Enforceability vs. Accountability in Electronic Policies

Travis D. Breaux and Annie I. Antón
North Carolina State University
Raleigh, North Carolina, USA
{tdbreaux, aianton}@eos.ncsu.edu

Clare-Marie Karat and John Karat
IBM TJ Watson Research Center
Hawthorne, New York, USA
{ckarat, jkarat}@us.ibm.com

***Abstract**: Laws, regulations, policies and standards are increasing the requirements complexity of software systems that ensure information resources are both available and protected. To motivate discussions as to how current policy models can address this problem, we surveyed several regulations, standards and organizational security policies to identify how elements in these documents affect both personnel responsibilities and software system security. We present a resulting taxonomy that distinguishes between enforceable and accountable policies and we discuss the value of both in achieving compliance.*

1. Introduction

Risk and compliance are providing new motivation for incorporating personnel responsibilities and non-functional requirements into electronic policies that govern software systems. Risk is the possibility that an organization will suffer harm or loss while compliance is the ability to hold a defensible position in a court of law. To demonstrate compliance, an organization must not only show which restrictions are in place to protect resources, but also how and why resources are used in their business processes. In terms of systems, this information is typically captured in policies and software requirements documents and elaborated in software design. However, using electronic policies similar to those proposed by Moffett and Sloman [7], this information can be distinguished within systems as objects in electronic policies. Moffett has more recently championed the idea of including high-level policies such as requirements in electronic policies [6].

The remainder of the paper is organized as follows: Section 2 provides the background for this work; Section 3 presents the definitions in our taxonomy distinguishing between enforceable and accountable policies; and Section 4 discusses related work.

2. Background

To develop an understanding of the relationship between organizational security policies and federal law, regulations and standards, we analyzed such documents to identify policy objects, e.g. rights and obligations (see Table 1). Guided by the *Introduction to Computer Security: The NIST Handbook* (NIST SP-800-12), we compared laws, regulations and standards according to their policy influence in three policy scopes: *Program Policies (**P**)* define high-level security goals, security personnel and their responsibilities; Issue Policies *(**I**)* address a single legal or technical security issue such as properly handling financial or health care information, contingency planning, or remote connectivity; and System Policies *(**S**)* are low-level technical policies that describe how to configure specific systems and applications.

		Policy Scope		
		P	***I***	***S***
Laws and Regulations	FISMA	✓		
	SOX	✓	✓	
	GLBA	✓	✓	
	HIPAA	✓	✓	✓
Standards	ISO 15408/CC			✓
	ISO 17799	✓	✓	
	NIST 800-12	✓	✓	
	NIST 800-27			✓

Table 1: Policy Influence from Laws, Regulations, Standards

The laws, regulations and standards we reviewed (see Table 1) contain several notable overlaps. For example, the U.S. *Federal Information Security Management Act* (FISMA), the *NIST Handbook* (NIST SP-800-12) and the ISO standard *Code of Practice for Information Security Management* (ISO 17799) all provide policy guidance to security programs. However, we observed that regulations, which are typically domain-sensitive, generally influence issue policies; for example, privacy in information sharing practices in the domains of healthcare (HIPAA) and finance (GLBA, SOX). Notably, HIPAA affects all policy scopes and even requires organizations to use role-based access control (RBAC) in systems. Finally, the *Common Criteria* (ISO 15408) and *Engineering Principles for Information Technology Security* (NIST SP-800-27) govern the software development process rather than policies in system administration.

Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)
0-7695-2598-9/06 $20.00 © 2006 **IEEE**

In addition to the aforementioned laws, regulations and standards, we analyzed three organizational security policies from large organizations in finance, government and technology. Two of these policies have been promoted as best-of-breed policies; the government policy by NIST [12] and the technology policy by a security consulting service owned by a Fortune 500 company. Each policy contains over 500 pages that cover program, issue, and system policy scopes. We chose to analyze the technology policy in detail since it serves the largest of the three organizations and it is actively used in security consulting services. From this policy, we categorized individual statements describing personnel responsibilities into the following four categories:

- **Classification:** Responsibilities to classify people and resources. Classes include roles for users, confidentiality for information flows, or valid purposes for data and application use.
- **Notification:** Responsibilities to notify personnel of security-related events such as new or existing vulnerabilities.
- **Review/Audit:** Responsibilities to review permissions and obligations to ensure minimal access to resources and evaluate compliance.
- **Documentation:** Responsibilities to document security-related decisions, such as granting permissions and assigning or implementing obligations.

The detailed classification of responsibilities from the analyzed document included only the program policy portion that accounts for less than 11% of the whole organizational security policy; the remaining 89% includes only issue and system policies. Our analysis suggests that program policies define an organization's security program from high-level goals and obligations that are delegated down a management hierarchy (e.g., from managers to subordinates) and are incrementally refined and implemented along the way as system policies. Issue policies are defined by specialists on a per-issue or per-regulation basis and distributed across departments in an organization.

3. Taxonomy of Policy Objects

From our analysis, we identified three types of policy elements that cover program, issue and system policies: obligations, recommendations and permissions (see Figure 1). *Obligations* describe the actions principals must perform and are distinguished in two ways: an obligation is called a *requirement* if the principal performing the action is a system or process; otherwise, the principal is a person and the obligation is called a *responsibility*. Dynamic

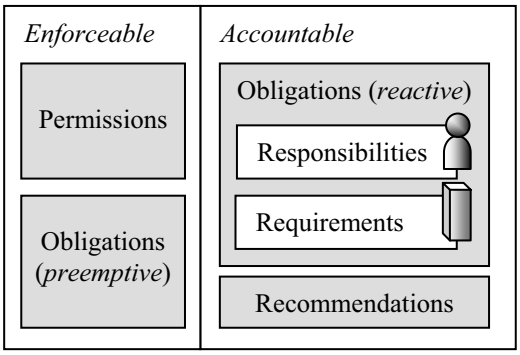

Figure 1: Policy Elements

requirements specify a degree of variability that is implemented by design using configuration directives such as changing parameters in a configuration file or database table, manipulating controls in a user interface or recompiling components from source code. Legacy systems have fixed or static requirements that cause these systems to dictate policy to an organization. In general, all principals are expected to comply with obligations. If a principal is non-compliant, then a *penalty* or *sanction* is imposed on the principal to compel compliance or to limit their non-compliant behavior. Obligations without penalties are called *recommendations*. Recommendations are implemented at the discretion of principals to increase security while offering principals flexibility in unforeseeable situations. Finally, *permissions* define allowable actions to facilitate organizational business practices.

We formalize obligations, recommendations and permissions using rules expressed as the triple $R_x = \langle C_x, t_x, f_x \rangle$ for a set C_x of constraints and, if every constraint in C_x is satisfied at some evaluation time, the action t_x is executed; otherwise the action f_x is executed. The subscript of the rule will later be used to distinguish between the conditions and actions of different rules. Obligations, recommendations and permissions are assigned to principals using constraints in C_x. Constraints may identify principals directly using a unique identifier or indirectly using roles [10] or context constraints [11]. Whereas direct assignment can occur offline, indirect assignment may require a runtime environment to evaluate constraints. Our previous work further shows that constraints on permissions and obligations can have deep structures [2, 3] that specify *when*, *how*, or *why* a principal is permitted or required to act. Rules or conditional actions have been proposed as a means to implement advanced review/ audit obligations [9]. We now discuss the anatomy of permissions, obligations and recommendations in terms of rules (see Figure 2).

Permissions are expressed using a simple rule R_P in which satisfying the *pre-condition* set C_P executes the action t_P to permit a principal to perform some action on an object. Example actions and objects include

reading or writing a file in a directory, inserting a record into a database, or transmitting packets on a network route. On the other hand, obligations are defined by three rules $\langle R_O, R_A, R_S \rangle$ for a *pre-condition* set C_O of constraints, which when all are satisfied, the action t_O is executed causing the principal to become obligated; the *achievement* set C_A of constraints that must be satisfied while the principal is obligated; and a *penalty* set C_S of constraints that when all are satisfied executes the action t_S imposing a penalty on the principal. Principals may require specific permissions to feasibly satisfy their obligations.

Obligations may be *maintenance* or *achievement* goals [1]. Maintenance goals require principals to maintain properties within a system by continually satisfying the constraints in the achievement set; depicted by the sequence of $t_O \rightarrow t_A$ invocations for an obligation in Figure 2. In this case, the penalty set C_S contains constraints that are satisfied by not satisfying constraints in the achievement set C_A. Example maintenance goals include monitoring system and network performance within runtime limits (e.g., number of errors, data loss, downtime) and can serve an important role in providing an overview of security. Alternatively, achievement goals require principals to perform actions to achieve a system state. In this case, the penalty set C_S contains constraints that are satisfied by passing a virtual or physical limit such as a deadline or resource quota.

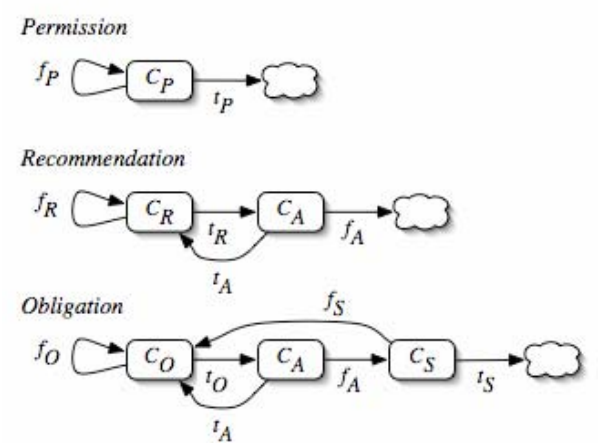

Figure 2: Anatomy of Policy Objects

Recommendations are like obligations but without penalties imposed on principals who do not maintain or achieve recommended states. Recommendations are defined by two rules $\langle R_R, R_A \rangle$ for a *pre-condition* set C_R of constraints that when all are satisfied executes the action t_R assigning the recommendation to a principal and an *achievement* set C_A of constraints that should be satisfied (maintained or achieved) to comply with the recommendation.

3.1 Enforceable vs. Accountable Policies

An *enforceable* policy requires a pre-emptive mechanism to irrefutably constrain or compel a principal's actions. Permissions are always enforceable since principals only receive permission after satisfying pre-conditions, whereas obligations are enforceable if they satisfy two conditions: 1) the satisfaction of the obligation is a constraint in a permission and 2) the performance of the action governed by the permission is the only means to violate the obligation. For example, an enforceable obligation includes requiring a principal to classify resources by level of confidentiality before they are permitted to share those resources. The obligation "to classify resources" is a constraint in the permission "to share resources" that is violated only if the obligated principal shares unclassified resources.

An *accountable* policy, on the other hand, only requires a reactive mechanism to determine if a principal is compliant. While penalties often compel principals to satisfy their obligations, they do not guarantee principals will do so. In fact, at the moment a principal is penalized for violating an obligation the system is already in an undesirable state. Despite this fact, penalties may include revoking permissions from principals to limit the impact of their non-compliance. Furthermore, recording instances of non-compliance provides important documentation that may be used to motivate restructuring accountable policies into enforceable policies or acquiring new technology to implement enforceable policies.

In some instances, the lack of available preemptive mechanisms to replace reactive mechanisms is only a technological limitation. For example, consider a quota policy governing a principal with permission to create files and write data to files in a file system. The intent of the policy is to limit the principal's ability to write data in excess of a pre-specified limit or quota using an obligation. Some file systems enforce this policy by tracking the principal's disk usage and coupling it with a pre-condition in the principal's write permission. However, in other file systems, the access control framework does not support such pre-conditions. In this situation, a maintenance obligation that uses a reactive mechanism may compel a principal to keep their disk usage under quota. The obligation is achieved when the principal is under quota and the penalty involves revoking the principal's write permission after their usage has exceeded quota. Another action must be executed to reinstate the principal's write permission when they *recomply* with the obligation. Accountable obligations can be used when the technology necessary to enforce policies cannot be adapted to legacy software systems.

Obligations including non-functional requirements or personnel responsibilities outside the scope of software systems are accountable through testimony. Non-functional requirements are properties in a software system to be achieved that, unlike constraints, are not directly testable such as safety, dependability, or confidentiality. Regardless, these obligations may still have satisfiable constraints to detect violations and penalize the principal. To satisfy the achievement set in this type of obligation, the principal must provide satisfactory testimony. If the obligation is a personnel responsibility outside the system scope, the testimony may be a single Boolean constraint satisfied by user feedback; for example, when a user accepts Terms and Conditions (TOC) before using a service or logging into a system. On the other hand, principals must provide structured testimony called a strategy to satisfy non-functional requirements. *Strategies* are a set of permissions and obligations that the principal believes will satisfy the non-functional requirement. There can be multiple strategies to implement a single requirement, which makes comparing strategies an important goal to evaluate their effectiveness. If the non-functional requirement is violated, a regulator or compliance officer may compare the strategy with best-practices to evaluate due-diligence. Electronically documenting these types of obligations is necessary to challenge a principal's claim when seeking to improve security or assign liability after a non-compliance event has been detected.

4. Related Work

Several policy models and languages interact with the distinction of enforceable vs. accountable policies. Park and Sandhu discuss the enforceable *pre-obligations* and accountable *ongoing-obligations* or maintenance obligations in $UCON_{ABC}$ [8]; however, it is unclear where accountable achievement obligations, possibly their notion of *duties*, fit into $UCON_{ABC}$. In the LGI model, Minsky and Ungureanu consider all obligations with sanctions *enforceable* [5], despite the lack of guarantees that a principal will satisfy the obligation which we classify as an accountable obligation. The Ponder language does not distinguish between accountable and enforceable obligations [4], though it appears this distinction is easily made in Ponder. The XACML language for access control includes obligations performed "in conjunction" with an authorization [13]; these obligations may be either enforceable, if they pre-condition the authorization, or accountable, if they follow the authorization. Since the exact order is unspecified we can only assume these obligations are accountable and that enforceable obligations will appear as conditions in XACML authorizations. Finally, unlike $UCON_{ABC}$ and LGI, Ponder and XACML do not explicitly account for sanctions or penalties imposed after violating accountable obligations.

Acknowledgements

The analysis leading up to this paper, presented in Section 2, began as collaboration between Travis D. Breaux, Clare-Marie Karat and John Karat in summer 2005 at the IBM Thomas J. Watson Research Center. The technical details in Section 3 were later developed by Travis D. Breaux and Annie I. Antón at NCSU and funded by the NSF grant *ITR: Encoding Rights, Permissions and Obligations: Privacy Policy Specification and Compliance* (NSF #032-5269).

References

[1] A.I. Antón, **"**Goal Identification and Refinement in the Specification of Software-Based Information Systems." Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1997.

[2] T.D. Breaux, A.I. Antón, "Deriving Semantic Models from Privacy Policies." In Proc. *IEEE 6th Workshop on Policies for Distributed Systems and Networks*, Stockholm, Sweden, pp. 67-76, 2005.

[3] T.D. Breaux, A.I. Antón, "Analyzing Goal Semantics for Rights, Permissions, and Obligations." In Proc. *IEEE 13th Requirements Engineering Conference*, Paris, France, pp. 177-186, 2005.

[4] N. Damianou, N. Dulay, E. Lupu, M. Sloman, "The Ponder Policy Language." In Proc. Work. Policies for Dist. Sys. and Nets., Bristol, UK, pp. 29-31, 2001.

[5] N.H. Minksy, V. Ungureanu, "Law-Governed Interaction: a Coordination and Control Mechanism for Heterogeneous Distributed Systems." *ACM Trans. on Soft. Engr. and Meth.*, 9(3), pp. 273-305, 2000.

[6] J.D. Moffett, "Requirements and Policies." In Proc. *Policy Workshop*, Bristol, U.K., 1999.

[7] J.D. Moffett, M.S. Sloman, "The Representation of Policies as System Objects." In Proc. *Conf. on Organizational Computing Systems*, Atlanta, Georgia, USA, pp.171-184, 1991.

[8] J.S. Park, R. Sandhu, "The $UCON_{ABC}$ Usage Control Model" *ACM Trans. on Information and System Security*, 7(1), pp. 128–174, 2004.

[9] T. Ryutov, C. Neuman, "The Specification and Enforcement of Advanced Security Policies" In Proc. *3rd Int'l Work. on Policies for Dis. Sys. and Net.*, Monterey, CA, USA, p. 128, 2002.

[10] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, "Role-based Access Control Models." IEEE Computer, 29(2), pp. 38–47, 1996.

[11] M. Strembeck, G. Neumann, "An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments." *ACM Trans. on Info. Sys. Security*, 7(3), pp. 392-427, 2004.

[12] "Information Security Policies and Procedures," Overseas Private Investment Corporation, Washington, D.C., USA, 2004.

[13] T. Moses (ed.) eXtensible Access Control Markup Language (XACML), version 2.0, Oasis Standard. http://xml.coverpages.org/xacml.html