Policies, Models, and Languages for Access Control

Sabrina De Capitani di Vimercati[1], Pierangela Samarati[1], and Sushil Jajodia[2]

[1] DTI - Università di Milano, 26013 Crema - Italy
{decapita, samarati}@dti.unimi.it
[2] George Mason University, Fairfax, VA 22030-4444
jajodia@gmu.edu

Abstract. Access control is the process of mediating every request to data and services maintained by a system and determining whether the request should be granted or denied. Expressiveness and flexibility are top requirements for an access control system together with, and usually in conflict with, simplicity and efficiency. In this paper, we discuss the main desiderata for access control systems and illustrate the main characteristics of access control solutions.

1 Introduction

One of the most important features of today's systems is the protection of their resources (i.e., data and services) against unauthorized disclosure (*secrecy*) and intentional or accidental unauthorized changes (*integrity*), while at the same time ensuring their accessibility by authorized users whenever needed (*no denials-of-service*) [30]. Considerable effort is being devoted to addressing various aspects of secrecy, integrity, and availability. However, historically, confidentiality has received the most attention, probably because of its importance in military and government applications. As a result, significant research has focused on achieving more expressive and powerful *access control systems*. Access control is the act of ensuring that a user accesses only what she is authorized to and no more. The development of an access control system requires the definition of the regulations according to which access is to be controlled and their implementation as functions executable by a computer system. This development process is usually carried out with a multi-phase approach based on the concepts of *security policy*, *security model*, and *security mechanism*. A policy defines the (high-level) rules according to which access control must be regulated. An access control model provides a *formal* representation of the access control security policy and its working. The formalization allows the proof of properties on the security provided by the access control system being designed [21]. A security mechanism defines the low level (software and hardware) functions that implement the controls imposed by the policy and formally stated in the model.

The traditional access control models used for describing the enforcement of confidentiality are based on the definition of access control rules, called *authoriza-*

S. Bhalla (Ed.): DNIS 2005, LNCS 3433, 225–237, 2005.

tions, which are of the form ⟨subject, object, operation⟩. These authorizations specify which operations can be performed on objects by which subjects. However, in today's systems the definition of an access control model is complicated by the need to formally represent complex policies, where access decisions depend on the application of different rules coming, for example, from laws practices, and organizational regulations. A security policy must then combine all the different regulations to be enforced [34] and, in addition, must consider all possible additional threats due to the use of computer systems. Given the complexity of the scenario, the simple authorization triple ⟨subject, object, operation⟩ is no more sufficient.

The remainder of this paper is organized as follows. Section 2 discusses the main features supported by modern access control policies and models. Section 3 presents recent approaches in the area of access control languages. Finally, Section 4 concludes the paper.

2 Policies and Models for Access Control

The access control service provided by the computer system should be expressive and flexible enough to accommodate all the different requirements that may need to be expressed, while at the same time be simple both in terms of use (so that specifications can be kept under control) and implementation (so to allow for its verification). In the following, we discuss the main features that an access control service should support.

2.1 Conditions and Groups

Even early approaches to authorization specifications allowed *conditions* to be associated with authorizations to restrict their validity. Conditions can make the authorization validity dependent on the satisfaction of some system predicates (*system-dependent* conditions) like the time or location of access. For instance, a condition can be associated with the bank-clerks' authorization to access accounts, restricting its application only from machines within the bank building and in working hours. Conditions can also constraint access depending on the content of objects on which the authorization is defined (*content-dependent* conditions). Content-dependent conditions can be used simply as way to determine whether or not an access to the object should be granted or as way to restrict the portion of the object that can be accessed (e.g., a subset of the tuples in a relation). This latter option is useful when the authorization object has a coarser granularity than the one supported by the data model [11]. Other possible conditions that can be enforced can make an access decision depend on accesses previously executed (*history dependent* conditions).

Another feature usually supported even by early approaches is the concept of *user groups* (e.g., Employees, Programmers, Consultants). Groups can be nested and need not be disjoint. Figure 1 illustrates an example of user-group hierarchy. Support of groups greatly simplifies management of authorizations, since a single authorization granted to a group can be enjoyed by all its members.

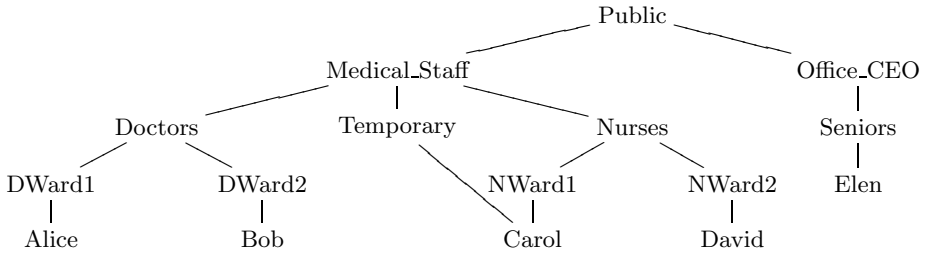

Fig. 1. An example of user-group hierarchy

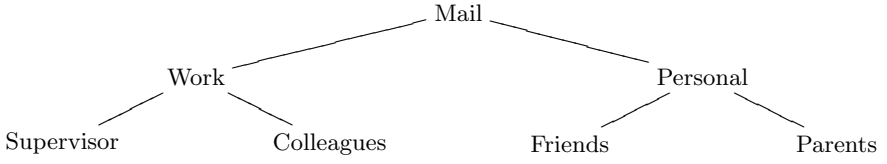

Fig. 2. An example of object hierarchy

Later efforts moved to the support of groups on all the elements of the authorization triple (i.e., subject, object, and operation), where, typically, groups are abstractions hierarchically organized. For instance, in an operating system the hierarchy can reflect the logical file system tree structure, while in object-oriented system it can reflect the class (is-a) hierarchy. Figure 2 illustrates an example of object hierarchy. Even operations can be organized hierarchically, where the hierarchy may reflect an implication of privileges (e.g., write is more powerful than read [29]) or a grouping of sets of privileges (e.g., a "writing privileges" group can be defined containing write, append, and undo [32]). These hierarchical relationships can be exploited *i)* to support preconditions on accesses (e.g., in Unix a subject needs the execute privilege on a directory to access the files within it), or *ii)* to support authorization implication, that is, authorizations specified on an abstraction apply to all its members. Support of abstractions with implications provides a short hand way to specify authorizations, clearly simplifying authorization management. As a matter of fact, in most situations the ability to execute privileges depends on the membership of users into groups or objects into collections: translating these requirements into basic triples of the form ⟨user, object, operation⟩ that then have to be singularly managed is a considerable administrative burden, and makes it difficult to maintain both satisfactory security and administrative efficiency.

2.2 Positive and Negative Authorizations

Although there are cases where abstractions can work just fine, many will be the cases where exceptions (i.e., authorizations applicable to all members of a group but few) will need to be supported. This observation has brought to the combined

support of both *positive* and *negative* authorizations. Traditionally, positive and negative authorizations have been used in mutual exclusion corresponding to two classical approaches to access control, namely:

Closed policy: authorizations specify permissions for an access. The closed policy allows an access if there exists a positive authorization for it, and denies it otherwise.

Open Policy: (negative) authorizations specify denials for an access. The open policy denies an access if there exists a negative authorization for it, and allows it otherwise.

The open policy has usually found application only in those scenarios where the need for protection is not strong and by default access is to be granted. Most systems adopt the closed policy, which, denying access by default, ensures better protection; cases where information is public by default are enforced with a positive authorization on the root of the subject hierarchy (e.g., Public).

The combined use of positive and negative authorizations was therefore considered as a way to conveniently support exceptions. To illustrate, suppose we wish to grant an authorization to all members of a group composed of one thousand users, except to one specific member Alice. In a closed policy approach, we would have to express the above requirement by specifying a positive authorization for each member of the group except Alice.[1] However, if we combine positive and negative authorizations we can specify the same requirement by granting a positive authorization to the group and a negative authorization to Alice.

The combined use of positive and negative authorizations brings now to the problem of how the two specifications should be treated:

- what if for an access no authorization is specified? (*incompleteness*)
- what if for an access there are both a negative and a positive authorization? (*inconsistency*)

Completeness can be easily achieved by assuming that one of either the open or closed policy operates as a *default*, and accordingly access is granted or denied if no authorization is found for it. Note that the alternative of explicitly requiring completeness of the authorizations is too heavy and complicates administration.

Conflict resolution is a more complex matter and does not usually have a unique answer [18, 25]. Rather, different decision criteria could be adopted, each applicable in specific situations, corresponding to different policies that can be implemented. Examples of different conflict resolution policies are given below.

[1] In an open policy scenario, the dual example of all users, but a few, who have to be denied an access can be considered.

Denials Take Precedence. Negative authorizations are always adopted when a conflict occurs (it satisfies the "fail safe principle"). In other words, the principle says that if we have one reason to authorize an access, and another to deny it, then we deny it.

Most Specific Takes Precedence. A natural and straightforward policy is the one stating that "the most specific authorization should be the one that prevails"; after all this is what we had in mind when we introduced negative authorizations in the first place (our example about Alice). Although the most-specific-takes-precedence principle is intuitive and natural and likely to fit in many situations, it is not enough. As a matter of fact, even if we adopt the argument that the most specific authorization always wins (and this may not always be the case) it is not always clear what more specific is:

- what if two authorizations are specified on non-disjoint, but non-hierarchically related groups (e.g., NWard1 and NWard2 in Figure 1)?
- what if for two authorizations the most specific relationship appear reversed over different domains? For instance, consider authorizations (Doctors, read+, Mail) and (Medical_Staff, read−, Personal); the first has a more specific subject, while the second has a more specific object (see Figures 1 and 2).

Most Specific Along a Path Takes Precedence. This policy considers an authorization specified on an element x as overriding an authorization specified on a more general element y only for those elements that are members of y because of x. Intuitively, this policy takes into account the fact that, even in the presence of a more specific authorization, the more general authorization can still be applicable because of other paths in the hierarchy. For instance, consider the group hierarchy in Figure 1 and suppose that for an access a negative authorization is granted to Medical_Staff while a positive authorization is granted to Nurses. What should we decide for Carol? On the one side, it is true that Nurses is more specific than Medical_Staff; on the other side, however, Carol belongs to Temporary, and for Temporary members the negative authorization is not overridden. While the most-specific-takes-precedence policy would consider the authorization granted to Medical_Staff as being overridden for Carol, the most-specific-along-a-path considers both authorizations as applicable to Carol. Intuitively, in the most-specific-along-a-path policy, an authorization propagates down the hierarchy until overridden by a more specific authorization [15].

Priority Level. The most specific argument does not always apply. For instance, an organization may want to be able to state that consultants should not be given access to private projects, *no exceptions allowed*. However, if the most specific policy is applied, any authorization explicitly granted to a single consultant will override the denial specified by the organization. To address situations like this, some approaches proposed adopting *explicit priorities*; however, these solutions do not appear viable as the authorization specifications may result not always clear.

Positional. Other approaches (e.g., [32]) proposed making authorization priority dependent on the *order in which authorizations are listed* (i.e., the authorizations that is encountered first applies). This approach, however, has the drawback that granting or removing an authorization requires inserting the authorization in the proper place in the list. Beside the administrative burden put on the administrator (who, essentially, has to explicitly solve the conflicts when deciding the order), specifying authorizations implies explicitly writing the ACL associated with the object, and may impede delegation of administrative privileges.

Grantor- or Time-Dependent. Other possible ways of defining priorities can make the authorization's priority dependent on the *time* at which the authorizations was granted (e.g., more recent authorizations prevails) or on priorities between the *grantors*. For instance, authorizations specified by an employee may be overridden by those specified by her supervisor; the authorizations specified by an object's owner may override those specified by other users to whom the owner has delegated administrative authority.

As it is clear from this discussion, different approaches can be taken to deal with positive and negative authorizations. Also, if it is true that some solutions may appear more natural than others, none of them represents "the perfect solution". Whichever approach we take, we will always find one situation for which it does not fit. Also, note that different conflict resolution policies are not mutually exclusive. For instance, one can decide to try solving conflicts with the most-specific-takes-precedence policy first, and apply the denials-take-precedence principle on the remaining conflicts (i.e., conflicting authorizations that are not hierarchically related).

The support of negative authorizations does not come for free, and there is a price to pay in terms of authorization management and less clarity of the specifications. However, the complications brought by negative authorizations are not due to negative authorizations themselves, but to the different semantics that the presence of permissions and denials can have, that is, to the complexity of the different real world scenarios and requirements that may need to be captured. There is therefore a trade-off between expressiveness and simplicity. For this reason, most current systems adopting negative authorizations for exception support impose specific conflict resolution policies, or support a limited form of conflict resolution. (e.g., see the Apache server [1] where authorizations can be positive and negative and an ordering can be specified dictating how negative and positive authorizations are to be interpreted). More recent approaches are moving towards the development of flexible frameworks with the support of multiple conflict resolution and decision policies.

2.3 Attribute-Based Specifications

In an open system like the Internet, the different parties (clients and servers) that interact with each other to offer services are usually strangers. They have no preexisting relationship and are not in the same security domain. Therefore, on the one side the server may not have all the information it needs to decide

whether or not an access should be granted. On the other side, however, the client may not know which information she needs to present to a (possibly just encountered) server to get access. All this requires a new way of enforcing the access control process, which cannot be assumed anymore to operate with a given prior knowledge and return a yes/no access decision. Rather, the access control process should be able to operate without a priori knowledge of the party requesting access and return the information of the requisites that it requires be satisfied for the access to be allowed [2, 17]. Also, the traditional "identity-based access control models" where subjects and objects are usually identified by unique names are not appropriate in this setting. Instead, attributes other than identity are useful in determining the party's trustworthiness. In this context, access restrictions to the data/services should be expressed by policies that specified the properties (attributes) that a requester should enjoy to gain access to the data/services. Some proposals have been developed that use *digital certificates*. Traditionally, the widely adopted digital certificate has been the *identity certificate*. An identity certificate is an electronic document used to recognize an individual, a server, or some other entity, and to connect that identity with a public key [3, 4, 8]. More recent research and development efforts have resulted in a second kind of digital certificate, the *attribute certificate* [14] that can be used for supporting and attribute-based access control. An attribute certificate has a structure similar to an identity certificate but contains attributes that specify access control information associated with the certificate holder (e.g., group membership, role, security clearance). One of the most important aspects that attribute-based access control policies should support is the ability to specify accesses to a *collection* of services based on a *collection* of attributes. In this context, logic programming provides a convenient, expressive, and well-understood framework in which to work with authorization policy. Jajodia et. al [33] propose a framework that models an attribute-based access control system using logic programming with set constraints of a computable set theory. More precisely, the set theory used in this approach is CLP($\mathcal{SET}$), the *hereditarily finite* and computable set theory developed by Dovier et al. [13]. Here, sets are constructed out of a finite universe by applying operators such as $\cap$, $\cup$, and so on. A policy can refer to both attributes and services, and a two sorted first order language with set variables is then used. The terms are constructed in the usual way by means of variables and functions. Also, the approach supports two kinds of predicates: those used to specify the computation domain and those used to specify its sub-domain of constraints. To reduce the runtime inefficiency of constrained logic programs, which is due to the backtracking through program clauses, two techniques are used. The first technique consists in transforming any attribute-based access control policy into one with less backtracking but the same semantics. The second technique consists in materializing commonly accessed predicates instances.

Bonatti and Samarati in [6] propose a uniform framework for regulating service access and information disclosure in an open, distributed network system like the Web. Access regulations are specified as logical rules, where some predicates

are explicitly identified. Attribute certificates are modeled as *credential expressions* of the form `credential_name(attribute_name`$_1$ `= value_term`$_1$`),..., attribute_name`$_n$ `= value_term`$_n$, where `credential_name` is the attribute credential name, `attribute_name`$_i$ is the attribute name, and `value_term`$_i$ is either a ground value or a variable. Besides credentials, the proposal also allows to reason about declarations (i.e., unsigned statements) and user-profiles that the server can maintain and exploit for taking the access decision. Communication of requisites to be satisfied by the requester is based on a filtering and renaming process applied on the server's policy, which exploits partial evaluation techniques in logic programs.

Although attribute-based access control polices allow the specifications of access control rules with reference to generic attributes or properties of the involved parties, they do not fully exploit the semantic power and reasoning capabilities of emerging web applications. The next step in the development of expressive and powerful access control models and policies should then be the support of access control rules based on the rich ontology-based metadata associated with both the subjects accessing the resources and the resources themselves [9].

3 Languages for Access Control

Languages for access control aim to support the expression and the enforcement of policies [30]. Many of these languages are used for expressing generic assertions about subjects (principals) such as the association of a principal with a public key, the membership of a principal in a group, or the right of a principal to perform a certain operation at a specified time [27]. They also serve for combining policies from many sources, and thus for making authorization decisions [5]. More broadly, the languages sometimes aim to support trust management [6, 31, 35]. Also, with the increasing number of applications that either use XML as their data model, or export relational data as XML data, it becomes critical to investigate the problem of access control for XML. To this purpose, many XML-based access control language have been proposed [7, 10, 20, 26]. In particular, one of the most relevant XML-based access control language is the eXtensible Access Control Markup Language (XACML). The purpose of XACML is the expression of authorization policies in XML against objects that are themselves identified in XML. XACML covers both an access control policy language and a request/response language. So besides defining who can do what and when by generating the corresponding policies, it is also possible to express access requests and responses in XACML. The eXtensible Access Control Markup Language (XACML) version 1.0 [26] has been an OASIS standard since 2003. Improvements have been made to the language and incorporated in version 2.0 [28].

Several of the most recent language designs rely on concepts and techniques from logic, specifically from logic programming: Li et al.'s D1LP and RT [22, 23, 24], Jim's SD3 [19], and DeTreville's Binder [12]. The expressive power and the formal foundations of logical formalisms are appealing in this context. Some

researchers and practitioners object that logic-based specifications may be complicated or even intimidating to some users. Security administrators and end users need simple and user-friendly approaches that allow them to easily understand the system behavior and maintain control over security specifications. It is tempting to conclude that logic-based approaches are not applicable, but then one would also give up all the advantages of logic-based formulations that enjoy a combination of clean foundations (hence, formal guarantees), flexibility, expressiveness, and declarativeness (so that users are not required to have any programming ability). On the contrary, we believe that a careful choice of syntax makes logic-based specifications accessible to a wide spectrum of users. In the following section, we present one of the most representative logic-based languages for access control.

3.1 A Flexible Authorization Framework

Jajodia et al. [18] worked on a proposal for a logic-based language that attempted to balance flexibility and expressiveness on one side, and easy management and performance, on the other. This language is a good representative for this line of work. It allows representing different policies and protection requirements, while at the same time providing understandable specifications, clear semantics, and bearable data complexity. Their proposal for a *flexible authorization framework* (FAF) corresponds to a polynomial (quadratic) time data complexity fragment of default logic.

In FAF, policies are divided into four decision stages, corresponding to the following policy components (Figure 4).

- *Authorization Table.* This is the set of explicitly specified authorizations.
- The *propagation policy* specifies how to obtain new derived authorizations from the explicit authorization table. Typically, derived authorizations are obtained according to hierarchy-based derivation. However, derivation rules are not restricted to this particular form of derivation.
- The *conflict resolution policy* describes how possible conflicts between the (explicit and/or derived) authorizations should be solved. Possible conflict resolution policies include *no-conflict* (conflicts are considered errors), *denials take precedence* (negative authorizations prevail over positive ones), *permissions-take-precedence* (positive authorizations prevail over negative ones), and *nothing-takes-precedence* (the conflict remains unsolved). Some forms of conflict resolutions can be expressed within the propagation policy, as in the case of overriding (also known as *most-specific-takes precedence*).
- A *decision policy* defines the response that should be returned to each access request. In case of conflicts or gaps (i.e. some access is neither authorized nor denied), the decision policy determines the answer. In many systems, decisions assume either the open or the closed form (by default, access is granted or denied, respectively).

Starting from this separation, the authorization specification language of FAF takes the following approach:

- The authorization table is viewed as a database.
- Policies are expressed by a restricted class of stratified and function-free normal logic programs called *authorization specifications*.
- The semantics of authorization specifications is the stable model semantics [16]. The structure of authorization specifications guarantees stratification and hence, stable model uniqueness and PTIME computability.

The four decision stages correspond to the following predicates. (Below s, o, and a denote a subject, object, and action term, respectively, where a term is either a constant value in the corresponding domain or a variable ranging over it).

`cando(o,s,±a)` represents authorizations explicitly inserted by the security administrator. They represent the accesses that the administrator wishes to allow or deny (depending on the sign associated with the action).

`dercando(o,s,±a)` represents authorizations derived by the system using logic program rules.

`do(o,s,±a)` handles both conflict resolution and the final decision.

Moreover, a predicate `done` keeps track of the history of accesses (for example, this can be useful to implement a Chinese Wall policy), and a predicate `error` can be used to express integrity constraints. In addition, the language has a set of predicates for representing hierarchical relationships (`hie`-predicates) and additional application-specific predicates, called `rel`-predicates. Hierarchical predicates represent hierarchical relationships within the different components of the system (objects, subjects, or access modes). For instance, in most realistic systems, data items are organized hierarchically. For example, in a file system, the basic objects are files, but these files are typically organized in a hierarchical directory structure. Similarly, in an object-oriented database, the objects being accessed are organized into an object hierarchy. In an analogous way, authorization subjects can be basic users or hierarchical groups in which they are organized. Application-specific predicates capture the possible different relationships, existing between the elements of the data system, that may need to be taken into account by the access control system. Examples of `rel`-predicates are `owner(user, object)`, which models ownership of objects by users, or `supervisor(user1, user2)`, which models responsibilities and control within the organizational structure.

Authorization specifications are stated as logic rules defined over the above predicates. To ensure stratifiability, the format of the rules is restricted as illustrated in Figure 3. Note that the adopted strata reflect the logical ordering of the four decision stages.

The authors of [18] present a materialization technique for producing, storing, and updating the stable model of the policy. The model is computed on the initial specifications and updated with incremental maintenance strategies.

Note that the clean identification and separation of the four decision stages can be regarded as a basis for a policy specification methodology. In this sense,

Stratum	Predicate	Rules defining predicate
0	`hie-`predicates	Base relations.
	`rel-`predicates	Base relations.
	`done`	Base relation.
1	`cando`	Body may contain `done`, `hie-` and `rel-`literals.
2	`dercando`	Body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive.
3	`do`	When head is of the form `do`$(\_, \_, +a)$ body may contain `cando`, `dercando`, `done`, `hie-` and `rel-` literals.
4	`do`	When head is of the form `do`$(o, s, -a)$ body contains just one literal $\neg$`do`$(o, s, +a)$.
5	`error`	Body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals.

Fig. 3. Rule composition and stratification of the proposal in [18]

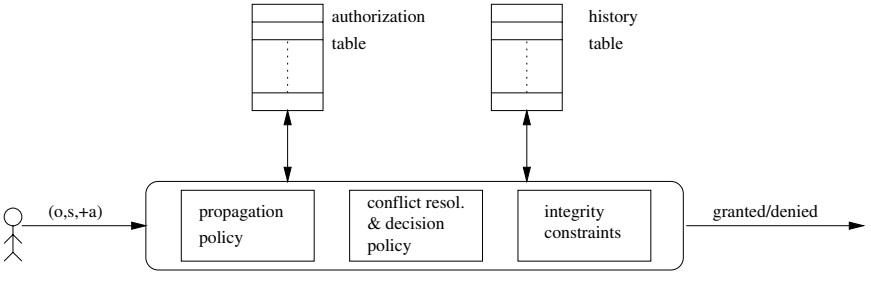

Fig. 4. Functional authorization architecture in [18]

the choice of a precise ontology and other syntactic restrictions (such as those illustrated in Figure 3) may assist security managers in formulating their policies.

4 Conclusions

Access control models, policies, and languages are constantly under development to obtain frameworks flexible and expressive enough so as to handle the specification and enforcement of security requirements of many emerging applications and real-world scenarios. In this paper, we presented the main features that modern access control models and policies should support and discussed recent proposals in the area of access control languages.

References

1. Apache http server version 2.0. http://www.apache.org/docs-2.0/misc/tutorials.html.
2. C. Bettini, S. Jajodia, S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management and security applications. In *Proc. 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems. Springer Verlag LNCS State-ofthe- Art series,*, 1998.
4. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of the 1996 IEEE Symposiumon Security and Privacy*, Oakland, CA, USA, May 1996.
5. P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1):1–35, February 2002.
6. P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002.
7. D. Box et al. *Web services policy framework (WS-Policy) version 1.1.*, May 2003. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp.
8. Y-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. Referee: trust management forweb applications. *WorldWide Web Journal*, 2(3):706–734, 1997.
9. E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati. Extending policy languages to the semantic web. In *Proc. of the International Conference on Web Engineering*, Munich, Germany, July 2004.
10. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.
11. C.J. Date. *An Introduction to Database Systems.* Addison-Wesley, 6th edition, 1995.
12. J. DeTreville. Binder, a logic-based security language. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2002.
13. A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraints logic programming. *ACM Transactions of Programming Languages and Systems*, 22(5):861–931, September 2000.
14. S. Farrell and R. Housley. An internet attribute certificate profile for authorization. RFC 3281, April 2002.
15. E.B. Fernandez, E. Gudes, and H. Song. A model for evaluation and administration of security in object-oriented databases. *IEEE Transaction on Knowledge and Data Engineering*, 6(2):275–292, 1994.
16. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
17. S. Jajodia, M. Kudo, and V.S. Subrahmanian. Provisional authorizations. In Anup Ghosh, editor, *E-Commerce Security and Privacy*, pages 133–159. Kluwer Academic Publishers, Boston, 2001.
18. S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, June 2001.

19. T. Jim. Sd3: A trust management system with certified evaluation. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2001.
20. M. Kudoh, Y. Hirayama, S. Hada, and A. Vollschwitz. Access control specification based on policy evaluation and enforcement model and specification language. In *Symposium on Cryptograpy and Information Security, SCIS'2000*, 2000.
21. C.E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, 1981.
22. N. Li, B.N. Grosof, and Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6(1):128–171, February 2003.
23. N. Li and J.C. Mitchell. Datalog with constraints: A foundation for trust-management languages. In *Proc. of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, New Orleans, LA, USA, January 2003.
24. N. Li, J.C. Mitchell, and W.H. Winsborough. Design of a role-based trust-management framework. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2002.
25. T. Lunt. Access control policies: Some unanswered questions. In *IEEE Computer Security Foundations Workshop II*, pages 227–245, Franconia, NH, June 1988.
26. OASIS. *eXtensible Access Control Markup Language (XACML) Version 1.0*, 2003. http://www.oasis-open.org/committees/xacml.
27. OASIS. *Security Assertion Markup Language (SAML) V1.1*, 2003. http://www.oasis-open.org/committees/security/.
28. OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2004. http://www.oasis-open.org/committees/xacml.
29. F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM TODS*, 16(1):89–131, March 1991.
30. P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.
31. K.E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *Proc. of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, Monterey, CA, June 2002.
32. H. Shen and P. Dewan. Access control for collaborative environments. In *Proc. Int. Conf. on Computer Supported Cooperative Work*, pages 51–58, November 1992.
33. L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proc. of the 2004 ACM Workshop on Formal Methods in Security Engineering*, Washington DC, USA, October 2004.
34. D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security*, 6(2):286–325, May 2003.
35. T. Yu, M. Winslett, and K.E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.