A HIERARCHICAL RELEASE CONTROL POLICY FRAMEWORK

Chao Yao[1], William H. Winsborough[1] and Sushil Jajodia[1],[2]
[1] *Center for Secure Information Systems, George Mason University, Fairfax, VA 22030-4444;* [2] *The MITRE Corporation, 7515 Colshire Dr. Mclean, VA 22102-7508*
cyao@gmu.edu, wwinsborough@acm.org and jajodia@mitre.org

Abstract With increasing information exchange within and between organizations, it becomes increasingly unsatisfactory to depend solely on access control to meet confidentiality and other security needs. To better support the regulation of information flow, this paper presents a release control framework founded on a logical language. Release policies can be specified in a hierarchical manner, in the sense that each user, group, division and organization can specify their own policies, and these are combined by the framework in a manner that enables flexibility within the context of management oversight and regulation. In addition, the language can be used naturally to specify associated provisions (actions that must be undertaken before the release is permitted) and obligations (actions that are agreed will be taken after the release).

This paper also addresses issues arising due to the fact that a data object can be released from one entity to another in sequence, along a release path. We show how to test whether a given release specification satisfies given constraints on the release paths it authorizes. We also show how to find the best release paths from release specifications, based on weights specified by users. The factors affecting weights include the subjects through which a path passes, as well as the provisions and obligations that must be met to authorize each step in the path.

Keywords: Policy; Release Control; Access Control

Introduction

There are increasing needs for collaboration between organizations. In order to cooperate on common tasks, organizations need to release data that are not normally made available externally. For example, a hospital needs to release individual medical records to another institute to facilitate medical research. Such data release might compromise organizational confidentiality or personal privacy. Release control systems offer a means to regulate such data release.

The principle of release control is already in common use. When an organization publishes documents, its security officers may manually verify it does not contain sensitive information. The process is automated in some network firewalls. By including application-level inspection, a network firewall can use a "dirty-word" list to identify and intercept sensitive objects. No matter which release control mechanisms organizations use, they need to specify release policies.

Release control can often better manage information-flow than can access control. Once data is retrieved from a controlled data source, access control no longer assists in regulating its further dissemination. By contrast, as long as the data remains within a release-controlled environment, its further propagation can be regulated through release control.

This paper is concerned with how to specify release policies. We borrow some techniques introduced for use in access control, which regulates the direct access of users to data. However, we adapt those techniques and introduce new ones for use in release control. This is necessary because of the following three characteristics of data release that distinguish it from access control.

First, the role of the *sender* is instrumental. Different senders should have different privileges. Many senders may be permitted to release objects that are routinely made available to the receiver; however, it may be important to permit certain subjects to release objects to receivers that are normally not permitted to receive them. For example, a project leader may be permitted to send a project report to a customer, while ordinary project team members may not.

Second, while a policy may not allow a data object to be passed directly from one user s to another r, there could nevertheless exist a path composed of legal releases though which the object could be passed from s to r. A release control policy should be able to regulate such release paths; this would not be a meaningful objective for an access control policy.

Third, subjects and sensitive data typically belong to units within an organization. Thus, each unit has its own domain comprising data and users over which it should have some control. For example, an accounting department may own sensitive financial data and should be able to control to whom those data can be released. On the other hand, higher authorities within the organization should be able to override or otherwise compose the policies of lower authorities, although they are not concerned about most of the details of policies of the lower authorities.

For example, the accounting department could have policies that permit expense reports to be sent to all employees within the organization, while the organization director's policy stipulates that certain reports can be sent only to a few members of upper management; obviously, the policies of the organization should be able to override those of the accounting department.

Based on the above observations, this paper presents a hierarchical release control policy framework. We introduce a logical language that allows users to specify release policies in a flexible and hierarchical manner.

A novel feature of our framework is that it combines subpolicies defined by various authorities within an organization. That is, each unit defines its own subpolicies and these are then combined in a hierarchical manner. For instance, the policies of an organization can be formed by composing the policies of the departments in the organization in a specified way. We show that this hierarchical composition of policies can easily be expressed with our logical language.

When users specify release policies, they should be able to use not only positive authorizations, but also negative authorizations (prohibitions). For example, a dirty-word list in a firewall is a kind of negative authorization. Release control systems should be able to support both negative and positive authorizations, and to resolve any conflicts that arise between them. They should also be able to define flexibly whether or not to release when no specific authorizations or prohibitions apply (called the *decision policy*). By borrowing techniques introduced by Jajodia et al. [JSSS01] for access control, our framework allows specifying both positive and negative release authorizations and incorporates notions of authorization derivation, conflict resolution, and decision policy.

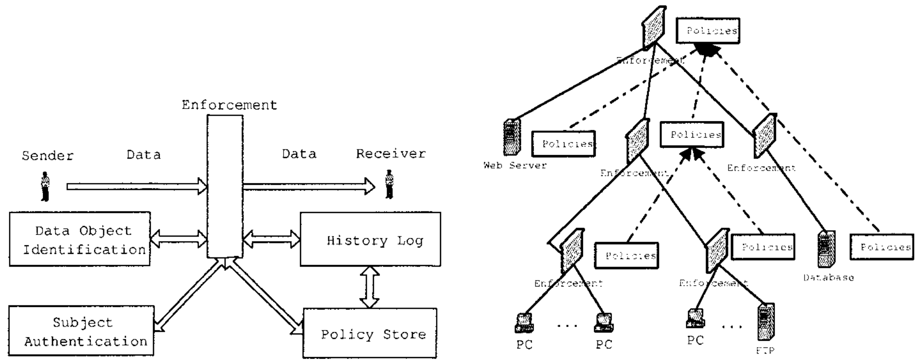
Often certain actions must be performed as a condition of release. Such actions are called *provisions* if they must be performed before release and *obligations* if they must be performed at some time after release. *Sender provisions* and *sender obligations* must be undertaken by the sender and *receiver provisions* and *receiver obligations*, by the receiver. Example sender provisions could include generating a log entry and attaching a copyright notice, a disclaimer, or a watermark to the released data; a sender obligation might be to transmit a follow-up customer-satisfaction survey form in 3 days; a receiver provision could be to present certain (additional) credentials; a receiver obligation could be to delete released data after 3 days. Our extended framework allows users to specify provisions and obligations in association with release control policy.

When an object is released to one receiver, one of the things that the receiver can potentially do with it is to release it further. We solve two problems related to such release paths. Sometimes confidentiality requires that certain release paths not be authorized. In this case, we need a way to detect them so that they can be blocked. On the other hand, a user may wish to discover legal paths that could be used to achieve a desired release. For the former purpose, our language allows defining policies to block undesired release paths. We show how to check whether such paths exist, enabling policy administrators to ensure they do not commit policy changes that permit illegal paths. For the latter purpose, we allow users to define weight functions associated with release paths based on various factors, and we show how to find the best release paths based on the assigned weights.

The paper is organized as follows. Section 1 gives an overview of our framework. Section 2 introduces the basic framework and language, and then extends them to support propagations and obligations. Section 3 presents technique for efficiently evaluating our policies. Section 4 discusses related work and Section 5 concludes.

1. System Architecture and Framework Overview

In this section, we first present system architectures for release control to understand the context in which release policies are applied. Then we overview the important components of the policy framework.



(a) General system architecture of release control

(b) Hierarchical release control system

Figure 1. System Architecture

1.1 System architecture

One general system architecture for release control is as shown in Figure 1(a). There is a filter-like enforcement point that intercepts each message that a sender attempts to release to a receiver. It invokes object-identification and subject-authentication components, and queries the policy store to determine whether the release is authorized. In an alternative architecture (not shown), the enforcement point does not actively intercept the messages, but passively responds to release queries and issues a permit token when the data release is authorized.

In some environments, sensitive data are distributed over an organization that consists of hierarchically organized units. A release control system in such an organization could be composed of smaller release control subsystems that correspond to the organizational units, as shown in Figure 1(b). Similarly, each unit may have a stake in the definition of policy. Thus, whether or not actual enforcement is distributed, it may be necessary to gather policy components that are defined in a distributed manner, so as to assemble the complete policy of the organization that is to be enforced consistently throughout.

1.2 Subjects and objects

Subjects in our framework can act as senders and as receivers. Subjects can be users, groups, or roles. In most applications, subjects and objects are organized into hierarchies. Policies are specified to explicitly authorize the release of an object o from sender s to receiver r. Implicit authorizations can be derived from these explicit authorizations in the usual manner by propagating them down the hierarchies. Such propagation enables higher entities in the hierarchy to generalize lower entities. Since we allow negative authorizations, conflicts may occur between derived authorizations. The policies used to resolve such conflicts can be specified using our language.

1.3 Policy authority

Policies in our framework consist of logic programs that define certain predicates used by the release control system. Many policy authorities contribute clauses. There are two kinds of predicate symbols in our language, *authority predicates* and *global predicates.* Each authority predicate belongs to a specified authority, and is denoted by an authority name followed by a predicate name, separated by a dot. Thus, our framework deviates from standard logic in that authority predicate symbols are structured (pairs). Authority can be looked upon as an administrator role that has the privileges to manage the policies of the corresponding organizational unit. Each release control system has its authorities organized in a hierarchy that has a single topmost authority, denoted a^T. This hierarchy usually resembles the structure of the organization. Only administrators associated with authority a can contribute or modify clauses that define the predicates of a. The body of such a clause can contain predicates of other authorities that are lower in the authority hierarchy than a, as well as global predicates, which are discussed in the following paragraph. The release decisions are determined by predicates of the topmost authority. This gives the topmost authority ultimate control over release decisions.

Global predicates are used to represent the object and subject hierarchies, as well as the association of subjects and objects with certain authorities. They are defined globally by the organization, rather than being associated with a particular authority. By contrast with authority predicates, global predicates are standard predicate symbols in the sense of being unstructured individuals.

Acct Authority

$acct.canrls(expenseDoc, manager, org2, +) \leftarrow .$

$acct.dercanrls(O, S, R, +) \leftarrow in(O, O'),\ acct.canrls(O', S, R, +)$

$acct.rls(O, S, R, +) \leftarrow acct.dercanrls(O, S, R, +)$

$acct.error \leftarrow path(O, S, org3), in(O, expenseDoc)$

Tech Authority

$tech.canrls(doc1, manager, org2, +) \leftarrow .$

$tech.rls(O, S, R, +) \leftarrow tech.canrls(O, S, R, +)$

Org Authority

$$org.rls(O, S, R, +) \leftarrow acct.rls(O, S, R, +),\ tech.rls(O, S, R, +),\ in(O, expenseDoc)$$

$org.rls(O, S, R, -) \leftarrow \neg org.rls(O, S, R, +)$

Figure 2. An example for release policy specifications

EXAMPLE 1 *Consider the clauses in Figure 2. There are three authorities in an organization, acct (the accounting department), tech (the IT department), and org (the whole organization). The topmost authority is $a^T = org$. The first group of clauses are contributed by acct. The first of these clauses says that manager can send the document type expenseDoc to another organization org2. The second indicates that positive authorizations propagate down the object hierarchy. The third makes acct's final release decision from the derived authorizations. This will be inte-*

grated below into the final decision for the system as a whole. The fourth expresses an integrity constraint that no sender can transmit to $org3$ *any object dominated by expenseDoc via any sequence of release. (Valid release specifications dom not make a.error true for any authority a.) The second group of clauses are contributed by tech:* $doc1$ *can be sent from manager to* $org2$*. The third group of clauses are contributed by org, the topmost authority. The first says that if a document is dominated in the object hierarchy by expenseDoc, then release of the document is permitted as long as both acct and tech authorize the release. The second clause contributed by org says that if a release is not authorized, then it is prohibited. This ensures that the policy defines a complete decision, meaning that all releases are either permitted or denied. (See Section 2.) In our context, for every o, s, r, exactly one of the atoms* $org.rls(o, s, r, +)$ *and* $org.rls(o, s, r, -)$ *is true. It determines the final release decision. From the clauses that make up this policy, it follows that if* $doc1$ *is an expenseDoc, then the release of* $doc1$ *from manager to* $org2$ *is authorized.*

1.4 PO actions

As discussed in Section , certain actions, called provisions and obligations, must be taken as a condition of release. We refer to them in general as PO-actions. Following Bettini et al. [BJWW02], PO-actions are represented using a special class of predicates and positive Boolean combinations of PO-actions are represented by PO-formulas, as introduced formally below in section 2.4. For example, the PO-formula $(Watermark \wedge Log) \vee SignContract$ expresses that to permit the release we must either embed a watermark into the data and record the release into log, or sign a contract. Although we defer the details to a later section, the basic idea is that we associate a PO-expression with each clause. This PO-expression is then used to define PO-formulas for atoms inferred by using the clause.

EXAMPLE 2 *From the following clauses, we can infer that* $acct.rls(doc1, manager, org2, +)$ *is true. Let us consider the PO-formula associated with this atomic formula. The PO-expression* $\top$*, which is associated with the first clause (see table below), represents that the PO-formula associated with* $in(doc1, expenseDoc)$ *is trivially true: no PO-actions are required. From the second clause, we associate* $acct.canrls(expenseDoc, manager, org2, +)$ *with* $Watermark$*. For the third clause, the formula variable* f_2 *in the PO-expression represents the PO-formula associated with (an instance of)* $acct.canrls(o', s, r, +)$*. Consider the instance of clause 3 obtained by substituting* o *by* $doc1$*,* o' *by* $expenseDoc$*, s by* *manager, and r by* $org2$*. In this case,* f_2 *is substituted by* $Watermark$*, because it is the PO-formula associated with the second positive atom in the body,* $acct.canrls(expenseDoc, manager, org2, +)$*. Replacing* f_2 *in the PO-expression* $Log \wedge f_2$ *accordingly, we obtain the PO-formula associated with* $acct.rls(doc1, manager, org2, +)$*, viz.,* $Log \wedge Watermark$*. From the fourth clause, we find that* $SignContract$ *is associated with* $acct.rls(doc1, manager, org2, +)$*. So* $acct.rls(doc1, manager, org2, +)$ *is associated with the PO-formula* $(Log \wedge Watermark) \vee SignContract$*, which represents the required PO-actions for the release authorization for* doc1 *from* $manager$ *to* $org2$*.*

Clause	**PO-expression**
$in(doc1, expenseDoc) \leftarrow$	$\top$
$acct.canrls(expenseDoc, manager, org2, +) \leftarrow$	$Watermark$
$acct.rls(O, S, R, +) \leftarrow in(O, O'),\ acct.canrls(O', S, R, +)$	$Log \wedge f_2$
$acct.rls(doc1, manager, org2, +) \leftarrow$	$SignContract$

2. Release Control Framework

We formalize the basic elements in our framework as follows.

DEFINITION 3 (DATA RELEASE DOMAIN) *A* data release domain *consists of a 4-tuple* $(Obj, Sub, PO, Auth)$ *in which:*

1 Obj *is a partially ordered set whose elements* $o \in Obj$ *are called objects;*

2 Sub *is a partially ordered set whose elements* $r, s \in Sub$ *are called subjects;*

3 PO *is a partially ordered set whose elements* $p \in PO$ *are called PO-actions;*

4 $Auth$ *is a partially ordered set whose elements* $a \in Auth$ *are called authorities and which has a maximal element,* $a^T \in Auth$*.*

We use $\sqsubseteq$ *to denote the order relation over each of these sets.*

In the remainder of this section, we present the language for basic authorization policies and then extend it to support policies with PO-actions.

2.1 Basic release specification language

We introduce a basic specification language to specify release authorization policies. We assume familiarity with the basic concepts of logic programming. (See for instance [Llo87].) The language contains the following symbols and constructs:

1 **Constant Symbols**: There is a constant for each $o \in Obj$, $s \in Sub$, and $a \in Auth$. In addition, $+$ and $-$ are also constants.

2 **Variable Symbols**: There are *subject/object* variables ranging over the sets Obj and Sub. We use O for variables range over Obj and S and R for variables ranging over Sub, often with subscripts or primes.

3 **Subject/Object Terms**: A *subject* (respectively, *object*) *term* is a subject (respectively, object) constant or variable. We use σ for sender subject terms, ρ for receiver subject terms, and ω for object terms.

4 **Predicate Symbols**: The set of predicate symbols is partitioned into predicates that have special roles in the release control policy and auxiliary predicates.

(a) A 4-ary predicate symbol, **a.canrls**, with the arguments $(\omega, \sigma, \rho, sign)$. It represents authorizations explicitly specified by the authority a.

(b) A 4-ary ternary predicate symbol, **a.dercanrls**, with the same arguments as **a.canrls**. It represents authorizations derived by the system from explicit authorizations, typically by propagating authorizations down the subject and object hierarchies.

(c) A 4-ary predicate symbol, **a.rls**, with the same arguments as **a.canrls**. It represents the final authorization decision made by the authority A.

(d) A 4-ary predicate symbol, **a.path**, taking the form $a.path(\omega, \sigma, \rho)$. It represents that there is a release path for the object ω from the sender σ to the receiver ρ. Such a path is a sequence of linked release authorizations from the authority a.

(e) A predicate symbol, **a.error**, used to detect certain integrity violations within the policy itself.

(f) A binary predicate symbol, **in**, with the form $in(\omega_1, \omega_2)$ or $in(\sigma_1, \sigma_2)$. It represents the domination relationship in a hierarchy.

(g) A binary predicate symbol, **dirin**, with the form $dirin(\omega_1, \omega_2)$ or $dirin(\sigma_1, \sigma_2)$. It represents the direct domination relationship in a hierarchy.

(h) A binary predicate symbol, **auth**, with the form $auth(\omega, a)$, $auth(\sigma, a)$, or $auth(\rho, a)$. It represents that the object or subject belongs to the authority.

(i) Other **user-defined** predicates to describe the properties of subject or objects.

We next define the core of our release policies. Later we extend this core so as to support provisions and obligations.

DEFINITION 4 (BASIC RELEASE SPECIFICATION) *A basic release specification RS is a set of Horn clauses over the language presented above. These clauses may include negated*[1] *literals in their bodies, and must satisfy the restriction that every variable that appears in the head of a clause also appears in the body. In addition, clauses must satisfy the restrictions presented in Figure 3, and RS must contain* $a^T.error \leftarrow a^T.rls(O,S,R,+), a^T.rls(O,S,R,-)$ *and* $a^T.rls(O,S,R,-) \leftarrow \neg a^T.rls(O,S,R,+)$. *If* $RS \models a^T.rls(o,s,r,+)$, *then the release for o from s to r is* authorized; *if* $RS \models a^T.rls(o,s,r,-)$, *then the release is* blocked.

Including $a^T.error \leftarrow a^T.rls(O,S,R,+)$, $a^T.rls(O,S,R,-)$ in the release specification makes that there don't exist conflicts. Including $a^T.rls(O,S,R,-) \leftarrow \neg a^T.rls(O,S,R,+)$ in the release specification makes it complete, as shown in [JSSS01]. This means that any specification RS makes one and only one of $a^T.rls(o,s,r,+)$ and $a^T.rls(o,s,r,-)$ true for each triple of constants (o,s,r).

Integrity rules. A clause with the head of *a.error* is called an *integrity rule.* An integrity rule derives an error every time the conditions in the body of the clause are satisfied. It imposes an constraint on the release specification. A valid release specification cannot make *error* true.

DEFINITION 5 (RELEASE PATH) *Given a set of release clauses, a* Release Path *from s to r for o is a sequence of subjects* $s_0, s_1, \ldots, s_n$ *such that* $s_0 = s$, $s_n = r$, *and* $a.rls(o,s,s_1,+)$, $a.rls(o,s_1,s_2,+)$, $\ldots$, $a.rls(o,s_{n-1},r,+)$.

Clauses that specify the *path* predicate are straightforward:

$$a.path(O,S,R) \leftarrow a.rls(O,S,R,+)$$
$$a.path(O,S,R) \leftarrow a.path(O,S,S')\ ,\ a.path(O,S',R)$$

EXAMPLE 6 *The following integrity rule prohibits there being a release path from staff to org1 for doc1:*

$$a^T.error \leftarrow org.path(doc1, staff, org1, +)$$

[1] As we discuss in Section 3.1, each permitted set of clauses is stratified, so all of the standard semantics of negation as failure coincide.

Predicate	**Clauses defining predicate**
rel-preds	Facts only (no clause body).
$a.canrls$	Body may contain rel-preds and arbitrary preds of a', $a' \sqsubseteq a$.
$a.over$	Body may contain $a.canrls$, rel-preds and arbitrary preds of a', $a' \sqsubseteq a$.
$a.dercanrls$	Body may contain $a.over$, $a.canrls$, rel-preds and arbitrary preds of a', $a' \sqsubseteq a$. Occurrences of $a.dercanrls$ must be in positive literals.
$a.rls +$ $a.rls -$	When head is of the form $a.rls(\omega, \sigma, \rho, +)$, body may contain $a.dercanrls$, $a.canrls$, rel-preds and arbitrary preds of a', $a' \sqsubseteq a$. When head is of the form $a.rls(\omega, \sigma, \rho, -)$, body contains just one literal, *viz.*, $\neg rls(\omega, \sigma, \rho, +)$, and ω, σ, and ρ are all variables.
$a.path$	Body may contain positive uses of $a.rls$ and $a.path$ only.
$a.error$	Body may contain positive uses of $a.path$, $a.rls$, $a.dercanrls$, $a.canrls$, rel-preds and arbitrary predcates of a', $a' \sqsubseteq a$.

Figure 3. Restrictions on the form of clauses defining the various predicates in our language. The predicates *in*, *dirin*, *auth* and user-defined predicates are called rel-preds. Predicates and literals associated with the authority a are called predicates and literals of a, respectively.

Disjunction of the authorizations of authorities:

$org.rls(O, S, R, +) \leftarrow acct.canrls(O, S, R, +)$
$org.rls(O, S, R, +) \leftarrow tech.canrls(O, S, R, +)$

Conjunction of the decisions of authorities:

$org.rls(O, S, R, +) \leftarrow acct.rls(O, S, R, +),\ tech.rls(O, S, R, +)$

Sender authority takes precedence:

$org.rls(O, S, R, +) \leftarrow acct.rls(O, S, R, +),\ auth(S, acct)$
$org.rls(O, S, R, +) \leftarrow tech.rls(O, S, R, +),\ auth(S, tech)$

Figure 4. Rules enforcing various composition policies.

DEFINITION 7 (VALID BASIC RELEASE SPECIFICATION) *A basic release specification RS is* valid *if* $RS \not\models a.error$ *for all authorities* $a \in Auth$.

2.2 Composition policies

Here we want to show that there are various ways in which a higher authority can compose authorizations defined by lower authorities. Figure 4 presents several possible composition policies in the context of our running example. As the topmost authority is *org*, truth values assumed by *org.rls* determine the final release decisions.

2.3 Legal release paths

Sometimes users may want the evaluation of release specifications to return not only release authorizations, but also release paths. Users may want to send o from s to r via other subjects in the case where direct release is not authorized. Given (o, s, r) and a set of subjects I as intermediate nodes of release paths, the problem is to find one or more legal release paths from s to r. Sometimes we may want to find an optimal path based on costs assigned to edges through which the path passes. We discuss algorithms to compute such paths in Section 3.2.

2.4 PO extension

We follow Bettini et al. [BJWW02] in associating PO-formulas with atoms entailed by the clauses in our policies. However, we take a slightly different approach: because we aim to enable higher authorities to override policy decisions made by their underlings, we provide a similar capability with respect to PO-actions. So while in the prior work each clause used in the proof of an atom could contribute PO-requirements to the PO-formula associated with the atom, in our framework, the decision whether to include PO-requirements associated with a given subtree in the proof is made by the PO-expression associated with the clause at which the subtree is rooted. PO-expressions generalize PO-formulas by allowing the use of *formula variables* whose values range over PO-formulas. Formula variables are disjoint from the standard subject/object variables, whose values range over the domain of interpretation.

PO-actions are represented by PO-atoms, which are constructed using special PO-predicate symbols, constants, and variables. The special symbol $\top$ is also a PO-atom, signifying that no PO-action is required. A PO-formula is either a PO-atom, a disjunction of PO-formulas, or a conjunction of PO-formulas. We next explain how we associate a PO-formula with each ground atom in the least Herbrand model of a release specification.

The policy author associates a PO-expression with each clause he authors. A PO-expression is either a PO-atom, a formula variable, a disjunction of PO-expressions, or a conjunction of PO-expressions.

Definition 8 (Release Specification) *A* release specification *consists of a basic release specification and a mapping associating a PO-expression with each clause in the former. Each formula variable occurring in a PO-expression must correspond to a positive literal occurring in the body of the clause with which the PO-expression is associated. (Neg-*

ative literals are not associated with PO-formulas.) This correspondence is established by numbering the formula variables and positive literals, and associating with one another the variables and literals that have the same number. For instance, the variable f_2 in Example 2 represents the PO-formulas associated with the ground instances of the second positive literal, $acct.canrls(o', s, r, +)$. Each object variable occurring in a PO-expression must occur in the clause with which the PO-expression is associated. This ensures that PO-formulas associated with ground atoms are also ground.

During policy evaluation, each ground instance of a clause defines a PO-formula by substituting for each formula variable in the clause's PO-expression the PO-formula associated with the corresponding atom in the body. The PO-formulas defined by all ground clause instances having the same head are then combined by disjunction to obtain the PO-formula associated with the ground atom that is that common clause head.

3. Evaluation of Release Specifications

This section discussed methods for the evaluation of release specifications. We first consider individual releases, and then turn to sequences of releases of a data object from one subject to another.

3.1 Materializing release specifications

The materialized authorizations of the highest authority are the evaluation results of a release specification. Recall that authority predicate symbols are pairs in our system. For example, *acct.canrls* is a different predicate from *tech.canrls*. Predicates with lower authority can appear in the body of clauses defining predicates with higher authority, whereas the reverse is prohibited. This, and the other restrictions presented in Figure 3, ensure the set of clauses given by a release specification is locally stratified. Therefore, the data complexity of its evaluation is quadratic time in the number of ground instances of the clauses in the specification, as illustrated in [Gel89]. As the number of variables in each clause is typically bounded by a small constant, the number of ground instances of each clause is bounded by a polynomial in the size of the specification (including the definitions of predicates representing the hierarchy). Thus, the complexity of evaluation is also bounded by such a polynomial.

For the extended framework, note that all PO-formulas are positive and hence consistent. Consequently, if a release specification's underlying basic release specification is RS, then if $RS \models a^T.rls(o, s, r, +)$, the corresponding release can be permitted, provided some combination of PO-actions is undertaken. The evaluation of the least Herbrand model of RS is not affected by the PO-expressions in the policy. We can evaluate the PO-formulas for each atom in the least Herbrand model in a bottom-up fashion, as illustrated in [BJWW02].

3.2 Computation of release path

Given a release specification RS, a triple (o, s, r), and a (possibly empty) set of subjects I, a basic path evaluation algorithm returns a possibly empty set of paths. Each path consists of a sequence of subjects, $s_0, s_1, \ldots, s_n$, such that $s_0 = s$, $s_n = r$, for each i, $0 \leq i < n$, $RS \models a^T.rls(o, s_i, s_{i+1}, +)$. Letting the PO-formula associated with $a^T.rls(o, s_i, s_{i+1}, +)$ be p_i, the PO-formula associated with the path $s_0, s_1, \ldots, s_n$ is $p = p_0 \wedge \ldots \wedge p_{n-1}$.

A basic path evaluation algorithm determines whether a release specification RS is valid and identifies all integrity violations within it. Such an algorithm can be obtained as follows. Given an object o, we compute all atoms $a^T.rls(o, s, r, +)$ entailed by RS, and find the PO-formula p associated with each such atom. Then we construct a directed graph whose nodes are given by Sub and which has edge (s, r) just in case $RS \models a^T.rls(o, s, r, +)$. We associate with this edge the PO-action associated with $a^T.rls(o, s, r, +)$. A release path corresponds to a path in this directed graph. There are well-known algorithms to check whether there exists a path between two nodes and to find all paths between two nodes.

When a user seeks a legitimate release path for getting a data object from a given sender to a given receiver, the user does not want a set of paths, but rather an optimal path based on certain assigned weights. The factors that affect such weights could be, for instance, the subjects through which the path passes and the PO-formulas associated with the edges in the path.

We now consider how to find an optimal path. We first assign weights to edges based on PO-actions and nodes (senders and receivers). We can still use the well-known algorithms for the shortest path. However, calculation of path weights is more complicated when basing them on PO-formulas associated with edges. A PO-formula expresses that one of a collection of sets of PO-actions needs to be executed, where each set corresponds to a disjunct in the DNF (disjunction normal form) of the PO-formula. So the weight assigned to a PO-formula is the minimum weight among the disjuncts of the DNF of the PO-formula. For example, given a PO-formula $(Log \wedge Watermark) \vee Sign$, if the weights of Log,

$Watermark$, and $Sign$ are 1, 2 and 3, respectively, then the weight of the disjunctive clause $Log \wedge Watermark$ is $1 + 2 = 3$ and the weight of the disjunctive clause $Sign$ is 2; hence the weight of the PO-formula is the minimum value 2.

To calculate a path's weight, we need to first find the PO-formula of the path. This is the conjunction of PO-formulas associated with edges in the path. Thus, the weight of a path is not simply the sum of the weights of its edges. The calculation of the weight of the PO-formula associated with a path can take advantage of pruning. For example, suppose there are two PO-actions Log and $Watermark$, where the weight of Log is less than that of $Watermark$. If one edge has a PO-formula $Watermark \vee Log$, and an adjacent edge has a PO-formula Log, then the PO-formula associated with the path consisting of these two edges is $(Log \wedge Watermark) \vee Log$. $Log \wedge Watermark$ certainly cannot have less weight than Log, so the formula can be pruned to be Log. This example also illustrates that, if the formula associated with an edge is $Log \vee Watermark$, even though the weight of Log is less than $Watermark$, we still need to keep track of both during the computation because we may subsequently encounter another edge in the path that has a PO-formula $Watermark$.

4. Related Work

Much of the extensive prior research in access control is highly relevant to release control. For instance, Jajodia et al. present an access control framework, FAF [JSSS01], that uses Horn clauses to express multiple access control policies (*e.g.*, open or closed, and positive or negative). They show that FAF specifications are locally stratified and can be evaluated in polynomial time. Our work is based on FAF. However, FAF does not address data release, nor does it allow policies to be composed of components specified by multiple authorities. It also does not allow provisions and obligations to be required as a condition of authorization.

Another important line of work in access control concerns the RBAC model[SCFY96, SBM99]. In RBAC, there is an administrator hierarchy [SBM99], where different administrators have different administrative privileges. However, in RBAC, even when multiple administrative roles have authority over various portions of the authorization specification, these portions are disjoint and the manner in which these portions are composed is somewhat trivial. Again, release is not addressed, nor typically are provisions and obligations.

Bettini et al. [BJWW02] study provisions and obligations in policy management. They introduce a framework for augmenting logical programs which associates a PO-formula with each policy clause. Our framework regarding provisions and obligations is quite similar, though PO-expressions we associate with clauses generalize PO-formulas in manner that makes our approach better in the context of multiple policy authors whose authorities are organized hierarchically. There are also related works [BdVS00, WJ02] that introduce policy algebras to combine authorization specifications for access control. Bonatti et al. [BdVS00] model policy as a set of ground terms over an alphabet for (subject, object, action) terms whereas Wijesekera and Jajodia [WJ02] model policies as non-deterministic transformers (relations) over a collection of subjects, objects, and action terms. Operands are policies, which are combined by operators such as addition, conjunction, and negation. Our framework language can be used to implement most policy algebra operators. Furthermore, we organize the specification of distributed policies in a hierarchical manner, which captures structural features of organizations, and hence is simple and manageable.

Another research area related to release control is flow control [Den76, Fol89, MMN90, ML97, SBCJ97]. This line of work focuses mainly on the context of multi-level security, concentrating on information flow between objects in programs. It seeks to control not only data release, but also the further propagation of any information derived from that data. In this sense it is much more ambitious than release control, which does not trace information flow through computations.

5. Conclusions

We have observed that the world's increasing reliance on information sharing heightens the need for mechanisms and models that protect confidentiality, and that access control alone is inadequate to modern requirements. We present a release control framework to better satisfy these needs. By providing for distributed, hierarchical specification of policy throughout an organization, and by allowing release policy clauses to include provisions and obligations that must be satisfied for release to be permitted, we create a framework that is both powerful and flexible. We have extended prior work associating provisions and obligations with authorizations in a manner that reflects our goal of enabling senior authorities to override policy authored by their juniors.

This paper concentrates on the specification of release control policies, rather than on their enforcement. Enforcement mechanisms of release control are more complicated than those of access control. This is because it is difficult to detect reliably when data transmission is occurring, and identify senders, receivers, and data objects on the fly, as data objects are transmitted. Further research in release control enforcement is clearly needed.

Acknowledgments

This work was partially supported by the NSF grants CCR-0113515, IIS-0242237, and IIS-0430402.

References

[BdVS00] Piero A. Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *ACM Conference on Computer and Communications Security*, pages 164–173, 2000.

[BJWW02] Claudio Bettini, Sushil Jajodia, Xiaoyang Sean Wang, and Duminda Wijesekera. Provisions and obligations in policy management and security applications. In *VLDB*, pages 502–513, 2002.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[Fol89] Simon N. Foley. A model for secure information flow. In *IEEE Symposium on Security and Privacy*, pages 248–258, 1989.

[Gel89] Allen Van Gelder. The alternating fixpoint of logic programs with negation. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania*, pages 1–10. ACM Press, 1989.

[JSSS01] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[Llo87] John W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer, 1987.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *SOSP*, pages 129–142, 1997.

[MMN90] Catherine D. McCollum, J. R. Messing, and LouAnna Notargiacomo. Beyond the pale of mac and dac-defining new forms of access control. In *IEEE Symposium on Security and Privacy*, pages 190–200, 1990.

[SBCJ97] Pierangela Samarati, Elisa Bertino, Alessandro Ciampichetti, and Sushil Jajodia. Information flow control in object-oriented systems. *IEEE Trans. Knowl. Data Eng.*, 9(4):524–538, 1997.

[SBM99] Ravi S. Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[WJ02] Duminda Wijesekera and Sushil Jajodia. Policy algebras for access control the predicate case. In *ACM Conference on Computer and Communications Security*, pages 171–180, 2002.