Understanding policy intent and misconfigurations from implementations: consistency and convergence

Prasad Naldurg[1], Ranjita Bhagwan[1], and Tathagata Das[2]

[1] Microsoft Research India, prasadn@microsoft.com, bhagwan@microsoft.com
[2] University of California at Berkeley, tathagata.das1565@gmail.com

Abstract. We study the problem of inferring policy intent to identify misconfigurations in access control implementations. This is in contrast to traditional role-mining techniques, which focus on creating better abstractions for access control management. We show how raw metadata can be summarized effectively, by grouping together users with similar permissions over shared resources. Using these summary statements, we apply statistical techniques to detect outliers, which we classify as security and accessibility misconfigurations. Specifically, we show how our techniques for mining policy intent are robust, and have strong consistency and convergence guarantees.

1 Introduction

Consider the following scenario, where an administrator is tasked with finding out whether the access permissions to shared files and folders in an organization comply with organizational policy. In particular, the administrator is interested in finding out if there are any misconfigurations, say in terms of users who have access to critical files and folders that they are not authorized to examine. It is difficult in general to prove that the access-control configuration instance complies with organizational policy. Most organizations do not have explicit policy manifests that outline all authorizations, and misconfigurations can occur in implementations due to various reasons, including incomplete policy specifications, oversight, partial implementation of policy changes, high rates of churn in organizational roles among employees, and others.

The problem of finding these incorrect permissions is important, and in previous work, we have built a tool Baaz, to find such instances [1]. Its goal is to analyze raw metadata and find misconfigurations automatically. Baaz has proved to be very successful in practice, finding many critical misconfigurations that were important enough to be fixed immediately. In this paper, we focus on the formal guarantees provided by our techniques. At the heart of our tool is a statistical outlier-detection algorithm that can be tuned to relevant datasets to improve precision and recall. We prove some very useful properties about the guarantees this algorithm can achieve in terms of policy consistency. The list of

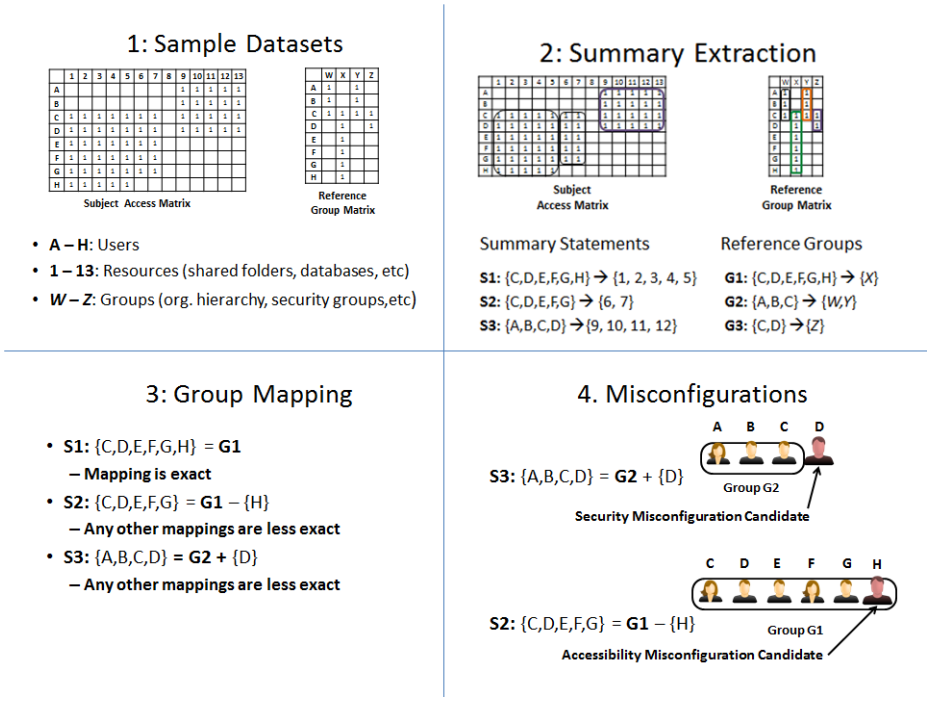

Fig. 1. Overview of the Baaz algorithm, with summary extraction, group mapping and misconfigurations

misconfiguration candidates output by Baaz will converge iteratively, independent of any external policy manifest.

We compare our methodology with role-mining [4, 5, 3, 6, 2], and call out a few crucial differences. Role mining is concerned with finding a natural collection of user-role and role-permission associations, to simplify the management of access control permissions according to organizational roles. As such, the two problems are semantically different. Role-mining techniques allow for noise in user-permission assignments, which is the property we are trying to find. Further, role-mining techniques are fragile and can lead to very different associations if a small number of (or even one) permissions change.

2 Understanding metadata

The first step in the Baaz misconfiguration detection algorithm is summary extraction. We start with a binary subject access matrix $A[u, o]$ (from raw metadata), with entries recording whether user $u \in U$ can access object $o \in O$ (See Figure 1.1). From this, we generate summary statements of the form $\langle \mathcal{U}_i \longrightarrow \mathcal{O}_i \rangle$, where users in set $\mathcal{U}_i$ can access all objects in set $\mathcal{O}_i$ (Figure 1.2). These object-

sets $\mathcal{O}_i$ across the summary statements are non-overlapping (mutually disjoint), and can be generated efficiently using a hash-table in linear time in $|O|$. Role-mining, on the other hand, looks for maximally-connected overlapping subject and object sets, which is NP-Complete.

These summary statements can act as placeholders for policy, providing *compactness* and *coverage*. They are compact in the sense that any pair of statements cannot be combined to create a more general and valid summary. They also cover the set of objects, i.e., any object that is shared by more than one user will necessarily appear in a (and only one) summary statement. These properties are very useful, as they minimize the number of summary statements that need to be examined for anomaly detection in the next step.

3 Detecting Misconfiguration Candidates

We briefly describe our main algorithms for detecting misconfiguration candidates. Refer Baaz [1] for full details.

Group Mapping (GM): The GM algorithm has two inputs, a subject dataset and a reference dataset. In a typical use-case (Fig. 1.2), the subject dataset is the list of user-sets U_i from the summary statements. The reference dataset is a matrix of user-set and group associations, obtained from a different source, such as the organization hierarchy, distribution lists, security groups, email groups, role charts etc. (e.g., Reference groups in Fig. 1.2) The goal of GM is to find the best mapping between the subject user-sets and the reference user-sets (Fig. 1.3). The main idea is that users in the same reference group should have the same access permissions and any inconsistencies detected in the closest subject user-sets are potential misconfigurations (Fig. 1.4).

The GM algorithm is implemented by adapting set-cover, by re-using mapped user-sets. We observe that there is a (deterministic) polynomial reduction from minimum set cover to group mapping. Since set cover is NP-hard, therefore group mapping is also NP-hard. Once we compute the mapping, we use a simple ranking function that prioritizes smaller inconsistencies. For accessibility misconfigurations this is computed as the fraction of users in the reference user-set who do not have access. For security misconfiguration, we call out the fraction of users in a subject user-set that cannot be mapped to well known reference user-sets. This threshold is fixed for any run of the algorithm, but can be tuned to control the quality of the output.

Object Clustering (OC): In OC, we use only the summary statements as input. The idea behind OC is to find pairs of summary statements that have overlapping (or nearly statistically identical) user-sets and object-sets, and call out the inconsistencies as misconfigurations. This threshold is also tunable.

4 Algorithm Properties

Once a misconfiguration candidate is proposed by either algorithm, an administrator may decide that it is valid (needs fixing), or incorrect, which may indicate

that a policy exception was encountered and needs to be archived, or that the reference dataset is stale. A natural question to ask is whether fixing one candidate could lead to other candidates involving the same objects and users, and if fixing that could lead to new candidates, or cause the original misconfiguration to reappear, and so on. We show that if we start with a fixed $A[u, o]$ and a fixed reference dataset, and implement the changes suggested by Group Mapping(GM) and/or Object Clustering(OC), the set of misconfigurations will *converge*.

We model the state of our algorithm as a 4-tuple $\langle \Sigma = \mathcal{S}, \mathcal{R}, \mathcal{M}, \mathcal{E} \rangle$. $\mathcal{S}$ is the set of summary statements obtained from $A[u, o]$. $\mathcal{R}$ is the reference dataset, and is a binary group-membership relation between the users and the set of reference groups $\mathcal{G}$. $\mathcal{M}$ is the set of misconfigurations output by the server algorithm and $\mathcal{E}$ is the set of invalid candidates or exceptions associated with the current state of the algorithm. The initial state is modeled as $\langle \mathcal{S}, \mathcal{R}, \phi, \phi \rangle$. This analysis assumes the reference dataset, as well as the thresholds are fixed and we are allowed to only fix misconfigurations by changing the subject dataset.

Lemma 1. *GM misconfigurations converge: The effect of applying GM to a state is equivalent to applying a monotonically decreasing (or order-preserving) function on the set of misconfiguration candidates.*

Proof. Any accessibility misconfiguration found in GM, will suggest that some users in a mapped reference group G_i that are not in the subject user-set U_i, should have access to the corresponding objects in O_i. These are the users that are missing from U_i but are members of group $\mathcal{G}_i$, suggesting that the implemented permissions have overlooked these group members. If an administrator fixes these permissions in $\mathcal{S}$, by granting these users permissions to O_i, then the set U_i will change in the corresponding summary statement in the next iteration to include these new users. Since G_k was already the best matching in R, no new misconfigurations will be flagged, and an existing summary statement will be updated.

Similarly if a security candidate is detected as some $T_i \subset U_i$ that has access to O_i and should not, fixing it by removing the access will cause it to disappear from any future reports, and the user-sets in the summary statement will have a better match in GM. If a candidate is marked as invalid/or as an exception, they will remain so in the future, and not reported. As the summary statements are over disjoint object sets, the set of GM misconfigurations cannot increase in any step.

Theorem 1. *GM algorithm has a steady state.*

Proof. To prove this statement, we rely on the well-known Knaster-Tarski theorem, which states that if L is a complete lattice, and $f : L \to L$ is an order preserving function (such as a monotonically decreasing function), then at least one fixpoint (where $f(l) = l, l \in L$) is guaranteed to exist. A complete lattice is a partially ordered-set in which all subsets have both a supremum and infimum.

We have already shown that GM is an order preserving function. Now, to use this theorem, we show that the set of misconfigurations $\mathcal{M}$ forms a complete lattice under $\subseteq$. Each misconfiguration is a triple drawn from the cross product of the sets $U \times O \times \{true, false\}$. A particular misconfiguration indicates that user $u \in U$ should or should not (given by the boolean values) have access to object $o \in O$. The set of misconfigurations in each state of the GM algorithm can be drawn from the powerset of this triple. It is well-known that the powerset of any given set, ordered by inclusion $\subseteq$ is a complete lattice, with the union of any two lattice elements being the supremum, and the infimum given by the intersection of any two subsets. Therefore $\mathcal{M}, \subseteq$ is a complete lattice.

Relations $\mathcal{S}$ and $\mathcal{R}$ are input parameters to GM. Relation $\mathcal{R}$ is fixed across all states for a sequence of repeated runs of algorithm. The set $\mathcal{E}$ does not change, as shown earlier. From Lemma 1, GM is an order-preserving function on $\mathcal{M}$, when changes in $\mathcal{S}$ are carried out appropriately. Therefore it must have a fixpoint, i.e., there exists a state σ_f such that $\mathbf{GM}(\sigma_f) = \sigma_f$.

Now, in OC we observe that it is possible for new candidates to arise in the course of fixing a misconfiguration. Consider the case where we remove users from a user-set U_j to fix a security misconfiguration between summary statements i and j. Now user-sets U_i and U_j are identical, and the number of objects in the merged object set may trigger a new security or accessibility issue in the next iteration, as the sizes of user and object sets get updated.

However, observe that fixing a misconfiguration in OC can never increase the number of summary statements. For a security misconfiguration, the misconfigured user will be removed from a user-set, and the remaining users and objects will be merged with an existing summary. Similarly, for an accessibility misconfiguration, the two summary statements will be merged once the misconfiguration is fixed. Therefore the number of summary statements will decrease every time a misconfiguration is fixed, even though the number of misconfigurations may temporarily increase. Eventually, for a fixed set of summary statements and a fixed threshold, the number of misconfigurations will also be fixed.

In practice, we have found that our datasets converge in a few (2 or 3) steps, and the summary statements and exceptions capture organizational policy details faithfully.

5 Conclusions

We show how iterative applications of our misconfiguration detection methods will always converge, leading to a master list of summary statements for any implementation in a finite number of steps. The summary statements thus generated are internally consistent, and they reconcile different aspects of access permission implementations. While we may not be able to guarantee completeness, these inferred policy statements are sound, and are validated by our real-world deployment scenarios, providing a grounding to argue about the quality of the inferred policy very strongly.

References

1. Das, T., Bhagwan, R., Naldurg, P.: Baaz: a system for detecting access control misconfigurations. In: Proceedings of the 19th USENIX conference on Security. USENIX Security'10 (2010)
2. Frank, M., Basin, D., Buchmann, J.M.: A class of probabilistic models for role engineering. In: CCS '08. ACM (2008)
3. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S., Lobo, J.: Mining roles with semantic meanings. In: Proceedings of the 13th ACM symposium on Access control models and technologies (2008)
4. Schlegelmilch, J., Steffens, U.: Role mining with orca. In: Proc. SACMAT '05. pp. 168–176 (2005)
5. Vaidya, J., Atluri, V., Warner, J.: Roleminer: mining roles using subset enumeration. In: CCS '06. pp. 144–153. ACM (2006)
6. Zhang, D., Ramamohanarao, K., Ebringer, T.: Role engineering using graph optimisation. In: SACMAT '07. pp. 139–144. ACM (2007)