The Policy Machine for Security Policy Management

Vincent C. Hu, Deborah A. Frincke, and David F. Ferraiolo
National Institute of Standards and Technology, 100 Bureau Dr. Stop 8930 Gaithersburg
Maryland 20899-8930, USA
{vhu, dferraiolo}@nist.gov
Department of Computer Science of the University of Idaho, Moscow Idaho 83844, USA
frincke@cs.uidaho.edu

Abstract. Many different access control policies and models have been developed to suit a variety of goals; these include **Role-Based Access Control**, **One-directional Information Flow**, **Chinese Wall**, **Clark-Wilson**, ***N*-person Control**, and **DAC**, in addition to more informal ad hoc policies. While each of these policies has a particular area of strength, the notational differences between these policies are substantial. As a result it is difficult to combine them, both in making formal statements about systems which are based on differing models and in using more than one access control policy model within a given system. Thus, there is a need for a unifying formalism which is general enough to encompass a range of these policies and models. In this paper, we propose an open security architecture called the *Policy Machine* (*PM*) that would meet this need. We also provide examples showing how the *PM* specifies and enforces access control polices.

1 Introduction

Access control is a critical component of most approaches to providing system security. Access control is used to achieve three primary objectives: (1), determining which subjects are entitled to have access to which objects (Authorization); (2) determining the access rights permitted (a combination of access modes such as read, write, execute, delete, and append); and (3) enforcing the access rights. An access control policy describes how to achieve these three goals; to be effective, this policy needs to be managed and enforced. There is a vast array of techniques that define and enforce access control policies within host operating systems and across heterogeneous bodies of data [NCSC98]. Although these techniques are successful in the specific situations for which they were developed, the current state of security technology has, to some extent, failed to address the needs of all systems [Spencer et al99, HGPS99] in a single notation. Access control policies can be as diverse as the applications that rely upon them, and are heavily dependent on the needs of a particular environment. Further, notations that easily express one collection of access control policies may be awkward (or incapable) in another venue. An example of this situation would be when a company's documents are under **One-direction Information Flow** [BL73, Biba77, Sand93] policy control at the development stage. When the development is finished, the documents that are available for use by employees, could then be required to be controlled by a role-based or **RBAC** [FCK95, SCFY96] policy. Most existing commercial technologies used to provide security to systems are restricted to a single policy model, rather than permitting a variety of models to be used [Spencer et al99]. For instance, Linux applies a **DAC** [NCSC87] policy, and it is difficult to implement **RBAC policy** (among others) in such a system. Further, if an organization decides to change from one policy model to another, it is quite likely that the new policy model will have to be implemented above the operating systems level, perhaps even as part of the application code or through an intermediary. This is inconvenient, subject to error, slow, and makes it difficult to identify or model the overall "policy" that is enforced by the system.

To meet this challenge, we have developed the *Policy Machine* (*PM).* The underlying concept of the *PM* relies on the separation of the access control mechanism from the access control policy [JSS97, JSSB97]. This enables enforcement of multiple access control policies within a single, unified system. Although complete policy coverage is an elusive goal, the *PM* is capable of expressing a broad spectrum of well-known access control policies. Those we have tested so far include: **One-directional Information Flow, Chinese Wall** [BNCW89]**, N-person Control** [NCSC91] **and DAC**. These were selected partly because they are so well known, and partly because they differ greatly from one another. A further advantage of *PM* is that it is highly extensible, since it can be augmented with any new policy that a specific application or user may require. This paper will demonstrate the functionalities of *PM*, and illustrate *PM*'s universal, compositional and combinational properties.

2 Policy Machine (PM)

In this section we describe our design of *PM*, and explain how we achieve our goal of a unified description and enforcement of policies. Our design of *PM* is related to the traditional **reference monitor** approach. A reference monitor is not necessarily a single piece of code that controls all accesses; rather, it is an abstraction or model of the collection of access controls [Ande72]. As we have applied the concept, *PM* does not dictate requirements for particular types of subject or object attributes nor the relationships of these attributes within its abstract database. Because of this generality *PM* can be used to model the implementation of a wide variety of access control policies, but *PM* is not itself a policy.

The structure of *PM* is based on the concept that all enforcement can be fundamentally characterized as either static (e.g., **BBAC**), dynamic (e.g., **Work Flow**), or historical (e.g., **Chinese Wall**) [GGF98, SZ97]. In the next five sections, we introduce the primary components of the *PM*: the *Policy Engine* (*PE*) with *PE* databases and the security policy operations, which is the *General Policy Operations* (*GPO*). We then discuss the Universal and Composition properties of the *PM* following each introduction.

2.1 Policy Engine (PE)

The task of the *PE* is to receive access requests and determine whether they should be permitted. Access requests are of the form <*User Identity*, *Requested Operations*, *Requested Objects*>. To determine the acceptability of a request, the *PE* executes three separate phases (Figure 1): the request management phase, the subject-object mediation phase, and the access history recording and database management phase to determine whether access is granted or denied. The request management phase is primarily used to assign the requesting user to a *subject.* The subject-object mediation phase is used to decide if the *subject* is permitted to access the requested *object.* The access history recording and database management phase determines if the granted access will affect the state of the *PM* if a historical policy is implemented.

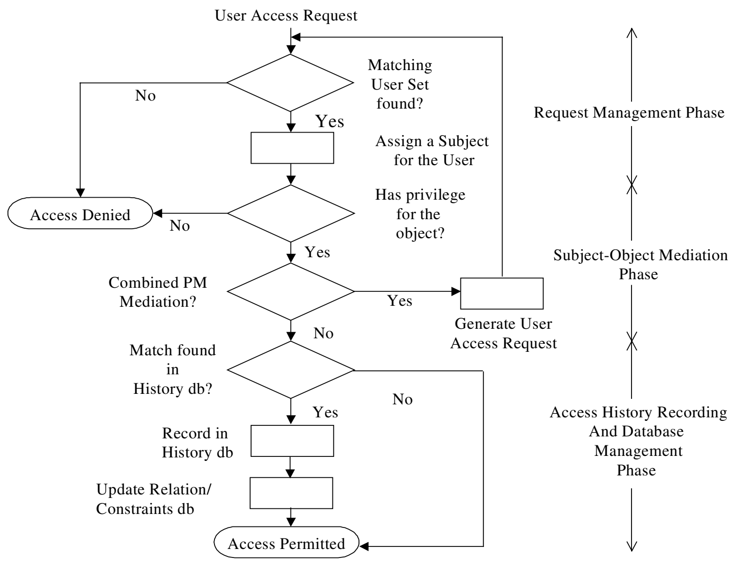
User Access Request
Matching User Set found?
No
Yes
Assign a Subject for the User
Has privilege for the object?
Access Denied
No
Yes
Combined PM Mediation?
Yes
Generate User Access Request
No
Match found in History db?
No
Yes
Record in History db
Update Relation/ Constraints db
Access Permitted
Request Management Phase
Subject-Object Mediation Phase
Access History Recording And Database Management Phase

Fig. 1. *PM* phases

Request Management Phase (RMp)

This phase is activated by the detection of a user request. The *PE* responds by creating a *subject* along with *operation*, and *object* specified within the user's request. For example, consider the following request:

(Mary; write; a_file) i.e., *user Mary* wants to write to *a_file*.

Currently, the *Relations database (Rdb)* has entries as follow:

Managers = (Mary), Programmers = (Managers, Bob, Sue) i.e., the *Managers user set* contains *user Mary*, and *user set Programmers* contains *user set Managers* and *users Bob* and *Sue*.

Thus, *user Mary* is assigned as:

subject = (Mary, Managers, Programmers); write; a_file

and is able to continue to the next phase.

Subject-Object Mediation Phase (SOMp)

The *SOMp* takes the input (*subject, operations* and *object*) created by the *RMp*, and the process authorizes access under two conditions. First, there is a match of the *subject*'s request and there is an entry in the *Privilege database* (*Pdb*). In this case, the requested access is authorized. Second, there exists no further subject-object mediation check that is required under a different *PM*. For example, assume that, in the *SOMp, PE* generated the following message when in the *RMp:*

subject = (Mary, Managers, Programmers); write; a_file

The *SOMp* will then check *Pdb*, and find entries:

(Programmers; all; a_file)

This means that the *Programmers user set* has *all* privileges to the file *a_file*. Therefore the request

subject = (Mary, Managers, Programmers); write; a_file

is authorized.

Access History Recording and Database Management Phase (AHRDMp)

This phase evaluates the relevance of the authorized *event* with respect to the history-based policies that are stored in the *History Based Relations database* (*HBRdb*). History-based policies are driven by an *event* and an *action* stored in the *HBRdb*. If the *event* received matches *events* stored in the *HBRdb* then *PE* in *AHRDMp* invokes the *action* associated with the *event*. The *action* either creates new constraints in the *Constraints database* (*Cdb)*, and/or updates relations in the *Pdb*. Consistency is checked whenever a relation or constraint is created. For example, *HBRdb* contains an entry:

event = (subject_a; all; x_file), response = (Generate Constraints = (((subject_a; all; y_file) ⊕ (:*:*)), ((subject_a; all;z_file) ⊕ (*:*:*)); Cdb)*

This means that *subject_a* is prohibited from any kind of access (by excluding (⊕) all (*)) to *y_file* and *z_file* if *x_file* has been accessed by *subject_a*. Entries for the constraints will be added into the *Cdb*. This example can be used for policies that require **SOD** constrains, such as **Chinese Wall** [BNCW89] policies.

2.1.1 PE Databases

PE databases provide relation, privilege, constraint, and history information for *PE* processes. The databases are maintained either by *GPO* or by the *AHRDMp*. In this section, we will describe each of the four types of databases.

Relations database *(Rdb)* maintains the *inheritance* relations between sets. For example, *Rdb* is used in the *RMp* to establish the active *user set(s)* of a *subject*. An example is in Figure 2.

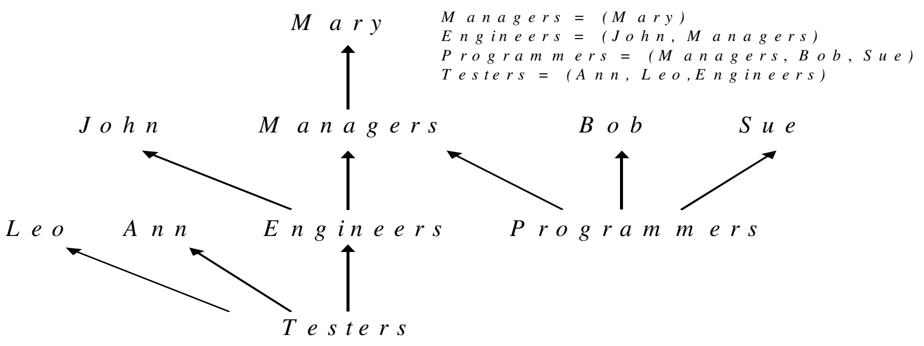
Managers = (Mary)
Engineers = (John, Managers)
Programmers = (Managers, Bob, Sue)
Testers = (Ann, Leo, Engineers)
Mary
John
Managers
Bob
Sue
Leo
Ann
Engineers
Programmers
Testers

Fig. 2. Set Relations database example

Constraints database (*Cdb*) contains constraint relations of *subject (sets)*, *object (sets)*, and *operation(s)*. It is used by *RMp* as a reference to establish a *subject* from a *user's* access request. An example entry:

(SS_1; P_1; OS_1), (SS_2; P_2; OS_2) i.e., The combination of *subject (sets) SS_1*, *operation(s) P_1* and *object (sets) OS_1* and the combination of *subject (sets) SS_2*, *operation(s) P_2* and *object (sets) OS_2* are mutually excluded.

Privilege database (*Pdb*) is accessed by *SOMp* for searching the *subject (sets)*, *operation(s)* and *object (sets)* relations. A relation in *Pdb* defines the privilege for the *subject (sets)*. An example of *Pdb* entry is:

Programmers; read, write, execute; file_a i.e., *subject set Programmers* can *read*, *write*, and *execute file_a*.

History Based Relations database (*HBRdb*) contains information, which is used in the *AHRDMp* to perform state transaction of a historical-related policy embedded in the database with the structure:

st_i, Event, Actions, st_j, where *st_i* is the current state of the embedded historical access control policies, *Event* is an authorized access :(*subject (sets)*, *operation(s)*, *object (sets)*). *Actions* are sets of actions *(a_1.....a_n)*, each *a_i* = (*dbname*, *Updates*), where the database *dbname* is the name of one of the *Cdb* or *Pdb* will be updated with the information in *Updates*. *st_j* is the next historical state *PM* will be when the *Event* occurred.

2.1.2 Universal Property

PM can support most major access models, making it a possible to express the corresponding access control policies that these models represent. In order to reduce the support needed to express a multitude of models, we take advantage of previous work which has shown that certain model categories may be simulated or represented by other model categories. For instance, Sandhu has shown in [Sand93] that **Lattice-based** access control models such as the **Bell-Lapadula** model [BL73] and **Biba** model [Biba77] may simulate **Information Flow** polices (including **MAC** policies) and **Chinese Wall** policies. In *PM*, **Lattices** are represented by a **set** implemented by data records in a database and embedded in the *PE*'s *Rdb* as described in section *2.1.1 Rdb*. The relations of the elements of the **Lattices** are checked in *RMp* as described in section 2.1 *RMp*. This allows us to describe a wide variety of models with a minimum of *PM* features.

As a second example, *PM* maintains records for access events and updates databases to reflect changes of the states caused by the events according to the historical policy states that are embedded in *HRBdb.* As indicated in section 2.1.1 *HRBd*, the historical related policy is implemented by a table structure of a database, the process in the *AHRDMp* tracks the historical state of the policy by referring the information in the table as described in section 2.1 *AHRDMp*. Therefore, policy models which can be mapped to state transition machine models can also modeled by *PM*; this allows us to support **Clark Wilson** [CWAC87] and **N-person Control** policy models, for example.

Gligor, Gavrila and Ferraiolo [GGF98] showed that **SOD** policies can be enforced either statically or dynamically. We can use either method by employing *PE*'s *Cdb* as described in section 2.1.1 statically and dynamically. The static *Cdb* constrains relations between subjects, objects and/or operations, and provides the tool for statically separating users of their privileges. The dynamic *Cdb* can be generated by the process of *AHRDMp* according to *the actions* stored in the historical policy database and can separate the users' privileges dynamically. Therefore, through the *Cdb* the *PM* is able to enforce policy models which require either static or dynamic **SOD. T**hese policies include **Work Flow** [AH96] and **RBAC.**

DAC access control policy can be achieved by appropriately managing the *Pdb* tables associated with each user. The *Pdb* table for each *user* should be set as a controlled *object* of the administrator's *PE.* These users will then have sole and total control of their own *Pdb* table, thereby allowing them to delegate the access privilege to other *users*.

Although a formal proof of correspondence is beyond the scope of this paper, the above paragraphs indicate how the properties and components of *PM* allow *PM* to model some major access control models.

2.2 General Policy Operations (GPO)

GPO is a set of operations for expressing access control policies. It allows users to specify, together with the authorizations, the policy according to which access control decisions are to be made. *GPO* has the following components: *Basic sets* define the basic elements and sets for the *GPO*. *Database query functions* represent restrictions or relationships between elements. *Administrative operations* are used for the administration of the *PM* databases. *Rules* are used to keep the transitions in the states of *PM* during the system operation.

Basic Sets

A set of basic sets, defines the *users*, *objects*, *operations* and their relations.

Examples:

US = the set of *user sets*, *(US_1, ……, US_n)*.

US_i = the set of *users*, *(u_1, …, u_n)*, where each *user* is assigned to *user set US_i*.

Database Query Functions

There are three types of database query functions:

1. *Set query* functions, for retrieving relations from *Rdb*. Examples:

member_of (*s*) denotes the members of set *s* (*s* is inherited by the members).
set_of (*m*) denotes the sets the member *m* is assigned to (inherit from).
transitive_member_of (*s*) denotes the transitive members of set *s*.
transitive_set_of (*m*) denotes the transitive sets the member *m* belongs to.

2. *Constraint query* functions, for retrieving constrains from *Cdb*. Examples:

userSet_constrained (*s*) denotes the constraint relations of the *user (set) s*.
objectSet_constrained (*o*) denotes the constraint relations of the *object (set) o*.
userSet_permitted (*s*) denotes the permitted relations related to the *user (set) s*.
objectSet_permitted (*o*) denotes the permitted relations of the *object (set) o*.

3. *Process mapping* functions, for retrieving information of current process. Examples:

operation_request (*u*) denotes the operation requested by *user* u.
object_request (*u*) denotes the *object* requested by *user u*.
subject_of (*u*) denotes the subject associated with *user u*.
active_userSets (*s*) denotes the set of active user(sets) associated with *subject s*.
access(s,p,o) return *1* if *subject s* is authorized to access *object o* by *operation p*.

Administrative Operations

A set of administrative commands. Examples:

addMember (x, y), *rmMember (x, y)*, new member *x* is added/removed to set *y* in *Rdb*.

addConstraints(*c*), *rmConstraintse*(*c*), constraints *c* is added/removed to *Cdb* database.

addRelations(*c*), *rmRelations*(*c*), relation *c* is added/removed to *Pdb*.

Rules

The rules represent assumptions about the *GPO*. Examples:

The member assigned to a given set is exactly the member directly inheriting the given set in Rdb, and the sets assigned to a given member are exactly the sets directly inherited by the member in Rdb.

Any two user sets assigned to a third user set do not inherit (directly or indirectly) one another in Rdb.

2.2.1 PM Composition

The *Administration Operations* of *GPO* allow users of *PM* to interact with *PM*; they are used to add, remove, and update a *PE* database entry. Before an *Administration*

Operation is executed, *GPO* will invoke the *Database Query Function*s built in the *GPO* to retrieve information required for the operation. *GPO* will then check the validation of the logics and formats governed by the *GPO's Rules*. The *Rules* of *GPO* guarantee the consistencies of the relations, constraints and privileges of the information stored in the *PE databases*.

Some of the functions for well-known access control policies can be built in the *GPO*. Through the *Administration Operations, PM* users/administrators can implement and manage an access control policy. For example, to add a relation for users in a Separation of Duty (**SOD)** policy, *PM* administrators can execute the command:

AddSODusers (x, y), i.e., add separation of duty (mutual exclusive) relation for *users x* and *users y*.

The *Administrative Operations* are a key feature of *GPO*, since they provide the *PM* administrator a tool for composing new access control policies. Through the *GPO's Rule*, the information stored in the *PE* database can be generated and modified without *PM* administrators having any known access control policy in mind.

The composition capability of *GPO* supports the important feature of *PM* that separates the security policies from the security mechanisms.

2.3 PM Combination

A key feature of *PM* is its ability to combine policies. Policies are established in different *PM*s by implementing their databases and *GPO*s. When policies are combined, *PM*s are essentially "chained together". In practice, this means that a "User Request" begins at the "first" *PM*, and it is passed along to subsequent *PM*s for examination. Each *PM* will in turn examine its own state of completion (i.e., ability to decide if there is enough information to process the request). If it has not reached a state of completion, it will pass the token as an input to the next *PM* (see Figure 3).

An *object* can be constrained under more than one policy, for example, a user may get past the **MAC** policy check, but may still need to be authorized again by other policies say, **RBAC** for allowing the requested operation to the object. For example, the sequence of checks could be: Can the user read secret information, is the user a doctor, is the patient assigned to the doctor for the operation/procedure.

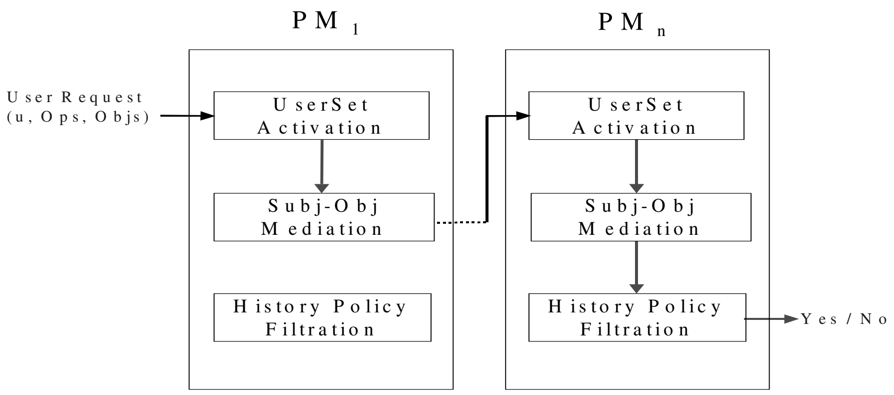

Fig. 3. *PM* combination

3 Conclusions

In this paper, we have demonstrated the concept of constructing universal policy architecture without detailed and formal specification. The issue of formal specification will be the subject of our planned future research. *PM* functions are executed through sequential phases (as described in Section 2) without recursive operations in the *PM* functions; therefore it is computable in polynomial time. Since *PM*'s database is always consistent (no conflict mapping of relations), users' access requests are handled conclusively; that is, they are either authorized or denied.

PM has significant theoretical and practical implications; it is an architecture that is capable of implementing virtually any access control policy. It also helps to promote interoperability and the use of innovative tools for policy generation and visualization that can be built on top of existing access control primitives and scaled to the largest virtual enterprise.

References

[AH96] Atluri V., Huang W., "An Authorization Model for Workflows". *Proceedings of the Fifth European Symposium on Research in Computer Security in Lecture Notes in Computer Science, No 1146*, September 1996.

[Ande72] Anderson J. P., "Computer Security Technology Planning Study," *ESD_TR_73-51, Vol. 1, Hanscom AFB, Mass.*, 1972.

[Bark97] Barkley J., "Comparing Simple Role Based Access Control Models and Access Control Lists", *Proceedings of the Second ACM Workshop on Role-Based Access Control*, November 1997, page 127-132.

[Biba77] Biba K. J., "Integrity Considerations for Secure Computer Systems," *ESD-TR-76-372, USAF Electronic Systems Division*, Bedford, Mass., April 1977.

[BL73] Bell D.E., and Lapadula L. J., "Secure Computer Systems: Mathematical Foundations and Model," *M74-244, MITRE Corp.*, Bedford, Mass., 1973.

[BNCW89] Brewer, D., Nash, M. "The Chinese Wall Security Policy." *Proc IEEE Symp Security & Privacy, IEEE Comp Soc Press,* 1989, pp 206-214.

[CW87] Clark D. D., and Wilson D. R., "A Comparison of Commercial and Military Security Policies," *Proc. of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, 1987, pp.184-194

[CWEO89] Clark D. D., and Wilson D. R., "Evolution of a Model for Computer Integrity", *NIST Special Publication 500-168, Appendix A*, September 1989

[FCK95] Ferraiolo D. F., Cugini J. A., Kuhn D. R., "Role-Based Access Control (RBAC): Features and Motivations", *Proc. of the 11th Annual Conference on Computer Security Applications. IEEE Computer Society Press*, Los Alamitos, CA. 1995. [GGF98] Gligor V. D., Gavrila S. I., Ferraiolo D., "On the Formal Definition of Separation-of-Duty Policies and their Composition", In *IEEE Symposium on Computer Security and Privacy*, April 1998.

[HGPS99] Hale J., Galiasso P., Papa M., Shenoi S., "Security Policy Coordination for Heterogeneous Information Systems", *Proc. 15th Annual Computer Security Applications Conference, Applied Computer Security Associates*, December 1999.

[ISW97] Irvine C. E., Stemp R., Warren D. F., "Teaching Introductory Computer Security at a Department of Defense University*", Naval Postgraduate School* Monterey, California, NPS-CS-97-002, April 1997

[JSS97] Jajodia S., Samarati P., and Subrahmanian V. S., ‘‘A Logical Language for Expressing Authorizations,” *Proc. IEEE Symp,* Oakland, Calif., May 1997.

[JSSB97] Jajodia S., Sammarati P., Subrahmanian V. S., and Bertino E., "A Unified Frame Work for Enforcing Multiple Access Control Policies*", Proc. ACM SIGMOD Conf. On Management of Data*, Tucson, AZ, May 1997.

[NCSC87] National Computer Security Center (NCSC). "A GUIDE TO UNDERSTANDING DISCRETIONARY ACCESS CONTROL IN TRUSTED SYSTEM", *Report NSCD-TG-003 Version1*, 30 September 1987.

[NCSC91] National Computer Security Center, "Integrity in Automated information System", *C Technical Report 79-91, Library No. S237,254*, September 1991.

[NCSC98] National Computer Security Center, "1998 Evaluated Products List", *Washington, D.C., U.S. Government Printing Office*.

[Sand93] Sandhu R. S., "Lattice-Based Access Control Models", *IEEE Computer, Volume 26*, Number 11, November 1993, page 9-19.

[SCFY96] Sandhu R. S., Coyne E. J., Feinstein H. L., and Youman C. E., "Role-Based Access Control Models", *IEEE Computer, Volume 29, Number 2*, February 1996, page 38-47.

[Spencer et al99] Spencer R., Smalley S., Loscocco P., Hibler M., Andersen D., and Lepreau J., "The Flask Security Architecture: System Support for Diverse Security Policies", *http://www.cs.utah.edu/fluz/flask*, July 1999.

[SZ97] Simon R.T., and Zurko M. E., "Separation of Duty in Role-Based Environments," *Proc. of the Computer Security Foundations Workshop X*, Rockport, Massachusetts, June 1997.