Detecting and Resolving Policy Misconfigurations in Access-Control Systems

Lujo Bauer Scott Garriss Michael K. Reiter

February 6, 2008
CMU-CyLab-08-004

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Detecting and Resolving Policy Misconfigurations in Access-Control Systems

Lujo Bauer[†] Scott Garriss[‡] Michael K. Reiter[§]

Abstract

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 paper we apply association rule mining to logs of granted requests 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 consent of the appropriate user, of course, and so a primary contribution of our work is to automatically determine from whom to seek consent and 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 a costly time-of-access delay by 44%, and can correctly predict 58% of the intended policy. These gains are achieved without increasing the total amount of time users spend interacting with the system.

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 such misconfigurations usually involves a human user to confirm that policy should be modified to permit the requested access. As a consequence, these misconfigurations are often disruptive and time-consuming, and can be especially 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 computers to read users' minds). In this paper we set out to show, however, that *most* such misconfigurations *can* be eliminated in advance. Eliminating a misconfiguration is a two-step process: first we must identify a potential misconfiguration, then attempt to resolve it by contacting the most appropriate human. The contributions of this paper are threefold: (i) to develop techniques to identify potential misconfigurations (Section 2), (ii) to develop techniques to resolve misconfigurations once they have been identified (Section 3), and (iii) to evaluate these techniques on a data set collected from a deployed system (Sections 2.3 and 3.3).

[†]CyLab, Carnegie Mellon University
[‡]Electrical & Computer Engineering Department, Carnegie Mellon University
[§]Computer Science Department, University of North Carolina at Chapel Hill

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* [1]. This technique enables the inference of if-then rules from a collection of multi-attribute 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 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, statistically significant rules can often repeatedly produce incorrect predictions. We address this through the use of a feedback mechanism described in Section 2.2.

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 increasing 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 our institution to control access to offices for approximately two years. This deployment is one in which misconfigurations that prolong access to an office substantially diminish the perceived usability of the system [5]. Grey controls access to 25 physical doors, and has a community of 29 users. Data drawn from logs allows us to reconstruct a detailed scenario of how policy was created and used over the course of 10,911 attempted accesses. This data shows that there is a high degree of resource sharing (22 of the 25 resources were accessed by more than one user). Additionally, Grey supports dynamic policy creation, which allows us to determine 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. For a particular, reasonable set of parameters, we correctly identify over 50% of intended, but not yet implemented policy, i.e., over 50% of the misconfigurations in the implemented policy. Using these predictions, our technique for resolving misconfigurations is able to proactively implement the needed policy for 44% 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.

2 Techniques for Identifying Misconfigurations

To identify potential policy misconfigurations, we first use *association rule mining* to detect statistical patterns, or *rules*, from 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 the 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 (Section 2.2).

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, e.g., 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 [1] to mine association rules, although other methods also exist. Apriori first builds all possible *itemsets*, or groups of attributes, that have occurred together in more than a certain fraction of the records. This fraction is known as the *support* of an itemset. 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.

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 towards rules that are true with high likelihood, while requiring higher support biases the output towards 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

The rules output by Apriori must, in turn, be used 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. These rules do not allow us to identify any possible misconfigurations, 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.

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 on 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 four resources A through D are commonly accessed together. This implies that Apriori will construct the itemset $A \wedge B \wedge C \wedge D$. Suppose that Apriori constructs the rules $A \wedge B \rightarrow D$ and $A \wedge C \rightarrow D$ from that itemset. 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.

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 0, $A \wedge B \rightarrow D$ 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.

2.3 Evaluation

In evaluating our prediction techniques we distinguish between several different types of policy. *Implemented policy* is the policy explicitly enacted via credentials that grant 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 enacted through credentials. To discover the intended policy, we distributed a questionnaire to the users of our system asking them whether and under what conditions they would be willing to grant authority that did not correspond to the policy they enacted during their use of 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.

Our evaluations take place in a simulated environment defined by the usage data that we have 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, e.g., 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. 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

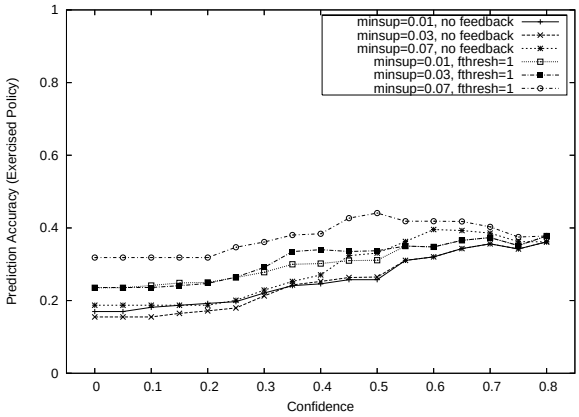

Figure 1: Prediction accuracy (exercised policy)

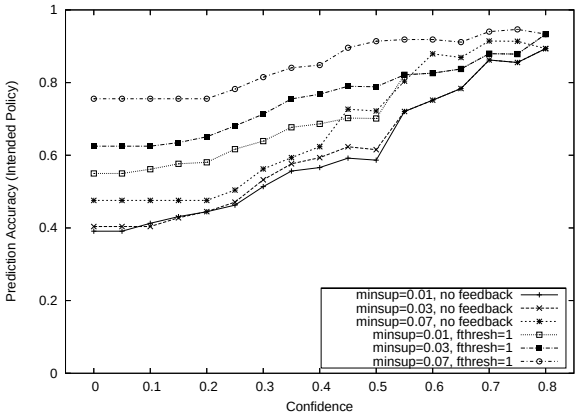

Figure 2: Prediction accuracy (intended policy)

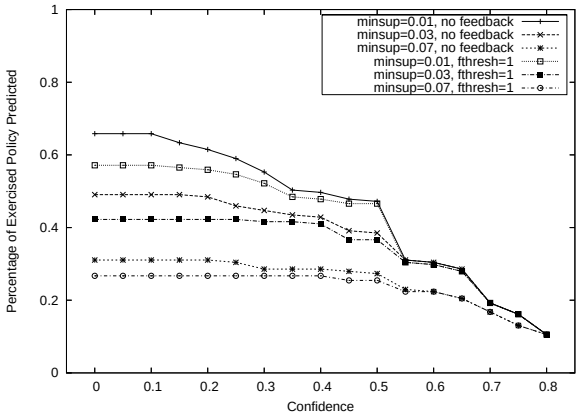

Figure 3: Coverage of exercised policy

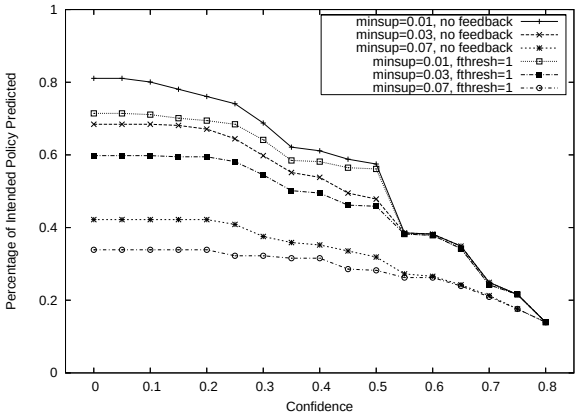

Figure 4: Coverage of intended policy

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 describes the percentage of target space predicted, i.e., the percentage of actual accesses present in the data set (or the percentage of intended policy) that we are able to predict.

Prediction accuracy First, 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). We evaluate three values for *minsup*: 0.01, 0.03, and 0.07, which represent supports of one, two, and three records. We evaluate confidence values ranging between 0.01 and 0.8. Confidence values higher than 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 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 1 and 2 show the prediction accuracy with respect to exercised and intended policy. As expected, including feedback in the prediction method improved accuracy for combinations of other parameters. 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 0.5. 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 data set, in which the exercised 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.

Prediction coverage We show prediction coverage for exercised and intended policy while varying the same parameters as when evaluating prediction accuracy. Our findings are shown in Figures 3 and 4. As expected, again, coverage decreases as we improve the accuracy of the rules. That is, the more accurate rules apply in a lower number of cases, and so predictions that would be made by a less accurate rule are missed. Interestingly, a sharp drop-off 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 parameters (minsup=0.01, minconf=0.4, and fthresh=1) our predictions cover 48% of the exercised policy and 58% of the intended policy.

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 5 shows these results for different values of *minconf* with *minsup* fixed at 0.01. Rules with a higher minimum confidence requirement 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 is available to the rule-mining algorithm. We conjecture that this is because although initially there are few data points on which to base predictions, the data points are 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 are representative of a very small but often exercised subset of the intended policy; the exercised policy is a larger fraction of the intended policy that covers those people and resources.

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 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.

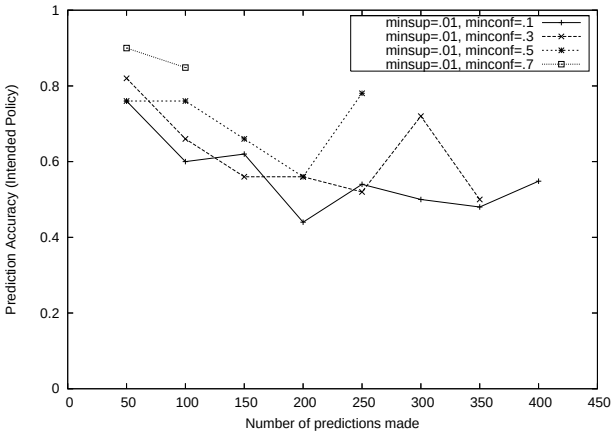

Figure 5: Prediction accuracy versus time

Varying the minimum support required before a rule was used did not have as great an impact on results as the other parameters. The higher coverage that resulted from a minimum support value of 0.01 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 0.01.

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 the extent to which the success of our technique on our dataset will 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. 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. We believe, however, that the latter scenarios reflect a minority of environments where distributed access-control is useful.

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, i.e., 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.

As our dataset represents a somewhat small deployment, an interesting question is to what extent our approach scales to larger policies. The complexity of mining rules grows with the number of attributes (i.e., resources in our system). As a result, scaling to environments with ever larger numbers resources will eventually pose a problem to a centralized rule miner. However, even in such a large organization, the useful patterns are likely to be with respect to localized resources (e.g., resources for an individual lab or building). An interesting avenue of future research would be to determine how the prediction process could be run over smaller subsets of data in very large

policies without overall loss of utility.

In addition to providing better scalability, dividing the resources into subsets prior to rule mining would enable the rule mining to be performed in a distributed fashion. As the rule mining algorithm would only need to know the access history pertaining to the resources in its subset, decentralization would help limit the privacy ramifications of using past actions as the basis of predictions. The prediction technique could work on anonymized data, but it would be impossible to resolve any potential misconfiguration without de-anonymization.

3 Techniques for Repairing Misconfigurations

Once a potential misconfiguration has been identified, it must be resolved in order to provide any benefit. This involves interacting with a user 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.

Following the example in the previous section, when Bob discovers that he cannot access the shared lab space, he must determine who to ask for assistance in resolving the conflict. 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 scenario, previous work relied on Bob's intuition to direct queries to the most appropriate principals [6], which would be Alice in this case.

However, we would like to resolve such misconfigurations proactively, i.e., at a time when Bob is not actively interacting with the system. Thus, obtaining the user's intuition would necessarily involve an additional user interruption. Instead, we attempt to determine which users are most likely to resolve the misconfiguration by analyzing past user behavior.

3.1 Users to contact

The challenge is to determine which user should be contacted to resolve a misconfiguration. If access-control policy is governed by a single administrator, then this process is straightforward. However, should more than one user have authority to edit policy, then the process is less clear.

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 embitter the users who are wantonly 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 appropriate 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 who 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 that 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 assistance 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

We measure the effectiveness of our resolution techniques using the simulated environment described in Section 2.3. Our objective is to evaluate the ability of the technique described in Section 3 to resolve misconfigurations, and to determine to what extent our techniques affect the usability of the system.

Our ability to resolve misconfigurations is measured by our *success rate*, which is the percentage of misconfigurations that it can resolve. 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 interaction; this effect is measured by counting the number of *user interruptions*. Finally, the *total user interaction time* estimates the extent to which users must interact with the system, both with and without our techniques.

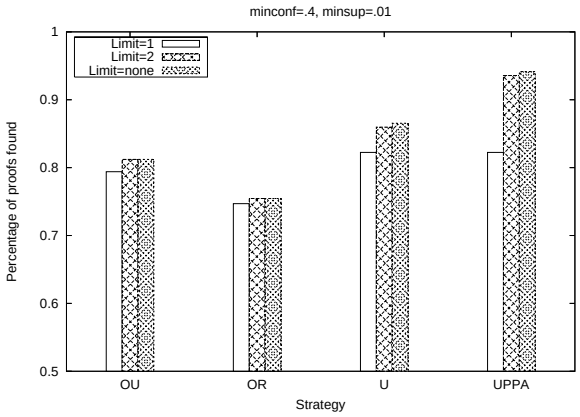

Figure 6: Success rate of resolution strategies

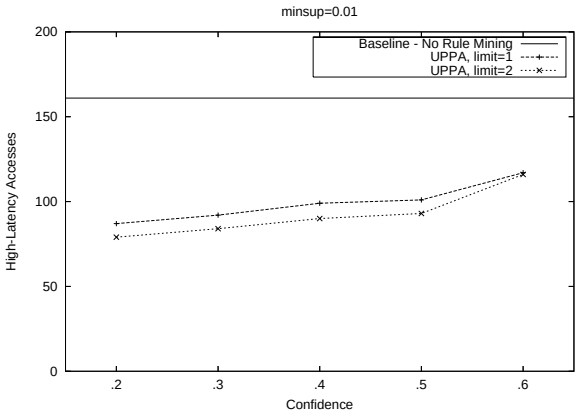

Figure 7: Number of high-latency accesses

Success rate We define success rate to be the percentage of misconfigurations (where a misconfiguration is a prediction that correctly identifies a discrepancy between implemented and intended policy). 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 with three different limits on the number of candidates to query.

Figure 6 shows the success rates obtained with these strategies when *minconf* and *minsup* are set to 0.4 and 0.01. 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 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 Section 3.3 and 3.3.

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 access-control policy, and this intervention is on the critical path to an access being allowed. At best, this is inconvenient both to 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 (e.g., flying), 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 minutes, 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 3) and the ability of the resolution process to succeed in

the cases where exercised policy is correctly predicted (Figure 6).

Figure 7 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, for a minimum confidence value of 0.4, and a limit of two user consultations reduces the number of high-latency accesses by 44%.

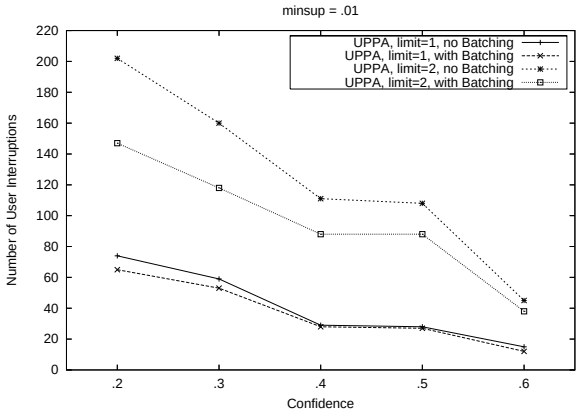

Figure 8: Proactive user interruptions

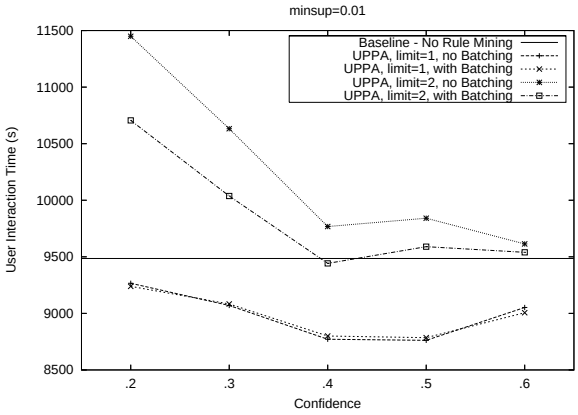

Figure 9: Total user interaction time

User interruptions In some situations, proactively resolving a misconfiguration may involve proactively querying users to determine the intended policy. We consider an interruption to be any query directed to a user as part of the resolution process. This overstates the extent to which a user must to be involved, because some queries directed to a user can be answered automatically by the user's phone, e.g., 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, and so we cannot simulate this distinction.

Often, several misconfigurations are detected at the same time, and their corresponding resolution processes each 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 the initial cost of interrupting the user.

Figure 8 shows the number of user interruptions for our data when consulting either one or two users chosen by the UPPA strategy and varying the minimum confidence values for rules. As expected, consulting two users instead of one increases the number of interruptions (and the success rate, as previously discussed). Batching is more effective when consulting two users and for low minimum confidence values; these settings result in more attempts to consult users, and thus more redundancy among those requests that can be reduced through batching.

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 7).

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 interruptions of similar length that occur off the critical path to an access [5]. In this evaluation we measure only the duration of the interruptions, and so our results are conservative: the actual benefit to users is likely to be significantly greater.

The latencies relevant to our investigation are (1) the latency incurred due to policy reconfiguration by the user whose access triggered the reconfiguration (Bob, as he waits for access to Alice's office); (2) the overhead incurred by the user who is reconfiguring policy (Alice, as she reconfigures her policy to allow Bob access); and (3) the delay in obtaining credentials that describe implemented policy. In our system, the latency incurred by the user who triggered policy reconfiguration ranged between 25 seconds and 18.6 hours, with a median of 98 seconds (the average, heavily influenced by a few outliers, was 53 minutes). The median time required to correct policy from the perspective of the user performing the correction was 23 seconds, only 5.8 seconds of which was spent selecting the appropriate correction (the rest was spent finding their phone, etc.). Finally, the delay of acquiring credentials that describe implemented policy, which each user incurred once for each resource she accessed, was 6.9 seconds on average. These particular latencies are specific to our system, but similar relationships between the latencies (e.g., delay incurred by user waiting for access is much greater than delay incurred by user modifying policy) likely hold in other scenarios as well.

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). 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 are assumed to take 6.9 seconds if the request can be completed without user interaction, and 98 seconds otherwise. Whenever a query requires user interaction, we assume that the user takes 23 seconds to respond to the query. We assume that each additional question posed to the user as part of a batch takes 6 seconds.

We calculate the total time users 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 9. 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 minimum confidence values. Notably, with a minimum confidence value of 0.4 and using batching, UPPA results in a slight overall time savings even if it is allowed to contact two principals.

3.4 Discussion

Each strategy for directing queries in our resolution mechanism is capable of resolving a majority of policy misconfigurations. In particular, the UPPA strategy, when allowed to consult two users, is able to resolve almost 95% 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. When comparing a user consultation limit of two to a limit of one on the basis of total user interaction time, the time associated with the additional user interaction is compensated by the savings resulting from fewer high-latency accesses.

To summarize: Our results show that a significant reduction—44%—in the number of high-latency accesses can be achieved without increasing 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 be aware of who resolved misconfigurations in the past. 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, our system provides integrated support for resolving misconfigurations at the time of access. Many systems do not support this functionality, and as a result, the duration of each high-latency access is likely to be dramatically higher. Our techniques significantly reduce the number of high-latency accesses, so the case for proactively resolving misconfigurations in such a scenario is likely to be even stronger than the one we present here.

4 Related Work

The vast majority of tools for empirical access-control policy analysis have been developed for firewalls (e.g., [4, 17, 12, 21, 2, 22]). 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, i.e., 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 [15], a system also using 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 [10] similarly seeks to detect router misconfigurations as statistical anomalies within a Bayesian framework. As in the aforementioned works in firewall analysis, though, whom to consult about apparent configuration errors and minimizing the costs of doing so were not issues in these works; these issues are central to ours, however. Moreover, there are numerous technical differences between our works and theirs, owing largely to the different

domains in which they are conducted.

Our techniques operate using only data describing 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 [19], SPKI/SDSI [18], RT [16], or PCA [3]. We utilize observed behavior to determine to whom a resolution request should be directed. Some systems, such as PeerAccess [20], 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 develop here. In particular, similar 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., [9, 13, 14, 11, 7, 16]). These mechanisms can be used to drive the correction of access-control misconfigurations once they are detected, though stop short of detecting those misconfigurations and predicting how they might be corrected, as we have studied here. Trust-X [8] proposes a mechanism in which subsets of policy cached from previous access-control decisions can be used to more quickly grant access in the future. This is effective when the implemented policies that govern different access requests share many common components. Our technique is complementary in that we focus on addressing the discrepancy between implemented policy and intended policy.

5 Conclusion

In various application contexts (e.g., health-care systems), the consequences of unnecessary delays for access to information can be severe. In such settings, it essential to eliminate access-control policy misconfigurations in advance of attempted accesses, and even in less critical environments, doing so can greatly improve the usability of the access-control system. In this paper 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 have demonstrated that by doing so, we can greatly reduce the critical-path delays to accesses, while inducing little additional overall work on users of the system. Specifically, we have shown using data from a deployed access-control system that our methods can reduce the number of accesses that would have incurred a costly time-of-access delay by 44%, and can correctly predict 58% of the intended policy. These gains are achieved without increasing 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

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings 20th International Conference on Very Large Data Bases, VLDB*, 1994.

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

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

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

[5] L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *Proceedings of the 3rd Symposium on Usable Privacy and Security*, July 2007.

[6] L. Bauer, S. Garriss, and M. K. Reiter. Efficient proving for practical distributed access-control systems. In *Proceedings of the 12th European Symposium on Research in Computer Security ESORICS*, 2007.

[7] M. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, 2004.

[8] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-X: a Peer to Peer Framework for Trust Establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.

[9] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security & Privacy*, 1996.

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

[11] N. C. Goffee, S. H. Kim, S. Smith, P. Taylor, M. Zhao, and J. Marchesini. Greenpass: Decentralized, PKI-based authorization for wireless LANs. In *Proceedings of the 3rd Annual PKI Research and Development Workshop*, 2004.

[12] S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, June 2000.

[13] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security & Privacy*, 2001.

[14] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. The STRONGMAN architecture. In *Third DARPA Information Survivability Conference and Exposition*, 2003.

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

[16] N. Li and J. C. Mitchell. Rt: A role-based trust-management framework. In *Proceedings of The Third DARPA Information Survivability Conference and Exposition*, 2003.

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

[18] R. L. Rivest and B. Lampson. SDSI—A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession, Apr. 1996.

[19] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2), 1996.

[20] M. Winslett, C. C. Zhang, and P. A. Bonatti. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005.

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

[22] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. FIREMAN: A toolkit for FIREwall modeling and ANalysis. In *IEEE Symposium on Security and Privacy*, pages 199–213. IEEE Computer Society, 2006.