Mining Hierarchical Temporal Roles with Multiple Metrics

Scott D. Stoller, Thang Bui

Mining Hierarchical Temporal Roles with Multiple Metrics*

Scott D. Stoller and Thang Bui

Department of Computer Science, Stony Brook University, USA

Abstract. Temporal role-based access control (TRBAC) extends role-based access control to limit the times at which roles are enabled. This paper presents a new algorithm for mining high-quality TRBAC policies from timed ACLs (i.e., ACLs with time limits in the entries) and optionally user attribute information. Such algorithms have potential to significantly reduce the cost of migration from timed ACLs to TRBAC. The algorithm is parameterized by the policy quality metric. We consider multiple quality metrics, including number of roles, weighted structural complexity (a generalization of policy size), and (when user attribute information is available) interpretability, i.e., how well role membership can be characterized in terms of user attributes. Ours is the first TRBAC policy mining algorithm that produces hierarchical policies, and the first that optimizes weighted structural complexity or interpretability. In experiments with datasets based on real-world ACL policies, our algorithm is more effective than previous algorithms at their goal of minimizing the number of roles.

1 Introduction

Role-based access control (RBAC) offers significant advantages over lower-level access control policy representations, such as access control lists (ACLs). RBAC policy mining algorithms have potential to significantly reduce the cost of migration to RBAC, by partially automating the development of an RBAC policy from an access control list (ACL) policy and possibly other information, such as user attributes [4]. The most widely studied versions of the RBAC policy mining problem involve finding a minimum-size RBAC policy consistent with (i.e., equivalent to) given ACLs. When user attribute information is available, it is also important to maximize interpretability (or "meaning") of roles—in other words, to find roles whose membership can be characterized well in terms of user attributes. Interpretability is critical in practice. Researchers at HP Labs report "the biggest barrier we have encountered to getting the results of role mining to be used in practice" is that "customers are unwilling to deploy roles that they

* This material is based on work supported in part by NSF under Grants CNS-1421893, CCF-1248184, and CCF-1414078, ONR under Grant N00014-15-1-2208, and AFOSR under Grant FA9550-14-1-0261. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

can't understand" [2]. Algorithms for mining meaningful roles are described in, e.g., [8,11].

Temporal RBAC (TRBAC) extends RBAC to limit the times at which roles are enabled [1]. TRBAC supports an expressive notation, called *periodic expressions*, for expressing sets of time intervals during which a role is enabled. A role's permissions are available to members only while the role is enabled. This allows tighter enforcement of the principle of least privilege.

This paper presents an algorithm for mining hierarchical TRBAC policies. It is parameterized by a policy quality metric. We consider multiple policy quality metrics: number of roles, *weighted structural complexity* (WSC) [8], a generalization of syntactic policy size, *interpretability* (INT) [8,11], described briefly above, and a compound quality metric, denoted WSC-INT, that combines WSC and INT. Our algorithm is the first TRBAC policy mining algorithm that produces hierarchical policies, and the first that optimizes WSC or interpretability.

Our algorithm is based on Xu and Stoller's elimination algorithm for RBAC mining [11] and some aspects of Mitra *et al.*'s pioneering algorithm for mining flat TRBAC policies (i.e., policies without role hierarchy) with minimal number of roles [6,7], which inspired our work. Our algorithm has four phases: (1) produce a set of candidate roles, (2) merge candidate roles where possible, (3) organize the candidate roles into a role hierarchy, and (4) remove low-quality candidate roles. The generated policy is not guaranteed to have optimal quality. Fundamentally, this is because the problem of finding an optimal policy is NP-complete (this follows from NP-completeness of the untimed version of the problem ([8]).

To evaluate the algorithm, we created datasets based on real-world ACL policies from HP, described in [2] and used in several evaluations of role mining algorithms, e.g., [8,11,7]. We could simply extend the ACLs with temporal information to create a temporal user-permission assignment (TUPA), and then mine a TRBAC policy from the TUPA and attribute data. However, it would be hard to evaluate the algorithm's effectiveness, because there is nothing with which to compare the quality of the mined policies. Therefore, we adopt a similar methodology as Mitra *et al.* [7]. For each ACL policy, we mine an RBAC policy from the ACLs and synthetic attribute data using Xu and Stoller's elimination algorithm [11], pseudorandomly extend the RBAC policy with temporal information numerous times to obtain TRBAC policies, expand the TRBAC policies into equivalent TUPAs, mine a TRBAC policy from each TUPA and fixed attribute data, and compare the average quality of the resulting TRBAC policies with the quality of the original TRBAC policy, with the goal that the former is at least as good as the latter.

We created two datasets, using different temporal information when extending RBAC policies to obtain TRBAC policies. For the first dataset, we use simple periodic expressions, each of which is a range of hours that implicitly repeats every day. For the second dataset, we use more complex periodic expressions based on a hospital staffing schedule.

In experiments using number of roles as the policy quality metric, Mitra *et al.*'s algorithm, designed to minimize number of roles, produces 34% more roles

than our algorithm, on average. In experiments using WSC-INT as the policy quality metric, our algorithm succeeds in finding the implicit structure in the TUPA, producing policies with comparable (for the first dataset) or moderately higher (for the second dataset) WSC and better interpretability, on average, compared with the original TRBAC policy.

We explored the effect of different inheritance types on the quality of the mined policy and found that weakly restricted inheritance leads to policies with significantly better WSC and slightly better interpretability, on average. We experimentally evaluated the benefits of some design decisions and quantified the cost-quality trade-off provided by a parameter to our algorithm that limits the number of candidate roles.

2 Background on TRBAC

An *RBAC policy* is a tuple $\langle User, Perm, Role, UA, PA, RH\rangle$, where *User* is a set of users, *Perm* is a set of permissions, *Role* is a set of roles, $UA \subseteq U \times Role$ is the user-role assignment, $PA \subseteq Role \times Perm$ is the permission-role assignment, and $RH \subseteq Role \times Role$ is the role inheritance relation (also called the role hierarchy). Specifically, $\langle r, r'\rangle \in RH$ means that r is senior to r', hence all permissions of r' are also permissions of r, and all members of r are also members of r'. A role r' is *junior to* role r if rRH^+r', where RH^+ is the transitive closure of RH.

A *periodic expression* (PE) is a symbolic representation for an infinite set of time intervals. The formal definition of periodic expressions in [1,7] is standard and somewhat complicated; instead of repeating it, we give a brief intuitive version. A *calendar* is an infinite set of consecutive time intervals of the same duration; informally, it corresponds to a time unit, e.g., a day or an hour. A sequence of calendars $C_1, \ldots, C_n, C_d$ defines the sequence of time units used in a periodic expression, from larger to smaller. A periodic expression has the form $\sum_{k=1}^{n} O_k \cdot C_k \ \triangleright \ d \cdot C_d$ where $O_1 = all$, O_k is a set of natural numbers or the special value *all* for $2 \leq k \leq n$, and d is a natural number. The first part of a PE (before $\triangleright$) identifies the set of starting points of the intervals represented by the PE. The second part of the PE (after $\triangleright$) specifies the duration of each interval.

For example, consider the sequence of calendars Quadweeks, Weeks, Days, hours, where a Quadweek is four consecutive weeks—similar to a month, but with a uniform duration. The periodic expression [*all* · Quadweeks + {1,3} · Weeks + {1,2,3,4,5} · Days + {10} · Hours $\triangleright$ 8 · Hours] represents the set of time intervals starting at 9am (the time intervals in each calendar are indexed starting with 1, so for Hours, 1 denotes the hour starting at midnight, 2 denotes the hour starting at 1am, etc.) and ending at 5pm (since duration is 8 hours) of every weekday (assuming days of the week are indexed with 1=Monday) during the first and third weeks of every quadweek.

A *bounded periodic expression* (BPE) is a tuple $\langle[begin, end], pe\rangle$, where *begin* and *end* are date-times, and *pe* is a periodic expression. A BPE represents the set of time intervals represented by *pe* except limited to the interval [*begin*, *end*].

A *role enabling base* (REB) is a set of BPEs, representing the union of the sets of time intervals represented by the BPEs.

A *temporal RBAC (TRBAC) policy* is a tuple $\langle User, Perm, Role, UA, PA, RH, IT, REBA\rangle$, where the first six components are the same as for an RBAC policy, IT is the inheritance type (described below), and $REBA$ is the role enabling base assignment (REBA), which is a mapping from roles in $Role$ to REBs [1]. A role r is enabled during the set of time intervals represented by $REBA(r)$.

We consider two types of inheritance [5]. In both cases, a senior role r inherits permissions from each of its junior roles r'. With *weakly restricted inheritance*, denoted by $IT = \text{WR}$, a permission inherited from r' is available to members of r during the time intervals specified by $REBA(r)$. With *strongly restricted inheritance*, denoted by $IT = \text{SR}$, a permission inherited from r' is available to members of r during the time intervals specified by $REBA(r')$.

A *temporal user-permission assignment (TUPA)* is a set of triples of the form $\langle u, p, reb\rangle$, where u is a user, p is a permission, and reb is a REB (even though reb is not associated with a role, we call it a REB, because it has the same type as a REB). We refer to such a triple as an *entitlement triple*. Such a triple means that u has permission p during the set of time intervals represented by reb. A TUPA should contain at most one entitlement triple for each user-permission pair. A TUPA can therefore be regarded as a mapping from user-permission pairs to REBs.

The meaning of a role r in a TRBAC policy π, denoted $[\![r]\!]_\pi$, is a TUPA that expresses the entitlements granted by r, taking inheritance into account. The meaning $[\![\pi]\!]$ of a TRBAC policy π is a TUPA that expresses the entitlements granted by π.

3 The Relaxed TRBAC Policy Mining Problem

A *policy quality metric* is a function from TRBAC policies to a totally-ordered set, such as the natural numbers. The ordering is chosen so that small values indicate high quality.

Number of roles is a simplistic but traditional policy quality metric.

Weighted Structural Complexity (WSC) is a generalization of policy size [8]. We adapt WSC to TRBAC. For a TRBAC policy π of the above form, the WSC of π is defined by $\text{WSC}(\pi) = w_1|Role| + w_2|UA| + w_3|PA| + w_4|RH| + w_5\text{WSC}(REBA)$, where the w_i are user-specified weights, $|s|$ is the size (cardinality) of set s, and $\text{WSC}(REBA)$ is the sum of the sizes of the REBs in $REBA$. The size of an REB is the sum of the sizes of the BPEs in it. The size of a BPE is the size of the PE in it (the beginning and ending date-times are always the same size, so we ignore them). The size of a PE is the sum of the sizes of the sets in it plus 1 for the duration, with the special value *all* counted as a set of size 1.

Interpretability is a policy quality metric that measures how well role membership can be characterized in terms of user attributes. *User-attribute data* is a tuple $\langle A, f\rangle$, where A is a set of attributes, and f is a function such that $f(u, a)$

is the value of attribute a for user u. An *attribute expression* e is a function from the set A of attributes to sets of values. A user u *satisfies* an attribute expression e iff $(\forall a \in A.\ f(u,a) \in e(a))$. For example, if $A = \{dept, level\}$, the function e with $e(dept) = \{\text{CS}\}$ and $e(level) = \{2,3\}$ is an attribute expression, which can be written with syntactic sugar as $dept \in \{\text{CS}\} \wedge level \in \{2,3\}$. We refer to the set $e(a)$ as the conjunct for attribute a. Let $[\![e]\!]$ denote the set of users that satisfy e. For an attribute expression e and a set U of users, the *mismatch* of e and U is defined by $\text{mismatch}(e,U) = |\,[\![e]\!] \ominus U|$, where the symmetric difference of sets s_1 and s_2 is $s_1 \ominus s_2 = (s_1 \setminus s_2) \cup (s_2 \setminus s_1)$. The *attribute mismatch* of a role r, denoted $\text{AM}(r)$, is $\min_{e \in E} \text{mismatch}(e, \text{asgndU}(r))$, where E is the set of all attribute expressions, and $\text{asgndU}(r) = \{u \mid \langle u, r\rangle \in UA\}$. We define policy interpretability INT as the sum over roles of attribute mismatch, i.e., $\text{INT}(\pi) = \sum_{r \in Role} \text{AM}(r)$.

Compound policy quality metrics take multiple aspects of policy quality into account. We combine metrics by Cartesian product, with lexicographic ordering on the tuples. Let $\text{WSC-INT}(\pi) = \langle \text{WSC}(\pi), \text{INT}(\pi)\rangle$.

A TRBAC policy π is *consistent* with a TUPA T if they grant the same permissions to the same users for the same sets of time intervals. When the given TUPA contains noise, it is desirable to weaken this requirement. A TRBAC policy π is ϵ-*consistent* with a TUPA T, where ϵ is a natural number, if they grant the same permissions to the same users for the same sets of time intervals, except that, for at most ϵ entitlement triples $\langle u, p, reb\rangle$ in T, the policy π either does not grant p to u or grants p to u at fewer times than reb [7]. Note that consistency is a special case of ϵ-consistency, corresponding to $\epsilon = 0$.

The *relaxed TRBAC policy mining problem* is: given a TUPA T and a policy quality metric Q_{pol}, find a TRBAC policy π that is ϵ-consistent with T and has the best quality, according to Q_{pol}, among policies consistent with T. Note that auxiliary information used by the policy quality metric, e.g., user-attribute data, is implicitly considered to be part of Q_{pol} in this definition. Note that the temporal part of T strongly influences π, even using WSC with $w_5 = 0$, because it determines how entitlements can be grouped in roles.

Suggested role assignments for new users. The system can compute and store a best-fit attribute expression e_r for each role r, i.e., an attribute expression that minimizes the attribute mismatch for r. When a new user u is added, the system can suggest that u be made a member of the roles for which u satisfies the best-fit attribute expression, and it presents these suggested roles in descending order of the attribute mismatch.

4 TRBAC Policy Mining Algorithm

Inputs to the algorithm are the TUPA T, the type of inheritance IT to use in the generated policy, the consistency threshold ϵ, and the policy quality metric Q_{pol}. User attribute data, if available, is used only indirectly, *via* the policy quality metric, if it considers interpretability.

```
R_init = new Set()
for u in U
  for ⟨P, reb⟩ in permREB(u, T)
                  ∪ permREB⁺(u, T)
    addRole(R_init, {u}, P, reb)
    for bpe in reb
      addRole(R_init, {u}, P, {bpe})

permREB(u, T) =
{⟨P, reb⟩ | (∃p.⟨u, p, reb⟩ ∈ T)
            ∧ P = {p | ⟨u, p, reb⟩ ∈ T}}

permREB⁺(u, T) =
{⟨P, reb⟩ | (∃p.⟨u, p, reb⟩ ∈ T)
            ∧ P = {p | ⟨u, p, reb′⟩ ∈ T
                       ∧reb ⊑ reb′}}
```

```
function addRole(R, U, P, reb)
// if there is an existing role with
// permissions P and REB reb,
// add users in U to it, otherwise
// create a new role with users U,
// permissions P, and REB reb.
if U, P, or reb is empty
  return
if ∃ r in R s.t. asgndP_0(r) = P
                 ∧ REBA(r) = reb
  asgndU_0(r).addAll(U)
else
  r = new Role()
  asgndP_0(r) = P
  asgndU_0(r) = U
  REBA(r) = reb
  R.add(r)
```

Fig. 1. Phase 1.1: Generate initial roles. "s.t." abbreviates "such that".

Phase 1: Generate roles. Phase 1 generates initial roles and then creates additional candidate roles by intersecting sets of initial roles.

Phase 1.1: Generate initial roles. Pseudocode for generating initial roles appears in Figure 1. The set of permissions P that each user u has for exactly the same REB *reb* are grouped to form the permissions of an initial role; this is the effect of using permREB in Figure 1. If there are any permissions that u has for a REB that semantically contains *reb*, then we also create another role that has those permissions in addition to permissions in P; this is the effect of using permREB$^+$. In addition, for each BPE *bpe* in *reb*, we create an initial role with permissions P and with REB $\{bpe\}$. The algorithm uses a semantic containment relation $\sqsubseteq$ on PEs, BPEs, and REBs: $x_1 \sqsubseteq x_2$ iff the set of time instants represented by x_1 is a subset of the set of time instants represented by x_2.

Phase 1.2: Intersect roles. Phase 1.2 starts to construct a set R_{cand} of candidate roles, by adding to R_{cand} all of the initial roles in R_{init} and all non-empty intersections of all subsets of the initial roles. In other words, for each subset of initial roles, if the intersection of their permission sets is a non-empty set P, and the intersection of their REBs is a non-empty REB *reb*, then create a candidate role with permissions P, REB *reb*, and the union of their user sets. REBs are intersected semantically, not syntactically; for example, if reb_1 represents 9am-5pm on Mondays and Wednesdays, and reb_2 represents 1pm-2pm on Mondays and Fridays, then their intersection is a REB that represents 1pm-2pm on Mondays. This phase is similar to role intersection in CompleteMiner [10] and the elimination algorithm [11].

This phase is expensive for large datasets. We use two techniques to reduce the cost when necessary; they provide a trade-off between cost and policy quality.

(1) Compute intersections for all pairs (instead of all subsets) of initial roles, as in FastMiner [10]. This reduces the worst-case complexity of this step and the overall algorithm from exponential to quadratic. (2) Compute intersections involving only the largest roles, specifically, roles whose relative size is in the top RIC (mnemonic for "role intersection cutoff"), where $0 \leq \text{RIC} \leq 1$. For example, RIC = 0.3 means that intersections are computed among roles whose size is in the top 30%. Role size is quantified as covEntit(r), defined below.

Phase 2: Merge roles. Phase 2 merges candidate roles to produce a revised set of candidate roles. We use three types of merges. (1) If candidate roles r and r' have the same same set of users U and the same REB *reb*, then they are replaced with a new role with users U, permissions $\text{asgndP}_0(r) \cup \text{asgndP}_0(r')$, and REB *reb*. (2) If candidate roles r and r' have the same users U and same permissions P, then they are replaced with a new role with users U, permissions P, and REB $reb(r) \sqcup reb(r')$. The function $\sqcup$ denotes semantic union of REBs; in other words, $reb_1 \sqcup reb_2$ is a REB that represents the set of time instants represented by reb_1 or reb_2. We distinguish two sub-cases. (2a) If reb_1 and reb_2 represent disjoint sets of time intervals, then $reb_1 \sqcup reb_2$ is simply $reb_1 \cup reb_2$. (2b) If reb_1 and reb_2 represent sets of overlapping or consecutive time intervals, then BPEs in them are merged, if possible, to simplify the result. For example, if reb_1 represents 9am-noon on weekdays, and reb_2 denotes noon-5pm on weekdays, then $reb_1 \sqcup reb_2$ contains a single BPE denoting 9am-5pm on weekdays.

Phase 3: Construct role hierarchy. Phase 3 organizes the candidate roles into a role hierarchy with full inheritance. A TRBAC policy has *full inheritance* if every two roles that can be related by the inheritance relation are related by it, i.e., $\forall r, r' \in R.\ [\![r]\!]_\pi \supseteq [\![r']\!]_\pi \Rightarrow \langle r, r' \rangle \in RH^*$. Guo *et al.* call this property *completeness* in the context of RBAC [3].

Phase 3.1: Compute inheritance. Phase 3.1 determines inheritance relationships between candidate roles, based on the requirement of full inheritance. Function isAncestorFullInher(r', r) tests whether r' is an ancestor of r with full inheritance; if IT = WR, the function avoids inheritance relationships that would lead to cycles in the role hierarchy.

$$
\begin{aligned}
&\text{isAncestorFullInher}(r', r) = \\
&\quad \text{asgndP}_0(r') \subseteq \text{asgndP}_0(r) \wedge \text{asgndU}_0(r) \subseteq \text{asgndU}_0(r') \\
&\quad \wedge (IT = \text{SR} \Rightarrow REBA(r') \sqsubseteq REBA(r)) \\
&\quad \wedge (IT = \text{WR} \Rightarrow \neg(\text{asgndP}_0(r) \subset \text{asgndP}_0(r') \wedge \text{asgndU}_0(r') \subset \text{asgndU}_0(r)))
\end{aligned}
$$

This function is called for every pair of candidate roles. If isAncestorFullInher(r', r) is true, and there is no intervening role $\bar{r}$ such that isAncestorFullInher($r', \bar{r}$) isAncestorFullInher($\bar{r}, r$), then r' is a parent of r. This phase produces maps *parents* and *children*, such that *parents*(r) and *children*(r) are the sets of parents and children of r, respectively.

Phase 3.2: Compute assigned users and permissions. Phase 3.2 computes the directly assigned users $\text{asgndU}(r)$ and directly assigned permissions $\text{asgndP}(r)$ of each role r, by removing inherited users and permissions from the role's originally assigned users $\text{asgndU}_0(r)$ and originally assigned permissions $\text{asgndP}_0(r)$.

Phase 4: Remove roles. Phase 4 removes roles from the candidate role hierarchy if the removal preserves consistency with the given ACL policy and improves policy quality. When a role r is removed, the role hierarchy is adjusted to preserve inheritance relations between parents and children of r, and the sets of directly assigned users and permissions of other roles are expanded to contain users and permissions that they previously inherited from r.

The order in which roles are considered for removal affects the final result. We control this ordering with a *role quality metric* Q_{role}, which maps roles to an ordered set, with the interpretation that large values denote high quality (note: this is opposite to the interpretation of the ordering for policy quality metrics). Low-quality roles are considered for removal first. We use a role quality metric that is a temporal variant of the role quality metric in [11] that gave the best results in their experiments. Specifically, $Q_{role}(r) = \langle \text{redun}(r), \text{clsSz}(r) \rangle$, where $\text{redun}(r)$ and $\text{clsSz}(r)$ are defined next, and the ordering on these tuples is lexicographic order.

The *redundancy* of a role r measures how many other roles also cover the entitlement triples covered by r. We say that a role r *covers* an entitlement triple t if $t \in [\![r]\!]_\pi$. Removing a role with higher redundancy is less likely to prevent subsequent removal of other roles, so we eliminate roles with higher redundancy first. The redundancy of role r, denoted $\text{redun}(r)$, is the negative of the minimum, over entitlement triples $\langle u, p, reb \rangle$ covered by r, of the number of removable roles that cover $\langle u, p, reb \rangle$ (we take the negative so that roles with more redundancy have lower quality). A role is *removable* in policy π, denoted $\text{removable}(r)$ (the policy is an implicit argument), if the policy obtained by removing r is ϵ-consistent with T.

The *clustered size* of a role r measures how many entitlements are covered by r and how well they are clustered. A first attempt at formulating this metric (ignoring clustering) might be as the fraction of entitlement triples in T that are covered by r. As discussed in [11], it is better for the covered entitlement triples to be "clustered" on (i.e., associated with) fewer users rather than being spread across many users. The clustered size of r is defined to equal the fraction of the entitlements of r's members that are covered by r. In the temporal case, each entitlement triple $\langle u, p, reb \rangle$ is weighted by the fraction of the time represented reb that is covered by $REBA(r)$.

$$\text{covEntit}(r) = \sum_{\substack{u \in \text{asgndU}(r) \\ p \in \text{asgndP}(r)}} \frac{\text{dur}(REBA(r))}{\text{dur}(T(u,p))} \quad \text{clsSz}(r) = \frac{\text{covEntit}(r)}{|\text{entitlements}(\text{asgndU}(r), T)|}$$

where $T(u,p)$ is the REB reb such that $\langle u, p, reb \rangle \in T$, $\text{dur}(reb)$ is the fraction of one time unit in calendar C_1 that is covered by reb, and $\text{entitlements}(U, T)$ is

the set of entitlement triples in T for a user in U. For example, if the sequence of calendars is $C_1 = \text{Year}, \ldots, C_n = \text{Hour}, C_d = \text{Hour}$, and *reb* is 9am-5pm every day, then $\text{dur}(reb) = 1/3$, since *reb* covers 1/3 of the time in a year.

Our algorithm may remove a role even if the removal worsens policy quality slightly. Specifically, we introduce a *quality change tolerance* δ, with $\delta \geq 1$, and we remove a role if the quality Q' of the RBAC policy resulting from the removal is related to the quality Q of the current RBAC policy by $Q' < \delta Q$ (recall that, for policy quality metrics, smaller values are better). Choosing $\delta > 1$ partially compensates for the fact that a purely greedy approach to policy quality improvement is not an optimal strategy.

Pseudocode for removing roles appears in Figure 2. It repeatedly tries to remove all removable roles, until none of the attempted removals succeeds in improving the policy quality. The policy π is an implicit argument to auxiliary functions such as removeRole and addRole. Function addRole(r) adds role r to the candidate role hierarchy: inheritance relations involving r are added, and the assigned users and assigned permissions of r's newly acquired ancestors and descendants are adjusted by removing inherited users and permissions. Removing a role r and then restoring r using addRole leaves the policy unchanged.

The following auxiliary functions are used in removeRole. isDescendant(r,r') holds if r is a descendant of r', as determined by following the parent-child relations in the *children* map. The set of authorized users of r, denoted authU(r), is the set of users in asgndU(r) or asgndU(r') for some r' senior to r; this is the same as in RBAC. The notion of authorized permissions must be defined differently in TRBAC than RBAC, because, with strongly-restricted inheritance, the inherited permissions of a role r may be associated with REBs different than $REBA(r)$. With strongly-restricted inheritance, the set of authorized permissions of r, denoted authP(r), is the set of permission-REB pairs $\langle p, reb \rangle$ such that (1) each directly assigned permission of r is paired with $REBA(r)$ and (2) each permission p inherited by r is paired with the semantic union of the REBs of the junior roles from which it is inherited. With weakly-restricted inheritance, authP(r) is the set of permission-REB pairs $\langle p, REBA(r) \rangle$ such that p is in asgndP(r) or asgndP(r') for some r' junior to r; we use a set of pairs for uniformity with the case of strongly-restricted inheritance.

5 Datasets

Our datasets are based on real-world ACL policies from HP, described in [2], and the high-fit synthetic attribute data for these ACL policies described in [11]; see those references for general information about the policies. As outlined in Section 1, for each ACL policy, we mine an RBAC policy from the ACLs and the attribute data using Xu and Stoller's elimination algorithm [11], and pseudorandomly extend the RBAC policy with temporal information several times to obtain TRBAC policies. For each ACL policy except americas_small, we create 30 TRBAC policies. For americas_small, which is larger, we create only 10 TRBAC policies, to reduce the running time of the experiments. We extend the RBAC policies in two ways, using different temporal information.

```
π = policy from Phase 3
q = Q_pol(π)
workL = list of removable roles in π
changed = true
while ¬empty(workL) ∧ changed
  sort workL in ascending order by Q_role
  changed = false
  for r in workL
    removeRole(r)
    // if ϵ-consistency is violated,
    // restore r.
    if |T \ [[π]]| > ϵ
      addRole(r)
      workL.remove(r)
    else
      // if policy quality improved,
      // keep the change.
      if Q_pol(π) < δq
        changed = true
        q = Q_pol(π)
        workL.remove(r)
      else
        // undo the change, i.e., restore r
        addRole(r)
```

```
function removeRole(r)
for parent in parents(r)
  // remove r from its parents
  children(parent).remove(r)
  for child in children(r)
    // if child is not a descendant of parent
    // after removing r, add an inheritance
    // edge between child and parent.
    if ¬ isDescendant(child,parent)
      children(parent).add(child)
      parents(child).add(parent)
  for u in asgndU(r)
    // if u is not authorized to parent after
    // removing r, add u to assigned users
    // of parent.
if u ∉ authU(parent)
    asgndU(parent).add(u)
for child in children(r)
  parents(child).remove(r)
  for p in asgndP(r)
    // if p is not fully authorized to child
    // after removing r, add p to assigned
    // permissions of child.
    if ⟨r, REBA(child)⟩¬ ∈ authP(child)
      asgndP(child).add(p)
R_cand.remove(r)
```

Fig. 2. Phase 4: Remove roles.

Dataset with simple PEs. A *simple PE* is a range of hours (e.g., 9am-5pm) that implicitly repeats every day. This dataset uses the same simple PEs as in [7], namely, $[6, 11]$, $[7, 10]$, $[8, 9]$, $[8, 11]$, $[9, 11]$, $[10, 11]$, $[10, 12]$, $[11, 13]$, $[14, 15]$, $[16, 17]$. These PEs are designed to cover various relationships between intervals, such as overlapping, consecutive, disjoint, and nested. We choose the number of PEs in each REB pseudorandomly using a similar probability distribution as in [7], namely, $pr(1) = 0.78, pr(2) = 0.2, pr(3) = 0.02$. We choose the specific PEs in each REB pseudorandomly using a uniform distribution.

Dataset with complex PEs. For this dataset, we use periodic expressions based on a hospital staffing schedule, based on discussions with the Director of Timekeeping at Stony Brook University Hospital. The periodic expressions are not taken directly from the hospital's staffing schedule, but they reflect its general nature. The schedule does not repeat every week, but rather every few weeks, because weekend duty rotates. Clinicians may work 3 days/week for 12 hours/day starting at 7am or 7pm, or 5 days/week for 8.5 hours/day starting at 7am, 3pm, or 11pm. We create two instances of each of these five types of work schedules, by pseudorandomly choosing the appropriate number of days of the week in each

of the four weeks of a Quadweek. Each REB is based on exactly one of the resulting 10 work schedules. Multiple PEs are needed to represent work schedules that wrap around calendar units; for example, a 7pm-7am shift is represented using two PEs, with time intervals 7pm-midnight and midnight-7am. The PEs are based on the following sequence of calendars: C_1=Quadweeks, C_2=Days, C_3=Hours, C_d=Hours. The days in a Quadweek are numbered 1..28. Including Week in the sequence of calendars is not helpful, because most workers' schedules do not repeat on a weekly basis.

6 Evaluation

The experimental methodology is outlined in Section 1. All experiments use quality change tolerance $\delta = 1.001$ (this value gave the best results for the experiments in [11]), $\epsilon = 0$, All and $w_i = 1$ for all weights in WSC. The policy quality metric is WSC-INT, and the inheritance type is weakly restricted, except where specified otherwise.

Our Java code, datasets, and an extended version of the paper are available at www.cs.stonybrook.edu/ stoller/policy-mining/. Periodic expressions are an abstract data type with two implementations: (1) simple PEs, as defined in Section 5, and implemented as pairs of integers, and (2) (general) PEs, as defined in Section 2, and implemented as arrays of arrays of integers. These implementations are used in the experiments in Sections 6.1 and 6.2, respectively. Running times include the cost of an end-to-end correctness check that checks equivalence of the input TUPA and the meaning of the mined TRBAC policy; the average cost is about 7% of the running time. The experiments were run on a Lenovo IdeaCentre K430 with a 3.4 GHz Intel Core i7-3770 CPU.

6.1 Experiments using dataset with simple PEs

In experiments on this dataset, role intersection is configured to use FastMiner for emea and americas_small, CompleteMiner for the other policies, and RIC = 1 for all policies.

Comparison of original and mined policies. Figure 3 shows detailed results from experiments on this dataset. In the column headings, μ is mean, σ is standard deviation, CI is half-width of 95% confidence interval using Student's t-distribution, and time is the average running time in minutes:seconds. There is no standard deviation column for INT, because interpretability is unaffected by the REBA and is the same for all TRBAC policies generated by extending the same RBAC policy. Ignore the last 2 columns for now. The averages and standard deviations are computed over the TRBAC policies created by extending each RBAC policy. The WSC of the mined TRBAC policy ranges from about 1% lower (for apj) to about 11% higher (for domino) than the WSC of the original TRBAC policy. The interpretability of the mined policy ranges from about 35% lower (for firewall-2) to about 1% higher (for apj) than the interpretability of the original

Dataset	Original Policy			Mined Policy						Time	Avg $\|R\|$	
	WSC		INT	WSC			INT				OurAlg	Mitra+
	μ	σ		μ	σ	CI	μ	σ	CI	μ		
americas_small	6975	7.5	189	7100	78	29	140	7	2.5	58:56	297	
apj	4879	10.0	385	4826	22	8.1	388	3.5	1.2	1:04	468	527
domino	449	2.5	23	499	70	26	20	1.5	0.57	0:02	30	40
emea	3929	4.4	32	4038	68	25	32	0.2	0.07	0:49	100	115
firewall1	1533	4.1	48	1653	58	22	44	3.7	1.4	1:45	97	130
firewall2	960	1.4	7	966	9.2	3.4	5	1.0	0.38	0:02	12	17
healthcare	168	1.4	14	168	3.9	1.5	14	0.42	0.16	0:01	15	25

Fig. 3. Results of experiments with simple PEs.

TRBAC policy. On average over the seven policies, the WSC is 3% higher, and the interpretability is 12% lower. Thus, our algorithm succeeds in finding the implicit structure in the TUPA and producing a policy with comparable WSC and better interpretability, on average, than the original TRBAC policy.

Comparison of inheritance types. We ran our algorithm again on the same dataset, specifying strongly restricted inheritance for the mined policies. This caused a significant increase in the WSC of the mined policies. The percentage increase averages 67% and ranges from 15% for apj to 140% for firewall-1. Intuitively, the reason for the increase is that, with strongly restricted inheritance, the temporal information associated with directly assigned and inherited permissions may be different, and this may prevent removing inherited permissions from a role's directly assigned permissions. Inheritance type has less effect on the average INT, increasing (worsening) it by about 9% on average, excluding the outlier firewall-2, for which the average INT decreases from 4 to 1.

Evaluation of choice of initial roles. We evaluated two ways of reducing the cost of the algorithm by creating fewer initial roles. (1) We modified Phase 1.1 to create fewer initial roles by removing the use of permREB$^+$ in Figure 1. Note that Mitra *et al.*'s algorithm does not use an analogue of permREB$^+$. This change increased the average WSC by 36% on average over the policies used in this experiment (all except emea and americas_small, which were omitted because of their longer running time), ranging from 13% for apj to 69% for healthcare. It increased (worsened) the average INT by 37% on average over those policies, ranging from 9% for apj to 67% for domino. (2) We modified Phase 1.1 to create fewer initial roles by removing the first call to addRole. Note that Mitra *et al.*'s algorithm does not include an analogue of this call. This change increased the average WSC by 8% on average over the policies used in this experiment (all except emea and americas_small), ranging from 7% for domino and firewall-2 to 9% for apj. It increased (worsened) the average INT by 7% on average over those policies, ranging from 0% for firewall-2 to 11% for domino.

Comparison with Mitra et al.'s algorithm. We ran Mitra *et al.*'s algorithm [7], and our algorithm with number of roles as policy quality metric (because Mitra

et al. use this metric), on our dataset with simple PEs. Their code supports only simple PEs, so we used only the simple PE dataset in the comparison. Their code, implemented in C, gave an error ("malloc: ...: pointer being freed was not allocated") on some TRBAC policies generated for emea and firewall-1; we ignored those results. Their code did not run correctly on americas_small, so we omitted it from this comparison.

The last two columns of Figure 3 show the numbers of roles generated by the two algorithms. Standard deviations are omitted to save space but are small: on average, 3% of the mean, for both algorithms. Mitra *et al.*'s algorithm produces 34% more roles than ours, on average. Our algorithm produces hierarchical policies, and their algorithm produces flat policies, but this does not affect the number of roles. There are many other differences between the algorithms, discussed in Section 7, which contribute to the difference in results. The above paragraph on evaluation of choice of initial roles describes two experiments that explore differences between our algorithm and Mitra *et al.*'s and quantify the benefit of those differences. The effects of some other differences between our algorithms, such as the use of elimination *vs.* selection in Phase 4, were investigated in the untimed case in [11] and likely have a similar impact here.

6.2 Experiments using dataset with complex PEs

In experiments on this dataset, role intersection is configured to use CompleteMiner for firewall2 and FastMiner for the other policies.

Comparison of original and mined policies. Figure 4 shows detailed results from experiments on this dataset. The original TRBAC policies here have higher WSC than the ones in Section 6.1, because complex PEs have higher WSC than simple PEs. For apj, emea, and firewall1, we created 5 TRBAC policies (instead of 30) from each, to reduce the running time of the experiments. The WSC of the mined TRBAC policy ranges from about 2% higher (for firewall2) to about 57% higher (for firewall1) than the WSC of the original TRBAC policy. The interpretability of the mined TRBAC policy ranges from about 34% lower (for healthcare) to about 2% higher (for apj) than the interpretability of the original TRBAC policy. On average over the six policies, the WSC is 20% higher, and the interpretability is 16% lower. Thus, our algorithm finds most of the implicit structure in the TUPA and produces a policy with moderately higher WSC and better interpretability, on average, than the original TRBAC policy. The results can be improved by using larger RIC, at the expense of higher running time.

The higher running times, compared to the dataset with simple PEs, are due primarily to the larger number of candidate roles created by role intersection (there are more overlaps between REBs in this dataset), and secondarily to the larger overhead of manipulating more complex PEs.

Benefit of general PEs. PEs can be translated into sets of simple PEs. For example, the REB $\{[all \cdot \text{Weeks} + \{1,2,7\} \cdot \text{Days} + \{1\} \cdot \text{Hours} \rhd 8 \cdot \text{Hours}]\}$ can be translated to the REB {[1,9], [25,33], [145,153]}. However, PEs are generally

Dataset	Original Policy			Mined Policy						RIC	Time
	WSC		INT	WSC			INT				
	μ	σ		μ	σ	CI	μ	σ	CI		μ
apj	16836	159	385	17434	337	419	391	1.9	2.3	0.7	50:43
domino	1156	49	23	1278	80	30	16	1.8	0.7	1	0:35
emea	5975	99	32	8683	284	353	32	0	0	0.7	201:10
firewall1	3712	97	48	5832	199	247	46	2.2	2.8	0.7	165:30
firewall2	1269	37	7	1291	52	19	5.3	0.68	0.3	1	1:00
healthcare	560	35	14	582	40	15	9.3	1.7	0.6	1	8:57

Fig. 4. Results of experiments with complex PEs.

more compact and efficient. In experiments with the healthcare policy, using this translation and simple PEs was about 19x slower than using general PEs.

Effect of role-intersection cutoff. We investigated the cost-benefit trade-off when varying the role-intersection cutoff RIC. Figure 5 shows running time and WSC as functions of RIC, averaged over three of the smaller policies (domino, firewall2, healthcare). The trade-off is favorable: as RIC decreases, running time decreases more rapidly than WSC increases. For example, at RIC = 0.8, running time is 40% lower than with RIC = 1, and WSC is only 13% higher.

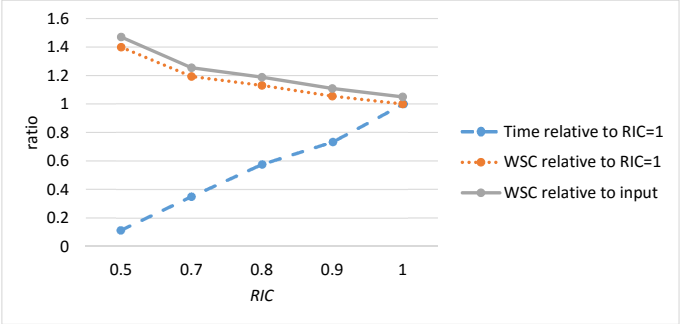

Fig. 5. Relative running time and relative WSC as functions of RIC.

7 Related Work

We discuss related work on TRBAC policy mining and then related work on RBAC mining. Role mining (for RBAC or TRBAC) is also reminiscent of some other data mining problems, but algorithms for those other problems are not well suited to role mining. For example, association rule mining algorithms are designed to find rules that are probabilistic in nature. They are not designed to produce a set of rules strictly consistent with the input that completely covers the input and is minimum-sized among such sets of rules.

7.1 Related Work on TRBAC Policy Mining

Mitra *et al.* define a version of the TRBAC policy mining problem and present an algorithm for mining a TRBAC policy from a TUPA [7]. It is an improved version of their earlier work [6].

Our algorithm is more flexible, because it can optimize a variety of metrics, including WSC and interpretability. Their algorithm is designed to optimize only the number of roles. The importance of interpretability is discussed in Section 1. WSC is a more general measure of policy size than number of roles and can more accurately reflect expected administrative cost. For example, the average number of role assignments per user is a measure of expected administrative effort for adding a new user [9], and this can be reflected in WSC by giving appropriate weight to the size of the user-role assignment.

Our algorithm produces hierarchical TRBAC policies. Their algorithm produces flat TRBAC policies. Role hierarchy is a well-known feature of RBAC that can significantly reduce policy size and administrative effort by avoiding redundancy in the policy.

Some other differences are: (1) Our algorithm determines which candidate roles to include in the final policy by elimination of low-quality roles, instead of selection of high-quality roles. We showed that elimination gives better results in the untimed case [11]. (2) Our algorithm creates more initial roles than theirs. The benefit of creating these additional initial roles is shown in Section 6.1 in the paragraph on evaluation of choice of initial roles. Their algorithm creates unit roles, which are similar to initial roles but have only one permission; our algorithm does not create unit roles. (3) Our algorithm performs fewer types of role intersections than theirs. Specifically, it omits types of role intersections that create PEs with time intervals that do not appear in the input, since these PEs are probably not natural (intuitive) ones in the application domain.

Our implementation supports periodic expressions for specifying temporal information, while theirs supports only ranges of hours that implicitly repeat every day. Design and implementation of operations on sets of PEs is non-trivial.

7.2 Related Work on RBAC Mining

A survey of work on RBAC mining appears in [4]. The most closely related work is Xu and Stoller's elimination algorithm [11]. We chose it as the starting point for design of our algorithm, because in the experiments in [11], it optimizes WSC more effectively than Hierarchical Miner [8], while simultaneously achieving good interpretability, and it optimizes WSCA, an interpretability metric defined in [8], more effectively than Attribute Miner [8].

Our algorithm retains the overall structure of the elimination algorithm but differs in several ways, due to the complexities created by considering time. Our algorithm introduces more kinds of candidate roles than the elimination algorithm, because it needs to consider grouping permissions that are enabled for the same time or a subset of the time of other permissions. Our algorithm attempts to merge candidate roles; the elimination algorithm does not. Construction of

the role hierarchy is significantly more complicated than in the elimination algorithm; for example, with strongly restricted inheritance, a permission p can be inherited by a role r from multiple junior roles with different REBs, which may together cover all or only part of the time that p is available in r. This also complicates adjustment of the role hierarchy when removing candidate roles. The role quality metric used to select roles for removal is more complicated, to give preference to roles that cover permissions for more times.

Acknowledgements. We thank the authors of [7]—Barsha Mitra, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya—for sharing their code and datasets with us and helping us understand their work.

References

1. E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.
2. A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proc. 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1–10. ACM, 2008.
3. Q. Guo, J. Vaidya, and V. Atluri. The role hierarchy mining problem: Discovery of optimal role hierarchies. In *Proc. 2008 Annual Computer Security Applications Conference (ACSAC)*, pages 237–246. IEEE Computer Society, 2008.
4. S. Hachana, N. Cuppens-Boulahia, and F. Cuppens. Role mining to assist authorization governance: How far have we gone? *International Journal of Secure Software Engineering*, 3(4):45–64, October-December 2012.
5. J. B. D. Joshi, E. Bertino, and A. Ghafoor. Temporal hierarchies and inheritance semantics for GTRBAC. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, pages 74–83. ACM, 2002.
6. B. Mitra, S. Sural, V. Atluri, and J. Vaidya. Toward mining of temporal roles. In *Proc. 27th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, volume 7964 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2013.
7. B. Mitra, S. Sural, V. Atluri, and J. Vaidya. The generalized temporal role mining problem. *Journal of Computer Security*, 23(1):31–58, 2015.
8. I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. B. Calo, and J. Lobo. Mining roles with multiple objectives. *ACM Trans. Inf. Syst. Secur.*, 13(4):36:1–36:35, 2010.
9. E. Uzun, D. Lorenzi, V. Atluri, J. Vaidya, and S. Sural. Migrating from DAC to RBAC. In *Proc. 29th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, volume 9149 of *Lecture Notes in Computer Science*. Springer, 2015.
10. J. Vaidya, V. Atluri, and J. Warner. RoleMiner: Mining roles using subset enumeration. In *Proc. 13th ACM Conference on Computer and Communications Security (CCS)*, pages 144–153. ACM, 2006.
11. Z. Xu and S. D. Stoller. Algorithms for mining meaningful roles. In *Proc. 17th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 57–66. ACM, 2012.