Specification of Attribute Relations for Access Control Policies and Constraints Using Policy Machine

Vincent C. Hu, David D. Ferraiolo, Serban Gavrila
National Institute of Standards and Technology, Gaithersburg. Maryland, USA
{vhu, dferraiollo, gavrila}@nist.gov

Abstract— **Attribute relations in access control mechanisms or languages allow accurate and efficient specification of some popular access control models. However, most of the access control systems including today's de-facto access control protocol and specification language, XACML, does not provide sufficient syntactic and semantic support for the specification of attribute relations in their scheme. In this paper, we show the deficiencies of XACML in specifying such capabilities in the implementations of the Multilevel Security, Hierarchical Role Based policies and Separation of Duty requirements of access control systems. In comparison, we then demonstrate the attribute relation mechanism provided by a relation-based access control mechanism – the Policy Machine.**

Keywords-access control; access control model

I. INTRODUCTION

A critical capability of an access control (AC) system is to allow an AC administrator to specify relations between AC attributes. With this capability, an AC system is able to maintain hierarchical orders of the attributes of the AC elements (subjects, actions, objects). The expression of privilege inheritance relations is essential for many popular AC models such as **Bell-La Padula**[1] and **Biba** [2] (**BLPB**) of **Multileve Security** (**MLS**)[3], and **Hierarchical Role Based Access Control** (**HRBAC**)[4] as well as constraint policies such as **Separation Of Duty** (SOD)[5].

The syntactic and semantic supports of attribute relation (AR) specifications in AC mechanisms or languages allow not only accurately specifying but also efficiently enforcing the relation-based AC models and policy constraints. The specific advantages of such capabilities include:

- Specifying hierarchical relations for the inherit or inherited privileges of subjects, actions, and objects in AC policies. For example, if subject *X* is **related** to subject *Y* then subject *X* **inherits** all the access privileges of subject *Y*.
- Efficient management of AC rules, such that AC policy administrators can modify privileges based on attribute groups and relations without leaking access permissions. Also, through a GUI, it is possible to display all the linkages of existing related attributes, thus providing a complete view of the current privilege assignments.
- Performance enhancement for evaluating access requests, because the AC system does not have to go through all the AC rules to collect attribute information for the grant decision if higher level attributes of the request can be found to match the rule.

eXtensible Access Control Markup Language (XACML) [6] is today's de-facto protocol and specification language scheme for AC implementations. XACML provides a flexible and mechanism independent representation of access rules that vary in granularity; It allows the combination of different authoritative domains' policies into one policy set for making access control decisions in a widely distributed system environment. However, XACML does not provide a scheme for specifying ARs; instead, ARs can be implemented in one of its architectural components (e.g., PEP) by ad-hoc applications.

In this paper, we first show the deficiencies of XACML in specifying ARs. We then demonstrate the virtues of an AR mechanism from a relation-based AC mechanism – *Policy Machine* (*PM*) [7, 8], which includes a server engine called *Policy Server (PS)* and a policy management system, called *General Policy Management System (GPMS)*. *PS* and *GPMS* together enable enforcement of multiple access control policies within a single, unified system. *PM* composes and combines access control policies from a relatively small set of atomic properties completely expressed with mappings and interrelationships of the ARs on three basic elements – *Subject Sets*, *Object Sets*, and *Operation Sets*. Mappings and interrelationships of ARs are enforced with a database and a fixed set of functions.

This paper contains six sections. Section I introduces the AR of AC policies. Section II explains AR implementation in the popular XACML scheme. Section III introduces the architecture and functions of *Policy Machine* (*PM*). Section IV demonstrates *PM*'s mechanism for specifying ARs for AC models and policy constraints. Section V compares *PM* with related work. Section VI is the conclusion.

II. ATTRIBUTE RELATIONS IN XACML

XACML provides an AC policy specification language in an XML scheme as well as generic architecture components (PDP, PEP, PIP, and PAP) for the AC enforcement functions. The regular expressions of XACML Version 2 are listed as following.

(1) PS: T+ PS + P + PCA + O
(2) T: S + R + A + E
(3) P: T + RL +RCA + O

(4) RL: T + C + E

where *PS* is the *PolicySet*, *T* is the *Target*, *P* is the *Policy*, *PCA* is the *Policy Combination Algorithm*, *O* is the *Obligation*, *S* is the *Subject*, *R* is the *Resource*, *A* is the *Action*, *E* is the *Environment*, *RL* is the *Rule*, *RCA* is the *Rule Combining Algorithm*, *C* is the *Condition*, and *E* is the *Effect* for the XACML language scheme.

Regular expressions (*2*) and (*4*) are used for composing AC rules by the basic AC elements: subjects, resources, actions, and environment variables. Regular expressions (*1*) and (*3*) are for associating (*2*) and (*4*) in two different levels. There is no grammar for the expression of ARs in these four regular expressions unless specified by enumerating every relation between attributes. Additionally, XACML allows functions to be implemented to handle ARs in a PEP or an extended function. And those two methods are ad-hoc efforts without formal and structural definition in the scheme. In comparison, we will introduce an AC mechanism that provides a well-defined framework for the specification of attribute relations in Section III.

Note that even though XACML Version 3 has more (and concise) elements in the language scheme than Version 2, for the purpose of explaining the ARs by the basic AC elements (i.e., subject, action, and object), we use XACML Version 2. The issues discussed in this paper apply to Version 3 as well.

A. *Specification of MLS and HRBAC Policies*

BLPB models for MLS policies require assigning **classes** (ranks) attributes to subjects and objects. Formal definitions are $Rs = \{\ldots(Sa_i, Sa_j)\ldots..\}$ and $Ro = \{\ldots(Oa_i, Oa_j)\ldots\}$, where *Rs* is a set of ARs for subject classes: for instance, Sa_i is the "Top Secret" class and Sa_j is the "Secret" class. *Rs* defines the "no read up" property of BLPB. In the same manner, Oa_i and Oa_j define the object classes and property. Instead of classes, HRBAC model uses Sa_i and Sa_j to define the hierarchical relation of privilege inheritance from Role Sa_j to Role Sa_i; for example, Role "Professor" inherits all Role "Teaching Assistant" privileges in a grading system. To specify and enforce these relations in XACML, AC policy authors need to specify all the possibilities including direct and indirect relations between the classes or roles. In the worst case, it requires $O(n^2)$ number of (*2*) type of statements to describe the relations for *n* number of classes or roles in the policy. Further, there is no semantic support for checking the correctness (e.g., cyclic assignment) of the specifications.

B. *Specification of Separation of Duty Policies*

When required to enforce SOD polices to prevent conflicts of interest or to control business processes, the access state of the AC system is dynamically dictated by some system variables. For example, a SOD policy constrains a subject's privileges (action and object pairs) not to exceed a predefined number, so that no subject should be assigned to more than *k* privileges. Another SOD policy guarantees that no less than *k* number of subjects can perform all of a set of privileges (i.e., requires at least *k* number of subjects to perform all of them). To specify and enforce these SOD policies, XACML needs to maintain counters for monitoring the number of privileges consumed by each subject currently in the system. Thus, the XACML's *obligation* and *environment* elements are used to update and retrieve (read in) the external counters, respectively. And to compose SOD policies, statements in regular expression (*4*) are needed for referencing the environment variables (e.g., external counters) and statements in (*3*) are used to store updated variables. However, the challenge is to accurately maintain the constraint variables (the number *k* in our examples), because a subject's access request can be granted from more than one type (*4*) statement. And (*4*) may be encompassed in (*1*) (*2*) or (*3*) statement, which provides no syntax for maintaining the ARs between (*4*)s. For example, a subject may be granted access both from Role *X* and Role *Y* to an object, and there is no way to specify the fact that *X* inherits *Y*, therefore, the privilege *k* for this subject is counted twice (which is supposed to be once) from both *X* and *Y* attributes in the same access session. It is hard to ensure a SOD policy is implemented without errors in XACML, because even though ARs can be specified in the language, there are no syntactic and semantic supports for the correctness of the specification unless by custom application through functions in PEP or PDP.

III. Attribute Relations in Policy Machine

NIST has initiated a project in pursuit of a standardized access control mechanism referred to as the *Policy Machine* (*PM*) [7, 8]. *PM* is based on the principle that the separation of access control policies from mechanisms allows enforcement of multiple policies within a single, unified system so that access control rules from different authorities may be integrated with each other. The *PM* architecture is composed of the *Policy Server* (*PS*) for PDP and PEP. *PS* includes *PS* processes and the *PS* database, and the *General Policy Management System* (*GPMS*). *PS* receives subject requests and performs the authorization process by referencing information from the *PS* database; it then generates a Boolean value (*grant* or *deny*) as a result. *GPMS* is the interface for *PM* administrators to configure and compose policies and to manage the *PS* database. *PM* categorizes subjects (users), objects (resources), and their attributes into policy classes, and appropriately enforces subsets of the policies in response to a subject's access request. The following fundamental data sets for *PM* processing are stored in the *PS* database:

S: The set of *PM* subjects (users) under the *PM*'s control

SA: The set of subject attributes of *S*

OP: The set of operations (access rights) permitted by the *PM*

O: The set of objects under the *PM*'s control

OA: The set of object attributes of *O*
PC: The set of policy classes the *PM* is implementing

Figure 1 shows the *PS* database model. The arrows denotes assignment mapping functions from one data set to another. For example the *uua* function maps subjects in *S* to subject attritutes in *SA*.

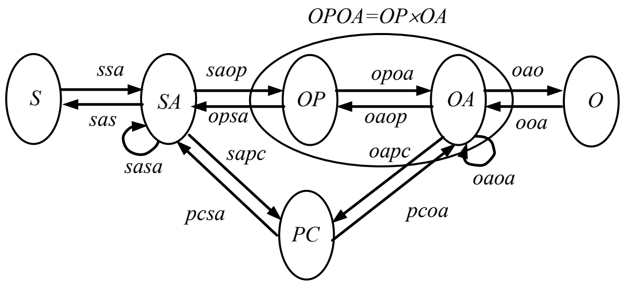

Figure 1. Set relations and functions of *PM*.

PM allows inheritance relations among subject attributes, and object attributes such that an element inherits the privileges from the elements that it is inherited from. The inheritance relation must not have cycles to be legitimate. A set of elements in an inheritance relation from one function to another function can be formally described by the union transitive closure of the two functions: $\cup_{y \in a(x)} b(y)$ denoted by the symbol "$x \rightarrow_a b$". For example, all inherited subject attributes SA_s of a subject *s* can be denoted by $s \rightarrow_{ssa} sasa$, and all inherited object attributes OA_o of an object *o* is $o \rightarrow_{ooa} oaoa$.

The atomic authorization process of *PM* is based on the above model and notation; the following formal definitions describe the *PS* authorization process:

For $s \in S$, $op \in OP$, $o \in O$, $pc \in PC$, $Grant\_instance\_of\_policy(s, op, o, pc)$ = True $\Leftrightarrow$ $\exists\ sa \in SA$ and $\exists\ oa \in OA$, such that

1) $sa \in (s \rightarrow_{ssa} sasa)$, *2)* $oa \in (o \rightarrow_{ooa} oaoa)$,

3) $sa \rightarrow_{op} oa$, *4)* $pc \in sa \rightarrow_{sapc} pcpc$, and

5) $pc \in oa \rightarrow_{oapcc} pcpc$.

PM only requires mapping the relations between elements to decide the permission of a subject's request. Through this mechanism, *PM* provides syntactic and semantic support of the AR specification.

IV. P ATTRIBUTE RELATIONS IN AC MODELS

This section demonstrates how *PM* specifies the MLS, HRBAC policies and SOD constraints by the AR assignments from the *PS* database and relation mapping functions. Subsection A demonstrates the implementation of simple BLBP model, and Subsection B shows the specification of SOD constraints as illustrated in Section II.

A. Specification of MLS and HRBAC Policies

PM can emulate MLS models by using its subject and object ARs. The subject security classes (labels) can be represented in *PM*'s *subject attributes*. Further, the objects security classes (labels) can be represented in *PM*'s *object attributes,* and the *subject attributes* are linked to the *object attributes* through *operations*. For example, to implement the Bell-La Padula model, *PM may* construct two sets of relations for each of the *subject attributes* and *object attributes* as shown in the simple example of Figure 2. The attribute with lower-case *r* in the attribute label of *subject attribute* and *object attribute* is for the read *privileges*, which are for the basic **confidential rule**. The attributes with lower-case *w* in the attribute label are for the **star** property of Bell-La Padula rules. In Figure 2, *TS* is *subject/object attribute* label for "Top Secret" *subject/object* class, *S* is for "Secret" class, and *C* is for "Confidential" class. *W* is for write privilege, *R* is for read privilege for each class (for example, *TSR* or *CW*). Each *subject/object* belonging to a class is assigned to both labels *w* and *r subject/object attribute* (for example, *TSr* and *TSw*). Assume that class *TS* dominates class *S*, and class *S* dominates class *C*; Subjects with the *Cw subject attribute* can write objects with the *object attribute Cw*, *Sw* and *TSw*. *Sw* can write *Sw* and *TSw*. *TSw* can only write *TSw*. *TSr* can read *TSr*, *Sr*, and *Cr*. *Sr* can read *Sr* and *Cr*. *Cr* can only read *Cr*. Note that a *subject/object* must be assigned to the same *r* and *w* group of *subject/object attributes* (*TS*, *S* or *C)*. For example, a subject should be assigned to the *Cw subject attribute* if she was assigned to the *Cr subject attribute* and vice versa.

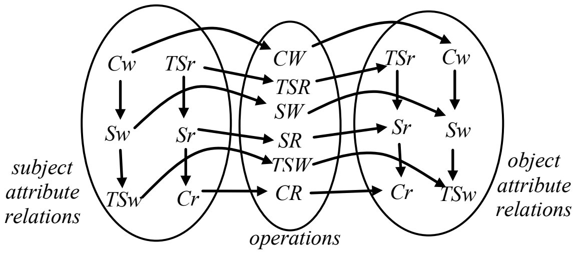

Figure 2. Simple Bell-La Padula Implementation.

Similar to BLBP models, the hierarchy of privilege inheritance for HRBAC can be directly specified by the *subject attributes* of *PM*, such that if *subject attribute x* dominates *subject attribute y*, then subject with role *x* inherits all the access privilege of subjects with role *y*. Figure 3 shows example attribute assignments of MLS and HRBAC of a *PM* system state. As the relation need only be assigned to directly related attributes, it only requires O(*n*) relation assignments if there are *n* classes for BLPA, or role inheritance relations for HRBAC. Thus the complexity is many times more efficient compared to O(n^2) assignment statements in Section II *A*.

Note that in this paper, we only focus on the efficiency and accuracy in **specifying** the AR required AC models and constraints. The process complexity (efficiency) for the enforcement of these models and constraints is either inevitable (e.g., collecting all the ARs in SOD models such as the examples in the next Subsection B) or algorithm /application dependent, thus not discussed in this paper.

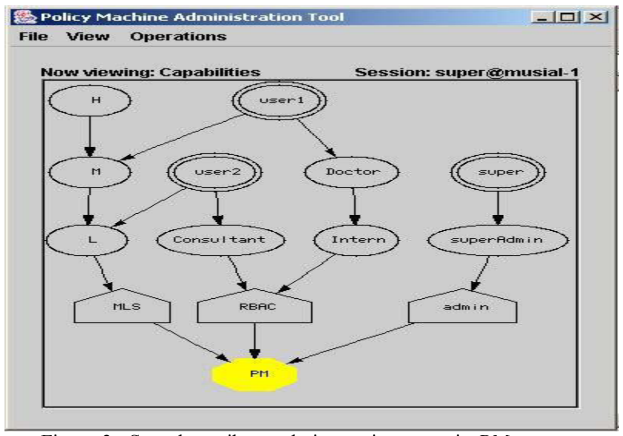

Figure 3. Sample attribute relation assignments in *PM*

B. *Specification of Separation of Duty Policies*

To enforce SOD, it is necessary to maintain all subject/object attribute relations for any subject or object if multiple attribute assignments are allowed. Hence, in addition to the basic relation mapping functions (in figure 1), to retrieve current mappings of ARs in the system, the function *sa_opoa(sa)* returns all (*op, oa)* pairs mapped to the *sa*.

For example, a SOD constraint specifies that no subject should be assigned to more than *k* privileges of a given set. Note that when *k* =1, this policy is a Privilege to Privilege Conflicts Policy (PPC), i.e. a set of privileges ($OP \times OA$) should not be assigned to the same subject. *PM* implements this policy by calculating the number of subject attributes the requesting subject is dominating or inheriting associated with the constrained privileges, and the number cannot exceed *k*. The rule is formally specified as:

$SoD_{PM} = \langle\, OPOA,\ k \,\rangle,\ OPOA = \{(op_1,\ oa_1),\ldots\ldots (op_n, oa_n)\},\ 1 \le k \le |OPOA|$, and

$\forall s \in S\ (\,|(\cap_{sasa(sa \in ssa(s))}\ sa\_opoa(sa)\,) \cap OPOA| \le k\,)$

Tuple SoD_{PM} contains the set of restricted privileges *OPOA,* and limited number of privileges *k*. $\cap$ used in $sa\_opoa(sa) \cap OPOA$ is because a subject may be assigned to duplicated privileges through different ARs. The example shows the SOD rule specifications by the *PM*'s standard *PS* functions based on the ARs. Without these functions, the complexity in specification is nontrivial.

V. Related Work

[9] proposed a Flexible Access Control Model (FACM), which provides user-friendly notation and presentation of ARs and constraints. However, the main usage of the graph representation is to help in the specification, design, rather than as a pure computational model, unlike *PM,* which provides computational functions in the *PS* server, and allows policy authors to specify AC rules by directly mapping ARs into rules semantic.

[10] proposed a Logical Framework for Reasoning about Access Control Models (ACMP) based on the C-Datalog program, which provide a precise mathematical foundation for reasoning about ARs. However, in addition to its logical programs are not being intuitive to most users, ACMP does not provide views of access instance and relations between attributes, unlike *PM*, which allows administrators to check/filter the relations at the point of view of any selected access element. This capability otherwise requires traceing through AC rules, and it is hard to achieve with the increased number of entries in the ACMP program.

VI. Conclusion

The flexibility and expressiveness of XACML make it complex to work directly with some AC mechanisms. Specifying ARs in XACML calls for completely specified relations for each and every directly or indirectly related attribute, thus produces a highly verbose document even if the actual policy rules are trivial. Because, *PM* is not a language, it is free from the syntactic and semantic complexity of a language. When describing hierarchical relations between attributes or policies, *PM* only requires adding links between them, therefore, avoiding the time delays due to the sequence of overhead algorithms. In supporting the enforcement of SOD policy constraint rules, *PM* provides an infrastructure that allows the efficient specification of rules to collect the attributes for the policy. *PM* also has a WYSIWYG graphic user interface (Figure 3) that visually aids in the management of policy documents. This feature is especially important when adding and deleting rules in the AC policies.

References

[1] Bell D.E. and Lapadula L. J., "Secure Computer Systems: Mathematical Foundations and Model," M74-244, MITRE Corp., Bedford, Mass., 1973 (also available as DTIC AS-771543).

[2] Biba K. J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division (also MTR3153, MITRE Corp.), Bedford, Mass., April 1977.

[3] NCSC, "Trusted Computer System Evaluation Criteria," National Computer Security Center, 1985.

[4] Ferraiolo et al, "Role-Based Access Control (RBAC): Features and Motivations," Proc. of the 11th Annual Conference on Computer Security Applications, Calif, pp 241-248, 1995.

[5] Jajodia et al, "A logical language for expressing authorizations," Proc. IEEE Symp. On Research in Security and Privacy, Oakland, Calif, pp 31- 42, May 1997.

[6] OASIS, "Extensible ACCess Control Markup Language (XACML), TC", www.oasisopen.org/committes/tc_home.php?wg_abbrev=xacml

[7] Hu et al, "The Policy Machine For Security Policy Management," Proc. ICCS Conference, San Francisco, 2001.

[8] Ferraiolo et al, "Composing and Combining Policies under the PolicyMachine," ACM SACMAT, 2005.

[9] Coetzee M. and Eloff J. H. P., "Virtual Enterprise Access Control Requirements," Proc. of SAICSIT, pp. 285-294, 2003.

[10] Bertino et al, "A Logical Framework for Reasoning about Access Control Models," ACM Transactions on Information and System Security, Vol. 6, No. 1, pp 71–127, February, 2003.