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 and standards are increasing the requirements complexity of software systems that ensure information resources are both available and protected. To accommodate these requirements and demonstrate compliance, we extend accountability in software systems to include personnel responsibilities as they interact with access control and authorization mechanisms. To this end, we distinguish between enforceable and accountable security policies and show the value of both in achieving compliance. We propose a policy model that leverages resource ownership to build accountability across permissions and obligations. The model accounts for the authorized delegation of permissions and obligations as well as the decisions made by authorized personnel when they interpret and refine high-level goals into permissions and obligations that satisfy those goals. Regulators and compliance officers can use the model to determine both how and why a particular resource is used to evaluate risk and increase security. We motivate our proposed model by analyzing security program, technical issue and system policies and standards from the ISO and NIST and U.S. regulations governing healthcare and finance.*

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. Risk and compliance require a broader definition of *accountability* beyond the traditional scope of access control and authorization. In the broader definition, organizations must demonstrate what restrictions are in place to protect resources, as well as how and why resources are used in an organization. This information is typically captured in software requirements documents and elaborated in software design. Using electronic policies similar to those proposed by Moffett and Sloman [25], this information can be integrated with an access control and configuration management framework to ensure compliance and accountability.

This paper proposes an enterprise-wide policy model that addresses accountability through ownership, permissions and obligations. Ownership allows those primarily responsible for resources to delegate responsibility for managing resources without delegating accountability for those resources. Where permissions describe what people and systems are permitted to do, obligations describe what people and systems must do, including personnel responsibilities that are typically not part of software system design. Laws, regulations and standards require that these obligations be accountable in the context of access control and authorization but current software systems exhibit limited support for this. In this work, we address policies in system administration and not policies governing the software development process.

The remainder of the paper is organized as follows: Section 2 provides the background and motivation for this work; Section 3 provides a review of related work in policy models and frameworks; Section 4 presents a scenario to contextualize remaining sections; Section 5 presents the main definitions in our policy model distinguishing between enforceable and accountable policies; Section 6 defines constraints motivated by our analysis; in Section 7 we discuss related work; and, finally, in Section 8 we discuss our model in the context of four categories of personnel responsibilities.

2. Background and Motivation

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. responsibilities and access control rules. Guided by the *Introduction to Computer Security: The NIST Handbook* (SP-800-12), we compared laws, regulations and standards according to their policy influence in Program Policies ***(P)***, Issue Policies ***(I)***, and System Policies ***(S)***; Table 1 shows checkmarks where each document influences these three policy scopes. *Program policies* ***(P)*** define high-level security goals, security personnel, and personnel responsibilities needed to implement a security program. *Issue policies* ***(I)*** are high-level policies that address a single legal or technical security issue such as properly handling financial or health information, contingency planning, patch-management or remote connectivity. *System*

*policies (**S**)* are low-level technical policies that describe how to configure specific systems and applications.

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 *Introduction to Computer Security: The NIST Handbook* (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 the influences on issue policies from regulations to be domain-sensitive; for example, privacy requirements for information sharing practices in healthcare (HIPAA) and finance (GLBA, SOX). Notably, HIPAA affects all policy scopes and requires organizations to use role-based access control (RBAC) in systems. Finally, the *Common Criteria* (ISO 15408) and *Engineering Principles for Information Technology Security* (SP-800-27) govern the software development process rather than policies in system administration.

		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

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 [37] 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 which 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. Where security reduces risk through obligations, an organization's business needs require flexibility through increased access. To accommodate the scope of organizational security policies in the context of laws, regulations and standards, we propose a policy model that emphasizes accountability by integrating ownership and personnel responsibilities with permissions.

3. Policy Models and Frameworks

Research in policy models and frameworks has been shifting focus to include broader issues in organizational management. We see evidence for this shift in both access control frameworks limited to permissions [8, 26, 27, 31] and more progressive approaches that also include obligations and delegation [16, 20, 23, 28]. Permissions describe what people and systems are permitted to do while obligations describe what people and systems must do given certain restrictions or constraints. In delegation, a person delegates their authority or responsibility to another person; the latter person acts on behalf of the former.

Role-based access control (RBAC) is a popular framework in which a role is defined as a job function and permissions are associated with roles [31]. Recent work shows RBAC with delegation [26, 27] and temporal constraints [7, 8] necessary to integrate aspects of management hierarchy and business context from large organizations. However, the relationship between roles, obligations and permissions is not clear.

A role is defined as a job function yet popular examples show roles as job titles [8, 26, 27, 31], which arguably represent *collections* of job functions or obligations. Our analysis in security policies and legislative requirements in HIPAA [35], however, require that permissions be assigned based on specific obligations. These obligations are not sufficiently distinguished by unique job titles. In addition, role hierarchies are insufficient to model management hierarchies since they do not distinguish between authority and mandate, as we do in Section 6.

Several policy languages represent obligations. Moffett and Sloman present a definition for delegating permissions and obligations [25] implemented in the Ponder language [15, 16]. The delegated responsibility fulfills a broader purpose, justification or cause implied by the act of delegation. While this relationship is consistently found in delegation, it also appears in situations when a person refines or implements their own obligations. We expand upon this distinction called instrumentation in Sections 6. Ponder does not represent requirements per say, but Moffett has since championed the need to include requirements in the scope of obligations [24]. In our analysis, we observed that modeling both system requirements and personnel responsibilities are necessary to ensure compliance. Finally, the EPAL language associates data access with intended use or purpose (e.g., marketing, payment, etc.) as a precondition or obligation [20]. The policy languages KAoS [33], REI [19] and P3P [14] model obligations but focus on concerns in web infrastructure and do not address administrative aspects of obligations.

Cholvy et al. define responsibility (obligation) as avoiding penalty states or providing notification or an account for one's actions [13]. We observed both distinctions in our analysis and incorporate them into our policy model. Minsky and Ungureanu define penalties as sanctions in the LGI model that involve actions performed after the expiration of a deadline [23]. The $UCON_{ABC}$ model by Park and Sandhu is limited to system requirements yet includes consequences if requirements are violated [28]. We observed that both accounting for unsatisfied obligations and imposing penalties are prescribed in organizational security policies, standards and law.

Rees et al. describe an information security framework called PFIRES that combines policy assessment and review to mitigate risk in organizational security [29]. The framework does not address policies as system objects, per se. However, they propose improving security by passing messages between personnel to communicate obligations and monitor compliance; these messages could be structured policy objects for accountability.

4. Policy Scenario

Consider a scenario where the Chief Security Officer (CSO) has been assigned the high-level security goal (a non-functional requirement) NFR_1 = "to ensure that information is secure." The CSO implements NFR_1 by assigning several new non-functional requirements including NFR_2 "ensure internal communications are confidential" to their IT security manager in charge of network security. The IT security manager responds by identifying all modes of "internal communications" relevant to satisfying their new obligation. As a result, the manager identifies internal web and e-mail servers among others that use TCP/IP network connections to share information between internal systems. The manager, with both authority over who administers these servers and knowledge of available security mechanisms in these systems, implements their obligation by assigning new functional requirements including FR_1 "ensure web servers use SSL for internal connections" and FR_2 "ensure mail servers use TLS for internal connections" to only relevant system administrators in different departments. One system administrator responsible for a mail server running Linux receives FR_2 and implements the requirement with a series of configuration directives which they apply to their system: FR_1 = "install latest OpenSSL libraries," FR_2 = "compile and configure Sendmail with TLS support", FR_3 = "generate X.509 certificates for Sendmail," etc.

At each level in the delegation hierarchy of obligations, the manager knows *what* goal their subordinate must achieve but may not have the technical knowledge to know *how* the subordinate will achieve this goal. Each person who creates an obligation owns that obligation and the decisions to implement an obligation by creating new obligations (or permissions) are called instrumentations. Tracing permissions and obligations through ownership, instrumentation and delegation are necessary to quickly identify how and why vulnerabilities are addressed to reduce risk and ensure compliance.

5. Security, Risk and Compliance

Security should be commensurate with risk. Increasing security reduces risk and reduces independence to conduct transactions in an organization. While transactions which make systems vulnerable are undesirable, those transactions which include innovation and crisis management are highly desirable yet unforeseeable. Consequently, policies must evolve to accommodate an organization's need for security and autonomy in a dynamic environment.

From our analysis, we identified three types of policy elements that support this evolution: obligations, recommendations and permissions (see Figure 1).

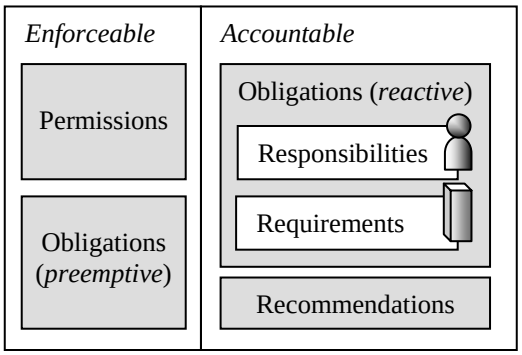

Figure 1: Policy Elements

Obligations describe the actions principals must perform and are distinguished in two ways: an obligation is called a *requirement* if the principal is a system or process; otherwise, the principal is a person and the obligation is called a *responsibility*. Requirements are implemented using configuration directives implemented by 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 requirements causing these systems to dictate policy to an organization. In general, obligations are used to establish baseline or minimal security since all principals are expected to comply with obligations. If a principal is non-compliant, then a *penalty* 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 baseline security while offering principals flexibility in unusual circumstances. Finally, *permissions* define allowable actions to facilitate organizational practices.

We formally define obligations, recommendations and permissions by rules expressed as triples $\langle C, x_T, x_F \rangle$ for a set C of constraints and if every constraint in C is satisfied execute the action x_T , otherwise execute the action x_F. Rules are evaluated using an application context E containing evidence from past and present system states in the form of key, value pairs (k, v). The values for each key may be obtained by executing non-trivial functions. Constraints in C are defined by the triple $\langle k, f, K \rangle$ for a key k, Boolean function f, and set K of constants. The constraint evaluates to true if $f(V, K)$ is true for $V = \{v \mid (k, v) \in E\}$. For example, a role-based mechanism defined by the constraint $(role, f_{role}, K)$ where $K = \{c\}$ for a role constant c is evaluated as follows: obtain the set of roles $R = \{r \mid (role, r) \in E\}$ for a principal's application context E; evaluate $f(R, K)$ by searching roles $r \in R$ and return true if the role constant c is equivalent to role r or its parent roles in the transitive closure of the role hierarchy, otherwise return false.

It is important to note that principals may be indirectly assigned permissions and obligations using constraints. Our previous work shows that these constraints have deep structures that require abstract mechanisms [9, 10, 11]. These constraints specify *when*, *how*, or *why* a principal is executing an action regardless of *who* the principal is. Alternatively, an object may be indirectly referenced by its attributes which may be shared by multiple objects. In both regards, these permissions and obligations are dynamically assigned to principals based on evidence in a specific context. For example, a role-based permission directly identifies only an authorized role and indirectly users who may change roles periodically. For this reason, we use the terms *assign* and *create* for permissions and obligations interchangeably; recognizing that assignment is abstract and may only be realized given a specific application context.

5.1 Permissions

Permissions are rules that determine when a principal is permitted to perform an action on an object. Formally, a permission $P = \langle C, x_T, x_F \rangle$ for a set

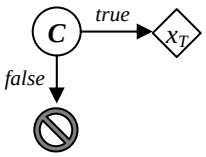

Figure 2: Anatomy of a Permission

C of constraints that, when all are satisfied, cause the action x_T to execute authorizing the principal p to perform an action a on an object o (see Figure 2). The action x_F denies permission. Evaluating a permission requires an application context E to minimally contain the intention, evidenced by $\{(principal, p), (action, a), (object, o)\} \subseteq E$. We assume a principal is denied the ability to perform an action if no permission granting the ability exists. Finally, the permission pool P^* contains all permissions within a single policy scope.

Example actions and objects include: a principal reads or writes to a *file* in a classified directory; selects, inserts or deletes a *record* from a database table; or routes *data* through a specific gateway. Alternatively, the objects may be permissions or obligations to be delegated or instrumented as described in section 6.

5.2 Obligations and Recommendations

Obligations are rules that require a principal to maintain or achieve a state within a software system. Presumably, the principal will perform or avoid performing certain actions to satisfy an obligation. Formally, an obligation is defined by three rules $\langle C, x_T, x_F\rangle$, $\langle A, a_T, a_F\rangle$, $\langle S, s_T, s_F\rangle$ for a condition set C of constraints (pre-conditions), which when all are satisfied, the action x_T is executed causing the principal to become obligated; an achievement set A of constraints that must be satisfied (maintained or achieved) while the principal is obligated; and a penalty set S of constraints that when all are satisfied executes the action s_T imposing a penalty on the principal. For convenience, we define an obligation $O = \langle C, A, S\rangle$ summarizing the constraint sets of the three rules. Valid constraints include temporal constraints between events or the effects of unspecified actions. If the events are actions performed by the principal within the scope of the system, the principal must have permissions to perform those actions in order to satisfy the obligation. Finally, O^* contains all the obligations defined within a single policy scope.

Obligations may be *maintenance* or *achievement* goals [1] (see Figure 3). Maintenance goals require

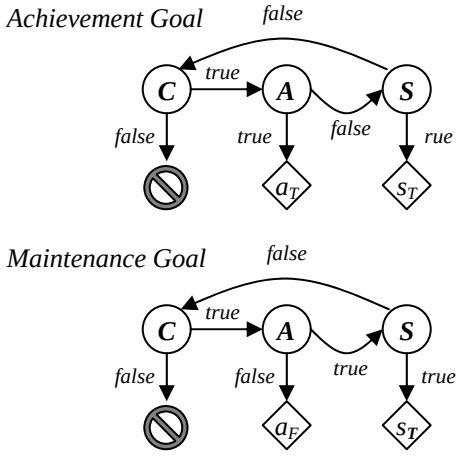

Figure 3: Anatomy of Obligations

principals to maintain properties within a system demonstrated by continually satisfying the constraints in achievement set A for an obligation O. Maintenance goals include monitoring performance within runtime limits (e.g., number of errors, data loss, downtime, or unauthorized disclosures) as part of a broad security overview. Formally, for a maintenance goal O, there is an unsatisfied constraint in the achievement set A if and only if all constraints in the penalty set S are satisfied. Notably, $a_F = s_T$ in a maintenance goal. On the other hand, achievement goals require principals to perform actions to achieve a system state. The penalty set S contains constraints that are satisfied by passing a virtual or physical limit such as a deadline or resource quota.

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

5.3 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 while 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, a principal is obligated 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. Formally, an obligation O is enforceable if there exists a permission P such that $A \subseteq C$ for $A \in O$ and $C \in P$ and the constraints in the penalty set $S \in O$ are satisfiable only by executing the governed action a for $(action, a) \subseteq E$.

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, obligations are *at least* accountable since at the moment a principal is penalized 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 constraint in the principal's write permission. However, in other file systems, the access control framework does not support such constraints. 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. Obligations should be used when the technology 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 testable. Regardless, these obligations may still have satisfiable constraints to detect violations and penalize the principal. For this type of obligation, a principal satisfies all constraints in the achievement set A by providing structured testimony such as a *strategy* described in Section 6. If the obligation is violated, a regulator or compliance officer may compare the strategy with best-practices. On the other hand, 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. Electronically documenting these types of obligations is necessary to challenge a principal's implementation to improve security or assign liability after a non-compliance event has already occurred.

6. Management and Accountability

Ensuring accountability requires documenting and maintaining all personnel decisions and relevant information used to allow or restrict access to resources. Permissions and obligations must be tightly coupled so that access to resources is traceable through personnel responsibilities and the authority to delegate those responsibilities. We propose several properties that must be preserved across the lifecycle of principals and objects. These properties allow users to determine which resources are accessed by whom and for what purpose. We address these properties in three management areas: ownership, instrumentation, and delegation. After defining these areas and relevant terms, we discuss these properties in delegation.

Ownership refers to the principal who is solely accountable and at least initially responsible for the proper management and security of objects they own. Ownership begins with the creation and ends with the destruction of an object. We define the ownership set N^* containing pairs (n, o) for principal owner n and object o. When an object is created, the system assigns the owner a set of maintenance permissions minimally including the permission to destroy the object. The system owns maintenance permissions and is responsible for destroying these permissions when an object is destroyed. The object type (e.g., permission, obligation, resource, etc.) determines how the object is maintained and which other maintenance permissions are assigned to the owner. Transferring ownership of an object to another principal requires transferring ownership of associated permissions and obligations governing that object and created by the owner. We do not consider the non-trivial downstream dependencies in instrumentation and delegation that occur when a principal destroys an object, however.

Instrumentation is the act of documenting both *how* and *why* permissions and obligations are assigned to principals. Obligations are instrumented with permissions and other obligations, called instruments, which answer *how* the instrumented obligation will be satisfied. The instrumented obligation answers *why* the instruments are assigned to principals. For example, in our scenario NFR_2 in part answers how NFR_1 will be satisfied while NFR_1 answers why NFR_2 has been assigned to the manager. We define the instrumentation set I^* containing pairs (O, I) for an obligation O instrumented by an instrument I which is either a permission or obligation. When a principal creates an obligation, the maintenance permission to instrument that obligation is assigned to the owner.

Obligations which are non-functional goals are not testable; however, they may be refined or partitioned by an instrument set called a *strategy*. Principals who develop strategies are documenting their assumptions with permissions and obligations which they claim satisfy the instrumented obligation. Strategies are tightly coupled with the obligations they instrument, so that satisfying the strategy satisfies the instrumented obligation. For an obligation O satisfied by a strategy $I_S = \{I \mid (O, I) \in I^*\}$, the achievement set $A \in O$ is satisfied if every achievement set $A_I \in I$ is satisfied for all $I \in I_S$ and the penalty set $S \in O$ is satisfied if any one penalty set $S_I \in I$ is satisfied for all $I \in I_S$.

Delegation is the act by which a principal (delegator) assigns permissions and obligations to other principals (delegatees). When a principal creates a resource, they receive a maintenance permission to create permissions for that resource. In some cases, it may be necessary for principals to allow other principals to create resource permissions by delegating the "permission to create permissions" to the other principals. We define the delegation set D^* containing permission pairs $\langle P, D \rangle$ for the permission P to create a permission which the delegator n assigns to the

delegatee as the new permission *D*. For the system *s*, we distinguish ownership by $\{(s, P), (n, D)\} \subseteq N^*$.

Principals exercise both their authority and their mandate when they delegate obligations. We distinguish *authority*, defined by the permission to delegate permissions and obligations, from *mandate*, defined by the valid obligation which a principal must instrument when delegating an obligation to their subordinate. We define the management set M^* containing the triples $\langle m, s, M \rangle$ for a principal manager *m*, principal subordinate *s* (person or system) and the set *M* of obligations such that the manager *m* is permitted to assign obligations to their subordinate *s* only by instrumenting obligations in *M*. The delegation of permissions and obligations are governed by separate management sets. We leave the complex semantic issues in determining whether a delegated obligation satisfies a particular mandate to the principals involved in the delegation.

To ensure accountability, we maintain properties in delegation that involve ownership and instrumentation. We separately consider properties when principals are: 1) delegating permission to access objects or delegating permission to create permissions; and 2) delegating obligations that are responsibilities or requirements. We assume that principals who instrument obligations have permission to do so.

6.1 Delegating Permissions

When a principal delegates a permission governing an object, we must ensure three properties: 1) the new permission instruments an existing obligation, 2) the permission is restricted to objects owned by the delegator and 3) the delegated permission may only narrow the scope of access. The first property is required in a fully accountable policy while the second and third properties result from using an abstract constraint mechanism.

In fully accountable policies, permissions are assigned to satisfy an obligation of the delegator, delegatee or another principal. The instrumentation set can account for this relationship; however, we enforce it by requiring that all permissions contain a constraint satisfied only if the instrumented obligation exists. When an obligation is revoked from a principal (e.g., destroyed), this constraint will no longer be satisfiable ensuring the permission becomes ineffective. Tightly coupling obligations with their instrumenting permissions supports *least privilege* and increases security in the event of ephemeral obligations. For each permission *P*, we define the constraint $\langle k, f, K \rangle$ where $(k, O) \in E$ for all $(O, P) \in I^*$ where obligation *O* is instrumented by the permission *P*; the constraint constants $K = \{O'\}$ for an obligation O' that the permission P must instrument; and *f(V, K)* returns true if $O' \in V$, otherwise it returns false.

Recall that in our rule-based approach, abstract mechanisms evaluate constraints in permissions, which permits defining permissions using classes of principals and objects. Since a permission can be defined for a class of objects, we ensure the scope of objects is limited to a specific owner. We define the constraint $\langle object, f, K \rangle$ where constraint constants $K = \{n\}$ for a principal owner *n*; the value set $V = \{o \mid (object, o) \in E\}$; and *f(V, K)* returns true only if $(n, o) \in N^*$ for $o \in V$. The constraint must persist for the lifetime of the permission. If the permission is delegated, then the new permission assigned to the delegatee must include the unedited constraint. If the permission is transferred to another owner, say as part of transferring ownership for the governed object, the constraint constant *n* must be updated for the new owner, accordingly.

When a principal delegates a permission *P* to create permissions for objects, the delegatee must only be permitted to narrow the scope of access in the permissions they create. For example, a delegator may wish to ensure that the delegatee creates permissions with a set of constraints C_1. In addition, the delegatee may wish to narrow the scope of access by adding permissions C_2. Ultimately, each permission created by the delegatee has the set *C* of conditions such that $C_1 \subseteq C$. We define the constraint $\langle k, f, K \rangle$ where $(k, c) \in E$ for all constraints $c \in C$ in the permission *P*; the constraint constants *K* contains the minimal set of constraints for the new permission; and *f(V, K)* is true only if $K \subseteq V$.

6.2 Delegating Obligations

When a principal delegates an obligation to a delegatee, we must ensure two properties: 1) the new obligation instruments an existing obligation within the delegator's mandate and 2) the principal owners of systems including applications and processes are obligated to satisfy requirements on those systems.

To ensure an obligation instruments an existing obligation, we can employ a constraint mechanism similar to the one in section 6.1 in the constraints of the delegated obligation. However, mandates further restrict such authority and require the instrumented obligation exists within a limited set called the mandate. For a manager *m* delegating an obligation *I* to a subordinate *s* under the mandate set *M*, we define the constraint $\langle principle, f, K \rangle$ where the constraint constants $K = \{m, I\}$ for the manager *m* and obligation *I*; the value set $V = \{p \mid (principal, p) \in E\}$; and *f(V, K)* returns true only if there exists $(O, I) \in I^*$ such that $O \in M$ for $\langle m, p, M \rangle \in M^*$. If the instrumented obligation is revoked or the mandate for the subordinate changes

and no longer includes the instrumented obligation, the delegated obligation will become ineffective.

System owners are initially responsible for implementing requirements, although in some cases, they may delegate these obligations to subordinates. The responsibility of a principal to achieve or maintain a requirement is defined by the same achievement set *A* and penalty set *S* for the requirement *O*. However, the condition set *C* for the responsibility obligates the owner of the system, whereas the requirement's condition set obligates the system, itself. We define the constraint ⟨*principle*, *f*, *K*⟩ where the constraint constants $K = \{s\}$ for a system *s*; the value set $V = \{p \mid (principal, p) \in E\}$; and *f(V, K)* returns true if $(p, s) \in N^*$, otherwise it returns false. If ownership is transferred to a new principal, the obligation to implement the requirement will obligate the new owner, accordingly.

7. Related Work

Several concepts in our policy model have previously appeared in requirements engineering and goal-based methods, access control in distributed systems, runtime and compile time configuration management, and policy compliance in general.

In requirements engineering, Antón et al. show how non-functional requirements can be implemented through functional requirements; a notion we capture in our definition of instrumentation in section 6 [2]. Mylopolous et al. propose the Secure Tropos framework for modeling ownership and delegation that defines obligations as trust in execution [17]. Secure Tropos models delegating permissions on condition of satisfying an obligation (enforceable obligations) and includes a design pattern for monitoring accountable obligations. Bandara et al. propose goal operationalization which includes delegation and instrumentation to refine high-level goals (obligations) using Event Calculus [3].

Permissions have received much attention in distributed systems as we now discuss. Delegating permissions generally means delegating the ability to create permissions. Barka and Sandhu extend this definition with the notion of *permanence* which we call transferring ownership, *totality* or the scope of access in delegated permissions, and cascading revocation in transitive delegation [5], which we do not address. Park et al. consider classification as a constraint in the context of role-based access control [27]. Bandmann et al. address delegating the ability to create permissions without delegating access to the resource [4] to ensure *least priviledge*. Park and Sandhu in the $UCON_{ABC}$ model identify obligations associated with consequences and obligations that precondition permissions [28], which we distinguish as *enforceable* obligations.

Review, audit and notification are emphasized in the broader context of policy and compliance. Schaad and Moffett formalize the t37 of review for delegated obligations [32]. Ryutov and Neumann describe the need for conditional actions to include event notifications [30]. Conditional actions are more general than permissions and obligations since they lack proper modality. In our experience, notifications are defined as obligations, but they may be expressed as conditional actions. Madigan et al describe a case study showing how combining obligations with procedures to implement those obligations improves policy compliance [21].

8. Discussion and Summary

Our policy model defines accountability through ownership, instrumentation and delegation and supports classification, notification, review/ audit and documentation as found in organizational security policies, regulations and standards.

Managers and resource owners are responsible for ensuring their subordinates and resources are properly classified. For example, principals are classified by the functions they perform (e.g., obligations), called *roles* [31, 36, 38]. Applications and data are classified by confidentiality, business need or intended purpose [35, 38]. Hosts, services and client applications are classified by vulnerability [22]. Constraint mechanisms use classes to restrict permissions and obligations. When the classification for principals or resources is changed, corresponding permissions and obligations are revoked or assigned based on the satisfiability of class-based constraints.

Security programs include obligations assigned to managers and resource owners to notify others of events including isolated security incidents or known vulnerabilities. Obligations can be used to monitor security requirements for incidents and send incident reports to resource owners and managers. Vulnerability announcements sent by third parties such as CERT or NIST can be conditionally assigned to managers and resource owners as new obligations.

Organizational security policies require that principals periodically review permissions to ensure that other principals have *least priviledge* to perform their job functions. Over time, the principal's job functions will change. For example, the principal completes a short-term project or the principal transfers to a new employee position. Regardless of the cause, the change in the principal's job function is evidence for restructuring their permissions. Instrumentation and mandates provides mechanisms to tightly couple permissions and obligations so that adding or removing

an obligation to/ from a principal immediately grants or revokes associated permissions (instruments).

Our model provides a multi-viewpoint solution to review and audit in security policies. At any moment, a principal in our model can view their obligations and identify the manager whose authority and mandate justifies the obligation. Resource owners can trace permissions on their resources to principals with access and the obligations of those principals that justify access. Managers can trace obligations they delegated to subordinates and evaluate the implementing strategy employed by those subordinates. The strategy can be compared with best-practices to improve security. Managers can also trace the permissions they delegated to other principals to evaluate the use of those permissions to balance risk with business needs. By highlighting ownership, authority and mandate, our model improves the ability of principals to review and audit policies in the security program.

The improved ability to audit the security program also satisfies the documentation requirement. Integrating non-traditional elements such as recommendations, responsibilities, and requirements with permissions in the access control framework provides documentation that is directly aligned with the runtime state of systems. Decisions to expose resources through permissions and protect resources through obligations are all documented and traceable between resources and principals. Cause, purpose or business needs as high-level responsibilities are directly mapped to low-level permissions and requirements.

Future work includes:

1) Determining to what extent policy is best implemented in an architecture vs. a policy language? The variety of components (e.g., file systems, databases, networks) that must interface to policies presents a challenge for the policy community. Each component dictates its own policy semantics governing behavior; therefore, should each component be designed to interface to a single policy language and policy engine or should a policy architecture use a pluggable design to configure otherwise naïve components or both?

2) A generalized framework is needed to support abstract constraint mechanisms. Many popular policy languages make assumptions about the structure of constraints (e.g., Boolean, temporal, nested structures) based on the scope of policies they address. However, the heterogeneity and momentum in developing new constraint mechanisms to accommodate new domain-specific problems requires increased flexibility during constraint specification in policies.

3) The relationship between requirements, policies and implementations shows that some requirements are realized as configurable parameters during runtime while others are realized as configurable components during compile time. The work done by Beigi et al. [6], Burgess and Ralston [12] and van der Hoek et al. [18] provides insight into policy-based configuration management, yet more work is needed to integrate requirements with configuration management and determine which approach is preferable in software architecture and design.

4) We are presently validating our policy model using HIPAA regulations. HIPAA delegates several permissions and obligations to individuals and health care managers and providers with constraints on events entailing the delegation of new rights and obligations. Using a request-for-services scenario between the individual and provider, we intend to show how ownership, instrumentation and delegation support accountability and compliance under HIPAA.

Acknowledgements

The analysis leading up to this paper began as collaboration between Travis D. Breaux, Clare-Marie Karat and John Karat in summer 2005 at IBM Thomas J. Watson Research Center. Discussion of that analysis appears in Section 2 and motivates the scenario in Section 3 and definitions in Section 4. The technical details in Section 5 and Section 6 for the policy model 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] A.I. Antón, W.M. McCracken, C. Potts. "Goal Decomposition and Scenario Analysis in Business Process Engineering." *Advanced Information Systems Engineering, 6th International Conference Proceedings (CAiSE `94)*, Utrecht, Netherlands, pp. 94-104, 6-10, 1994.

[3] A.K. Bandara, E.C. Lupu, J. Moffett, A. Russo, "A Goal-based Approach to Policy Refinement." In Proc. Policies for Dis. Sys. and Nets, Yorktown Heights, NY, USA, pp. 229-239, 2004.

[4] O. Bandmann, M. Dam, B.S. Firozabadi, "Constrained Delegation." In Proc. *IEEE Symp. on Security Privacy*, pp. 131-140, 2002.

[5] E. Barka, R. Sandhu, "Framework for Role-based Delegation Models." In Proc. 16th Annual Conf. on Computer Security Applications, pp. 168-176, 2000.

[6] M.S. Beigi, S. Calo, D. Verma, "Policy Transformation Techniques in Policy-based Systems Management" In Proc. *IEEE 5th Workshop on Policies for Distributed Systems and Networks*, Yorktown Heights, NY, pp. 13-22, 2004.

[7] E. Bertino, P.A. Bonatti, E. Ferrari, "TRBAC: A Temporal Role-Based Access Control Model" *ACM Trans. on Info. and Sys. Sec.*, 4(3), pp. 191-233, 2001.

[8] R. Bhatti, A. Gafoor, E. Bertino, J.B.D. Joshi, "X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-wide Access Control." *ACM Trans. Info. Sys. Security*, 8(2), pp. 187-227, 2005.

[9] 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.

[10] 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.

[11] T.D. Breaux, A.I. Antón, "Mining Rule Semantics to Understand Legislative Compliance." In Proc. *ACM Workshop on Privacy in Electronic Society*, Alexandria, Virginia, USA, pp. 51-54, 2005.

[12] M. Burgess, R. Ralston, "Distributed Resource Administration using Cfengine." Software-Practice and Experience, 27(9), pp. 1083-1101, 1997.

[13] L. Cholvy, F. Cuppens, C. Saurel, "Towards a Logical Formalization of Responsibility." In Proc. *6th Int'l Conf. on Aritficial Intelligence and Law*, Melbourne, Australia, pp. 233-242, 1997.

[14] L. Cranor, M. Langheinrich, M. Marchiori, M. Pressler-Marshall, J. Reagle, "The Platform for Privacy Preferences 1.0 (P3P1.0) Specification", W3C Recommendation, 2002.

[15] 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.

[16] N. Dulay, E. Lupu, M. Sloman, N. Damianou, "A Policy Deployment Model for the Ponder Language" In Proc. *IEEE/IFIP Int'l Sym. on Intenerated Net. Mgmt.* Seattle, WA, USA, pp. 529-543, 2001.

[17] P. Giorgini, F. Massacci, J. Mylopoulos, N. Zannone, "Modeling Security Requirements through Ownership, Permission and Delegation." In Proc. 13th IEEE Int'l Reqs. Engr. Conf., Paris, France, pp. 167-176, 2005.

[18] A. van der Hoek, A. Carzaniga, D. Heimbigner, A. Wolf, "A Testbed for Configuration Management Policy Programming." IEEE Trans. on Soft. Engr., 28(1), pp. 79-99, 2002.

[19] L. Kagal, T. Finn, A. Joshi, "A Policy Based Approach to Security for the Semantic Web" *In Proc. 2nd Int'l Semantic Web Conf.*, Sanibel Island, FL, USA, pp. 402–418, 2003.

[20] G. Karjoth, M. Schunter, M. Waidner, "Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data" *In Proc. 2nd Int'l Work. on Privacy Enhancing Tech.*, San Francisco, CA, USA, LNCS 2482, pp. 69–84, 2002.

[21] E.M. Madigan, C. Petrulich, K. Motuk, "The Cost of Non-Compliance: When Policies Fail" In Proc. *32nd ACM Conf. on User Services*, Baltimore, MD, USA, pp. 47-51, 2004.

[22] P. Mell, T. Grance, "*Use of the Common Vulnerabilities and Exposures (CVE) Vulnerability Naming Schemes*". *NIST SP-800-51*, Gaithersburg, MD, USA, 2002.

[23] 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.

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

[25] 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.

[26] S. Oh, R. Sandhu. "Role Administration: A Model for Role Administration Using Organization Structure." In Proc. 7th ACM Sym. on Access Control Models and Tech., Monterey, CA, USA, pp. 155–162, 2002.

[27] J.S. Park, K.P. Costello, T.M. Neven, J.A. Diosomito, "A Composite RBAC Approach for Large, Complex Organizations." In Proc. 9th ACM Sym. On Access Control Models and Technologies, Yorktown Heights, NY, USA, pp. 163-172, 2004.

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

[29] J. Rees, S. Bandyopadhyay, E.H. Spafford, "PFIRES: A Policy Framework for Information Security." Communications of the ACM, 46(7), pp. 101-106, 2003.

[30] 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.

[31] 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.

[32] A. Schaad, J.D. Moffett, "Delegation of Obligations." In Proc. *IEEE 3rd Policies for Dis. Sys. and Net.*, pp. 25-35, 2002.

[33] A. Uszok, J. Bradshaw, R. Jeffers, "KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services" *In Proc. 2nd Int'l Conf. on Trust Mgmt.*, Oxford, UK, pp. 16–26, 2004.

[34] "Evaluation Criteria for Information Technology Security", International Organization for Standards (ISO), ISO/IEC 15408, 1999.

[35] "Standards for Privacy of Individually Identifiable Health Information." 45 CFR Part 160, Part 164 Subpart E. In Federal Register, vol. 68, no. 34, February 20, 2003, pp. 8334 – 8381.

[36] "Standards for the Protection of Electronic Protected Health Information" 45 CFR Part 164, Subpart C. In Federal Register, 68(34), February 20, 2003, pp. 8334 – 8381.

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

[38] "An Introduction to Computer Security: the NIST Security Handbook," *NIST SP-800-12*, Gaithersburg, MD, USA, 1995.