A Framework Integrating Attribute-based Policies into Role-Based Access Control [*]

Jingwei Huang, David M. Nicol, Rakesh Bobba and Jun Ho Huh
Information Trust Institute, University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL 61801, USA
{jingwei,dmnicol,rbobba,jhhuh}@illinois.edu

ABSTRACT

Integrated role-based access control (RBAC) and attribute-based access control (ABAC) is emerging as a promising paradigm. This paper proposes a framework that uses attribute-based policies to create a more traditional RBAC model. RBAC has been widely used, but has weaknesses: it is labor-intensive and time-consuming to build a model instance, and a pure RBAC system lacks flexibility to efficiently adapt to changing users, objects, and security policies. Particularly, it is impractical to manually make (and maintain) user to role assignments and role to permission assignments in industrial context characterized by a large number of users and/or security objects. ABAC has features complimentary to RBAC, and merging RBAC and ABAC has become an important research topic. This paper proposes a new approach to integrating ABAC with RBAC, by modeling RBAC in two levels. The aboveground level is a standard RBAC model extended with "environment". This level retains the simplicity of RBAC, supporting RBAC model verification/review. The "underground" level is used to represent security knowledge in terms of attribute-based policies, which automatically create the simple RBAC model in the aboveground level. These attribute-based policies bring to RBAC the advantages of ABAC: they are easy to build and easy to adapt to changes. Using this framework, we tackle the problem of permission assignment for large scale applications. This model is motivated by the characteristics and requirements of industrial control systems, and reflects in part certain approaches and practices common in the industry.

[*]This material is based upon research sponsored by the U.S. Department of Energy under Award Number DE-OE0000544, and by the U.S. Air Force Research Laboratory and the U.S. Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: [Access controls]

Keywords

RBAC, Attribute-Base Access Control, Role Engineering, Industrial Control Systems

1. INTRODUCTION

A recent study [18] shows that the adoption of role-based access control (RBAC) [7, 19] is quickly growing, and that RBAC has become the most popular access control model. RBAC is simple, reflects organizational structure, and is easy to administer and review. However, it also has weaknesses: it is difficult and costly to build a good RBAC instance, and a pure RBAC system lacks flexibility to efficiently adapt to changing users, objects, and security policies. Particularly, it is impractical to manually make (and maintain) user to role assignment and role to permission assignment for dynamic applications and/or large scale applications with a large number of users or objects. Recently introduced attribute-based access control (ABAC) has features that are complimentary to RBAC. It is straightforward to use ABAC to represent policy based on the attributes of users, objects, and the access environment, and it is easy to revise policy to adapt to a changing application; however, ABAC is typically more complex than RBAC w.r.t. policy review. Researchers are exploring ways to integrate RBAC and ABAC [1, 11, 12, 15, 16].

We propose a new approach that integrates attribute-based policies into RBAC, designed for supporting RBAC model building for large scale applications. We model RBAC in two layers. One layer, called the"aboveground" level, is a traditional RBAC model extended with environment constraints. This layer retains the simplicity of RBAC, and allows routine operations and policy review. In a second layer, called the "underground layer", we focus on how to construct attribute-based policies to automatically create the primary RBAC model on the aboveground level. This second layer takes the advantages of ABAC, eases the difficulty of RBAC model building, particularly for large scale applications, and brings flexibility to adapt to dynamic applications. In this way, the proposed approach combines the advantages of both RBAC and ABAC.

Prior work (e.g. [1, 11]) focuses on rule-based automatic user-role assignment, thereby addressing the difficulty in user-role assignment when there is a large dynamic population of users. We extend this by defining attribute-based policies for both user-role assignment and role-permission assignment. This extension addresses the challenge posed by a large number of security objects observed in the context of industrial control systems (ICS), by automated role-permission assignment. Many existing research only considers the attributes of users; we consider the attributes of users, roles, objects, and the environment.

This work is motivated by characteristics of ICS and the requirements for a large scale RBAC model for the same [22]. However, the model we propose is generic.

The contents are organized as follows: §2 briefly discusses the related research; §3 presents the aboveground level of RBAC model; §4 presents the underground level of RBAC model; §5 presents the application of this proposed framework in ICS; §6 further discusses our model by comparing it with most relevant previous work; §7 gives a summary and the directions for future research.

2. RELATED RESEARCH

Kuhn et al [12] suggested that combining the best features of RBAC and ABAC can provide effective access control for distributed and changing applications. They compared the strength and weakness of RBAC and ABAC with respect to simplicity of security administration, easiness of reviewing permissions assigned to users, and flexibility to adapt to rapid changing applications; then they presented a spectrum of possible ways to combine RBAC and ABAC; finally, they revealed that standards organizations are developing a policy-enhanced RBAC standard to accommodate attribute-based features. They presented a very interesting picture of the landscape of combining RBAC and ABAC. However, the attributes they considered are limited to user-centered attributes. The attributes of objects and environment are also important to access control, and should be considered.

Earlier, Al-Kahtani and Sandhu [1] proposed a model of rule-based automatic user-role assignment for RBAC, called RB-RBAC, to overcome the difficulty of manual user-role assignment for service-providing enterprises which typically have a huge number of users. In this model, users are dynamically assigned roles by using rules, based on users' attributes. This model also has the limitation of considering only the attributes of users; furthermore, attributes are expressed in propositional logic, thus being less expressive than what we permit (first order logic). In addition their approach to representing mandatory access control is to create roles of *read* and *write* for every node in a security lattice. This approach can lead to a large number of roles; more importantly, the roles are created based on the general security classification lattice rather than specific job functions; this makes it difficult to realize the principle of least privilege.

Similar to [1], Kern and Walhorn [11] also adopted a rule-based approach to user-role assignment, supporting automated administration of roles in large organizations. They pointed out that dynamic user-role assignment as in [1] creates difficulties in reviewing permission assignments and in evaluating the impact of a new rule or revision of rules. To overcome these problems, they proposed static user-role assignment. In this way, the rule-based user-role assignment is separated from run-time RBAC system. This approach has been applied in a bank and identity management solution of an IT service provider.

OASIS issued a standard RBAC profile for XACML [16]. They adopted an approach of representing roles as an attribute rather than entities or subjects of access. Their focus is on using standard XACML [17] to represent the standard (core and hierarchical) RBAC [2], rather than to combine the strength of RBAC and ABAC.

Most recently, Ni and Bertino [15] proposed a new access control language *xfACL*, attempting to combine the benefits of XACML and RBAC, and emigrate their drawbacks. Their focus is on language specification.

In the context of access control for relational databases, Giuri and Iglio [8] introduced the concept of "restricted privilege" to extend a privilege with a condition, called "privilege restriction", under which the object(s) can be accessed with the access mode specified in a privilege. A restricted privilege is a triple of: an access mode (operation type), a single object / or an object collection of the same object type, and a privilege restriction; a privilege restriction is a logic expression on the attributes of an object in the object collection or the context, and if the logic expression is true, then the single object or the objects in the collection satisfying that condition can be accessed in the specified access mode. So, a privilege is a single permission or a group of permissions of the same access mode (operation type) on a set of objects selected by the privilege restriction. Further, they introduced "parameterized privilege" to allow a privilege restriction to contain variables; then, they extend concept of role into "role template" which contains parameterized privileges. A striking feature of their approach is that the concepts of *parameterized privileges* and *role template* allow them capture the common access control patterns over different attribute values, and represent roles and associated permissions in a compact manner. This model can be regarded as a standard hierarchical RBAC model extended with attribute-based constraints on permissions.

Chae and Shiri [4] proposed a variant of RBAC to categorize objects in hierarchical classes (more exactly, groups), to enable association of an object group rather than an individual object with an operation in a permission, and to allow authorization propagation through object group hierarchy. Different from [1, 11, 12], this model deals with the difficulty of handling a large number of objects. Jaeger et al [9] also proposed aggregating objects into groups however their aggregation of objects and operations into groups was based on similarities of objects (e.g., objects with similar operations) and similarities of operations (e.g., operations corresponding to common operative types such as read).

RBAC has been extended in serval directions, particularly with the context or environment of access. Examples include: temporal RBAC [10], location sensitive RBAC [6], context constrained RBAC [14], and others. They provide us clues in modeling the context of access.

With respect to access control in ICS, Bertino [3] discussed the requirements and possible approaches for critical infras-

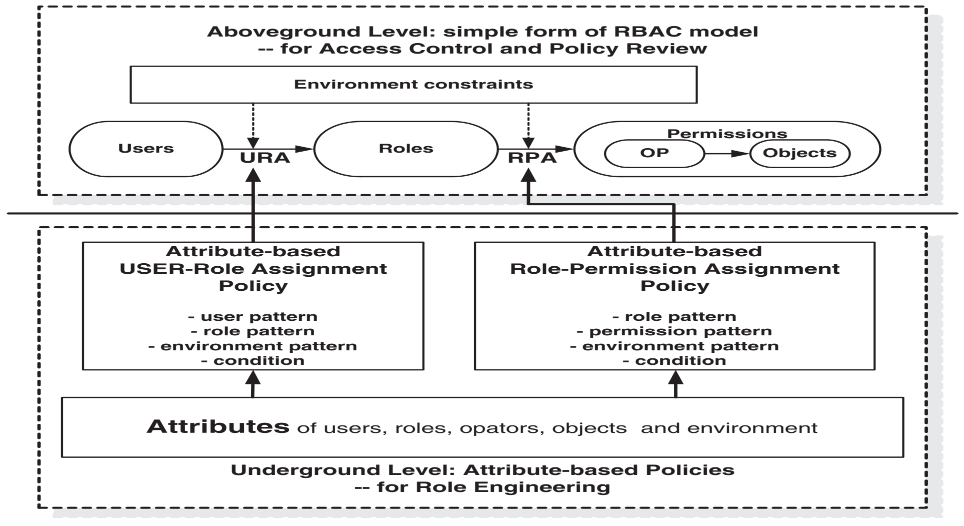
Aboveground Level: simple form of RBAC model
-- for Access Control and Policy Review
Environment constraints
Users
URA
Roles
RPA
Permissions
OP
Objects
Attribute-based USER-Role Assignment Policy
- user pattern
- role pattern
- environment pattern
- condition
Attribute-based Role-Permission Assignment Policy
- role pattern
- permission pattern
- environment pattern
- condition
Attributes of users, roles, opators, objects and environment
Underground Level: Attribute-based Policies
-- for Role Engineering

Figure 1: A two-layered framework integrating Attribute-based policies into RBAC

tructures, particularly addressed the need of attribute-based and context based user-role assignment.

Another area of relevant research is "role engineering", an approach to defining roles and assigning permissions to those roles [5]. Role engineering can be regarded as the process of building a RBAC model instance, including role definition, structuring roles, permission definition, assignment of permissions to roles, and identification of constraints. There are two basic approaches of role engineering: top-down and bottom-up [5, 13, 24]. In the top-down approach, to define roles, role engineers, together with business administration staff, analyze business processes in an organization, identify job functions in business process, decompose job functions into smaller units, identify actions and objects of operations in each work step, identify constraints, finally assign permissions to roles. Top-down approach can be labor-intensive, time-consuming, and expensive. To overcome the problems with top-down approach, the bottom-up approach is developed [13, 24], using machine learning technologies to automatically discover roles from the existing user-permission assignments, so called "role mining". However, the bottom-up approach also has some significant limitations: 1) as commonly recognized, the meaning of the roles discovered by role mining are difficult to interprete with respect to business processes, thus making it difficult to manage and maintain such roles; 2) each role mining algorithm developed so far typically requires user-permission association data in some form, and lacks flexibility to consider various constraints and attributes of interest; 3) the outcome of role mining is not guaranteed to be completely consistent with the existing access control policies; 4) because the outcome of role-mining depends on the input user-permission associations, it is necessary to review (and revise) the input data to make it represent expected access control policy. So, even though roles can be identified automatically, the process to collect, review and revise input data can be labor-intensive and time-consuming as in the case of the top-down approach.

In our specific context of RBAC modeling, the application we target has a very large number of security objects (order of millions), and that there are complex policies beyond the concept of role applied to the access of those objects. Therefore, on one hand, it is impractical to assign each permission (an operation on an object) to a role manually; on the other hand, role mining approach appears insufficient to meet the complex needs of the domain.

3. ABOVEGROUND LEVEL: TABLES

In this paper, we mainly use first order logic to make formal descriptions, and follow the convention that all unbound variables are universally quantified in the largest scope. The aboveground level is a simple and standard RBAC model, but extended with constraints on attributes of the "environment". We use the notion of environment to represent the context of a user's access, such as the time of access, the access device, the system's operational mode, and so forth.

The model is formally described as a tuple,

$$\mathcal{M} = (U, R, P, O, OP, EP, URA^e, RPA^e),$$

where U is a set of *users*. A user could be either a human being or an autonomous software agent; R is a set of *roles*. A role reflects a job function, and is associated with a set of permissions; O is a set of *objects*. Objects are the resources protected by access control; OP is a set of *operators*. An operator represents a specific type of operations; P is a set of *permissions*. A permission is defined by a legitimate operation, which comprises an operator and an object; EP is a set of predefined *environment state patterns*. We use *environment state* to model the context of an user's access. Each environment state pattern (*environment pattern*, hereafter) defines a set of environment states; URA^e is the extended *user-role assignment* relation, which is basically a mapping from users to roles, associated with certain environment patterns; RPA^e is the extended *role-permission assignment* relation, which is basically a mapping from roles to permissions, also associated with certain environment patterns. All sets are finite. In the following, we give some further formal description of *environment*.

3.1 Environment

We represent the context of access as environment, and model an environment as a vector of *environment attributes*, each of which is represented by an environment variable (called an attribute name) associated with an attribute value in a domain. An environment is defined by n attributes, let $v_i \in D_i, i = 1, ..., n$, be the ith environment variable, where D_i is the domain of that environment variable; then a vector $(v_1, ..., v_n)$, in which all variables are instantiated is called an *environment state* (also denoted as s.) The set of all possible environment states is denoted by E. Choice of environment attributes (and hence environment state) is domain dependent. Environment attributes, particularly the dynamic attributes, are gathered by an access control engine at runtime.

Example (environment state): Assume that the environment is defined by three attributes: mode, access location and access time; then mode = "normal" and access_location = "station 1" and access_time = "8:00AM Monday" is an environment state.

An *environment pattern*, denoted as e, is treated as an individual in domain EP, but is semantically defined by a first-order logical expression of assertions involving environmental attributes. An environment pattern defines a set of environment states, in which every environment state satisfies the environment pattern, i.e.

$$\{(v_1, ..., v_n) | e(v_1, ..., v_n)\}.$$

Hereafter sometime we directly use e to denote the set of environment states defined by the environment pattern.

Examples (environment pattern): $Access\_location = station\_1$ and $access\_time \in [8:00, 22:00]$ is an environment pattern, which defines the set of all the environment states, having any mode, *access_location* at *station_1*, and *access_time* between 8:00AM and 10:00PM.

We say that an environment state s matches environment pattern e, iff: $e(s)$ is true. An environment pattern can be empty, denoted by ϕ, which is most general; every state matches ϕ. We say that e_1 is subsumed by e_2, iff: the set of all environment states that match e_1 is a subset of the set that match e_2. That is, $subsume(e_2, e_1)$, iff: $\{s|e_1(s)\} \subseteq \{s|e_2(s)\}$. The relation is reflexive, transitive, and anti-symmetric.

3.2 User-role assignments

A particular user-role assignment associates a user, a role, and an environment pattern :

$$URA^e \subseteq U \times R \times EP, \tag{1}$$

where EP is the set of all environment patterns that have been defined for the system of interest.

The semantics of a user-role assignment, $(u, r, e) \in URA^e$, is defined as:

$$match(s, e) \rightarrow has\_role(u, r), \tag{2}$$

which states that if the real environment state s matches the given environment pattern e, then user u is assigned to role r. We assume that the RBAC engine could understand the semantics of each environment pattern as defined.

Basic RBAC models define user-role assignments simply as a mapping from users to roles,

$$URA \subseteq U \times R. \tag{3}$$

We have extended this notion with a dependency on the environment, as a means to integrate certain extended RBAC features of context and constraints. The environment pattern associated with a user-role assignment is the environment-dependent condition which is sufficient for the assignment. This feature can be regarded as constrained user-role assignment. If there are no constraints on user-role assignments, the associated environment patterns are simply empty, so the model becomes the common one.

In this model, the relation between URA^e and URA is:

$$(u, r) \in URA \leftrightarrow (\exists e, (u, r, e) \in URA^e). \tag{4}$$

User-Role assignments may be expressed in tabular form. Table 1 shows an example, with an environment extension.

3.3 Role-permission assignments

A role-permission assignment associates a role, a permission, and an environment pattern. Thus the set of all such assignments is a subset

$$RPA^e \subseteq R \times P \times EP. \tag{5}$$

The semantics of a role-permission assignment, $(r, p, e) \in RPA^e$, is defined as:

$$match(s, e) \rightarrow has\_permission(r, p), \tag{6}$$

which states that if the real environment state s matches the pattern e, then permission p is assigned to role r.

Similar to user-role assignments, we've extended the common role-permission assignment with environment patterns. The relation between RPA^e and RPA is:

$$(r, p) \in RPA \leftrightarrow (\exists e, (r, p, e) \in RPA^e). \tag{7}$$

As with user-role assignments, the role-permission assignment may also be organized in tabular fashion.

4. UNDERGROUND LEVEL: POLICIES

The underground level of RBAC model focuses on the security policies used to construct the aboveground level of RBAC model. Those security policies are a formalism of a role engineers' tacit knowledge about access control based on the attributes of users, roles, objects, and the environment, and the relation among them. We attempt to explicitly represent the implicit knowledge used to construct a RBAC model, and to integrate the extensions to standard RBAC models in an attribute-based paradigm.

In the following, we treat all users, roles, permissions, operators, and the security objects as "*objects*" (in the sense of the object-oriented design) and each of which has certain attributes. Notion *obj.attr*, or equivalently *attr*(*obj*), denotes the attribute *attr* of object *obj*.

The attributes needed in RBAC are typically domain dependent, and need to be customized for each specific target system. Some examples of attributes are as follows. The attributes of users may include "id", "department", "security clearance", "knowledge domain", "academic degree" or "professional certificate". A role may have attributes such as "name"; "type" reflecting job function types such as "manger", "engineer", and "operator"; "security level"; "professional requirements"; "direct superior roles" and "direct subordinate roles" (if role hierarchy is modeled). Objects may have attributes such "id", "type", "security level", "state". Operators may have attributes like "name", and "type". The environment attributes may include "access time", "access location", "access application", "system mode", "target value", and so forth.

Table 1: Extended User-Role Assignment Table

User Id	Role Id	Environment Pattern
com:ab:zn1:amy	Manager.Zone1	Device = "Station_1.2" & Time = "Weekday"
com:ab:zn1:ben	Engineer.Zone1	Device = "Station_1.2" & Time = "Weekday" & Mode = "normal"
com:ab:zn1:jim	Engineer.Zone1	Mode = "emergency"
com:ab:zn1:bob	Operator.Zone1	Device = "Station_1.2" & Time = "Weekday" & Mode = "normal"
...	...	...

4.1 Role-Permission Assignment Policy

The role-permission assignment policy is a set of rules. Each rule has the following structure:

```
rule_id {
  target {
    role_pattern;
    permission_pattern {
      operator_pattern;
      object_pattern;
    };
    environment_pattern;
  }
  condition;
  decision.
}
```

where, all of the *patterns* and the *condition* are FOL (First Order Logic) expressions; an *environment pattern*, as presented in § 3.1, defines a set of environment states; similarly, a *role pattern* defines a set of roles by specifying their common attributes; a *permission pattern*, consisting of an operator pattern and one object pattern, defines a set of permissions by specifying their common features w.r.t. the attributes of the operator and the object in a permission; an *operator pattern* defines a set of operators (or operation types); each *object pattern* defines a set of objects; the *target*, which is, formally, the Cartesian product of the above sets of roles, permissions, and environment patterns, defines *the range of (role, permission, environment_pattern) triples to which this rule applies*; the *condition* is a logical expression defining a relation among the attributes of both the roles, the permissions (operators and objects), and the environment. This expression is the *condition under which a role-permission assignment can be made*; the *decision* is the role-permission assignment. This form of rules states that when the condition is true, a role covered by the role pattern can be assigned with a permission covered by the permission pattern in the specified environment pattern.

Let $\mathit{pattern}^R(r)$ denote a role pattern, refereing the role as r, e.g. r.type="engineer"; $\mathit{pattern}^P(op, o)$ denote the general form of a permission pattern, referring to permissions of form $p(op, o)$. A permission pattern actually consists of zero or one operator pattern and zero or one object pattern; $\mathit{pattern}^E(\varepsilon)$ represents a predefined environment pattern ε; $\mathit{condition}(r, p(op, o), \varepsilon)$ denotes a logical expression, describing a relation among the attributes of role r, object o, and environment ε.

The semantics of a role-permission assignment rule is defined as follows.

$$(\forall r, op, o)(\mathit{pattern}^R(r) \wedge \mathit{pattern}^P(p(op, o)) \wedge \mathit{pattern}^E(\varepsilon) \wedge \mathit{condition}(r, p(op, o), \varepsilon) \supset (r, p(op, o), \varepsilon) \in RPA^e) \quad (8)$$

which states that for any role r, satisfying the given role pattern $\mathit{pattern}^R(r)$, for any permission $p(op, o)$, satisfying permission pattern $\mathit{pattern}^P(p(op, o))$, if *condition(r,p(op,o),ε)* is true, then role r is assigned with permission *p(op,o)* in environment pattern ε.

A pattern can be empty, ϕ. An empty pattern defines the most general pattern, which every element in a domain matches. For example, if role pattern is empty, then the defined role-permission assignment rule is applied to all of the roles in R. Therefore, if the target of a rule is empty, then the rule is most general and applicable to all combination of role, permission, and environment; on the other hand, a rule is most specific, (i.e. only applicable to a single specific combination,) if the target of the rule is defined most specifically, i.e. the role pattern is exactly defined as a specific role instance, the permission pattern is exactly defined as a specific permision/operation, and the environment pattern is also most specific. Generally, the target of a rule defines a specified range of (role, permission, environment) triples to which the rule applies.

In this framework, a role could be defined based on a role template. Different from the concept of "role" in RBAC, a role template is associated with a set of permission patterns rather than permissions. The idea is developed to address scaling issues that arise in the ICS domain. We will give detailed discussion on role template and "proto-permission" (a specific form of permission pattern) in §5.

Examples of attribute-based RPA policies are given in §5 paragraph (f).

For simplicity, we only use positive rules. If (r, p, e) is in RPA^e, then role r is assigned with permission p in environment e; if not, by the close world assumption, we conclude that role r does not have permission p in environment e.

In role engineering, the defined role-permission assignment policy (rule set) is applied to all possible combinations of (role instance, permission, environment pattern), and if a combination can be inferred by any rule, then it is included in the RPA^e on the aboveground level of RBAC model.

4.2 User-Role Assignment Policy

User-role assignment is highly dependent on business rules and constraints. In our view, the task of assigning users to roles can be approached like the role-permission assignment problem, in terms of policies that enforce those rules and constraints. Such policies would be formulated in terms of user and role attributes, and would be crafted to enforce things like Separation of Duty. However, unlike role-permission assignment, user assignment may have to balancing competing or conflicting policy rules against each other. Correspondingly a complete policy oriented formulation will need to specify how to combine rules and arrive at a final assignment decision. In what we present below, an attribute based user-role assignment policy is used only to identify *po-*

tential assignments. A rule-combining algorithm is used for making the final assignments.

Similar to role-permission assignment rule, the user-role assignment rule consists of:

```
rule_id {
    target {
        user_pattern;
        role_pattern;
        environment_pattern;
    }
    condition;
    decision.
}
```

where, similarly, an user/role/environment pattern defines a set of users/roles/environment_states by specifying the common attributes; all of the *patterns* and the *condition* are expressions in first order logic; differently, the *decision* of a rule is *to mark an (user,role,environemt_pattern) triple as a potential assignment*.

Let $pattern^U(u)$ denote a user pattern, referring to user u; $pattern^R(r)$ denote a role pattern, referring to role r; $pattern^E(\varepsilon)$ denote the specification of an environment pattern; $condition(u, r, \varepsilon)$ denote a logical expression, describing a relation among the attributes of user u, role r, and environment pattern ε.

The semantics of a user-role assignment rule is defined as follows.

$$(\forall u, r)(pattern^U(u) \land pattern^R(r) \land pattern^E(\varepsilon) \land condition(u, r, \varepsilon) \to (u, r, \varepsilon) \in temp\_URA^e), \quad (9)$$

which states that for any user u satisfying $pattern^U(u)$, any role r satisfying $pattern^R(r)$, if the condition that states a relation among the attributes of u, r, and ε is true, then $(u, r, \varepsilon) \in temp\_URA^e$, which means that according to this rule, u can be assigned to r in environment ε.

Example:

```
rule:{
    target:{
        role_pattern(r): r.type = "chemical engineer";
        environment_pattern(e):{
            Device = "Station_1.2"
            and Time = "Weekday"
            and Mode = "Normal" }
    }
    condition:{
        knowledge_match(u,r);
        security_req_match(u,r);
        u.base_plant = "Houston";
    }
    decision: add (u,r,e) in URAe.
}
```

In this rule, knowledge_match(u, r) is a function which returns true if the professional knowledge of user u matches the knowledge requirements of role r; security_req_match(u, r) is a function which returns true if the user meets the security requirements of role r. A necessary condition for considering assigning a user u to role r is that the user work at the Houston plant (where presumably one finds Station_1.2.)

The rule has no filter on the users it considers. The rule applies to roles with job_type attribute "chemical_engineer". Now suppose that there is a role "Engineer.Zone.1.2" designed to work in Zone 1, on Station_1.2, on weekdays; this role has job_type attribute chemical_engineer, requires professional knowledge in Chemical Engineering, and at least a Bachelor's degree in that field. Going through employees one finds John, who works at the Houston plant; John has a Bachelor degree on Chemical Engineering; and suppose that John sufficiently meets the security requirements for role "Engineer.Zone.1.2". The patterns for user and role cover these instances, the condition of the rule is true, and so the assignment of John to that role will be marked as possible. Of course, there may be other roles that John is suitable for, and there may be other employees suitable for role "Engineer.Zone.1.2", so a later step is needed to select among all assignments marked as possible, by considering some constraints such as separation of duty.

The following is the pseudo-code of rule combining algorithm.

```
rule_combining(temp_URAe) {
    for (i = 0; i < constraints.length(); i++){
        if ( ! satisfy(temp_URAe, constraints[i])){
            add i in conflictList;
        }
    }
    if ( conflictList.length() == 0) {
        URA^e = temp_URAe;
    } else {
        if (constraintConflictResolution(temp_URAe,
                    conflictList, temp2_URAe))
            URAe = temp2_URAe;
        else
            notify(temp_URAe, conflictList);
    }
}
```

Note that the constraint conflict resolution algorithm is dependent on constraints, which in turn are domain-dependent, so it is difficult to give a general algorithm for constraint conflict resolution. If conflict cannot be resolved by algorithm, the algorithm will inform RBAC administrator, and the conflict resolution needs to be performed manually (or with tools we have not identified) by the role-engineer.

5. CASE STUDY : LARGE SCALE ICS

This section presents how the proposed framework might be applied to construct a RBAC model for industrial control systems (ICS). Our knowledge of these systems is informed by close collaboration our institute has with vendors of commercial ICS systems.

Problems: The target application domain of ICS has the following features. There is a very large number of security objects (order of millions) with complex relations among them; on the other hand, many objects and operations applied on them are similar, and there are patterns to follow; security objects are organized in hierarchical structures; each role or security object may have a security level; users dynamically change over time as business changes; security objects also change over time due to device replacement or maintenance; access to control processes and devices is through some Human-Machine Interfaces and software applications; each protected point of access to a control system, called "*point*", or "control blocks", contains information about the status of a control process or device, and is used to set target control values or control parameters; all those points are important parts of the security objects to be protected by the target access control system; runtime operation environment (with dynamic attributes), e.g. access location and/or access time, is a sensitive / important factor in access control; different zones have similar structure, i.e. roles, operations and objects are similar in different zones; zones and devices may have operation "modes"; control stations play an important role in access control.

A specific challenge we face is to construct an RBAC framework that is compatible with access control mechanisms used in the modern ICS, which are a mixture of different mechanisms including station-based, group-based, attribute-based, and (simplified) lattice-based. The underlying reason for the compatibility requirement is to recognize the practical need for an incremental transition path. Another major challenge is how to define roles and assign fine-grained permissions to roles for a large scale application effectively, efficiently, and in an automated fashion as much as possible.

5.1 RBAC Model Building Process

In the following, we briefly present how to apply the proposed framework in building RBAC model instance for ICS. More detailed discussion and design of RBAC for ICS can be found in [23].

(a) **Identify security objects and object hierarchy**: In a plant, security engineers identify devices and points to be protected by RBAC as a set of security objects, and organize those objects in hierarchical structures (an acyclic graph, generally). In ICS, it is common to find that the control devices and other assets involved in monitoring and controlling the physical process and in overall management of a plant are organized in an hierarchical and modular manner as per ISA-88 and ISA-95 standards. This hierarchical structure grouping objects is called "object hierarchy" in this paper.

(b) **Identify operation types**: All types of the operations applied to each security object are identified, e.g. a specific type of control parameter may have the types of operations such as "read" and "reset". Minimally, an operation type could be defined by just an *operator*; generally, an operation type could be defined by a *permission pattern*.

(c) **Identify *role templates***: ICS tend to have very well defined job functions, with well defined access needs, to monitor and control the physical processes. For example, "Operator" and "Engineer" are very well understood job functions in the ICS community even though there might be some variations in their functions across different types of ICS. In the world of electric power "operator" is actually a licensed position. Such well-defined job functions with well defined access needs are good candidates to define "roles". However, in practice users performing these well defined job functions are limited in scope to either a particular sub-process or a particular geography in accordance with the organization of the plant. In other words, the job functions identified in ICS, such as 'Operator" and "Engineer", perform certain types of operations; what objects those operations can apply to are dependent on the attributes of users and those objects. From this point of view, we defined in our previous work [23, 22]: *role template* and *proto-permission* that leverage some common characteristics of ICS to simplify the creation and management of roles. Here we show how our framework can be used to represent these concepts and realize a manageable RBAC solution for ICS. Different from a permission that comprises a pair of *operator* and *object*, a *proto-permission* consists of an *operator* and the object type of the operand. So, a *proto-permission* represents just an operation type, rather than a specific operation on a specific object. Proto-permission is a specific form of permission pattern. Different from the concept of "role" in RBAC, a *role template* is associated with a set of *proto-permissions*. Therefore, a *role template* tells what types of operations can be performed by a role created from the role template. Let RT denote the set of all role templates; PP denote the set of proto-permissions. Every *role template* has an attribute (or equivalently function) of *proto-permission set*, denoted as *pps*; every proto-permission has an attribute (or function) of *operator*, and an attribute (or function) of object type, denoted as *objType*. These can be formally described as follows.

$$pps : RT \rightarrow 2^{PP}; \tag{10}$$

$$op : PP \rightarrow OP; \tag{11}$$

$$objType : PP \rightarrow TYPES. \tag{12}$$

We use concept *role template* to formally represent each well defined job function in ICS, and use *proto-permission* to represent each allowed operation type associated with a role template. Assume that the plant has a number of basic types of job functions such as "Operator", "Engineer", and "Manager". They are identified as role templates. Each of them associates with a set of operation types, e.g. "Engineer" has operation type "reset_parameter" on objects of type "XYZ", "Manager" has operation type "view_schedule" on "System", et al. All identified operation types should be covered by role templates.

(d) **Identify *roles* and their *privilege ranges***: Based on business workflow analysis, a number of roles can be identified for each role template. Each role has an assigned working (access) environment pattern such as access through station B in zone 1 on daytime. Furthermore, as discussed earlier, in practice each role only has access to a range of objects; this range of objects accessible to a role is called the *privilege range* of the role. Formally,

$$pr : R \rightarrow 2^{O}. \tag{13}$$

Privilege range is used to define the boundary of objects for which a role is responsible, and is used as constraints in role-permission assignment. A role has access to an object only if the object is within the role's *privilege range*. If the *privilege range* is not concerned, then it can be simply set as the whole set of objects.

Privilege range can be defined over *object hierarchy*. An *object hierarchy* (denoted as OH) is simply a subset of the power set of all *objects* considered, i.e. $OH \subseteq 2^{O}$. A node in the *object hierarchy* is called an "*object group*", which is a subset of the *objects*, i.e. og $\in 2^{O}$. In an *object hierarchy*, that an *object group* is a child of another in OH means that the former is a subset of the latter.

For example, the privilege range of an instantiated role "engineer_A.1.2.1" might be defined as:

$$\begin{aligned} &pr(\text{“}engineer\_A.1.2.1\text{'}) = \\ &\qquad objectgroup(A.1.2.1) \cup objectgroup(C.1.2) \\ &\qquad\qquad - objectgroup(C.1.2.1.3) \\ &\qquad - \{x \mid x.type = \text{“}XXX\text{”} \wedge x.Security\_level > 100\}. \end{aligned} \tag{14}$$

A label such as $C.1.2.1.3$ describes a node (object group) identified by a path through a hierarchy. Starting at node C, one selects an immediate child labeled "1" with respect to C's parentage, having parentage C, from that node one selects the node labeled "2" with respect to $C.1$'s parentage, having parentage $C.1$ and so on. This scheme reveals immediately that $C.1.2$ is higher up in the hierarchy than $C.1.2.1.3$ as one passes through the former node to reach the latter. In the above example, the subtracted sets are "exceptions", which can be a single node in an object hierarchy, or a set which is defined by a logical expression as needed.

Back to the task of role identification, as an example, a role called "Engineer_Chem_Zone1_Daytime" may be created, based on role template "Engineer". This role has privilege range that covers all Chemical engineering related objects in zone 1, and this role works in a predefined environment pattern called "Daytime_Zone1". There could be other "Engineer" roles such an on electrical engineering, in zone 2, and working in evening shift, and so forth;

(e) **Analyze security policies and identify attributes**: The security engineers analyze all security policies and requirements applied to the plant, and list all of the concerned attributes of users, roles, objects, and access environment. For example, *privilege range* is a critical attribute of a role. There are many other attributes that may be of concern, as the examples given in §4. A particular attribute that needs to be considered is "security level". It is common to assign a security object a "security level", requiring a subject of access to have a corresponding security level.

(f) **Develop *attribute-based policies***: The security engineers construct the underground level of RBAC model by developing attribute-based policies for role-permission assignment and user-role assignment.

First, let us consider a simple yet general case: (1) a role can only perform the types of operations, specified by the proto-permissions of the role's template; (2) privilege range constraint: a role can access an object only if that object is within the privilege range of that role; (3) security level constraint: to access an object, the role's security level needs to be greater than or equivalent to the one of the object. This can be represented with the attribute-based policy for role-permission assignment:

```
rule:{
    target:{}
    condition:{
        memberOf(o, r.pr);
        r.securityLevel >= o.securityLevel;
        memberOf(pp, r.template.pps)
            and op = pp.op and o.type = pp.objType;}
    decision: add (r, p(op,o), \phi) in RPAe.
}
```

The formal semantics of this rule is as follows:

$$\begin{aligned}(\forall r, pp, op, o)(&\mathit{memberOf}\,(o, r.pr)\\ &\wedge r.\mathit{securityLevel} \geq o.\mathit{securityLevel}\\ &\wedge \mathit{memberOf}\,(pp,\ r.\mathit{template.pps})\\ &\wedge op = pp.op \wedge o.\mathit{type} = pp.\mathit{objType}\\ &\rightarrow (r, p(op, o), \phi) \in RPA^e), \qquad (15)\end{aligned}$$

which states that for all *role* r, *proto-permission* pp, *operator* op, and *object* o, if o is within the privilege range of role r, the security level of r is greater than or equivalent to the one of o, *proto-permission* pp is within r's *role template*'s *proto-permission set*, and op is the *operator* specified in pp, as well as the type of o is the object type specified in pp, then role r is assigned with permission (instance) $p(op, o)$ without any environment constraints (where environment pattern is empty).

Nevertheless, in reality, role-permission assignment must satisfy more constraints. Let us look at the example below. (1) using role template, "*Engineer*", and using a proto-permission that allows to perform "*reset_parameter_T*" type of operations on objects of type "ObjectType_YYY" ; (2) privilege range constraint: a role can access an object only if that object is within the privilege range of that role; (3) security level constraint: to access an object, the role's security level needs to be greater than or equivalent to the one of the object; (4) environment constraint: the operation must be performed in the mode of "normal"; (5) environment constraint: access time must be in daytime shift; (6) environment constraint: the operation must be performed from station "station_X"; (7) station privilege range constraint: the object to be accessed must be within the privilege range of the operation station; (8) parameter range constraint: the target value of the parameter to be set must be within a specified interval [68,73]; (9) professional domain constraint: the role's professional domain must match with the object's domain. These security requirements can be represented as an attribute-based rule as follows:

```
rule: {
    target: {
        role_pattern(r): r.template.id = "Engineer";
        operator_pattern(op): op = "reset_parameter_T";
        object_pattern(o): o.type = "ObjectType_YYY";
        environment_pattern(e): {
            e.mode = "Normal";
            e.accessTimeStart = 8AM;
            e.accessTimeEnd = 16PM;
            e.station = "Station_X";
            e.targetValueInf = 68;
            e.targetValueSup = 73;
        }
    }
    condition: {
        memberOf(o, r.pr);
        memberOf(o, e.station.pr);
        r.securityLevel >= o.securityLevel;
        r.profDom = o.profDom;
    }
    decision: add (r, p(op,o), e) in RPAe table.
}
```

The role-permission assignment policy consists of a set of rules like the one above. As stated earlier, we use only positive rules, therefore, a permission is assigned to a role if any rule grants the assignment.

User-role assignment rule can be specified based on the attributes of users and roles in a similar manner, as shown in §4.2.

(g) **Create RBAC assignment tables**: Use the specified attribute-based policies to create role-permission assignment and user-role assignment in the aboveground level.

This task can be illustrated by considering how the above example rule is used to make role-permission assignment. For each pair of role and permission, if the pair or the role's assigned working environment does not match the target of the rule, then skip this pair; otherwise, continue to evaluate the condition part of the rule. If the condition is true, then assign the permission to the role in that assigned environment, in the role-permission assignment table of the aboveground level. Consider role "Engineer_Chem_Zone1_Daytime", and a permission "reset_parameter_T" on object "point_1.2.7" (saying it represents the 7th point in zone 1 sector 2), in the environment pattern as stated in the example rule. This (role, permission) pair is within the target; assume that object "point_1.2.7" is within the privilege ranges of the role and the access station; the role's security level dominates the object's; the professional domains match; then the permission is assigned to the role in the specified environment pattern. For another role, saying, "Engineer_Chem_Zone2_Daytime", having privilege in zone 2 which does not cover "point_1.2.7", thus that permission cannot be granted.

(h) **Verify/review RBAC model**: Formally verify / review the RBAC policy represented by the aboveground level of RBAC model instance against security policies and requirements by using logic [21]. Detailed description on policy review will be presented in another paper.

(i) **Repeat the process**: If the above logical verification and review fail, go back to an earlier step to revise the RBAC model, and then verify and review again.

Detailed role engineering process for ICS is out of the scope of this paper, and will be discussed in another paper.

5.2 Discussion on Case Study

Migration to RBAC from a legacy system could be a great challenge for ICS. The proposed framework could support building RBAC model for ICS and the migration. We highlight some major features as follows.

Expressibility: The proposed framework is general enough to cover the required features of the targeted access control systems in ICS; [20]; User groups are modeled by *role template*; the types of operations conducted by a user roup is modeled by *proto-permission*; station-based access constraints, access application constraints, temporal constraints, "mode" constraints, "parameter range' constraints, and others are modeled as environment constraints.

Support for role engineering: Role engineering is widely recognized as difficult; it becomes even more challenge for ICS due to the large scale and dynamic features. The proposed framework enables automatic user-role assignment and role permission assignment, through attribute-based polices. This approach could largely help to overcome the problems of manual user-role assignment and role-permission assignment for large scale applications in ICS.

Simplicity: The proposed framework can integrate the existing mechanisms and concepts in ICS uniformly, in the form of attribute-based policies; thus avoiding complexity caused by ad hoc representation and management. Attribute-based policies can express security policies and requirements in a straightforward manner; thus easier to construct and maintain. The simplicity of the aboveground level eases RBAC model review.

Flexibility: The attribute-based policies have flexibility to adapt to the dynamically changing users, objects, security policies and requirements, and even business processes.

Verifiability: The logic representation of the attribute-based policies provides a basis for formal verification of the RBAC model.

6. FURTHER DISCUSSION

In our approach to RBAC modeling, in the underground level, we develop attribute-based policies for user-role assignment and role-permission assignment separately, and use these policies to create the aboveground level of RBAC in a simple and standard form. The policy form of role definition and user-role assignment makes the model easier to build and easier to change for dynamic applications. The simple form of RBAC in the aboveground makes the model easy to operate and easy to perform access review. In this way, this proposed framework combines the advantages of both ABAC and RBAC.

In the following, we clarify our contributions with respect to prior art. Our framework of combining ABAC with RBAC is new compared to all of the strategies summarized by Kuhn et al in the table 1 of [12] and other research presented in Section 2. Using the mapping notion of [12], our framework has a 2-level and 2-step architecture: (1) underground level: $U, A^U, A^R, E \rightarrow R$; $R, A^R, A^O, E \rightarrow perm$; (2) aboveground level: $U, E \rightarrow R$; $R, E \rightarrow perm$, where $A^U/A^R/A^O$ is the set of attributes of users/roles/objects. In addition, past research tends to focus on the attributes of users (including [12]), but our model considers the attributes of all relevant entities including users, roles, objects, and the environment.

Al-Kahtani and Sandhu's research [1] and Kern and Walhorn's work [11] focused on addressing the difficulty of user-role assignment. We move further to provide a solution for the problems of role-permission assignment in large scale applications with millions of objects.

Chae and Shiri [4] addressed the problem of dealing with large number of objects in a manner somewhat similar to ours. They proposed to organize objects in "classes" (more exactly, groups) with hierarchal structure; a permission is defined as association of an operation (operation type) with a "class" (group) of objects, and a role having that permission can perform that operation over all objects in that "class" (group). In our model, a proto-permission is defined as association of an operation type and an object type; a role having a proto-permission can perform a specified type of operations over the objects of a specified type in the privilege range of that role; a privilege range can be defined arbitrarily as needed, typically based on an existing object group in the object hierarchy and modified with "exceptions" defined with relevant attributes. The main advantage of our model is that we are not only able to support a object hierarchy but also deal with exceptions to inheritance in that hierarchy.

Our research is also relevant to role engineering. As discussed in §2, role engineering is a difficult task. One of the reasons underlying the difficulty of role engineering is that the simplicity of RBAC transforms the complex access control knowledge into the tacit knowledge owned by role engineers who define roles. We envision that the explicit representation of that knowledge will ease the difficulty of RBAC system building, enable to formally verify RBAC model, and make it easy to adapt to the changes. Our approach to RBAC model building can be regarded in the stream of the "top-down" approach of role engineering. A top-down approach mainly includes role name identification from business process analysis, permission identification, and role-permission assignment, and the approach is typically carried out manually; we assume that from business process analysis, role names, role templates (or types of roles), proto-permission (or types of operations), and the attributes concerned are identified manually; then we develop the attribute-based policies and use the policies to automatically create user-role assignment and role-permission assignment. In the course of using RBAC, some security requirements or regulations may change. It is commonly recognized that it is difficult to change a RBAC model instance, particularly a large one. By using our framework, role engineers can more easily change the attribute-based policies, then use the updated policies to generate the user-role assignment and role-permission assignment in the aboveground level.

Finally, we find that no single existing RBAC model readily meets all needs of the ICS domain, such as unifying all access mechanisms in use and automatic role-permission assignment. Our proposed framework could work well in the context, as discussed in §5.2.

Although motivated by ICS, our proposed framework is generic, and can be used in other areas, such as health information networks, military and government information management. Particularly, access control for cloud-based information and computing services across security domains is a great challenge; our proposed framework can be used in a unified form of attributed-based policies to combine RBAC with traditionally used mandatory access control, including non-hierarchical caveats, and other security polices; the framework also helps to deal with a great number of security objects, including both information and cloud services.

7. CONCLUDING REMARKS

This paper proposed a new approach to combining ABAC and RBAC, that brings together the advantages of both the models. We developed our model in two levels: aboveground and underground. The aboveground level is a simple and standard RBAC model extended with environment constraints, which keeps the simplicity of RBAC, and supports straightforward security administration and review; in the underground level, we explicitly represent the knowledge for RBAC model building as attribute-based policies, which are used to automatically create the simple RBAC model in aboveground level. The attribute-based policies bring the advantages of ABAC: they are easy to build and easy to change for a dynamic application. We showed how the proposed approach can be applied to RBAC system design for large scale ICS applications. Regarding future work, we will continue to make formal analysis of the the properties of the proposed model, further explore approaches of formal verification of a RBAC model instance against security requirements and higher level of policies, and work out the proposed framework based role engineering process. Another direction to go is to extend the proposed model to cover more features such as operations with multiple objects.

Acknowledgments

We would like to thank Tom Markham, Julie Hull, Alex Chernoguzov, Kevin Staggs, and Himanshu Khurana for their valuable discussion and insight.

8. REFERENCES

[1] M. A. Al-Kahtani and R. Sandhu. A model for attribute-based user-role assignment. ACSAC '2002.

[2] ANSI. *American National Standard for Information Technology - Role Based Access Control.* 2004.

[3] E. Bertino. Policies, access control, and formal methods, 2012. Chapter in Handbook on Securing Cyber-Physical Infrastructure (in print).

[4] J. H. Chae and N. Shiri. Formalization of rbac policy with object class hierarchy. In *Proceedings of the 3rd international conference on Information security practice and experience*, ISPEC'07, pages 162–176, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] E. J. Coyne and J. M. Davis. *Role Engineering for Enterprise Security Management.* Artech House, Inc., Norwood, MA, USA, 1st edition, 2008.

[6] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. Geo-rbac: A spatially aware RBAC. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[7] D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the NIST-NSA Conference*, pages 554–563, 1992.

[8] L. Giuri and P. Iglio. Role templates for content-based access control. RBAC '97, pages 153–159, New York, NY, USA, 1997. ACM.

[9] T. Jaeger, A. Prakash, J. Liedtke, and N. Islam. Flexible control of downloaded executable content. *ACM Trans. Inf. Syst. Secur.*, 2:177–228, May 1999.

[10] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. on Knowl. and Data Eng.*, 17:4–23, January 2005.

[11] A. Kern and C. Walhorn. Rule support for role-based access control. SACMAT '2005, pages 130–138. ACM, 2005.

[12] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding attributes to role-based access control. *Computer*, 43(6):79 –81, June 2010.

[13] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with multiple objectives. *ACM Trans. Inf. Syst. Secur.*, 13:36:1–36:35, December 2010.

[14] G. Neumann and M. Strembeck. An approach to engineer and enforce context constraints in an rbac environment. SACMAT '03, pages 65–79. ACM, 2003.

[15] Q. Ni and E. Bertino. xfacl: an extensible functional language for access control. SACMAT '11, pages 61–72. ACM, 2011.

[16] OASIS. Core and hierarchical role based access control (RBAC) profile of XACML v2.0, Feb. 2005.

[17] OASIS. eXtensible Access Control Markup Language (XACML) version 2.0, OASIS standard, 2005.

[18] A. C. O'Connor and R. J. Loomis. 2010 economic analysis of role-based access control, nist report, 2010.

[19] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29:38–47, February 1996.

[20] UIUC-ITI. RBAC driven least privilege architecture for control systems solution requirements specification, July 2011. Technical Documentation.

[21] UIUC-ITI. RBAC policy audit and review, Nov. 2011. Technical Documentation.

[22] UIUC-ITI. An RBAC specification for industrial control systems, Oct. 2011. Technical Documentation.

[23] UIUC-ITI and Honeywell Labs. Towards an RBAC for distributed control systems, Apr. 2012. Technical Report, `http://users.crhc.illinois.edu/pvt/rbac-for-dcs-tr.pdf`.

[24] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: A formal perspective. *ACM Trans. Inf. Syst. Secur.*, 13:27:1–27:31, July 2010.