

A Formal Authorization Policy Model

Jie Dai
Computer Science Department
Central Michigan University
jdai@cps.cmich.edu

Jim Alves-Foss
Computer Science Department
University of Idaho
jimaf@cs.uidaho.edu

Abstract

This paper presents a formal model that interprets authorization policy behaviors. The model establishes a connection of applying authorization policies on an administration domain with dissecting the domain into the authorized, denied, and undefined divisions. This connection enables us to analyze authorization policy development problems such as policy merge, inconsistency, ambiguity, and redundancy by examining the domain elements mapped into each of the divisions. In addition, three distinct authorization values are assigned to the divisions based on the permission of access control, and are used to calculate partition index of each rule or policy for measurement purpose. The entire measurable model provides a method to analyze and develop correct and conflict free authorization policies.

Keywords: formal modeling, access control policies

1 Introduction

Among various security policies, authorization policies are a core requirement for a computer or network system to prevent illegal usages. In the system, unauthorized users should not be allowed to access any resources by any means. This is conventionally realized by using authentication and access control mechanisms, which are jointly treated as authorization mechanisms. These mechanisms are actually implementations of authorization policies, and therefore, to some extent, we can say that the security of a computer or network system depends on the quality of its authorization policy establishment, which can be evaluated by the consistency, ambiguity, and redundancy of specification, and the correctness of implementation.

Unfortunately, the establishment of high-quality authorization policies is a complicated issue. Unlike other procedural functionalities of system or application software, authorization is characterized by complex logic deduction. It has to deal with not only general logic problems such as negation, hierarchy, and default, but also authorization specific logics such

as policy merge, policy incorporation, and delegation. Currently, researches on these logic problems are separated. For instances, scientists in knowledge representation arena focus on the development of general logics including non-monotonic logic [2], descriptive logic [14], and modal logic [8], to solve the description and reasoning of negation, hierarchy and default problems; scientists in *logic programming* (LP) [23] focus on a generic formalism and language for automatic handling of the logic problems by a machine; and computer security scientists focus on establishing models for security properties [10, 6, 24, 15] and approaches dealing with delegations [1, 7, 20] and conflicts [19, 4]. However, in addition to all of the efforts in each of the related areas, we still need an integral theoretical guidance for the software development of authorization policies from specification to executable programs.

This paper presents our work of establishing a formal model in order to provide a measurable method for developing authorization policies correctly and consistently. This work is conducted under the assumption that authorization policies can be safely separated from and integrated into other functionalities of system or application software and treated independently in a logical way. Interested readers who are concerned about how to realize the separation (resp. integration) may refer to [12, 11].

2 Safe policy development

A logical model providing guidance to incrementally build up and safely maintain authorization policies for a software system should be able to interpret access control behaviors of the policies in all forms, i.e. independent of their expressions. In our model, a policy is viewed as a set of partitions (combinations of the authorized, denied, and gray divisions) of an administration domain by disjoint access operations. When multiple policies are added or old policies are altered, new partitions of the administration domain are generated by policy merge. In addition, consistency, ambiguity and redundancy of various policies should be able to be analyzed before merge

by examining the elements mapped to the authorized, denied and gray divisions of a domain and help determine what merge strategies should be chosen in order to reduce redundancy and avoid conflicts and ambiguity. Thirdly, we found that a partition index value would be helpful for us to know the effects of each of the policies and predict the correctness of their merge in an engineering development process. For instance, if policy P_1 has partition index value of 0.2, and P_2 has 0.3, then the index value will be no more than 0.2 for the intersection of the two policies, and at least 0.3 for the union of the two policies. In this paper, we only showed the calculation of such index value as permission of access control at the rule level of our model, but it is similar to calculate it at the policy level except for that a policy may have different value for each access operation involved.

An engineering based policy development approach is developed in [13], which views a policy development process as the hierarchical composition of the policies from specification to implementation by people including administrators, users and programmers. The top-level of the policies corresponds to policy specifications. Through some middle levels of refinement and transformation, the bottom-level of the policies corresponds to executable programs. A safe policy development process is that different policies at the same level should be conflict free, non-ambiguous and non-redundant and lower level policies conform to the higher level policies, i.e. authorized division of the lower level policies should be included in that of the higher level policies. A case study has been shown in [11] introducing how to use our model to help develop and evaluate the established authorization policies for a certificate based distributed file system.

3 Formal model for authorization policies

In computer science, systems of logic such as *First Order Logic* (FOL), and LP are usually used for software systems to conduct formal specification and verification, as well as knowledge representation and automatic reasoning. In this paper, our problem domain of the treated software systems is focused on authorization, where a set of authorization policies can be analyzed as specification and developed as executable programs. Due to the complexity of the authorization behaviors in a computer or network system, where policy merge and incorporation across autonomous domains are involved, it is desirable to establish a unified interpretation on authorization policy specifications and programs in order to analyze

their behaviors and evaluate the development. Therefore, we have established a formal model which has the following major characteristics:

Generality. The model provides a unified interpretation of rule combinations and policy merge for authorization specifications and programs. It can be used to interpret policy problems such as inconsistency, ambiguity, redundancy, policy merge either within a single administration domain or across domains.

Computability. In order to make authorization a scientific subject to be studied and provide a means to facilitate the evaluation of correct development of authorization programs, our model provides a measurement for authorization policies based on their permission of access control.

3.1 A computable authorization logic model

The formal authorization logical model is comprised of three definitions on *administration domain division*, *authorization rule*, and *partition value*. It is defined on the basic data types including a set of subjects, S , a set of objects, O , an administration domain that is the product of the subject and object sets, $D = S \times O$, and a set of access operations, X .

DEFINITION 1 (Administration Domain Division)
The administration domain division is a sub-domain of administration domain (D). It contains all the elements of D that have been mapped by an access $x \in X$ to the type of authorized access (A_d), denied access (N_d), or undefined access (G_d), which has authorization value of 1, $\frac{1}{2}$ or 0 respectively. If an element of $(s, o) \in D$ ($s \in S$, and $o \in O$) is mapped to the authorized division A_d , then s is authorized to perform access x on object o ; If it is mapped into N_d the access is denied; If it is mapped into G_d , it is not specified either authorizing or denying access.

Therefore, the union of the authorized, denied, and undefined divisions for an access operation would be the administration domain D , i.e. $A_d \cup N_d \cup G_d = D$. The G_d is needed when policies in two different domains merge with each other in order to allow interoperation between two federated systems, in which case access rights are not specified for the entire joint domain by original policies. The authorization values assigned to the divisions indicate that the permission of access control for A_d is more than that of G_d , which is more than that of N_d .

DEFINITION 2 (Authorization Rule) *The authorization rule R is a function Γ taking an access operation $x \in X$ and the applied administration domain D to generate a domain partition, I , specifying the placement of elements of D into the divisions of A_d , G_d and N_d . The possible types of partition are: A (authorization), G (gray), N (deny), AG (authorization-gray), AN (authorization-deny), GN (gray-deny), and AGN (authorization-gray-deny). Partition A (G or D) means that all elements of D fall into a single authorized (resp. undefined or denied) division; partition AG (AN or GN) means that some elements of D fall into authorized division and others into the gray division (resp. authorized and denied, or gray and denied); partition AGN means that there are elements falling into all three authorized, gray and denied divisions on D . Authorization rule can be represented with the following formula:*

$$R \equiv \Gamma(x, D), \Gamma : X \times D \rightarrow I, \text{ where}$$

$$I = \{A, G, N, AG, AN, GN, AGN\}$$

The domain partition of a rule also indicates the classification (A , G , N , AG , AN , GN , or AGN) for the rule. For instances, the A rule authorizes all subjects an access to all objects on an administration domain. Such rules are not needed for access control in a computer system, as they define no restrictions. The GN rule defines a restriction that some subjects are denied an access to some objects and others are undefined. This happens when people want to specify operation restriction to a few subjects, such as users with lower classification level are not allowed to read higher classified resources. The AGN rule defines restriction that some subjects are authorized, undefined or denied an access to some objects. This is the case when people use both authorization and explicit negation in policy specification. AG , GN , and AGN rules are called partially defined rules; A , N , and AN rules are called complete rules; and G rules are called empty rules.

DEFINITION 3 (Partition Value) *The domain partition I is assigned an authorization value v , which can be calculated based upon division values. This partition value measures the corresponding rule in terms of permission on access control. The calculation is summarized in Figure 1, where the norm is calculated as the number of the elements in the domain or divisions in this paper.*

With the partition value, we may compare two rules or one rule in different development stages in terms of

their permission of access control, and therefore, provide an approach in testing and analyzing the derived rules.

3.2 Rule combination

When no authorization rules have been applied, an administration domain has a single gray division, where more than one rule can be applied based on security requirements. Each new rule or modification of old rules will result in a different partition of the domain by merging the old partition with the partition of the new rule or modification. There are varieties of *partition merge* (Definition 4) depending on how rules are combined. Commonly seen *rule combinations* (Definition 5) are intersection (\cap), union (\cup) priority aggregation [17], and product [5, 21], each of which has its own distinct way to generate a new partition. In order to simplify the illustration, only partition merges by rule intersection and union are introduced in this paper.

DEFINITION 4 (Partition Merge). *The partition merge of a domain is a remapping of all elements of the domain to its A_d , G_d and N_d divisions by a rule combination.*

DEFINITION 5 (Rule Combination). *The rule combination is a symmetric operation denoted with an infix operator (normally C). The combination of two rules (R_1CR_2) in a domain D generates a new rule R_3 with partition (I_3) of D by merging the two partitions (I_1 and I_2) of the combining rules with respect to an access x .*

Figures 2 and 3 specify how an element of D is remapped by rule intersection and union. For instance, if (s, o) belongs to the authorized division A_d by R_1 but the gray division G_d by R_2 , then (s, o) belongs to the gray division after R_1 intersects with R_2 and the authorized division after R_1 unions with R_2 . From Definition 2, one knows that there are seven basic types of authorization rules (A , G , N , AG , GN , AN , and AGN) which in total can generate twenty-eight new types of rules with their partitions that can be calculated by Definitions 4 and 5 and authorization value by Definition 1. For instance, $(AGN)_1C(AGN)_2$ is a new rule derived by combing a rule of type AGN with another rule of the same type.

In order to justify our definition on partition merge by rule intersection and union, we thereby define the completeness and disjointness of the partition merge and consistent rule.

$$\begin{aligned}
v &= 1, \text{ when } I \text{ is } A \\
v &= \frac{1}{2}, \text{ when } I \text{ is } G \\
v &= 0, \text{ when } I \text{ is } N \\
v &= \frac{\|G_d\|}{\|D\|} \times \frac{1}{2} + \frac{\|N_d\|}{\|D\|} \times 0 = \frac{\|G_d\|}{\|2D\|}, \text{ when } I \text{ is } GN \\
v &= \frac{\|A_d\|}{\|D\|} \times 1 + \frac{\|G_d\|}{\|D\|} \times \frac{1}{2} = \frac{\|A_d\|}{\|D\|} + \frac{\|G_d\|}{\|2D\|}, \text{ when } I \text{ is } AG \\
v &= \frac{\|A_d\|}{\|D\|} \times 1 + \frac{\|N_d\|}{\|D\|} \times 0 = \frac{\|A_d\|}{\|D\|}, \text{ when } I \text{ is } AN \\
v &= \frac{\|A_d\|}{\|D\|} \times 1 + \frac{\|G_d\|}{\|D\|} \times \frac{1}{2} + \frac{\|N_d\|}{\|D\|} \times 0 = \frac{\|A_d\|}{\|D\|} + \frac{\|G_d\|}{\|2D\|}, \text{ when } I \text{ is } AGN
\end{aligned}$$

Figure 1 Calculation of authorization value for partition index

For an element $(s, o) \in D$, if

$(s, o) \in (A_d)_1$ and $(s, o) \in (A_d)_2$, then $(s, o) \in (A_d)_3$
 $(s, o) \in (A_d)_1$ and $(s, o) \in (G_d)_2$, then $(s, o) \in (G_d)_3$
 $(s, o) \in (A_d)_1$ and $(s, o) \in (N_d)_2$, then $(s, o) \in (N_d)_3$
 $(s, o) \in (G_d)_1$ and $(s, o) \in (A_d)_2$ or $(G_d)_2$, then $(s, o) \in (G_d)_3$
 $(s, o) \in (G_d)_1$ and $(s, o) \in (N_d)_2$, then $(s, o) \in (N_d)_3$
 $(s, o) \in (N_d)_1$ and $(s, o) \in (A_d)_2, (G_d)_2, \text{ or } (N_d)_2$, then $(s, o) \in (N_d)_3$

$(A_d)_1, (G_d)_1$ and $(N_d)_1$ are the divisions of D by partition I_1 of R_1
 $(A_d)_2, (G_d)_2$ and $(N_d)_2$ are the divisions of D by partition I_2 of R_2
 $(A_d)_3, (G_d)_3$ and $(N_d)_3$ are the divisions of D by partition I_3 of R_3

Figure 2 Partition merge by rule intersection

For an element $(s, o) \in D$, if

$(s, o) \in (A_d)_1$ and $(s, o) \in (A_d)_2, (G_d)_2, \text{ or } (N_d)_2$, then $(s, o) \in (A_d)_3$
 $(s, o) \in (G_d)_1$ and $(s, o) \in (A_d)_2$, then $(s, o) \in (A_d)_3$
 $(s, o) \in (G_d)_1$ and $(s, o) \in (G_d)_2$ or $(N_d)_2$, then $(s, o) \in (G_d)_3$
 $(s, o) \in (N_d)_1$ and $(s, o) \in (A_d)_2$, then $(s, o) \in (A_d)_3$
 $(s, o) \in (N_d)_1$ and $(s, o) \in (G_d)_2$, then $(s, o) \in (G_d)_3$
 $(s, o) \in (N_d)_1$ and $(s, o) \in (N_d)_2$, then $(s, o) \in (N_d)_3$

$(A_d)_1, (G_d)_1$ and $(N_d)_1$ are the divisions of D by partition I_1 of R_1
 $(A_d)_2, (G_d)_2$ and $(N_d)_2$ are the divisions of D by partition I_2 of R_2
 $(A_d)_3, (G_d)_3$ and $(N_d)_3$ are the divisions of D by partition I_3 of R_3

Figure 3 Partition merge by rule union

DEFINITION 6 (Completeness of Partition Merge)
A partition merge is complete if any element of the

applied administration domain after merge belongs to $A_d, G_d, \text{ or } N_d$ of the domain.

DEFINITION 7 (Disjointness of Partition Merge) A partition merge is disjoint if the A_d , G_d , and N_d of the applied administration domain after merge are disjoint.

DEFINITION 8 (Consistent Rule) A consistent rule is a rule that an access cannot be both authorized and denied, both authorized and gray, or both denied and gray on a domain.

In other words, the A_d , G_d , and N_d divisions of a consistent rule on an applied domain with respect to an access are disjoint. It is generally assumed that each rule that is a basic unit of a policy is defined consistent. Based on the assumption, we can prove that the defined partition merge by rule intersection and union is complete and disjoint.

PROPERTY 9. The partition merge by rule intersection or rule union is complete and disjoint.

PROOF. From Definitions 1 and 2, an administration domain D can be dissected into the maximum of three divisions, A_d , G_d , and N_d by a rule. In Definition 5, all elements of D in rule intersection and union are remapped by the rules defined in Figures 2 and 3, where each element is definitely remapped to a new division of A_d , G_d , or N_d only. Therefore, the partition merge defined is complete. In addition, based on the assumption that each authorization rule applied on D is consistent and there is no overlapping in remapping by Definition 5, the partition merge is disjoint.

With the theory on rule combination established, we may interpret policy merge with our authorization logical model.

4 Policy merge

In current computer and network systems, multiple authorization policies (Definition 10) usually coexist and interact with each other. This has caused inconsistency, redundancy and ambiguity in simple aggregation of different policies, and makes it indispensable for a mechanism, policy merge, to combine different policies in a safe way.

Policy merge is a strategy that coordinates multiple individual policies according to user's requirements. The strategy can be implemented by using meta-policy [18] or directly modifying the old policies. Policy merge can be applied into two cases: a) multiple policies are required within a single administration domain and b) policies across different

administration domains need to incorporate (or called interoperation between federated systems).

Within a single administration domain of case a), there is a unified classification of users and resources, therefore, policy merge and evaluation can be conducted through the rule combination introduced previously except for that multiple access operations are involved (Definition 11). In case b) however, there may exist different standards for user clearance and resource classification, as a result, our solution requires one conduct policy merge by converting the problems of multiple autonomous domains to those of a single domain (Definition 12) and then treat them similarly as in case a).

DEFINITION 10 (Authorization Policy) The authorization policy P is a set of disjoint access x_i and associated rule r_i pairs defined on an administration domain. In the following formalization, X is the set of accesses that are involved in a set of rules R constituting the policy.

$$P \equiv \prod_{i=1}^n (x_i, r_i), \text{ where } x_i \in X, \text{ and } r_i \in R$$

DEFINITION 11 (Single-domain Policy merge) The merge of two authorization policies on a same domain generates a new policy by a merge strategy. In the following formula, P_1 and P_2 stand for two different policies. An infix operator M denotes a merge strategy, which is a symmetric operation applied on P_1 and P_2 , and can be decomposed into an access operation merge M_x and a rule merge M_r . X_1 (or X_2) is the set of accesses and R_1 (or R_2) the set of associated rules defined by P_1 (or P_2). x_p (or x_q) is an instance of X_1 (or X_2), and r_p (or r_q) an instance of R_1 (or R_2).

$$P_1 M P_2 \equiv \prod (X_1 M_x X_2, R_1 M_r R_2), \text{ where}$$

$$P_1 \equiv \prod_{p=1}^m (x_p, r_p), x_p \in X_1, r_p \in R_1 \text{ and}$$

$$P_2 \equiv \prod_{q=1}^n (x_q, r_q), x_q \in X_2, r_q \in R_2.$$

M is a set of user defined merging strategies including policy union, intersection, product, and priority. A different M has different M_x and M_r based on user's security requirements. For instance, in priority merging [3, 19, 4] there are various strategies based on recency, specificity, authorship, and distance between the policies. We argue that it is the differences in M that have induced various versions of

policy merge and solutions to the problem of policy conflicts, which can be solved during the runtime or at the policy specification time. In the following, an example shows how to use our model to analyze authorization policies and their merge by policy intersection and union.

EXAMPLE 1. There are two policies P_1 and P_2 . Access x_i (or x_j) is associated with rule r_1 (or r_2) defined by P_1 (or P_2). Policy merges, M , of P_1 and P_2 by intersection \cap or union \cup are shown in Figure 4, where merge results are presented separately by the equality of x_i and x_j . The entries in column Mx show the involved access operations after merge and ϕ in M_x means that none of the original accesses is involved; and the entries in the column M_r show the new rules applied after merge, and ϕ in M_r denotes that none of the original rules would apply in anyway in the merged policy.

M		M_x	M_r or C
\cap	$x_i = x_j$	x_i or x_j	$r_1 \cap r_2$
	$x_i \neq x_j$	ϕ	ϕ
\cup	$x_i = x_j$	x_i or x_j	$r_1 \cup r_2$
	$x_i \neq x_j$	x_i and x_j	r_1 and r_2

Figure 4 Policy intersection and union

The secure interoperation of two federated systems is realized by safe policy merge in original distinct systems. This can be treated as applying additional interoperation policies (denoted as H) to each of the related administration domains in a proper way, as direct modification of original policies in each of the domains can be seen as adding a modification policy to the policies. It is a common practice that H should not affect the authorization and deny of the original domains. This is the *principle of autonomy* that has been summarized by Gong and Qian [16]. The added rules of H may permit accesses to the resources of one domain to the principals or subjects in another domain; or deny such accesses. Although only H directly interacts with the original policies of a domain, the policies in another domain also matter for the security of the interoperation.

DEFINITION 12 (Policy Merge Across Domains) Policies P_1 and P_2 of two domains D_1 and D_2 are merged through the additional interoperation policy H . The merge generates two new policies (P'_1) and (P'_2) that must be safe on a new joint domain $D = D_1 \cup D_2$. All original rights of P_1 (or P_2)

on D_2 (or D_1) are gray. H is defined on D . The safe P'_1 (or P'_2) is derived from the merge of P_1 , H and P_2 on domain D . In the following formula, M_1 (M_2) is the policy merge strategy used to generate P'_1 (P'_2):

$$P'_1 = (P_1 M_1 H) M_1 P_2, \text{ and } P'_2 = (P_2 M_2 H) M_2 P_1$$

The secure interoperation of federated systems is nontrivial, as problems such as how to join the original domains still exist. However, with our policy model, we can represent and interpret the problems with searching for a policy merge operator along with additional interoperation policies which can produce safe dissection of the joint domain. In the following section, we specify the properties of safe policy merge strategies.

5 Inconsistency, ambiguity, and redundancy

Inconsistency (Definition 13), ambiguity (Definition 14) and redundancy (Definition 15) of authorization policies are the primary problems that affect quality of the policies and security of a system. However, these problems are common in a multiple-policy environment, where partition overlapping of different policies on an administration domain and joining of domains occur. In order to explore the nature of the problems, hereby, we interpret them with our formal authorization policy model, and then in the next section, briefly introduce our solution to these problems by applying the established theory in the policy development process.

The following definitions of consistency, ambiguity, and redundancy are made under the assumption that each policy is separately defined complete and disjoint on an administration domain.

DEFINITION 13 (Consistency of Two Policies). Two policies P_1 and P_2 are consistent if the intersection of the authorized division (A_d) of policy P_1 (or P_2) with the denied division (N_d) of policy P_2 (or P_1) with respect to any access on a domain is empty. The consistency can be formalized as the following, and the subscript i stands for which policy the division belongs to.

$$(A_d)_1 \cap (N_d)_2 = \emptyset \text{ and } (A_d)_2 \cap (N_d)_1 = \emptyset, \text{ where } (A_d)_i \neq \emptyset, (N_d)_i \neq \emptyset, \text{ and } i = 1 \text{ or } 2$$

DEFINITION 14 (Ambiguity of Two Policies). Two policies P_1 and P_2 are ambiguous if the authorized (A_d) or denied (N_d) division of policy P_1 (P_2) overlaps

with the gray division (G_d) of policy P_2 (resp. P_1) with respect to any access on a domain D . The ambiguity can be formalized as the following, and the subscript j denotes four distinct cases of division intersection .

$$(A_d)_1 \cap (G_d)_2 = \chi_1, \text{ or } (A_d)_2 \cap (G_d)_1 = \chi_2, \text{ or } (N_d)_1 \cap (G_d)_2 = \chi_3, \text{ or } (N_d)_2 \cap (G_d)_1 = \chi_4, \text{ where } \exists \chi_j, \chi_j \subset D, \text{ and } j=1,2,3, \text{ or } 4$$

DEFINITION 15 (Redundancy of Two Policies). Two policies P_1 and P_2 are redundant if the authorized (A_d) or the denied (N_d) division of policy P_1 (P_2) is a subset of the authorized or the denied division of policy P_2 (resp. P_1) with respect to any access on a domain. The redundancy can be formalized as the following, and the subscript i stands for which policy the division belongs to.

$$(A_d)_1 \subseteq (A_d)_2, \text{ or } (A_d)_2 \subseteq (A_d)_1, \text{ or } (N_d)_1 \subseteq (N_d)_2, \text{ or } (N_d)_2 \subseteq (N_d)_1, \text{ where } (A_d)_i \neq \emptyset \text{ and } (N_d)_i \neq \emptyset, \text{ and } i=1 \text{ or } 2.$$

The following example shows the meaning of the definitions.

EXAMPLE 2. On an administration domain of $D = \{a, b, c\}$ (each domain element is now denoted with a single letter to simplify the illustration), policy P_1 has a partition of $(A_d)_1 = \{a\}$, $(G_d)_1 = \{b\}$, and $(N_d)_1 = \{c\}$, policy P_2 has a partition of $(A_d)_2 = \{a, b, c\}$, $(G_d)_2 = \{\}$, and $(N_d)_2 = \{\}$. P_1 and P_2 are conflict to each other for $(A_d)_2 \cap (N_d)_1 = \{c\}$, ambiguous for $(A_d)_2 \cap (G_d)_1 = \{b\}$, and redundant for $(A_d)_1 \subseteq (A_d)_2$.

6 Related work

Modeling authorization policies is not an innovative idea. The formalisms give mathematical generalization and representation of the problem domain and therefore provide rigorous ways for people to analyze it. Bell [3] presented a model for the “multi-policy machine” at runtime, where he used a state machine to describe the authorization system and viewed a policy as a function taking as parameters of the request, system state, and decision to get a value v of authorization decision. His model described runtime policy enforcement but did not have mechanism to analyze static policies. Similarly, McLean [21] used a state machine to describe access control policies at runtime. Bidan et al [5] proposed a more restrictive alternative to Bell’s model and identified the set of combination operators enabling security officers to combine security policies in a controlled and secure way. However, they have not provided semantics of the proposed combiners,

including union, intersection, product of classifications, and the denial of classification.

In [25], Woo and Lam defined an authorization policy a tuple of $(P+, P-, N+, N-)$ over a set of subjects S , and a set of objects O . There is no explanation on how to use their model to explain rule combination, policy merge, policy consistency, ambiguity, and redundancy in the literature. Additionally, their model is not computable.

Gong and Qian [16] have done fundamental research on the computational issues in secure interoperation. Based on their work, we applied our model and converted interoperation across different domains into a policy-merging problem of a single domain.

Cuppens et al [9] have presented a modal logic to describe security policies. The normative conflicts that they focused on refer to the contradiction that happens between obligatory policies and authorization policies. They provided a meta-strategy to solve the conflicts. This meta-strategy can be formalized with our model.

Research conducted by Lupu and Sloman [21] also show that authorization policy conflicts are closely related to the overlapping of subject and object domains. They have developed a domain nesting theory to explain modality conflicts. This informal theory can be represented with our logical model concerning the division of administration domains into authorized, gray, and denied.

7 Conclusion

This paper presented a formal authorization policy model and its interpretations for authorization policy behaviors. This fundamental research supported our authorization policy engineering [11] for generating correct and conflict free policies before they are integrated into a system. The model is computable in terms of restriction on access for each rule or policy. This is particularly valuable for an engineering process towards a goal of measurement. In our analysis regarding consistency, ambiguity, and redundancy for a nontrivial authorization logic program, called *Virtual*, and calculation of the restriction value for the rules and policies expressed by the logic program in its development process, the model helped us better understand the written policies and build up challenges and test cases to validate our policies. However, in order to apply the model in the real world, more work needs to be done. For instances, a tool based on the model needs to be built to compute the elements in each division of a partition and the index value, which may also be based on complexity

of the authorization logic rather than permission of access.

References

- [1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A calculus for access control in distributed systems," *ACM Transactions on Programming Languages and Systems*, 15(4): 706-734, Oct. 1993
- [2] G. Brewka, J. Dix, and K. Konolige, "Nonmonotonic Reasoning: An Overview," CSLI Publications, Center for the Study of Language and Information, Stanford, California, 1997.
- [3] D. E. Bell, "Modeling the Multipolicy Machine," *Proceedings of the New Security Paradigm Workshop*, pages 2-9, Aug. 1994.
- [4] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo, "A Logical Framework for Reasoning on Data Access Control Policies," *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, Los Alamitos, 1999, pp. 175-189.
- [5] Bidan, C., V. Issarny, "Dealing with Multi-Policy Security in Large Open Distributed Systems", *Proceedings of the 5th European Symposium on Research in Computer Security*, pp 51-66, September 1998.
- [6] D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified exposition and multics interpretation," *Tech. Report MTR-2997*, The MITRE Corporation, Bedford, MA, July 1975.
- [7] Matt Blaze, Joan Feigenbaum, Jack Lacy, "Decentralized Trust Management," *Proceedings of 1996 IEEE Conference on Privacy and Security*, Oakland, 1996.
- [8] B. F. Chellas, "Modal Logic: An Introduction," Cambridge University Press, Cambridge, UK, 1980.
- [9] F. Cuppens, L. Cholvy, C. Saurel, and J. Carrere, "Merging Security Policies: Analysis of a Practical Example," *Proceedings of the 11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
- [10] D. D. Clark, and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *IEEE Security and Privacy Symposium*, p. 184-194, April 1987.
- [11] J. Dai, "Logic Based Policy Engineering in Distributed Authorization," Dissertation, Computer Science Department, University of Idaho, Idaho, Nov. 2001.
- [12] J. Dai, and J. Alves-Foss, "Authorization Policy Engineering in the CBASS," Proc. of the *25th Anniversary Annual International Computer Software and Applications Conference (COMP.SAC 2001)* in Chicago, USA Oct 8-12, 2001.
- [13] J. Dai, and J. Alves-Foss, "Logic Based Authorization Policy Engineering," Proc. of *The 6th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, July 2002.
- [14] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, "Reasoning in Description Logics," *Principles of Knowledge Representation and Reasoning*, edited by G. Brewka, *Studies in Logic, Language and Information*, CLSI Publications, pp 193-238, 1996.
- [15] J. Goguen, and J. Meseguer, "Security Policies and Security Models," *Proceeding of 1982 IEEE Symposium on Security & Privacy*, Oakland, CA., IEEE Computer Society, April 1982.
- [16] L. Gong, and X. Qian, "Computational Issues in Secure Interoperation," *IEEE Transaction on Software Engineering*, vol. 22, No. 1, Jan 1996.
- [17] B. N. Grosf, "Courteous Logic Programs: Prioritized Conflict Handling for Rules," *IBM Research Report RC 20836*, Dec. 30 1997, revised from May 8 1997, available at: <http://www.research.ibm.com/rules/papers.html>
- [18] Hilary H. Hosmer. "Metapolicies II," *Proceedings of 15th National Computer Security Conference*, pages 369--378, Baltimore, MD, October 1992.
- [19] S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logic Language for Expressing Authorizations," *Proceedings of 1997 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 4-7, Oakland, California, page 31-42.
- [20] N. Li, J. Feigenbaum, and B. N. Grosf. "A Logic-Based Knowledge Representation for Authorization with Delegation (Extended Abstract)," *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, June 1999, pp. 162-174, Full paper available as IBM Research Report RC21492.
- [21] E. Lupu and M. Sloman, "Conflict Analysis for Management Policies," *Fifth IFIP/IEEE International Symposium on Integrated Network Management IM'97*, San-Diego, May 1997
- [22] J. McLean, "The algebra of security," *Proceedings of the 9th IEEE Symposium on Research in Security and Privacy*, pages 2--7, Oakland, California, April 18-21 1988.
- [23] U. Nilsson, and J. Maluszynski, "Logic, Programming and Prolog," 2nd edition, John Wiley & Sons Ltd., 1995, Available at <http://www.ida.liu.se/~ulfnl/lpp/copyright.html>.
- [24] I. Sutherland, "A model of information," *Proceedings of Ninth Nat'l Computer Security Conference*, Faithersburg, Md., 1986.
- [25] T. Woo, and S. S. Lam, "Authorization in Distributed System: A Formal Approach", *Proceedings of IEEE Symposium of Research in Security and Privacy*, Oakland, CA, May1992.