CLASSIFYING INFORMATION FOR EXTERNAL RELEASE

S. Dawson[1] S. De Capitani di Vimercati[2] P. Lincoln[1] P. Samarati[1,3]
(1) SRI International, Menlo Park, CA, USA
(2) Università di Brescia, Brescia, Italy
(3) Università di Milano, Milano, Italy

Abstract Organizations in the private, public, and governmental sectors are more and more required to make their data available on the Internet. This demand often involves archival data maintained at the organization, which must be disclosed selectively. We illustrate an approach to classifying information for external release that guarantees protection of sensitive data while maximizing information release.

1. INTRODUCTION

The widespread use of the Internet and the World Wide Web has introduced a globally interconnected society, in which information is available from everywhere at any time. As a result, organizations are often required to publish their data on the network to make them accessible to the external world. This export process often involves historical data, once considered protected and available only internally, that can (or must) be made available outside. Such information sharing and dissemination is clearly selective: not all data can be cleared for release. Some data are private and hence should not be released, some data should be released only to specific classes of users. Publication of the database must then ensure that sensitive information is not improperly disclosed. Ensuring this protection requires withholding from recipients data they are not cleared to see as well as additional data that, although not themselves sensitive, might allow recipients to infer sensitive information. At the same time, it is desirable to avoid unnecessary (i.e., not needed for protection) denials to publish information. As a matter of fact the first reason for undergoing the release process is to make data available to others.

Constraints establishing what data can or cannot be released are often stated with reference to classes of data recipients (a user by user specification would be rather impractical, if at all applicable). Recipient classes may bear relationships with eachother (e.g., data released to a class of users may be a subset of data released to another class of users). This consideration makes mandatory policies, based on the classification of subjects and data, particularly appealing to the problem under consideration. Mandatory policies are also appealing since they allow a clear and formal definition of the constraints. However, although mandatory policies have been largely investi-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35515-3_53

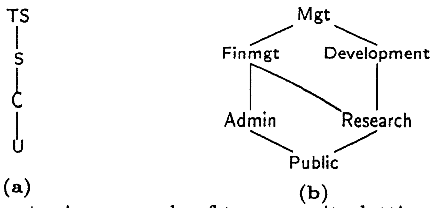

Figure 1 An example of two security lattices

gated and several multilevel models and systems have been proposed [Castano et al., 1995, Jajodia and Sandhu, 1991, Lunt et al., 1990, Sandhu and Chen, 1998, Winslett et al., 1994], the support for inference protection (i.e., indirect disclosure) remains to date very limited (we refer the reader to [Dawson et al., 1999a, Jajodia and Meadows, 1995] for a discussion on related work).

We illustrate an approach to the classification of archival relational databases that must undergo external release. The reason for considering relational databases is that most archival data are indeed relational. We start by investigating different kinds of constraints that can be imposed on information being released, including different types of protection constraints (direct data classification, or indirect classification requirements due to inference) as well as visibility constraints, consideration of which allows the data holder to express the fact that some information should be made available to given classes of subjects. Constraints can be content-dependent, that is, the classification of a data element may depend on its value or the value of other related elements. We present an approach to enforce the constraints that guarantee protection from improper disclosure while maximizing information release.

2. BASIC CONCEPTS AND SCENARIO

We briefly introduce preliminary definitions and concepts of mandatory access control, relational model, and multilevel relations.

Mandatory access control policies are based on definition of access classes, often called *security levels*, and on the assignment of security levels to *subjects* (processes) and *objects* (data) in the system. The security level of a subject, also called *clearance*, reflects the subject's trustworthiness not to improperly disclose information it acquires. The security level of an object reflects the sensitivity of the information in the object. Security levels are partially ordered according to a dominance relationship $\succeq$, where $x \succeq y$ is read *x dominates y*. Security levels together with the dominance relationship defined on them form a security lattice, $\mathcal{L}$. Figure 1 illustrates an example of two security lattices: a totally ordered lattice (Figure 1(a)) with classes top secret (TS), secret (S), confidential (C), and unclassified (U), where TS $\succeq$ S $\succeq$ C $\succeq$ U; and a partially ordered lattice (Figure 1(b)) with Public (publicly available information) as bottom and Mgt (Management restricted information) as top. Accesses are regulated so that information can flow only upwards (from lower to higher sensitivity). In particular, a subject can read only objects at its level or below (*no read up* principle) and write only objects at its level or above (*no write down* principle).

A relational DBMS defines a *database* in terms of *relation schemas*, and *relations* (instances) over them. Formally, let $\mathcal{A}$ be a set of attributes and $\mathcal{D}$ a set of domains. A relation schema R is a finite set of attributes $\{A_1, \ldots, A_m\} \subseteq \mathcal{A}$ each of which is defined on a domain $D_i, i = 1, \ldots, m$. Notation $R(A_1, \ldots, A_m)$ is used to

Department

	name	workarea
t_1	R&D	software
t_2	Security	protection

Technology

	code	description	dept
t_1	tec1	rout. protocol	R&D
t_2	tec2	compiler	R&D
t_3	tec3	sec. tools	Security

Project

	code	topic	platform	requisite	type	techno
t_1	p_1	network	Unix	LANnet	pubdomain	tec1
t_2	p_2	OS	Mac	C	reserved	tec2
t_3	p_3	acc. control	Windows	encry. algo.	reserved	tec3

Employee

	code	name	specialty	rank	salary	dept	prj
t_1	e_1	Bob	network	1	1000	R&D	p_1
t_2	e_2	Tom	opsys	3	3500	R&D	p_2
t_3	e_3	Sam	models	4	3800	Security	p_3

(a)

Department$^\lambda$

name	workarea
R&D Public	software Public
Security Public	protection Public

Technology$^\lambda$

code	description	dept
tec1 Development	rout. protocol Development	R&D Development
tec2 Development	compiler Development	R&D Development
tec3 Development	sec. tools Development	Security Development

Project$^\lambda$

code	topic	platform	requisite	type	techno
p_1 Public	network Research	Unix Research	LANnet Research	pubdomain Finmgt	tec1 Development
p_2 Finmgt	OS Finmgt	Mac Mgt	C Finmgt	reserved Finmgt	tec3 Mgt
p_3 Finmgt	acc. control Finmgt	Windows Mgt	encry. algo. Finmgt	reserved Finmgt	tec2 Mgt

Employee$^\lambda$

code	name	specialty	rank	salary	dept	prj
e_1 Public	Bob Public	network Finmgt	1 Public	1000 Admin	R&D Admin	p_1 Development
e_2 Public	Tom Public	opsys Finmgt	3 Finmgt	3500 Finmgt	R&D Admin	p_2 Mgt
e_3 Public	Sam Public	models Finmgt	4 Finmgt	3800 Finmgt	Security Admin	p_3 Mgt

(b)

Figure 2 An example of database (a) and multilevel database (b)

represent a relation schema R over the set $\{A_1, \ldots, A_m\}$ of attributes. A tuple t over a set of attributes $\{A_1, \ldots, A_k\}$ is a function that associates with each attribute A_i a value $v \in D_i$. Expression $t[A]$ denotes the value v associated with attribute A in t and $R_i.A$ denotes attribute A in R_i; relation names may be omitted when clear from the context. A relation r over relation schema $R(A_1, \ldots, A_m)$ is a set of tuples over the set of attributes $\{A_1, \ldots, A_m\}$. A database $\mathcal{B}$ over a set of relation schemas $\{R_1, \ldots, R_n\}$ is a set of relations $\{r_1, \ldots, r_n\}$, where each r_i is a relation over R_i. Figure 2(a) illustrates a simple example of database storing information about departments, technology, projects, and employees of an organization.

The application of a mandatory policy to a relational database requires the classification of the information in it. Such classification can be applied at different granularity levels (e.g, relation, tuple, or single elements). To make our approach as general as possible, we consider a classification at the element level. Given a relation schema $R(A_1, \ldots, A_m)$ and a security lattice $\mathcal{L}=(L, \succeq)$, a *multilevel relation* with element-level labeling over R, denoted r^λ, is a pair (r, λ), where r is a relation over R and λ is a *labeling function* $\lambda : r \rightarrow L$ such that $\lambda(t[A_i]) = l$ if and only if $t[A_i] \in r$ is classified at level $l \in L$. Accordingly, a *multilevel database* $\mathcal{B}^\lambda$ with element-level labeling over a set of relation schemas $\{R_1, \ldots, R_n\}$ is a set of multilevel relations $\{r_1^{\lambda_1}, \ldots, r_n^{\lambda_n}\}$, where $r_i^{\lambda_i}$ is a multilevel relation over R_i, and $\lambda = \lambda_1 \cup \ldots \cup \lambda_n$. Figure 2(b) illustrates a multilevel version of the database in Figure 2(a), where security levels are taken from the lattice in Figure 1(b).

Figure 3 depicts the basic scenario we consider. We are given an archival relational database, a set of classification constraints, and a security lattice for the interpretation of the mandatory policy. The goal of our approach is to produce a classified database

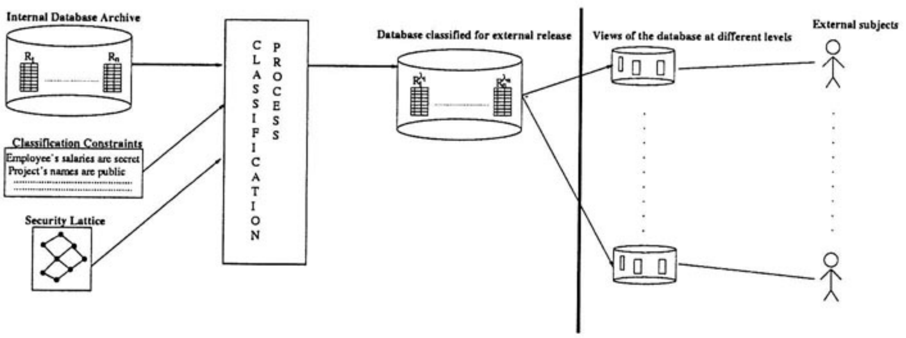
Internal Database Archive
Classification Constraints
Employee's salaries are secret
Project's names are public
Security Lattice
CLASSIFICATION PROCESS
Database classified for external release
Views of the database at different levels
External subjects

Figure 3 Classification of information for external release

that provides the required protection, while maximizing information visibility. The interpretation of the classified database will then produce different database views for different subjects, where each view contains all and only the data whose classification is dominated by the subject clearance. According to how data classifications are determined by the classification process, each view is a subset of the actual database obtained by eliminating all the data that the subject is not cleared to see as well as additional data that would have allowed the subject to infer sensitive information. From an architectural point of view, we can imagine this classification process to be executed by a *wrapper* protecting the database [Dawson et al., 2000].

3. PROBLEM DEFINITION

We investigate the different kinds of classification constraints that data holders may express to impose restrictions on the data release. We also discuss completeness and consistency properties of the constraints and define the concepts of correctness and minimality of a solution.

Classification constraints. Classification constraints specify properties that the classification assigned to data should satisfy. We identify two main classes of classification constraints: *protection constraints* and *visibility constraints*.

Protection constraints impose minimum bounds on levels that can be assigned to elements and possible dominance relationships that must be satisfied between them. Among these constraints we can distinguish *basic*, *association*, *inference*, and *integrity* constraints. Basic constraints specify the minimum classification that can be assigned to an element. For instance, "$\lambda(\texttt{E.salary}) \succeq$ Finmgt where $\texttt{E.rank} \geq 3$" states that the salary of employees with `rank` greater than or equal to 3 should be classified at least Finmgt. Association constraints specify minimum bounds for classifications of *set* of elements, by referring to their least upper bound. Intuitively, they specify the minimum clearance necessary to view all the elements in the set. For instance, "$\texttt{lub}\{\lambda(\texttt{E.name}), \lambda(\texttt{E.salary}) \succeq$ Admin" states that only subjects cleared Admin or above can have visibility of both employee's `name` and `salary` (subjects at other levels might be able to see one of the two attributes but not both). Inference constraints specify relationships that must be satisfied between the classification of an attribute (or set of attributes) and the classification of another attribute. They reflect interrelationships between data that must be taken into account in the classification to prevent indirect (inference) disclosure. For instance, if by knowing the value of a project's `topic` one could infer the project's `requisite`, constraint "λ(P.topic)$\succeq\lambda$(P.requisite)" should be imposed. This way, subjects not cleared to view a project's requisite will not be able to see the topic, and will not therefore be able to make inferences. Integrity constraints specify conditions on the classifications imposed by the multilevel relational data model, rather than from the data. For instance, *primary key* constraints state that all key attributes must have the same level, while *reference* constraints state that the level of foreign key attributes must be greater than or equal to the level of the corresponding key attributes [Lunt et al., 1990]. All these constraints define (directly or indirectly) lower bounds on the levels that can be assigned to elements. Intuitively, if we think of all the information unclassified (bottom) at the beginning of the process, the consideration of protection constraints will bring to increasing the levels of the elements.

Visibility constraints impose *upper bounds* on the classification of attributes. Intuitively they require that certain data *be visible* to certain subjects. For instance, "Public$\succeq$E.name" states that employees' names should be visible to `Public` subjects. With respect to inference problems, visibility constraints can also be used to represent the fact that certain information is already available to certain subjects, and therefore a higher classification assigned by the process would not reflect real knowledge.

In our approach, classification constraints are represented in a uniform and general way as SQL-like statements of the form

set ⟨`labeling expression`⟩ **in** ⟨`relations_list`⟩ **where** ⟨`condition`⟩

where `labeling expression` is an expression of the form $X\succeq Y$, where X is a security level $l \in L$ or is of the form lub$\{\lambda(A_1), \ldots, \lambda(A_n)\}$, with $A_i \in \mathcal{A}$, and Y is a security level $l \in L$ or is of the form $\lambda(A)$ and $A_1, \ldots, A_n, A$ disjoint; `condition` is a boolean expression of mathematical relationships between attributes, or attributes and ground values; and `relations_list` is the set of relations involved in the constraint. The meaning of a constraint is that `labeling expression` must be satisfied on the classification of elements in `relation_list` in all the tuples satisfying the `condition`. Figure 4 illustrates an example of possible classification constraints defined over the database in Figure 2(a) and security lattice in Figure 1(b). For simplicity, field **in** is omitted, and field **where** is omitted when there is no condition associated with the constraint (which intuitively is to be read as condition always true). Constraints, which have been distinguished according to the typologies discussed above, are of immediate interpretation. Completeness constraints guarantee completeness of the specifications and are discussed next.

Constraint completeness and consistency. Given a database $\mathcal{B}$, a security lattice $\mathcal{L}$, and a set C of classification constraints over $\mathcal{B}$ and $\mathcal{L}$, our objective is to produce a multilevel database $\mathcal{B}^\lambda$ that satisfies the set C of constraints. To guarantee the existence of such a database $\mathcal{B}^\lambda$, the set of constraints must be *complete* and *consistent.* A set of constraints is complete if it defines a classification for each possible element in the database. Intuitively, if the set is not complete there may exist elements for which no level is produced. Completeness, if not provided in the initial specifications, can be trivially guaranteed by adding a default constraint "**set** $\lambda(A)\succeq\bot$" for each attribute A in the database whose occurrence may not be completely classified, where $\bot$ denotes the bottom of the lattice. Such a default constraint clearly does not affect the specifications provided as input, since each level in the lattice dominates $\bot$. A set of constraints is consistent if there exists a labeling that satisfies all the constraints, that is, if it possible to find a solution $\mathcal{B}^\lambda$. Unlike completeness, consistency cannot be guaranteed since the presence of both lower and upper bound constraints could force,

```
Visibility (upper bound) constraints
1) set Public⪰λ(E.code)
2) set Public⪰λ(P.code) where type = 'pubdomain'
3) set Research⪰λ(P.topic) where type = 'pubdomain'

Basic classification constraints
4) set λ(E.dept)⪰Admin
5) set λ(E.salary)⪰Admin where E.rank < 3
6) set λ(E.salary)⪰Finmgt where E.rank ≥ 3
7) set λ(E.specialty)⪰Admin
8) set λ(E.prj)⪰Development
9) set λ(T.description)⪰Development
10) set λ(T.dept)⪰Development
11) set λ(T.code)⪰Development
12) set λ(P.code)⪰Finmgt
13) set λ(P.type)⪰Finmgt
14) set λ(P.techno)⪰Research
15) set λ(P.topic)⪰Research
16) set λ(P.platform)⪰Research
17) set λ(P.requisite)⪰Research

Inference and association constraints
18) set lub{λ(E.specialty), λ(D.workarea)}⪰λ(P.topic)
     where E.prj = P.code ∧ E.dept = D.name
19) set λ(P.topic)⪰λ(P.requisite)
20) set lub{λ(P.requisite), λ(P.platform)}⪰λ(T.descr)
     where P.type = 'reserved' ∧ P.techno = T.code
21) set λ(T.description)⪰λ(D.workarea)
     where T.dept = D.name
22) set λ(E.rank)⪰λ(E.salary) where E.rank ≥ 3
23) set lub{λ(E.salary), λ(E.name)}⪰Admin
```

```
Integrity constraints
24) set λ(E.dept)⪰λ(E.name)
25) set λ(E.salary)⪰λ(E.code)
26) set λ(E.rank)⪰λ(E.code)
27) set λ(E.prj)⪰λ(E.code)
28) set λ(E.name)⪰λ(E.code)
29) set λ(E.specialty)⪰λ(E.code)
30) set λ(D.workarea)⪰λ(D.name)
31) set λ(T.description)⪰λ(T.code)
32) set λ(T.dept)⪰λ(T.code)
33) set λ(P.topic)⪰λ(P.code)
34) set λ(P.platform)⪰λ(P.code)
35) set λ(P.requisite)⪰λ(P.code)
36) set λ(P.techno)⪰λ(P.code)
37) set λ(P.type)⪰λ(P.code)
38) set λ(E.dept)⪰λ(D.name)
     where E.dept = D.name
39) set λ(E.prj)⪰λ(P.code)
     where E.prj = P.code
40) set λ(T.dept)⪰λ(D.name)
    where T.dept = D.name
41) set λ(P.techno)⪰λ(T.code)
     where P.techno = T.code

Completeness constraints
42) set λ(E.code)⪰Public
43) set λ(E.rank)⪰Public
44) set λ(E.name)⪰Public
45) set λ(P.code)⪰Public
46) set λ(D.name)⪰Public
47) set λ(D.workarea)⪰Public
```

Figure 4 Possible classification constraints on the relations in Figure 2(a)

on certain elements, dominance relationships that cannot be satisfied. For instance, constraints "set λ(E.name)$\succeq$Admin" and "set Public$\succeq\lambda$(E.name)" are clearly inconsistent, since no classification of `E.name` is possible which satisfies both of them. Our classification process enforces consistency checking at the beginning of the labeling computation, returning an error and avoiding the execution of the process in presence of inconsistencies (see Section 4).

Minimal classification. Given a database $\mathcal{B}$ and a set C of classification constraints, there might be more than one multilevel database $\mathcal{B}^{\lambda}$ satisfying C. Although each of them will provide the required protection, not all of them are equally satisfactory. We are interested in solutions that, while protecting from improper information disclosure (i.e., satisfying the constraints) minimize the information loss due to the classification, where by information loss we intend the nonvisibility of certain data by given subjects. Although the notion of information loss is difficult to make both sufficiently general and precise, it is clear that a first requirement in minimizing information loss is to prevent *overclassification* of data: no database elements should be assigned security levels higher than necessary to satisfy the classification constraints. Consistently, we define minimal a solution $\mathcal{B}^{\lambda}$ such that no other solution $\mathcal{B}^{\lambda'}$ exists which assigns a security level lower than or equal to that assigned by $\mathcal{B}^{\lambda}$ to every element in $\mathcal{B}$. A formal definition of minimal classification can be found in [Dawson et al., 1999b].

4. CONSTRAINTS ENFORCEMENT

Our classification process is based on a previous proposal presented by us in [Dawson et al., 1999a], exploiting a graph interpretation of the constraints, which we have now extended to the consideration of conditions. A set C of classification constraints over relation schemas $\{R_1, \ldots, R_n\}$ and security lattice $\mathcal{L}=(L, \succeq)$ is represented as a graph, called the *constraint graph*, as follows. There is a node for each security level $l \in L$ and for each attribute A appearing in some relation schema R_i. Each

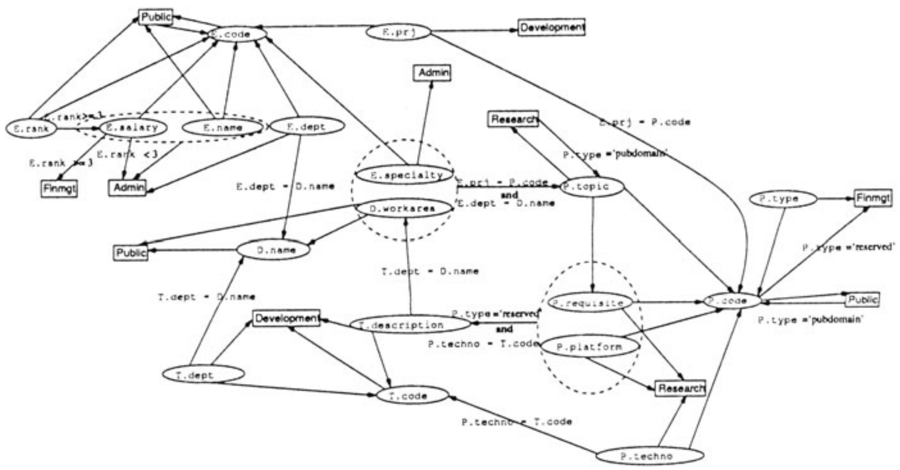

Figure 5 Graphical representation of the constraints in Figure 4

constraint $c \in C$ with a labeling expression of the form $X \succeq Y$ and condition s is represented by an edge labeled s incident to node Y. The starting point of the edge is X, if X is a security level or attribute; it is a hypernode containing all attribute nodes A_i, if $X = \texttt{lub}\{\lambda(A_1), \ldots, \lambda(A_n)\}$. Figure 5 illustrates the constraint graph corresponding to the classification constraints in Figure 4. Attributes are represented as circles, levels as squares, and hypernodes as dashed ellipses. Note that, for the sake of readability, some level nodes have been duplicated.

The classification process, sketched in Figure 6, is divided in two main phases: the first phase enforces upper bound constraints (step 5) and propagates them (step 6), while the second phase (step 7) enforces lower bound constraints and determines the final classification. Both phases enforce constraints by traversing the constraint graph, where for traversal purposes, arcs leaving from a hypernode are interpreted as leaving from each of the nodes in it. Traversal is performed following (possibly in reverse order) the topological order of the graph calculated without consideration of upper bound constraints. Upper bound constraints are ignored in the determination of the topological order as they are evaluated only once pushing them into the node incident of the arc (step 5). At the beginning of the process, the classification λ of each node is initialized to bottom level $\perp$ (step 1), while its maximum allowed level l_{max} is initialized to top level $\top$ (step 2). Constraints are indexed (step 3) so that for each node A, set *Constr*$[A]$ contains all the constraints whose left-hand side includes A. Each attribute A also maintains a flag (*done*$[A]$), initially set to false, that indicates whether all the elements of the attribute have been classified. After this initialization process the set of constraints is interpreted as a graph and its strongly connected component (SCC) determined (step 4), so to allow topological order based-traversals. For instance, a possible topological order for the graph in Figure 5 is: *scc*[1] := {`P.platform`}; *scc*[2] := {`E.specialty`}; *scc*[3] := {`P.requisite`,`P.topic`,`D.workarea`,`T.description`}; *scc*[4] := {`P.techno`}; *scc*[5] := {`T.dept`}; *scc*[6] := {`T.code`}; *scc*[7] := {`P.type`}; *scc*[8] := {`E.prj`}; *scc*[9] := {`P.code`}; *scc*[10] := {`E.dept`}; *scc*[11] := {`D.name`}; *scc*[12] := {`E.name`}; *scc*[13] := {`E.rank`}; *scc*[14] := {`E.salary`}; *scc*[15] := {`E.code`}. Step 5 enforces upper bound constraints by pushing the level imposed by them (left-hand side) into the in-

```
/* Input:  A database $\mathcal{B}$, a security lattice $\mathcal{L}$, and a set $C$ of constraints */
/* Output: A corresponding multilevel database $\mathcal{B}^\lambda$ satisfying $C$ for external release */
1. Set the level $\lambda$ of all elements in $\mathcal{B}$ to $\bot$
2. Set the maximum level $l_{max}$ of all elements in $\mathcal{B}$ to $\top$
3. For each $A \in \mathcal{A}$ do$Constr[A]$:= $\{c \in C \mid A \in lhs(c)\}$; $done[A]$ := FALSE
4. Define the graph $G$ representing $C$ and find a topological order among the SCCs of $G$.
   Let $numscc$ be the number of SCCs
5. /* Push upper bound constraints into hit nodes */
   For each upper bound constraint $c \in C$ do
     For each $t \in$ evaluate($c,\mathcal{B}$) do $l_{max}(t[rhs(c)])$ := $l_{max}(t[lhs(c)])$
6. /* Traverse the graph in topological order pushing upper bound constraints forward */
   For $i$ := 1,... , $numscc$ do
     For each $A \in scc[i]$ do
       For each $c \in Constr[A]$ do
         For each $t \in$ evaluate($c,\mathcal{B}$) do
           If $rhs(c) \in L$ and $\neg(l_{max}(t[lhs(c)]) \succeq rhs(c))$ then return (inconsistency)
           else lower $l_{max}$ of $t[rhs(c)]$ to the g.l.b. between its current value
             and the lub of the $l_{max}$ of the elements $t[lhs(c)]$
             If $rhs \in scc[i]$ then repeat the step for $rhs(c)$
7. /* Traverse the graph in reverse topological order pushing lower bound constraints backward.
     Within each SCC execute a forward propagation process. */
   For $i$ := $numscc$,... ,1 do
     For each $A \in scc[i]$ do
       $done[A]$ := TRUE
         For each $c \in Constr[A]$ do
           If $done[rhs(c)]$ then
             For each $t$ $\in$evaluate($c,\mathcal{B}$)
               If $|lhs(c)|$= 1 then assign $\lambda(t[A])$ the lub between its current value and the $\lambda$ of $rhs(c)$.
               else assign $\lambda(t[A])$ the lub between its current value and the minimum level
                 that $A$ can assume without violating the constraint
           else $done[A]$ := FALSE
         If $\neg done[A]$ then
           For each $c \in Constr[A]$ do
             For each $t \in$ evaluate($c,\mathcal{B}$) do determine a minimal level $l$ that $t[A]$ can assume by trying
               levels walking down lattice $\mathcal{L}$ one step at the time from $l_{max}$ and checking consistency
               by traversing the SCC forward. Lower other elements in the SCC as needed.
             $\lambda(t[A])$ := $l_{max}(t[A])$
         $done[A]$ := TRUE
```

Figure 6 Classification Algorithm

terested element (right-hand side). Then, the graph is traversed in topological order pushing the constraints forward and possibly lowering the maximum allowed level of nodes encountered (step 6). Note that traversal of an arc requires determining the elements actually involved in the dominance relationship expressed by the arc. Function evaluate($c,\mathcal{B}$) performs such a computation in the algorithm. Intuitively, evaluate executes a SQL query on the database and returns the tuples to which the constraint applies (i.e., those on which the condition evaluates to true). Note that the propagation of upper bound constraints is deterministic as no choices can be taken during the propagation but lowering the l_{max} of the hit elements to the greatest lower bound of their current value and the maximum level being pushed through, where the level being pushed through may increase when traversing arcs corresponding to lub constraints (i.e., constraints with more than one attribute on the left-hand side). Possible inconsistencies in the constraints are found in this phase as they cause attempts to push into a constant level node l a maximum allowed level that does not dominate l. At the end of this phase, each element in the database has associated a maximum allowed level l_{max}, which must dominate the final level λ that will be produced. For instance, with reference to our example, the maximum level for t_1[P.topic], and t_1[P.requisite] is set to Research; while the maximum level for

t_1[P.code], t_1[E.code], t_2[E.code], and t_3[E.code] is set to Public. The maximum of all other elements remains in-varied and equal to the top level Mgt.

The second phase (step 7) enforces lower bound constraints, increasing the classification λ of each element until it reaches its final level. A major complication in lower bound constraint propagation is the presence of cycles, intuitively SCC, which are therefore treated separately.

Let us first assume there are no cycles. The graph is traversed in reversed topological order (backtraversing edges) raising the classification λ of the nodes encountered (left-hand side of the constraint being traversed) to be the least upper bound between its value and the level being pushed backwards. Note that, unlike with upper bounds, such propagation along the graph is not deterministic as lub constraints can be satisfied in several ways. For instance, constraint lub$\{\lambda(\text{E.salary}), \lambda(\text{E.name})\}\succeq$Admin could be solved by raising the λ of elements on either E.salary or E.name to satisfy the constraint, or by distributing the "load" on E.salary and E.name. In our approach, the choice of how to solve a lub constraint is based on the topological order: we force the lub constraint on the last attribute in the left-hand side to be flagged done (intuitively the one belonging to the SCC with lowest index). Thus, according to the topological order previously illustrated, the constraint above is solved by first considering elements on E.salary and then elements on E.name. Note that traversal in reversed topological order guarantees that we will (back)proceed from a node when only the corresponding attribute (more precisely, the elements of the attribute) has reached its final level, that is, all arcs leaving from it have been considered. This guarantees that each constraint is evaluated only once.

For cyclic constraints, this backward propagation process is not directly applicable. The problem with cycles is that constraints have to be evaluated more than once (as passing through a cycle the level being pushed backward could increase) and a further pass could invalidate previous choices that increased the classification of certain elements, meaning the classification being produced would not be minimal anymore. Adapting backward propagation to acyclic constraints would then require a considerable effort and log maintenance to enforce backtracking of previously determined λ. To avoid this complicated and expensive process, we deal with cycles in a special way, enforcing on them a forward "push-down" propagation. When entering a SCC with more than one attribute (cyclic constraint) we traverse the graph forward and try to lower the maximum level of the elements we encounter, as low as possible. In particular, for each element, the process starts from the element's current l_{max} and walks down the lattice to a minimal level that can be assigned to the element without violating the constraints. To illustrate, consider the cycle formed by the constraints $\{c_{18}, c_{19}, c_{20}, c_{21}\}$ and assume that l_{max} of all elements to be top level Mgt. When encountering the first attribute in the SCC, let it be P.requisite, the algorithm will try to lower the current maximum level of elements on it by attempting one level at the time walking down the lattice to the lowest possible level. First Finmgt is considered and the attempt to lower the l_{max} of t_i[P.requisite], $i = 1, \ldots, 3$, is evaluated with respect to its consequences on the other elements involved in the SCC. Consideration of the other constraints "set λ(P.requisite)$\succeq$Research", "set λ(P.requisite)$\succeq\lambda$(P.code)", and " set lub$\{\lambda(\text{P.requisite}), \lambda(\text{P.platform})\}\succeq\lambda$(T.description) where P.type = 'reserved' $\wedge$ P.techno = T.code" returns satisfaction without requiring changes. Continuing down the lattice, Admin, is considered. However, the attempt to lower the maximum level of t_i[P.requisite], $i = 1, \ldots, 3$, to Admin fails because it would violate constraint "set λ(P.requisite)$\succeq$Research". Hence, level Research is tried. The lowering attempt succeeds only for element t_1[P.requisite], which will then be assigned maximum level Research. It fails instead for the values of P.requisite in tuples t_2 and t_3 which will be assigned level Finmgt. In a similar way, the process computes the final level of all elements over attributes involved in the considered cycle.

Figure 2(b) illustrates a multilevel database $\mathcal{B}^\lambda$ produced by enforcing the constraints in Figure 4 on the database $\mathcal{B}$ in Figure 2(a).

5. CONCLUSIONS

We have presented an approach for classifying archival relational databases that must undergo external release. Our approach takes into account different kinds of content-dependent constraints and produces a classification that provides the required protection while maximizing information visibility. Future work includes the consideration of dynamic and longitudinal databases, where data updates need to be considered.

Acknowledgments

This work was supported in part by the National Science Foundation under grants ECS-94-22688 and CCR-9509931, and by DARPA/Rome Laboratory under contract F30602-96-C-0337.

References

[Castano et al., 1995] Castano, S., Fugini, M., Martella, G., and Samarati, P. (1995). *Database Security.* Addison-Wesley.

[Dawson et al., 1999a] Dawson, S., De Capitani di Vimercati, S., Lincoln, P., and Samarati, P. (1999a). Minimal data upgrading to prevent inference and association attacks. In *Proc. of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Philadelphia.

[Dawson et al., 1999b] Dawson, S., De Capitani di Vimercati, S., and Samarati, P. (1999b). Specification and enforcement of classification and inference constraints. In *Proc. of the 20th IEEE Symposium on Security and Privacy*, Oakland.

[Dawson et al., 2000] Dawson, S., Qian, S., and Samarati, P. (2000). 'providing security and interoperation of heterogeneous systems. *Distributed and Parallel Databases*, 8(1):119–145.

[Jajodia and Meadows, 1995] Jajodia, S. and Meadows, C. (1995). Inference problems in multilevel secure database management systems. In *Information Security: An Integrated Collection of Essays*, pages 570–584.

[Jajodia and Sandhu, 1991] Jajodia, S. and Sandhu, R. (1991). Toward a multilevel secure relational data model. In *Proc. of the 1991 ACM SIGMOD Conference*, pages 50–59.

[Lunt et al., 1990] Lunt, T., Denning, D., Schell, R., Heckman, M., and Shockley, W. (1990). The seaview security model. *IEEE Transactions on Software Engineering*, 16(6):593–607.

[Sandhu and Chen, 1998] Sandhu, R. and Chen, F. (1998). The multilevel relational (mlr) data model. *ACM Transactions on Information and System Security*, 1(1).

[Winslett et al., 1994] Winslett, M., Smith, K., and Qian, X. (1994). Formal query languages for secure relational databases. *ACM Transactions on Database Systems*, 19(4):626–662.