A LEARNING-BASED APPROACH TO INFORMATION RELEASE CONTROL

Claudio Bettini
DICO, University of Milan, Italy, and
Center for Secure Information Systems, George Mason University, Virginia
bettini@dico.unimi.it

X. Sean Wang
Department of Computer Science, University of Vermont, Vermont, and
Center for Secure Information Systems, George Mason University, Virginia
xywang@cs.uvm.edu

Sushil Jajodia
Center for Secure Information Systems, George Mason University, Viginia
jajodia@gmu.edu

Abstract: Controlled release of information from an organization is becoming important from various considerations: privacy, competitive information protection, strategic data control, and more. In most organizations, data protection is afforded only by using access control. However, it can be argued that access control suffers from at least two problems. First, effective access control assumes a perfect categorization of information ("who can access what"), which is increasingly difficult in a complex information system. Second, access control is not effective against insider attacks, where users with legitimate access rights send out sensitive information, either with malicious intent or by accident. Information release control is viewed as complementary to access control, and aims at restricting the outgoing information flow at the boundary of information systems. This paper presents an architectural view of a release control system. The system emphasizes the role of automated

learning for release control constraints. This has resulted from the realization that the most difficult task of effective release control is how the release control constraints are specified. In a learning-based system, data mining and machine learning techniques are employed to generate release control constraints from samples provided by the security officer. The system applies continuous learning to adjust the release control constraints to reduce both mistakenly released and mistakenly restricted documents. This paper also provides a specific example on how to learn keyword-based release control constraints.

Keywords: Data protection, Release control

1. INTRODUCTION

Release control refers to the process of checking the output data generated upon a user request to determine if the information is appropriate for release across a security boundary. Nowadays, organizations have vast amounts of information that is shared with other organizations or even the general public. Such sharing takes the form of public web pages, financial reports, technical white papers, biographies of personnel, etc. Furthermore, the knowledge workers of the organizations send out messages for collaboration purposes. Such information sharing needs to be done in a controlled fashion, taking into account security and other considerations.

For information security, a number of aspects have been considered in the literature. These aspects include (1) secure communication, (2) perimeter control, (3) reliable authentication, (4) authorization to information cells (such as files, relational tables, columns in tables, and XML nodes), and (5) partitioning of the information into cells. According to Wiederhold [Wie00], aspect (5) is often under-emphasized and requires a highly reliable categorization of all information to those cells. It is in the management of that categorization where many failures occur. In other words, if some information is miss-categorized, which is highly likely in a complex organization, it is possible for the sensitive information to be released to unauthorized users.

A more serious threat to sensitive information comes from careless or malicious *insiders,* individuals or organizational entities with authorized access to the system. This type of threats are potentially very damaging since the access control rules (on which organizations rely heavily) are not effective. Insider threats need to be countered using several different methods, from intrusion analysis to information forensics. An important tool to counter such threats is release control, which blocks the information at the *gate* from “inside” to “out side.”

In addition to the aforementioned security considerations, some legal concerns of information release need to be addressed. Depending on the category of information, an organization may wish to append disclaimers, copyright and other legal notices, and even watermarks. Release control can be used to address this problem.

Release control thus needs to become an important component for securing and managing information of an organization. From a technical perspective, the release control system is best done by separating "release control constraints", which state what need be controlled, from "checking algorithms", which monitor the outgoing data against these constraints. This separation allows more convenient management of the release control system to fit the ever-changing organizational security and legal needs, and allows more opportunities for the introduction of new checking algorithms.

In view of the aforementioned technical considerations, for an effective and efficient release control system, at least two issues need to be considered carefully. The first is how the release constraints are established and refined, and the second is how the checking of the outgoing information can be done efficiently and in a meaningful way.

In this paper, we present an architectural view of the release control system and then focus on the first issue. One relevant source for coming up with release constraints is clearly the data store. Indeed, in addition to the ability of simply adding release constraints derived from experience or high-level requirements, security officers may query the data store to determine what needs to be controlled at the release point. We call this the "manual" method. For example, access control rules are usually adopted to restrict the access to the data store [JSSS01]. When some data have restricted access, it is likely that the release of such data should be checked. In addition, information that might be inferred through integrity constraints from restricted data should also be automatically added to the release constraints store. Furthermore, data that are similar to such restricted data may also need to be checked at the release point. Due to the involved complexity of these tasks, "automated tools" are necessary.

We introduce an automated learning approach to help acquire release control constraints. In a learning-based approach, the security office can give an initial sample set of documents, including both "cleared for release" documents and "restricted" documents. The system will try to learn an initial set of release control constraints from the given sample set. As the time goes by, when more and more documents are released and restricted (some of the releasing and restricting are certified by the security officer), the learning process will periodically adjust the release control constraints to do a better job: reducing the mistakenly released documents as well as the mistakenly restricted documents. In this paper, we outline such a learning based system

architecture, and provide a specific example for learning release control constraints.

The remainder of the paper is organized as follows. In Section 2, we give a formal foundation for specifying release control constraints. We also give a language that can be used to represent release control constraints on XML documents. In Section 3, we outline an architecture for release control and emphasize the role of learning in the whole architecture. In Section 4, we give a specific example of learning keyword-based release control constraints. We discuss related work in Section 5 and conclude the paper in Section 6.

2. FORMAL FRAMEWORK

For a uniform treatment, we assume the data to be checked before release is in XML format (the underlying data may be a table resulting from a database query, semistructured data or a full text document). The data are checked against a number of release constraints.

2.1 Release Constraints

A *release constraint* (*RC*) is a pair $\langle R, CI \rangle$, where *R* is a *matching rule* and *CI* a set of *controlled items.*

Each RC is evaluated against the data being released and it prevents its release if satisfied. Formally, each RC gives a mapping that assigns each document with a label in **{Restrict, Release}**. A document *Doc* can be released if for each release constraint *RC*, *RC*(*Doc*)=**Release**, i.e., in the data, the controlled items do not appear in the way given by the corresponding matching rules.

The set *CI* of controlled items is a set of tuples of the form $\langle A_1{:}a_1, A_2{:}a_2, \ldots, A_n{:}a_n \rangle$, where A_j are variable symbols and a_j their values. Values can range over several simple domains (including integer, string, Boolean, etc.) or even complex domains (admitting single-valued, multi-valued, or possibly structured attribute values). Variable symbols may actually denote attribute names as well as paths in the document structure. In the simplest case, variable symbols are omitted and each tuple is a list of keywords.

Syntactically, the attribute part *A* of each pair can be specified by an XPath expression, while the value part *a* can be specified by a regular expression if it is a string (e.g., to denote any word starting with a certain prefix) or, in the case it is a numerical value, it can be specified by a simple condition (*op k*) with $op \in \{<, =, >, \leq, \geq\}$ and *k* being a number (e.g., to denote all values greater than or equal to a certain constant).

The matching rule R specifies how a document should be checked against the controlled items. As an example, when the data to be checked are answers from a database query, they can be represented as a sequence of tuples. Then, a simple matching rule may check if one of these tuples contains the attribute values specified in one of the tuples in the set of controlled items. In the case of data containing unstructured text, the rule may specify, for example, that the words in each tuple of the set of controlled items should not appear together in any k-words portion of the text, where k is a parameter defined by the specific rule. In other words, the set of controlled items essentially lists the pairs attribute-value involved in a release constraint, while the matching rule specifies their relationships.

Example 1. Consider a corporate database that includes data about Employee's salaries and assume an internal policy prohibits the release of the salary of Mr. Woo. The administrator will set up a RC with $CI = \{\langle Employee{:}Woo,\ Salary{:}75{,}000\rangle\}$ and R equal to the simple rule on relational query results checking all the query results being released for tuples containing the attribute values specified in CI. In this case, the system will check each tuple returned, as part of a database query result, making sure the values *Woo* and *75,000* do not appear in it.

Note that this is a conceptual model of RCs. In practice, controlled items will have compact representations, in the form of SQL queries, general XQuery expressions including predicates, strings with metacharacters or other representations.

The above formal model can be easily extended to represent overriding constraints by allowing negative terms in control items (in the form $\neg A{:}a$). For example, in protection software, among all documents rejected because containing the word 'chip', those in which the occurrence of 'chip' is always part of the sequence 'chocolate chip' should be released. This can be modeled by the above extension.

2.2 A language for matching rules on XML documents

In this subsection, we focus on the problem of representing matching rule when the data to be checked in contained in XML documents. We design a language that has the following expressiveness properties, which we consider necessary for our task.

- The language should be able to express the maximum distance between all pairs of values as a pre-determined upper bound. In terms of XML

this may be the number of edges in the XML tree that separate the nodes storing the values.

- The language should be able to express the presence of certain nodes (with particular labels, semantic tagging, degree, type, e.g., leaf, root, etc.) in the path between the pair of nodes for which we consider the distance in the XML structure.
- The language should be able to express relationships between nodes. Example of relationships include the difference in depth in the XML tree, the difference in the semantic tagging possibly attached to the nodes, the presence of a common ancestor with a specific tag, relationships between node attributes, etc.
- The language should be able to discriminate the order of the values in the XML structure. The order may be particularly relevant in overriding. In our chocolate-chip example above, the word 'chocolate' must appear before the word 'chip'.

Following the above requirements, we represent each matching rule as a conjunction of a *cardinality rule* and one or more *node-relation* rules.[1]

A *cardinality rule* has the form $NumVar \geq k$ where k is a positive integer and *NumVal* is a language keyword denoting the number of different values specified in the controlled items that should be observed in the document. Hence, a cardinality rule specifies that the number of these values must be at least k, not considering negative terms in the controlled items representing overriding conditions. If no cardinality rule is given, a default cardinality rule is assumed with $k=1$; this is the most conservative choice.

Example 2. Given a set of controlled items $(A_1{:}a_1, \neg A_2{:}a_2, A_3{:}a_4, A_4{:}a_4)$, a cardinality rule $NumVar \geq 2$ would restrict any document that would contain at least 2 of the positive controlled items; i.e., either $A_1{:}a_1$ and $A_3{:}a_3$, or $A_1{:}a_1$ and $A_4{:}a_4$ or $A_3{:}a_3$ and $A_4{:}a_4$, but not containing $A_2{:}a_2$. In the case of XML documents by containment of $A{:}a$ we mean the presence of a node/attribute identified by A and having text/value a.

A *node-relation* rule $NR(A_1{:}a_1, \ldots, A_k{:}a_k)$ is represented by a Boolean *node formula.* A *node formula* is recursively defined from a set of *atomic distance formulas* and *atomic node formulas* by applying the standard conjunction and negation operators as well as quantification on nodes. Existential quantification has the form: $\exists N \in P{:}F(N)$, where N is a node appearing in the path P of the XML tree and $F(N)$ is a node formula. Paths can be defined as explained below for *atomic distance formulas.* Since

[1] Alternative matching rules (disjunctive behavior) can be established by adding more release control rules with the same controlled items.

negation is allowed, universal quantification and disjunction are implicitly part of the language.

An *atomic distance formula* has the form:

- $|P_2||P_1| \; op \; |P_2|$, where P_1, P_2 are paths in the XML document tree and *op* is in $\{<,=,>,\geq,\leq\}$,
- $|P| \; op \; k$, where P is a path in the XML document tree, and k is a non-negative integer.

The notation $|P|$ stands for the length of path P in the XML tree of the document. For example, the root node of an XML document is easily characterized by having a path of length *'0'*.

Paths P are either among $P(a_1),...,P(a_k)$, which denote the paths from the root of the XML tree to the nodes associated with a subset of values $a_1,...,a_k$ among the controlled items, or they can be paths obtained from them by applying the following operators:

- $P_1 \cap P_2$ (Path Intersection)
- $P_1 - P_2$ (Path Difference)
- $Prefix_k(P)$ (Path Prefix of length k)
- $Suffix_k(P)$ (Path Suffix of length k)

Each *atomic node formula* compares a node property with a constant value or with the property of a different node. The set of atomic node formulas includes the following:

- Comparison of attribute values
  - $AttrValue(N_1,attrName)$ *op* $AttrValue(N_2,attrName)$, where *op* is in $\{<,=,>,\leq,\geq,\neq\}$ (applies only if the attribute has a numeric value).
  - *AttrValue*(*N,attrName*) *op k,* where k is an integer and *op* is in $\{<,=,>,\leq,\geq,\neq\}$ (applies only if the attribute has a numeric value).
  - $AttrValue(N_1,attrName)$ *rel* $AttrValue(N_2,attrName)$, where *rel* is in $\{=,\neq,substr,...\}$ (applies only if the attribute has a string value).
  - *AttrValue*(*N,attrName*) *rel string,* where *string* is any string and *rel* is in $\{=,\neq,substr,...\}$ (applies only if the attribute has a string value). For example, using conjunction, this can be used to check if a set of nodes has the same attribute value.
- Comparison of node meta properties
  - $Order(N_1)$ *op* $Order(N_2)$, where *op* is in $\{<,=,>,\leq,\geq\}$ and *Order*() is not an attribute, but a function assigning a unique value to each node in the XML tree, according to a specific order (e.g., preorder). This may be used to check if a certain word appears before or after another one in the document.

- *Tag(N) rel string,* where *string* is any string and *rel* is in {=, ≠, *substr*,...}. For example, using conjunction, this can also check if a set of nodes has the same tag <*Table*>.
- *Degree*(*N*) *op k,* where *Degree*(*N*) denotes the number of children of *N* in the tree, *k* is a non-negative integer and *op* is in {<,=,>, ≤, ≥, ≠}. As an example, a leaf node can be characterized by having degree '*0*'.

Example 3. The following matching rule

$\exists N \in Suffix_2(P(N_1))$:*AttrValue(N, "attrName") superstring "relString"*

is satisfied if either the parent node or the parent of the parent node of N_1 has a value for the attribute *"attrName"* which contains the string *"relString"*. Note that N_1 is identified by testing whether it contains one of the values in the controlled items.

Note that paths in atomic distance formulas can be considered constants when a specific XML document is considered. This is different from nodes in atomic node formulas that can be both constants and variables. Indeed, they are constants if the nodes are among the ones corresponding to the values $a_1,...,a_k$ in the controlled items, and variables otherwise. If they are variables they will appear in the node formula under quantification.

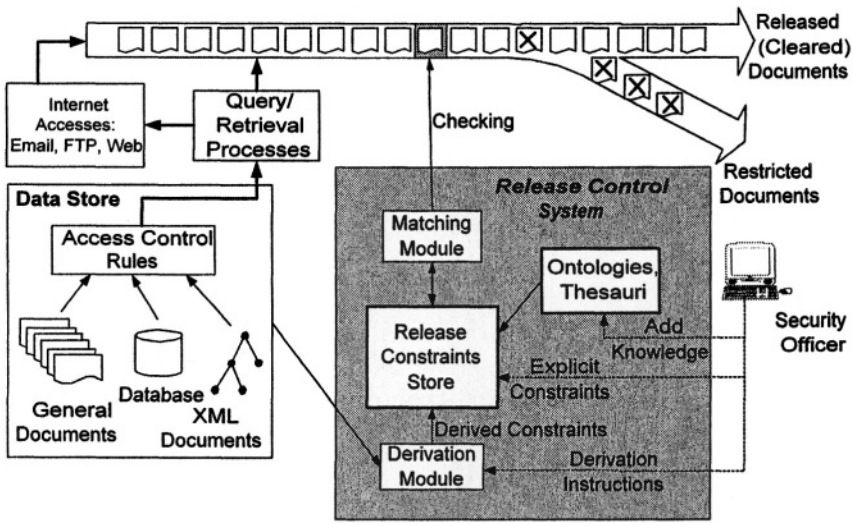

Figure 1. General architecture for release control.

3. RELEASE CONTROL ARCHITECTURE

The architecture we are proposing is based on three basic component: (i) the flow of documents that are going to be released, (ii) a Data Store from which the documents are extracted/derived, and (iii) a Release Control System monitored by a security officer. Figure 1 illustrates these components and their internal structure, ignoring at this level the machinery needed for the learning process.

3.1 The Basic Components

The Data Store can include different types of data sources like standard relational databases, XML or other document type repositories. Documents to be released may be obtained through queries on different data sources in the Data Store as well as through access to internet/intranet services. Databases in the Data Store as well as other sources can be assumed to be protected by usual access control systems.

The main modules in the Release Control System are the *Release Constraints Store* and the *Matching Module.* The first module is the repository of release constraints and includes constraints explicitly inserted by the security officer, constraints derived from the data store with processes guided by the security officer, and constraints derived from examples of restricted and released documents by a learning process, which will be explained later in this section. An example of derivation of constraints from the data store is using the information about access control rules in order to derive sensible associations of terms directly or indirectly bounded to items whose access is forbidden by those rules. Ontologies and thesauri can also be used to derive new release constraints by identifying "semantic" similarities to the given ones. Optimization modules (not depicted in the figure) will also operate on the Release Constraints Store. For example, the set of Release Constraints can be reduced considering the subsumption relationships along the hierarchies of matching rules and controlled items. As an example if RC_1 says that each tuple in the outgoing data should not contain the strings *Woo* and *75k,* and RC_2 says that the string *Woo* should not appear anywhere in the outgoing data, RC_1 can be disregarded.

Given a set of release constraints in the Release Constraints Store, the Matching Module is responsible for checking each one of them against the outgoing documents, and for blocking the release of those for which any of the constraints is satisfied. Since we assume documents in XML form, the module must contain a matching algorithm with a XML parsing component. A basic version of the algorithm may consider an explicit listing of controlled items in each release constraint, and hence, it will perform

keyword-based matching. Clearly, appropriate indexing on the keywords appearing in the release constraints will be necessary, so that all applicable release constraints are efficiently selected upon reading one of the sensitive keyword in the outgoing data. Efficient techniques should also be devised in order to keep track of the current position within the list of controlled items in each release constraint. More sophisticated versions of the algorithm will consider working with compact representations of the controlled items (possibly in the form of queries).

3.2 Integration of a Learning Module

While the security officer in some cases may be able to explicitly provide both controlled items and associated matching rules, we believe there are a large number of documents to be restricted for which only more vague criteria are available. For this reason, our framework proposes the integration in the above architecture of a *Learning Module* that has the main goal of learning release constraints. In particular, in this paper we will show how the learning component can generate specific matching rules starting from controlled items, some domain knowledge, and a training set containing documents already marked as restricted or released.

In principle, given a sufficiently large training set of positive and negative examples we may ask a learning algorithm to derive controlled items and matching rules accordingly to the syntax described above. In practice, this is not a realistic requirement: the learning process, and, in particular, the extraction of features from the examples must be guided by some knowledge about the specific domain. One possibility is to start with a possible set of "critical" correlated keywords from the security officer and with a set of parameterized matching rules. For example, the security officer may consider the distance of two keywords in a document to be a relevant criterion, while the number of occurrences of keywords not to be a relevant one. In this case, the upper bound on the "distance" becomes a parameter to be tuned by the learning process.

The main issue is how to choose appropriate parameterized rules so that the algorithm may minimize the rate of mistakenly released and mistakenly restricted documents by tuning the parameters. In order to illustrate the general idea in the specific context of our matching rules, we give an example, by considering only cardinality rules. As we have observed in Section 2.2, the default and most conservative cardinality rule $NumVal \geq k$ is obtained by using $k=1$. The value of k may actually be used as a parameter in the learning process. For example, from the training set it may be observed that all correctly restricted documents contain at least 2 terms of the controlled items, while many mistakenly restricted ones contain

only one. The value of *k* may then be raised to 2. Of course, there are several hypotheses on the document corpus and on the learning algorithm (including the size of the training set) that should hold to preserve a correct behavior of the system while reducing its mistakes.

In the context of security, it may be desirable to implement a learning process that preserves the following "conservativeness" property:

> *All documents that have been certified as restricted by the security officer will be restricted by the system.*

Preserving this property implies that any derivation of new rules or refinement of existing rules must lead to a global set of release constraints that is still able to correctly classify documents that are known to be restricted.

Figure 2 depicts the Learning Module and its integration in the overall architecture, including a Monitoring Tool that will be discussed below..

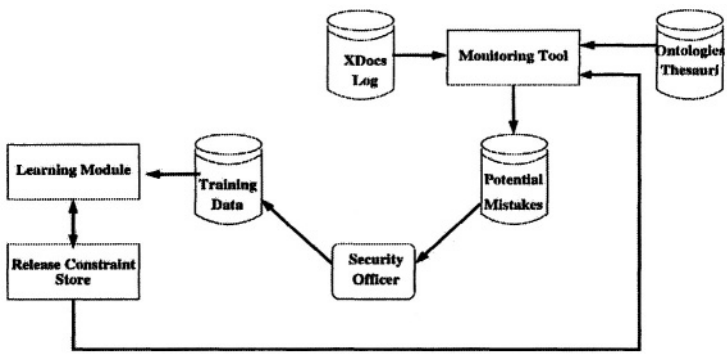

Figure 2. The Learning Module and the Monitoring Tool

3.3 The Learning Module

The learning module has two main functionalities:

- *It derives release constraints for the initial set-up of the system.* As mentioned above, performing this task requires a training set of documents marked to be restricted or released, approved by the security officer; it also requires some domain knowledge, possibly in the form of controlled items and/or parametric rules. We impose that the rules obtained by the learning algorithm will preserve the conservativeness property, i.e., the system using these rules would correctly restrict at least all documents marked to be restricted in

the training set. The algorithms and strategies involved in this task under specific assumptions are described in Section 4.

- *It helps refining the system behavior upon the identification during system operation of mistakenly released and mistakenly restricted documents.* This task is based on the assumption that the security officer monitors the system behavior and provides feedback to the learning module by dividing samples of the processed documents in four categories: correctly restricted (CRes), correctly released (CRel), mistakenly restricted (MRes), and mistakenly released (MRel). It can be considered a form of online learning since the process may automatically start once a large enough set of these samples becomes available.[2] There are essentially two approaches to perform this task: (i) Re-applying the learning process used in the initial set-up considering as the training set the new samples as well as all documents whose classification has been verified in the past by the security officer. When the set becomes too large, many strategies are possible, including, for example, the use of a sliding time window for the past, or of a "least recently used" strategy. The rules obtained through this process replace the previous ones. (ii) Refining the current rules using only the new set of CRes, CRel, MRes, and MRel documents. This is potentially a more efficient approach, but details on how it can be done are very dependent on the learning algorithm.

3.4 The Monitoring Tool

A critical system will have a huge number of documents flowing through it, and specific strategies must be devised to monitor its behavior in order to provide feedback to the learning module. The most trivial strategy consists of periodically extracting samples and forwarding them to the security officer, but it is likely to be unsatisfactory, since any significant frequency of sampling involves an unsustainable amount of work for the security officer.

In our architecture we propose to introduce a monitoring tool that filters the documents based on a "similarity" metric so to drastically reduce the number of documents to be examined by the security officer. Note that in principle the tool should be applied both to restricted documents to identify potentially mistakenly restricted ones, and to released documents to identify potentially mistakenly released ones. However, while mistakenly restricted documents may have other ways to be recognized (e.g. feedback from users whose requests have been refused) and are less critical, the problem is

[2] Actually, considering the conservativeness property, even a single example of mistakenly released document can be useful to refine the rules.

serious for released ones. Also, each restricted document is associated with the release constraint (controlled items and matching rules) that has prevented its release. When the security officer examines the document this association can help recognizing the reason for the sensitivity and, in case of a mistaken restriction may lead to drop or explicitly modify a rule. Released documents on the other side have no attached information. Our focus is on tools to detect potentially mistakenly released documents.

The monitoring tool considers "similarity" of released documents to restricted ones based on the closeness of the document to the classification threshold. This technique is still based on learning and details really depend on the specific learning algorithm, but, intuitively, it works as follows: essential features of documents to be restricted and to be released can be represented by numerical values, and an n-dimensional boundary that separates the two types of documents can be found. Then, a document being examined has the same features computed and the closeness to this boundary can be evaluated. Intuitively, documents that are close to the boundary are similar both to restricted and released ones. The monitoring tool should rank the documents according to their closeness to the boundaries, so that the security officer can dynamically and asynchronously examine them.

The monitoring tool also uses other techniques to identify potential MRel documents:

- The use of ontologies and thesauri to substitute words or structures with those appear in controlled items.
- The explicit relaxation of some of the rules. For example, increase distance (for distance based condition), decrease cardinality, allow regular expression in the string literals, dropping rules by making them always satisfied (e.g., indefinite order, infinite distance).

Intuitively, the application of these techniques, as well as of the learning based one, should be guided by the goal of identifying a very small fraction of the released documents. This is both required due to the limited resources and, more importantly, by the implicit assumption that security officer policies are quite conservative: it is more likely that few restricted documents have been incorrectly released.

These components of the architecture operate asynchronously with respect to the main Release Constraint System.

4. LEARNING FOR KEYWORD-BASED RELEASE CONTROL CONSTRAINTS

As we mentioned in our discussion of the release control system architecture (Section 3), learning for the purpose of obtaining specific

release control constraints plays an essential role. We also mentioned that domain experts or security officers should guide the learning by giving relevant features that the learning process should focus on. In this section, we study, in more detail, such a *feature-based learning* when keywords-based release control constraints are considered.

In general, a feature-based learning requires that domain experts provide certain domain knowledge to the task of learning. The domain knowledge specifies what types of features are important for the task at hand. A learning mechanism is to identify the specific features, within the given types, that can be used to distinguish between different sets of documents.

We believe this approach is rather useful in practice. Indeed, in information retrieval techniques, features of texts are routinely used in deciding the relevance of documents. For example, text appearing in subject line, or title, or abstract, may be more important than that appearing in the body of a document. However, the features used in information retrieval are usually "hard coded" into the information retrieval systems. This may be reasonable for documents that do not have much of structure. When a document is represented in XML, features of various types need to be considered.

Often, specific domains of applications determine the features that are important in XML documents. For example, in applications where documents are structured as a hierarchical tree (e.g., parts contain chapters, chapters contain sections, and section contain paragraphs), it is important to talk about contents belonging to a particular level of the hierarchy (e.g., two sections of the same chapter). Hence, it is important for the domain experts to specify certain 'generic' types of features that are important for the domain.

For release control, the above discussion about domain-specific feature implies the following strategy for generating release control rules. Step 1, the domain experts specify certain type of features that are important. Step 2, the learning system discovers the conditions on the features for the release control purpose. In the following, we illustrate this approach on keyword-based release control.

4.1 Keyword-based features

In this approach, we assume the controlled items are simply keywords, and a feature specifies the particular relationship based on particular ways of appearance of these keywords in a document. The "particular ways" of appearance is a type of feature given by a domain expert. We codify such features by using the notion of a *feature function.*

Formally, we define a *feature function* as a function f such that, given an XML document *Doc* and m nodes of *Doc,* it returns a tuple of n values, i.e., $f(Doc, N_1, ..., N_m) = (a_1, ..., a_n)$.

Note that the functions we define are on the nodes of XML documents. We still call them keyword-based since we will use the appearance of keywords to determine which nodes in the XML document to consider, as we will show later.

Example 4. The following are three example features:

- Distance feature: $dist(Doc, N_1, N_2) = D$, where D is an integer parameter, and $dist(Doc, N_1, N_2)$ is defined as $|P(N_1)| + |P(N_2)| - 2*|P(N_1) \cap P(N_2)|$ and $P(N_i)$ means the path from the root of the XML document *Doc* to node N_i. This feature extracts the distance between the two nodes when the document tree is viewed as a graph.
- Least common ancestor feature: $lca(Doc, N_1, N_2)=T$, where T is the tag of the node defined by $Suffix_1(P(N_1) \cap P(N_2))$. Here T is a string and $P(N_i)$ is the path from the root of *Doc* to node N_i. This feature extracts the tag value of the lowest (in terms of the node level) common ancestor of the two given nodes.
- Ordering: $ocd(Doc, N_1, N_2)=R$, where $Order(N_1)$ R $Order(N_2)$. Here, R is one of the relational comparison symbols. This feature gives the order relationship between two nodes.

In the above examples, each feature function only returns one value (more specifically, a tuple with one value in it).

The above feature functions can naturally be used for specifying conditions on the appearance of keywords in XML documents. For example, given two keywords K_1 and K_2, we want to talk about the appearance of them within certain distance in an XML document. To do this, we only need to specify a condition involving the distance of the nodes that "contain" K_1 and K_2, respectively.

More formally, we define the notion of *occurrences of keywords* in an XML document as follows: Given a set of keywords $\kappa = \{K_1, ..., K_n\}$ and a document *Doc,* an *m-occurrence of* κ *in* Doc, where m is an integer no greater than the number of elements in κ, is a partial mapping *occ* from κ to the nodes of *Doc* such that $occ(K_i)$ contains K_i for each $i=1,..., n$ if $occ(K_i)$ is defined, and the number of K_i with $occ(K_i)$ defined is exactly m. Here, "contains" means either an attribute of $occ(K_i)$ has the value K_i or the value of $occ(K_i)$ is exactly K_i. To simplify the presentation, in the following, we use "contains" to mean that the value of the node is exactly the keyword.

Example 5. In the XML document in Figure 3, there are seven 2-occurrences of *{"A", "B", "K_3"}*. Note that in the tree representation of the XML document in Figure 3, we only show the node values in the XML tree. Other information of the XML documents is omitted. The labels within the nodes (within the circle) are meant to identify the nodes for presentation purpose, and are not part of the XML documents.

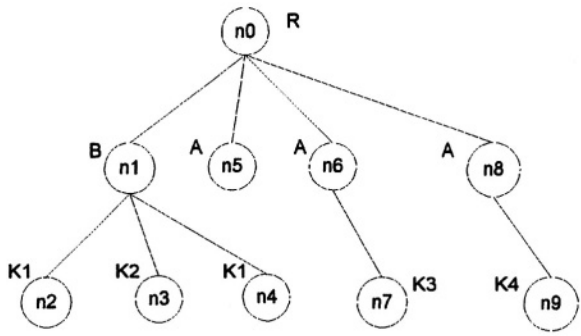

Figure 3. An XML document.

A partial occurrence *occ* of κ in *Doc* is said to be a *maximum* one if *occ* is defined on the maximum number of keywords in κ among all such partial occurrences.

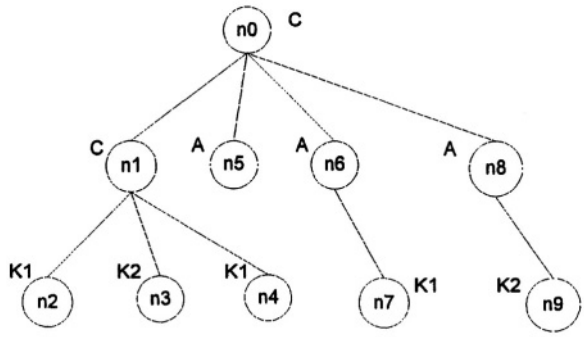

Figure 4. Another XML document.

Now we are ready to extract keyword-based features from XML documents. Assume the feature function takes the form $f(Doc, N_1, \ldots, N_m)=(a_1, \ldots, a_n)$. Then the features extracted from the document via function f is the set of n-tuples given as follows: $(a_1, \ldots, a_n)$ is one such n-tuple if and only if there exists an m-occurrence *occ* of κ in *Doc* such that $f(Doc, occ(K_1), \ldots, occ(K_m))=(a_1, \ldots, a_n)$.

Example 6. Given the distance feature function and the set of keywords $\{$"K_1", "K_2"$\}$, the set of features extracted from the XML document in Figure 3 is $\{(2)\}$. The distance feature of the same keywords extracted from the XML document in Figure 4 is $\{(2), (4)\}$.

Together with a set of keywords (used as controlled items), a Boolean condition based on the features extracted via feature functions can be used as a release control constraint. More specifically, given a document, we extract the features using the feature functions, and then test the Boolean condition on the features. If anyone of the specific features satisfies the condition, we should block the release of the document.

Example 7. Suppose a release control constraint specifies the distance between keywords "K_1" and "K_2" to be less than *3*. Then, none of the documents in Figures 3&4 can be released since each one contains at least one occurrence with distance 2. Suppose another release control constraint specifies the lowest common ancestor of keywords "K_1" and "K_2" to have value *"B"*. Then the document in Figure 3 cannot be released while that in Figure 4 can be.

4.2 Learning for keyword-based release control constraints

A security officer can certainly set up a set of release control constraints by giving a set of keywords and a condition on the keyword-based features (extracted from a set of feature functions). In a practical system, however, it is more likely that system need to learn from a set of examples to establish the specific release control constraints. In this subsection, we will show how this may be done for keyword-based release control constraints.

As mentioned earlier, we assume that (1) a set of feature extraction functions are set up by domain experts as the likely types of features to be concerned by the release control constraints; (2) the security officer gives a collection of keyword sets to be controlled, and (3) a set of documents is provided as learning samples (i.e., the documents either cleared for release or restricted by the security officer). Given (1), (2) and (3) above, the task of learning is to give a set of release control constraints by specifying conditions on the feature values extracted by the feature functions. We now outline a method to do this. The key issue is to convert this learning problem to one where traditional learning algorithms can apply.

Assume κ is the set of keywords we are concerned with. For each feature function $f(Doc, N_1, ..., N_m)=(a_1, ..., a_n)$, we create the following attributes (as in a relational database schema): For each i, $i=1, ..., n$, and each subset $\{K_1,$

..., K_m} (of size m) of κ, we get an attribute name $f[a_i, K_1, \ldots, K_m]$. We use a relational schema of all such attributes. Furthermore, we add a document ID as an extra attribute to the relational schema.

Example 8. Consider the keyword set $\{K_1, K_2, K_3\}$ and the three features given in Example 4. Then we have the following 18 attributes (in addition to the document ID attribute):

$dist[D, K_1, K_2]$	$lca[T, K_1, K_2]$	$ocd[R, K_1, K_2]$
$dist[D, K_2, K_1]$	$lca[T, K_2, K_1]$	$ocd[R, K_2, K_1]$
$dist[D, K_1, K_3]$	$lca[T, K_1, K_3]$	$ocd[R, K_1, K_3]$
$dist[D, K_3, K_1]$	$lca[T, K_3, K_1]$	$ocd[R, K_3, K_1]$
$dist[D, K_2, K_3]$	$lca[T, K_2, K_3]$	$ocd[R, K_2, K_3]$
$dist[D, K_3, K_2]$	$lca[T, K_3, K_2]$	$ocd[R, K_3, K_2]$

Of course, certain relationships between the parameters may lead to a reduction of the number of attributes. For example, since the value for $dist[D, K_1, K_2]$ is the same as $dist[D, K_2, K_1]$, we can omit one if we already have the other. Another example is that if we know $ocd[R, K_1, K_2]$, then we implicitly know $ocd[R, K_2, K_1]$, and we can omit one of these two attributes. From this reasoning, we can reduce the number of attributes in the example to nine (in addition to the document ID attribute).

By using the training samples, we set up feature tuples in the above relational schema as follows. Given an XML document *Doc* and a maximum occurrence *occ* of κ in *Doc*, we generate a tuple as follows: For attribute $a_j[K_1, \ldots, K_m]$, it gets a **null** value if the number of keywords *that occ* is defined on is less than m; otherwise, the attribute gets the corresponding value from $f(occ(K_1), \ldots, occ(K_m))$.

Example 9. Consider keyword set $\kappa = \{K_1, K_2, K_3\}$ and the XML document in Figure 3. Consider the occurrence κ in the document such that K_1 is mapped to n_2, K_2 to n_3 and K_3 to n_7. Then the value for $dist[D, K_1, K_2]$ is 2, while the value for $ocd[R, K_2, K_3]$ is "<".

In the above method, each occurrence of a set of keywords in an XML document sets up a feature tuple in the given relational schema.

Given a set of documents and a set of keywords, we can set up a set of feature tuples for each occurrence of keywords in each document. When we have a set of documents that need to be restricted (determined by the security officer), then the tuples obtained by that document form a set of "restricted" feature tuples. When we have a set of documents that can be released, the tuples obtained by that document form a set of "releasing" feature tuples. Note, however, that there is an apparent difference in the semantics of the

feature tuples in the above two sets. In the "restricted" set of feature tuples, a document needs to be restricted even if only one tuple (among all those that belong to the same document) is "dangerous".

Example 10. Suppose we want to restrict a document from being released when the keywords **"K_1"** and **"K_2"** appear within distance *3*. In this case, both XML documents in Figures 3&4 should be restricted. However, not all the distance features of *"K_1"* and *"K_2"* in Figure 4, namely *2* and *4,* satisfy the restricting constraint.

By reducing the learning task to two sets of feature tuples as given above, we can now apply any traditional learning algorithm [Mit97, Qui96]. In general, the learning algorithm will produce a *classification* condition on the parameter values. That is, given a feature tuple in the given relational schema, the classification condition gives either **restrict** or **release**. For each document, if any of its feature tuple results in the value **restrict**, then the document should be restricted from release. This classification condition will be used as the matching rule in a release control constraint and the corresponding keywords will be the controlled items.

5. RELATED WORK

The concept of information release control has been explicitly introduced recently in [Wie00, Mon01, RW01], but a general formal framework does not currently exist.

Some form of control over outgoing data has been performed since a long time in different contexts, but it has been mostly based on basic filtering tools, and heuristics have been directly coded into programs. An attempt to specify rules in a high level language is represented by *Felt* [Swa94] which, among its features, provides language statements to identify words, or parts of words in documents and to drop or substitute these words. Restricting rules are then compiled into a program for automatic matching. Despite we are not aware of any structured formal framework for release control as the one we are proposing, we should mention the very active research field of information filtering which also includes publication/subscription systems.

An information filtering system is an information system designed for unstructured or semistructured data [BC92], as opposed to typical database applications that work with highly structured data. With respect to the general information retrieval paradigm, in which a large body of data has to be searched against a specific user search criteria, in information filtering it is usually the case that there are a large number of specifications about

information needs of a large number of people and/or tasks, and they all have to be matched against the same text data, in most cases dynamically produced and distributed by some data sources. Publication/subscription systems (see, e.g., [FJL+01] and [ASS+99]) are an instance of information filtering applications. For example, consider the task of sending to each user subscribing to a news service the subset of the daily news specified in his/her profile. The analogy with our work is quite clear: the set of release constraints can be considered a set of subscriptions, and any matching against the outgoing data leads to a specific action, which usually is preventing the release of the data. Despite this analogy, our goal is to deal with a more heterogeneous set of data that includes also structured data resulting from database queries.

Some work has been done specifically on XML documents filtering for publication/subscription systems [AF00, DFFT02, PFJ+01]. We are considering the algorithms and techniques proposed in this area for their adaptation to implement the matching module of our architecture. However, it is still not clear if the algorithms can be adapted to our language for matching rules and if they are compatible with the specific requirements that a security application imposes.

Alternative approaches for the matching module are represented by continuous query techniques [CCC+02, CDTW00, MSHR02].

Our work is also related to what is commonly known as Internet filtering software. Filtering or blocking software restricts access to Internet content through a variety of means. It may scan a Web site's content based on keywords, phrases or strings of text. It may also restrict access based on the source of the information or through a subjective ratings system assigned by an anonymous third party. Mostly, this software has been focused on blocking pornographic content, and it has not been considered very successful until now, either for under-blocking or over-blocking Internet content. This is partly due to the way blocking criteria have been devised and partly from the inherent complexity of the task. Despite some aspects are very related, we are considering several issues about the treatment of structured and semi-structured data while the data considered by these systems is usually unstructured, or the structure it has it is totally unknown. Regarding our learning-based approach, the general techniques to learn the release constraints from a training set of positive and negative examples are well known [Qui96, Mit97]. Learning has been extensively applied in text categorization and text filtering [Seb02], but efforts to study and apply learning techniques for the categorization and filtering of XML documents have just recently started and pose many open questions. We plan to investigate further the problem of learning release constraints and of refining

the constraints considering, in particular, recent statistical learning approaches [CST00].

Regarding our intention of integrating keyword-based techniques with text categorization algorithms, we will take into account the results of the Genoa Technology Integration Experiment [Mon01] performed as part of a DARPA project on boundary control. In the experiment keyword-based and NLP (Natural Language Processing) techniques were compared in their accuracy on a concrete experimental scenario; the experiment involved a corpus of heterogeneous documents that had to be controlled for the release of potentially sensitive information for terrorist attacks. (The scenario referred in particular to the Aum Shinrikyo Japanese cult that bombed the Japanese metro system with an anthrax pathogen.)

Finally, a large amount of work has been done in the past decade on word sense disambiguation (see, e.g., [Les86]), and ontology-based reasoning (see, e.g., [GL02]) which are important issues for any content-based document management application and, in particular, for a more accurate release control. An example of use of word sense disambiguation and natural language processing in boundary control, limited to text documents, is the Genoa Technology Integration Experiment [Mon01] performed as part of a DARPA project on boundary control. In our case ontology-based reasoning is used to add relevant rules and/or feature functions over XML documents.

6. CONCLUSION

In this paper, we presented an architecture for controlled information release. We emphasized on the automated learning (for release control constraints) in the whole infrastructure for release control. We formalized the release control constraints and presented a learning strategy for keyword-based release control constraints.

The above release control system can be useful in a traditional data system, such as database system, FTP directories, and web sites. More recent applications, such as web services, can also benefit from release control. Web services [New02] are an emerging paradigm for internet computing heavily based on XML and on the SOAP protocol. Web services present to the network a standard way of interfacing with back-end software systems, such as DBMS, .NET, J2EE, CORBA objects, adapters to ERP packages, and others. While standards are currently under definition for authentication and authorization, as well as for encryption, controlling the information that is released through a web service to the general internet or to a restricted subset of cooperating processes will be one of the major issues that will also probably affect the success of the new paradigm. While the major objectives

of the proposed project are not to develop a specific technology for web services, we envision a very interesting integration of the technologies that may emerge from our results into a web service architecture.

Content-based firewall is another interesting application of our release control system. Current firewall systems are mostly based on selecting incoming and outgoing packets based on source/destination IP and port numbers. Filtering software based on dirty-word checking or virus identification in some cases have been integrated. The content analysis is however quite primitive both in the definition of the filtering criteria and in the matching algorithms. We advocate an approach that incorporates the release control into firewall systems to allow more advanced monitoring on the contents that are released through the firewall.

ACKNOWLEDGEMENTS

This work was supported by the NSF under grant IIS-0242237. The work of Bettini was also partly supported by Italian MIUR (FIRB "Web-Minds" project). The work of Wang was also partly supported by NSF Career Award 9875114. The authors would like to thank Nicolò Cesa-Bianchi of the University of Milan for insightful discussions on computational learning techniques.

REFERENCES

[AF00] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of 26th International Conference on Very Large Data Bases,* pages 53—64, USA, 2000.

[ASS+99] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC),* pages 53—62, May 1999.

[BC92] N. J. Belkin and W. B. Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM,* 35(12):29—38, December 1992.

[CCC+02] Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams – A new class of data management applications. In *Proceedings of the 28th International Conference on Very Large DataBases (VLDB),* pages 215—226, 2002.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data: May 16—18, 2000, Dallas, Texas,* pages 379—390, 2000.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Machines (and other kernel-based learning methods),* Cambridge University Press, UK, 2000.

[DFFT02] Yanlei Diao, Peter Fischer, Michael Franklin, and Raymond To. Yfilter: Efficient and scalable filtering of xml documents. In *Proceedings of the International Conference on Data Engineering (ICDE),* pages 341—342, 2002.

[FJL+01] Francoise Fabret, H. Arno Jacobsen, Francois Llirbat, Joao Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast Publish/Subscribe systems. In *Proceedings of ACM International Conference on Management of Data (SIGMOD),* pages 115—126, 2001.

[GL02] Michael Gruninger and Jintae Lee. Ontology: applications and design. *Communications of the ACM,* 45(2):39—41, February 2002.

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

[Les86] Michael E. Lesk. Automated sense disambiguation using machine-readable dictionaries: How to tell a pinecone from an ice cream cone. In *Proceedings of the SIGDOC Conference,* 1986.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[Mon01] Eric Monteith. Genoa TIE, advanced boundary controller experiment. In *17th Annual Computer Security Applications Conference.* ACM, 2001.

[MSHR02] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD),* pages 49—60, 2002.

[New02] Eric Newcomer. *Understanding Web Services.* Addison Wesley, 2002.

[PFJ+01] Joao Pereira, Francoise Fabret, H. Arno Jacobsen, Francois Llirbat, and Dennis Shasha. Webfilter: A high-throughput XML-based publish and subscribe system. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB),* pages 723—725, September 2001.

[Qui96] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys,* 28(1):71—72, March 1996.

[RW01] Arnon Rosenthal and Gio Wiederhold. Document release versus data access controls: Two sides of a coin? In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM),* pages 544—546, November 5—10, 2001.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys,* 34(1):1—47, 2002.

[Swa94] V. Swarup. Automatic generation of high assurance security guard filters. In *Proc. 17th NIST-NCSC National Computer Security Conference,* pages 123—141, 1994.

[Wie00] Gio Wiederhold. Protecting information when access is granted for collaboration. In *Proc. of Data and Application Security, Development and Directions, IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security,* pages 1—14, 2000,