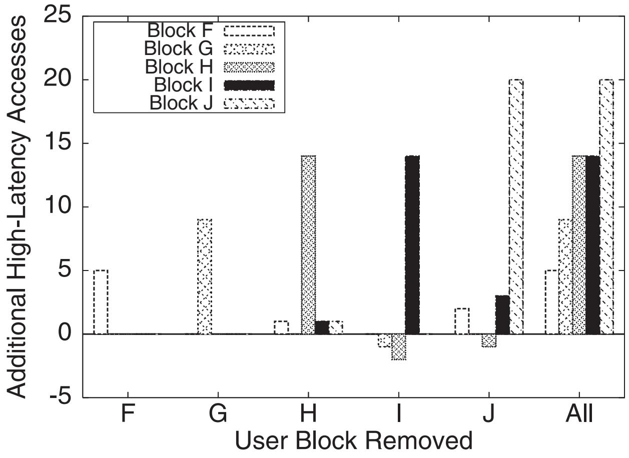

Fig. 15. Additional high-latency accesses incurred by the removal of each user block.

in an overall decrease in user-interaction time relative to the baseline scenario without rule mining. Additionally, withholding blocks A and C reduces the user-interaction time by approximately 1% relative to the scenario where no blocks are removed. The data from these blocks results in the construction of several relatively inaccurate rules, so removing them decreases the number of incorrect predictions that must be resolved. This gain in user-interaction time comes at a cost of no additional high-latency accesses for block A, and two additional high-latency accesses for block C. Thus the performance of the rule mining is strictly superior without block A in that it reduces user-interaction time without incurring any additional high-latency accesses.

4.2.2. User Partitioning. We repeat the previous evaluation, but partition the dataset by users rather than resources. Again, we measure the number of high-latency accesses and total user-interaction time. Figure 15 depicts the number of high-latency accesses incurred by removing one user block at a time. If Alice is a member of block J, her high-latency access of door1 will only be attributed to block J. In contrast to resource partitioning, we see (by inspecting the "all" tic) that the benefit of using rule mining is spread more evenly across all blocks of users, though the users who access the most

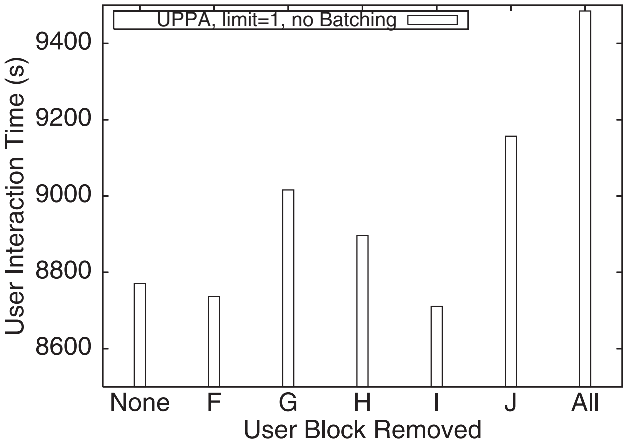

Fig. 16. Total user-interaction time with the removal of each user block.

distinct resources still gain the greatest benefit. Again, the negative effects of removing a block of users are largely constrained to the users included in that block.

Interestingly, removing block I actually *decreases* the number of high-latency accesses that occur in block G by two and block H by one. Similarly, removing block J decreases the high-latency accesses in block H by one. Removing a block of data often changes the percentage of cases in which a rule holds (i.e., its confidence). In this manner, removing blocks I and J increases the confidence of a few rules enough to exceed *minconf* and be used in the prediction process. These new rules identify several additional misconfigurations that result in a further decrease in the number of high-latency accesses.

The estimated user-interaction time with each user block withheld is shown in Figure 16. As was the case for resource blocks, the user-interaction time with each user block withheld remains lower than the baseline in which rule mining is not used. Removing blocks F and I results in less user interaction than the scenario where all data is used for rule mining. Since the removal of blocks F and I do not result in any additional high-latency accesses outside of their blocks, we can conclude that, in our dataset, these blocks are poor indicators of the policy for users in other blocks.

4.2.3. Partitioning Based on Total Accesses. We also evaluate a partitioning strategy that sorted based on total accesses rather than unique accesses. While the resulting blocks are similar, they are not identical. For example, one of the users who accessed the fewest unique resources (and was therefore located in block F) accessed these resources extremely often. When sorting on total accesses, this user would fall into block J. While specific results differ when partitioning by total accesses, we find that our overall conclusions hold with this partitioning strategy as well.

4.2.4. Discussion. Given the partitioning strategies described above, rule mining continues to produce useful results even when a fifth of the input data is removed. In particular, we found that removing blocks A, F, and I results in no change in or fewer high-latency accesses in other blocks as well as a slight decrease in the total user-interaction time. The removal of the other blocks does result in additional high-latency accesses outside of the removed block; however, the effects are generally small. Removing block J results in the largest increase, with four additional high-latency accesses. Removing any one of blocks B, D, E, G, H, or J slightly increases the user-interaction time (by a maximum of 6% for block D), but in each case the interaction time remains below that of the scenario where rule mining is not used. These results indicate that, for our dataset, subsets of users or resources can be removed for privacy or scalability reasons without significantly impacting the experience of the remaining users.

5. FURTHER DISCUSSION

5.1. Large Policies

As our dataset represents a somewhat small deployment, an interesting question is to what extent our approach scales to larger systems. There are two aspects of scalability to consider: the extent to which rule mining can identify useful patterns in the logs of large systems and the computational complexity required to mine these patterns. As the number of resources a user may access will generally not increase proportionally with the size of the organization, the matrix of past accesses is likely to be more sparsely populated in large systems than in ours. Thus, any resource that is shared by a large number of users will produce rules that are clearly distinctive in terms of confidence and support. However, data from large systems will, in absolute terms, contain a greater number of spurious patterns. Further research is necessary to determine whether tuning the rule-mining parameters to ignore spurious patterns would negatively affect our ability to mine rules concerning resources shared by a small group of users.

As the cost of mining rules grows with the number of attributes (i.e., resources in our system), scaling to environments with ever larger numbers of resources will eventually pose a problem to a centralized rule miner. However, recent algorithms can efficiently mine rules on datasets containing 10,000 attributes and 50,000 records [Yuan and Huang 2005]. In even larger organizations, the useful patterns are likely to be with respect to localized resources (e.g., resources for an individual lab or building). The data for the rule-mining process could therefore be partitioned along these boundaries and mined on different machines. In addition to improving scalability, decentralizing the data may mitigate users' privacy concerns and remove the requirement that a single entity have a global view of the entire system.

5.2. Data Representation for Rule Mining

In our approach, the input to the rule-mining algorithm is simply a collection of Boolean values, each indicating whether a particular user has accessed a particular resource. This representation, while effective in our scenario, excludes other potentially useful information. In particular, this representation does not include how often a user accessed a resource, the time of the most recent access, or any explicit structure between resources. This information is relevant to the prediction of future accesses, and if included, could improve our ability to proactively resolve misconfigurations.

For example, consider a scenario in which both Alice and Bob have each accessed resources A and B many times. Should Alice access resource C once, our technique would predict that Bob would be now likely to access C. This prediction may be incorrect if Alice was granted temporary access to C due to exceptional circumstances. In this case, the discrepancy between the number of times Alice accessed A and B and the number of times she accessed C might indicate that the rule $A \wedge B \rightarrow C$ should be ignored until Alice has accessed C a few more times. The drawback to this approach is that delaying predictions until more data becomes available could impede our ability to resolve misconfigurations before the time of access.

Additionally, the time elapsed since a user last accessed a resource can indicate the relevance of a rule mined from that data. Continuing the above example, the rule $A \wedge B \rightarrow C$ is more likely to reflect current access-control policy if Alice accessed C yesterday than if she accessed C over a year ago. As described, our techniques do not distinguish between the two scenarios. Thus, while Alice's ability to access C may be temporary, our technique assumes that access is granted permanently. To counter this, access data could be discarded after it reaches a certain age. This would allow the system to adapt to situations in which access is revoked after a period of time. The length of this interval would need to be carefully selected to balance the desire to mine rules from as much data as possible with the desire to quickly adapt when access is revoked.

The third type of information that our data representation does not capture is any explicit structure between resources. For example, given that Alice is authorized to write to dir/fileA, the properties of the file system may imply that she is also authorized to list the contents of dir, but do not imply anything about her ability to write to dir/fileB. Our techniques treat dir, dir/fileA, and dir/fileB as independent resources. This approach has two drawbacks: (1) it requires logs of multiple accesses to construct rules that are obvious from the structure of the resources (e.g., dir/fileA → dir) and (2) by considering each resource independently, it may be unable to mine more general trends, for example, dir → doorA. This could occur if Alice writes to dir/fileA, Bob writes to dir/fileB, and both Alice and Bob access doorA. When each resource is considered in isolation, there is insufficient data to discover the relationship between dir and doorA. One way of addressing this is to consider the structure of the resources when constructing the data that is given to the rule-mining algorithm, for example, each write to dir/fileA could also count as a listing of dir. Another approach to making use of external information about dependencies between resources would be to encode these dependencies directly as rules (e.g., dir/fileA → dir) that are evaluated with higher priority than other rules. Further investigation is necessary to determine if this approach is sufficiently general to cover all possible resource structures.

5.3. Directing Resolution Requests

Although our techniques for directing resolution requests are relatively simple, we showed that they were able to successfully resolve 92% of the correctly identified misconfigurations. However, some scenarios may necessitate more advanced heuristics for directing requests. For example, consider a scenario in which multiple administrators govern a single resource, but each administrator delegates only to a nonoverlapping subset of users. The resource could be a lab space shared between the research groups of two professors. Each professor is willing to delegate authority to access the lab to his or her students, but not to the students of the other professor. If we predict that a new student will access the lab, our technique would simply direct the resolution request to the administrator who has resolved the most misconfigurations in the past, which would probably be the professor with the larger research group. A more advanced heuristic might also consider the new student's access history and the rule used to make the prediction to determine to which group the student belongs.

Another drawback of directing resolution requests to the administrator who has resolved the most related requests in the past is that it effectively penalizes an administrator for providing assistance (though this was not severe in our dataset; see Figure 10). This problem may be particularly acute in a scenario where several administrators share equal responsibility for a resource. One possible way to address this concern would be to create a group containing the most helpful administrators in the candidate list, then randomly select an administrator from that group. If the group-selection parameters are chosen correctly, this approach could ensure that requests are spread equally among the administrators responsible for the resource.

6. RELATED WORK

Related work falls, broadly speaking, into three categories: work that uses similar data-mining or machine-learning techniques to analyze access-control policy, work that focuses on assisting users in policy administration, and distributed access-control systems to which our techniques are applicable.

6.1. Policy Analysis

The objectives of our work are related to those of *role mining* (e.g., Kuhlmann et al. [2003]; Schlegelmilch and Steffens [2005]; Molloy et al. [2008, 2009]), in which machine-learning techniques are used to extract roles from implemented policy. These roles may then serve as a guide when migrating a legacy system to one that supports role-based access control. Vaidya et al. [2007] formally defined the role-mining problem and bound its complexity. Similarly to role mining, our approach seeks to identify similarities in the input data that are indicative of shared policy. As with some approaches to role mining, a decision that our algorithm frequently has to make is whether two similar but slightly different patterns are different because of misconfiguration or are intended to be different. Unlike role mining, our techniques do not seek to characterize the implemented access-control policy. Instead, we aim to discover the portions of intended access-control policy that have not been implemented. Additionally, our techniques operate on the basis of observed behavior and are agnostic to the policy-specification language.

Many tools for empirical access-control policy analysis have been developed for firewalls (e.g., Bartal et al. [1999]; Mayer et al. [2000]; Hazelhurst et al. [2000]; Wool [2001]; Al-Shaer and Hamed [2004]; Yuan et al. [2006]). These tools generally provide ways to test or validate firewall policy against administrator intentions or other rules. Our work differs from these in multiple ways. First, since in the firewall setting there is typically one authority for the proper access-control policy (the human administrator or a high-level specification of that policy), there is no analog in that domain to a central concern here, namely, determining with whom to inquire about a potential policy change and minimizing the costs for doing so. Second, because it has the benefit of a policy authority that can be consulted freely, firewall analysis has focused on detecting traffic permitted in violation of policy, that is, to improve security, at least as much as what additional traffic should be allowed. The central technique we employ here, namely, learning from allowed accesses to predict others that are likely to be intended, focuses exclusively on improving the *usability* of discretionary access controls granted in a least-privilege manner.

The use of data-mining algorithms for detecting misconfigurations has recently been treated in several domains. Perhaps most closely related to our work is Minerals [Le et al. 2006], a system which uses association rule mining to detect router misconfigurations. By applying association rule mining to router configuration files in a network, Minerals detected misconfigurations such as router interfaces using private IP addresses that should have been deleted, user accounts missing passwords, and BGP errors that could result in unintentionally providing transit service or that could open routers to attack. The work of El-Arini and Killourhy [2005] similarly sought to detect router misconfigurations as statistical anomalies within a Bayesian framework. As in the aforementioned work in firewall analysis, though, whom to consult about apparent configuration errors and minimizing the costs of doing so were not issues considered in those articles; these issues are central to ours, however.

6.2. Policy Administration

Jaeger et al. [2003] sought to characterize the ways in which implemented policy may conflict with constraints that specify what accesses are prohibited. Their tool, Gokyo, allows the administrator to view the conflicts, and determine the least complex way of correcting them, possibly by marking them as explicit exceptions to a specified policy.

This tool could assist users of our techniques to resolve misconfigurations without introducing conflicts.

Many accesses in the context of health-care systems are granted by exception rather than because they are consistent with implemented access-control policy [Bhatti and Grandison 2007]. Using access-control logs that are annotated with whether or not an access was granted by exception, PRIMA uses heuristics to extract the exceptions that occur often enough to warrant further attention [Bhatti and Grandison 2007]. An administrator must then determine if the exception represents a misconfiguration. Applied frequently, PRIMA will help administrators to refine the implemented policy so that it expands to include exceptions that are consistent with the intended policy. As with our work, success is measured in terms of the coverage of intended policy. Our techniques differ from theirs in that we use data mining to identify exceptions and we attempt to correct the implemented policy prior to the time of access.

6.3. Distributed Access-control Systems

Our techniques operate using only data that describes past access attempts and who was contacted to resolve any misconfigurations. As such, our techniques should apply to systems employing a wide variety of access-control frameworks, such as RBAC [Sandhu et al. 1996], SPKI/SDSI [Rivest and Lampson 1996], RT [Li and Mitchell 2003], or PCA [Appel and Felten 1999]. We utilize observed behavior to determine to whom a resolution request should be directed. Some systems, such as PeerAccess [Winslett et al. 2005], explicitly encode hints as to how queries should be directed. Such hints could be considered in conjunction with observed behavior.

Though we have conducted our study in the context of Grey, there are numerous systems that we believe are well equipped to utilize the techniques we have developed here. In particular, similarly to Grey's support for dynamic delegation, many other systems enable retrieving credentials remotely from other parties as a means for satisfying access-control policy (e.g., Blaze et al. [1996]; Jim [2001]; Keromytis et al. [2003]; Goffee et al. [2004]; Becker and Sewell [2004]; Li and Mitchell [2003]). These mechanisms could be used to drive the correction of access-control misconfigurations once they are detected, though they do not detect those misconfigurations or predict how they might be corrected, as we have studied here.

7. CONCLUSION

In various application contexts (e.g., healthcare systems), the consequences of unnecessary delays for access to information can be severe. In such settings, it is essential to eliminate access-control policy misconfigurations in advance of attempted accesses; even in less critical environments, doing so can greatly improve the usability of the access-control system. In this article we have shown how to eliminate a large percentage of such misconfigurations in advance of attempted accesses using a data-mining technique called *association rule mining*. We demonstrated that, by doing so, we can greatly reduce the critical-path delays to accesses, while causing little additional overall work for users of the system. Specifically, for the access-control testbed from which our dataset was drawn, we showed that our methods can reduce the number of accesses that would have incurred a costly time-of-access delay by 43%, and can correctly predict 58% of the intended policy. These gains are achieved without impacting the total amount of time users spend interacting with the system. To accomplish these results, we contributed both new uses of rule mining and novel approaches for determining the users to which to make suggestions for changing policy. These results should be applicable to a wide range of discretionary access-control systems and settings, but particularly for systems that provide automated support for resolving misconfigurations via dynamic delegation.

REFERENCES

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*.

AL-SHAER, E. S. AND HAMED, H. H. 2004. Discovery of policy anomalies in distributed firewalls. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*.

APPEL, A. W. AND FELTEN, E. W. 1999. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*.

BARTAL, Y., MAYER, A. J., NISSIM, K., AND WOOL, A. 1999. Firmato: A novel firewall management toolkit. In *Proceedings of the IEEE Symposium on Security and Privacy*.

BAUER, L., CRANOR, L. F., REITER, M. K., AND VANIEA, K. 2007a. Lessons learned from the deployment of a smartphone-based access-control system. In *Proceedings of the 3rd Symposium on Usable Privacy and Security*.

BAUER, L., GARRISS, S., MCCUNE, J. M., REITER, M. K., ROUSE, J., AND RUTENBAR, P. 2005. Device-enabled authorization in the Grey system. In *Information Security: Proceedings of the 8th International Conference,* Lecture Notes in Computer Science, vol. 3650. Springer, Berlin, Germany, 63–81.

BAUER, L., GARRISS, S., AND REITER, M. K. 2007b. Efficient proving for practical distributed access-control systems. In *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS)*.

BECKER, M. AND SEWELL, P. 2004. Cassandra: Flexible trust management, applied to electronic health records. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*.

BHATTI, R. AND GRANDISON, T. 2007. Towards improved privacy policy coverage in healthcare using policy refinement. In *Proceedings of the 4th VLDB Workshop on Secure Data Management*.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*.

EL-ARINI, K. AND KILLOURHY, K. 2005. Bayesian detection of router configuration anomalies. In *Proceedings of the ACM SIGCOMM Workshop on Mining Network Data*.

GOFFEE, N. C., KIM, S. H., SMITH, S., TAYLOR, P., ZHAO, M., AND MARCHESINI, J. 2004. Greenpass: Decentralized, PKI-based authorization for wireless LANs. In *Proceedings of the 3rd Annual PKI Research and Development Workshop*.

HAZELHURST, S., ATTAR, A., AND SINNAPPAN, R. 2000. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the International Conference on Dependable Systems and Networks*.

JAEGER, T., EDWARDS, A., AND ZHANG, X. 2003. Policy management using access control spaces. *ACM Trans. Inform. Syst. Sec. 6,* 3, 327–364.

JIM, T. 2001. SD3: A trust management system with certified evaluation. In *Proceedings of the IEEE Symposium on Security and Privacy*.

KEROMYTIS, A. D., IOANNIDIS, S., GREENWALD, M. B., AND SMITH, J. M. 2003. The STRONGMAN architecture. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*.

KUHLMANN, M., SHOHAT, D., AND SCHIMPF, G. 2003. Role mining—revealing business roles for security administration using data mining technology. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*.

LE, F., LEE, S., WONG, T., KIM, H. S., AND NEWCOMB, D. 2006. Minerals: Using data mining to detect router misconfigurations. In *Proceedings of the SIGCOMM Workshop on Mining Network Data (Mine Net)*. 293–298.

LI, N. AND MITCHELL, J. C. 2003. Rt: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*.

MAYER, A., WOOL, A., AND ZISKIND, E. 2000. Fang: A firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy*.

MOLLOY, I., CHEN, H., LI, T., WANG, Q., LI, N., BERTINO, E., CALO, S., AND LOBO, J. 2008. Mining roles with semantic meanings. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*. 21–30.

MOLLOY, I., LI, N., LI, T., MAO, Z., WANG, Q., AND LOBO, J. 2009. Evaluating role mining algorithms. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*. 95–104.

RIVEST, R. L. AND LAMPSON, B. 1996. SDSI—a simple distributed security infrastructure. Presented at the International Cryptology Conference. Rump session.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *IEEE Comp. 29,* 2.

SCHLEGELMILCH, J. AND STEFFENS, U. 2005. Role mining with ORCA. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT)*.

VAIDYA, J., ATLURI, V., AND GUO, Q. 2007. The role mining problem: Finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*.

WINSLETT, M., ZHANG, C. C., AND BONATTI, P. A. 2005. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*.

WOOL, A. 2001. Architecting the Lumeta firewall analyzer. In *Proceedings of the 10th USENIX Security Symposium*.

YUAN, L., MAI, J., SU, Z., CHEN, H., CHUAH, C.-N., AND MOHAPATRA, P. 2006. FIREMAN: A toolkit for FIREwall modeling and ANalysis. In *Proceedings of the IEEE Symposium on Security and Privacy*.

YUAN, Y. AND HUANG, T. 2005. A matrix algorithm for mining association rules. In *Proceedings of the International Conference on Intelligent Computing* (ICIC). Lecture Notes in Computer Science, vol. 3644, Springer, Berlin, Germany.

Received October 2008; revised December 2009; accepted April 2010