Chapter 17

PROTECTING DEDUCTIVE DATABASES FROM UNAUTHORIZED RETRIEVALS

Steve Barker

Abstract An approach is presented that addresses the problem of protecting deductive databases from unauthorized disclosures of the information they contain. Our treatment of this problem involves rewriting a database in a form that is guaranteed to allow users to access only the subset of the logical consequences of the database that they are authorized to see. Security requirements can be seamlessly added to the database, and decisions on the legitimacy of access requests can be made by using efficient computational methods for which attractive technical results exist. The approach enables the specification of security requirements to be exploited to help to improve the efficiency of query evaluation, and naturally extends to enable deductive databases to be protected against unauthorized update and revision requests.

1. INTRODUCTION

Little attention has thus far been given to the problem of securing the information contained in deductive databases. In part, this may have been due to the hitherto limited use that has been made of deductive databases in commercial environments. The practical value of deductive databases is, however, increasingly being recognized [13], and deductive databases are predicted to become increasingly important in the future [11]. For deductive databases to become significant in practice, the problem of protecting the information they contain must be addressed.

In [6] and [9], modal logics are considered for specifying the confidentiality of the information contained in a deductive database. However, the suggested approaches are not especially compatible with the methods of representation and computation that deductive databases typically employ. In contrast, our approach uses specifications of security that are consistent with the standard formulation of a deductive database

as a *function-free normal clause theory* [10] and enables standard methods of computation to be exploited to protect deductive databases from unauthorized access requests. Due to space limitations, in this paper we only consider the protection of deductive databases from unauthorized read requests. The extensions required to protect a deductive database from unauthorized modifications are described in [4].

We use a *role-based access control (RBAC)* [14] policy to protect a deductive database. More specifically, we formulate security policies that are based on the $RBAC_1$ model described in [14]. That is, we consider *RBAC* models that permit user and permission assignments on objects to be specified, and include role hierarchies. Amongst other attractions, we believe that the "high-level" nature of authorizations in *RBAC* makes it more suitable for protecting deductive databases than the *discretionary access control (DAC)* policies that DBMSs have traditionally used to protect other types of database.

The rest of this paper is organized thus. In Section 2, some basic notions in deductive databases, theorem-proving and security are outlined. In Section 3, the representation of an *RBAC* model as a clause form theory is discussed. Section 4 describes how a deductive database may be specified as being protected from unauthorized read requests. In Section 5, computational issues are considered. Finally, in Section 6, some conclusions are drawn and suggestions for further work are made.

2. PRELIMINARIES

A deductive database, D, consists of a finite set of ground atomic assertions (i.e., facts) and a finite set of deductive rules. The set of facts is referred to as the *extensional database (EDB)* and the deductive rules are referred to as the *intentional database (IDB).*

To protect D from unauthorized disclosures of information, our approach is to express D in a form (defined below) that ensures that retrievals of information are only possible if the $RBAC_1$ security theory, that defines authorized accesses, permits the read access on D. Henceforth, we will refer to the secure form of D protected from unauthorized read requests as a read *protected database* and we denote an arbitrary read protected database by $\boldsymbol{D*}$. To denote specific instances of D and $\boldsymbol{D*}$, we use D_i and $\boldsymbol{D_i*}$, respectively, where i is a natural number. We also use $\boldsymbol{S*}$ to denote an arbitrary $RBAC_1$ security theory and $\boldsymbol{S_i*}$ to denote a specific instance of an $RBAC_1$ theory where i is a natural number. As we will see, theorem-proving techniques may be used on $\boldsymbol{D*}$

$\cup S*$ to determine whether or not a user's request to perform a read operation on $D*$ is authorized by $S*$ or not.

The protected databases and $RBAC_1$ security theories that we consider consist of a finite set of function-free *normal clauses* [10]. A normal clause takes the form: $H \leftarrow L1,L2,...,Lm$. The *head* of the clause, H, is an *atom* and $L1,L2,...,Lm$ is a conjunction of *literals* that constitutes the *body* of the clause. The conjunction of literals must be true (proved) in order for H to be true (proved). A literal is an atomic formula or its negation; in this context negation is *negation as failure* [10], and the negation of the atom A is denoted by *not A*. A clause with an empty body is an *assertion* or a *fact.*

Since we consider function-free theories, the only terms that appear in $D* \cup S*$ are constants and variables. Henceforth, we will denote the constants that appear in $D* \cup S*$ by symbols that appear in the lowercase in literals. The variables that appear in the literals in $D* \cup S*$ will be denoted by using uppercase symbols (possibly subscripted).

Since $D* \cup S*$ is expressed in normal clause logic, it follows that the *well-founded semantics* [16] may be used for the associated declarative semantics, and that *SLG-resolution* [8] may be used for the corresponding procedural semantics.

When SLG-resolution is used with the normal clause theory $D* \cup S*$, a *search forest* is constructed starting from the *SLG-tree* with its root node labeled by the *goal clause* $Q \leftarrow Q$ [8]. From the soundness and (search space) completeness of SLG-resolution (for flounder-free computations), Q is true in $WFM(D* \cup S*)$(where $WFM(D* \cup S*)$ is the well-founded model of $D* \cup S*$) iff there is an SLG-derivation for $Q \leftarrow Q$ on $D* \cup S*$ that terminates with the answer clause $Q \leftarrow$. That is, $Q \in WFM(D* \cup S*)$ iff the body of $Q \leftarrow Q$ is reduced to an empty set of literals. In contrast, Q is false in $WFM(D* \cup S*)$ iff all possible derivations of $Q \leftarrow Q$ either finitely fail or fail infinitely due to positive recursion; Q has an undefined truth value in all other cases. In the case where Q has an undefined truth value, SLG-resolution produces conditional answers of the form $Q \leftarrow \delta$ viz. Q is true if δ is true where δ is a nonempty set of *delayed* negative literals [8].

The soundness of SLG-resolution is important from a security perspective since it ensures that no unauthorized access request is permitted from an SLG-derivation on $D* \cup S*$; completeness is important since it implies that non-floundering SLG-resolution is sufficiently strong to ensure that no authorized access request is ever denied.

Given that the only terms that are included in $D* \cup S*$ are constants and variables, it follows that $D* \cup S*$ satisfies the *bounded-term-size property* [17]; SLG-resolution is guaranteed to terminate for theories

that satisfy this property [8]. Moreover, SLG-resolution has polynomial time *data complexity* [17] for function-free normal theories [15].

We assume that a *security administrator (SA)* is responsible for specifying $D* \cup S*$. We also assume that a *closed policy* [7] is to be used for protecting a deductive database. It is, however, entirely straightforward to modify our approach to implement an *open policy* [7] or any number of hybrid (i.e., open/closed) policies for protected databases.

Whilst we recognize that the session concept [14] is an important aspect of RBAC, we will not consider role activation/deactivation in the discussion below. In the examples of the evaluation of access requests that we consider later, we simply assume that a user has active the set of roles that is necessary to read from a protected database. However, it should be noted that role activation/deactivation can be naturally accommodated in the approach we describe (see [5]).

3. $RBAC_1$ AS A LOGICAL THEORY

The minimum requirements of any *RBAC model* are that it provides means for specifying that users are assigned to roles and permissions to perform operations on database objects are associated with a role. The database objects to be protected in a deductive database are *n*-ary predicates.

The assignment of users and permissions to roles is represented in our approach by an *SA* including definitions of *ura(U,R)* and *rpa(R,P,O)* predicates in a clause form theory that represents an application-specific $RBAC_1$ security policy, an $RBAC_1$ theory.

In the *ura(U,R)* relation, the predicate name *ura* is shorthand for *user-role assignment;* instances of *ura* are used in an $RBAC_1$ theory to represent that user *U* is assigned to a role *R*. Similarly, *rpa(R,P,O)* stands for *role-permission assignment*; instances of *rpa* in an $RBAC_1$ theory are used to specify that the role *R* is assigned the permission to perform a *P* operation on a database object *O*. Since we only consider protection against unauthorized retrieval requests, *P=read* in this paper.

The *ura(U,R)* and *rpa(R,P,O)* relations are defined by a *SA* using normal clauses. For example, $ura(bob,r1) \leftarrow$ specifies that the user *Bob* is assigned to the role *r1*; $rpa(r1,read,q(V,Y,Z)) \leftarrow V \neq a$ specifies that role *r1* is assigned read permission on any instance of *q(V,Y,Z)* such that *V* is not equal to *a*; $rpa(R,read,r(X,Y)) \leftarrow$ specifies that all roles are permitted to read all instances of *r* (i.e., read access on *r* is publicly accessible); and $rpa(r1,read,s(a,Y,Z)) \leftarrow Z < 20$ specifies that the role

r1 has the read permission on instances of *s* such that the first argument of *s(V,Y,Z)* is constrained to be *a* and *Z* values are less than 20.

Additional authorization rules can be straightforwardly specified by formulating *ura* or *rpa* definitions in terms of *ura*, *rpa, not ura* or *not rpa* conditions. Since constants and variables are used in their specification, *ura* and *rpa* definitions can be as specific or as general as is required.

In addition to user-role and permission-role assignments, role hierarchies are the other key component of $RBAC_1$. Role hierarchies are used to represent the idea that, unless constraints are imposed, "senior roles" (more powerful roles) inherit the (positive) permissions assigned to roles that are "junior" (less powerful) to them in a hierarchy (but not conversely). In cases where the unconstrained upward inheritance of positive permissions is not appropriate, only minor modifications of our representation of $RBAC_1$ are required to enforce alternative policies.

To represent an $RBAC_1$ role hierarchy, a *SA* uses ground instances of a binary relation to describe the pairs of roles that are involved in a "seniority" relationship in the partial order $(R,>)$ that represents a role hierarchy; R is a set of roles; and $>$ is a "senior to" relation.

In more formal terms, a role *R1* is *senior to* role *R2* in a role hierarchy, *RH*, iff there is a path from *R1* to *R2* in *RH* such that $R1 > R2$ holds in the partial order describing *RH*. The reflexive, antisymmetric and transitive *senior to* relation (i.e. $>$) may be defined in terms of an irreflexive and intransitive relation, "directly senior to". The *directly senior to* relation, denoted by $\rightarrow$, may be defined (since $>$ is not dense) in the following way (where $\wedge$ is logical 'and', $\neg$ is classical negation, and $Ri\ (i \in \{1,2,3\})$ is an arbitrary role):

$$\forall R1,R2\ [R1 \rightarrow R2 \text{ iff } R1 > R2 \wedge R1 \neq R2 \wedge \neg\exists R3[R1 > R3 \wedge R3 > R2 \wedge R1 \neq R3 \wedge R2 \neq R3]]$$

A *SA* uses ground instances of a binary *d-s* predicate in an $RBAC_1$ theory to record the pairs of roles that are involved in a "directly senior to" relationship. That is, the assertion $d\text{-}s(r_i,r_j)$ is used to record that role r_i is directly senior to the role r_j in an $RBAC_1$ role hierarchy.

The following set of clauses define the *senior to* relation (where '_' is an anonymous variable):

```
senior-to(R1,R1)←d-s(R1,_)
senior-to(R1,R1)←d-s(_,R1)
senior-to(R1,R2)←d-s(R1,R2)
senior-to(R1,R2)←d-s(R1,R3),senior-to(R3,R2)
```

The *senior-to* predicate is used in the definition of the *permitted* clause that follows:

$$permitted(U,read,O) \leftarrow ura(U,R1),senior\text{-}to(R1,R2),rpa(R2,read,O)$$

The *permitted* clause expresses that a user U is authorized to perform the *read* operation on an object O if U is assigned to a role $R1$ that is senior to the role $R2$ in an $RBAC_1$ role hierarchy associated with the $RBAC_1$ theory, and $R2$ has been assigned the *read* permission on O.

In implementations of our approach, *senior-to* should be stored as a *persistent relation* [1] that is recomputed only when changes are made to the set of *d-s* assertions in an $RBAC_1$ theory; it is not computed each time *permitted(U,read,O)* is evaluated.

The clauses defining *senior-to* are included in every instance of an $RBAC_1$ theory; the application-specific *d-s*, *ura* and *rpa* clauses define a particular instance of an $RBAC_1$ theory. For all "meaningful" $RBAC_1$ theories, the application-specific *d-s*, *ura*, and *rpa* clauses will be *acyclic* [2]. Although *senior-to* violates the acyclicity property, it is nevertheless negation-free. It follows therefore that any instance of an $RBAC_1$ theory is *locally stratified* and has a unique 2-valued *perfect model* [12] that coincides with the total well-founded model of the theory [8]. An important corollary of $RBAC_1$ theories being categorical and having a 2-valued model is that $RBAC_1$ theories define a consistent and unambiguous set of authorizations.

4. READ PROTECTED DATABASES

To see what is involved in protecting a deductive database D from unauthorized read requests, consider an *IDB* clause in D having the following general form:

$$H \leftarrow A1,...,Am,not\ B1,...,not\ Bn \qquad (m \geq 0,\ n \geq 0)$$

This type of clause is interpreted declaratively as stating that: H is true (or provable) in D iff $\forall Ai\ (i \in \{1,..,m\})\ Ai$ is true (or provable) in D and $\forall Bj\ (j \in \{1,..,n\})\ Bj$ is false (or not provable) in D.

When a clause defining H is to be included in the protected form, D_*, of D, the required reading of the protected form of H is that: H is true (or provable) in $D*$ as far as a user U of $D*$ is concerned if $\forall Ai\ (i \in \{1,..,m\})\ Ai$ is true in D^* and U is authorized (from $S*$) to know that Ai is true in D^* and $\forall Bj\ (j \in \{1,..,n\})$ either Bj is not true (not provable)

in $D*$ or U is not authorized (from $S*$) to know that Bj is true in $D*$. If U is not authorized to know that a literal C is true in $D*$ then we regard C as being false in $D*$ as far as U is concerned.

To define the read protection on H we use the following form of the *holds* predicate:

$$holds(U,read,H,D*,S*) \text{ iff } [S* \models permitted(U,read,H) \text{ and } D* \models H]$$

The $holds(U,read,H,D*,S*)$ definition expresses that U can read H (i.e., can know that H is true) from a protected database $D*$ iff U has the read permission on H from $S*$ and H is true in $D*$.

From the definition of *holds* we also have:

$$\neg holds(U,read,H,D*,S*) \text{ iff } [S* \not\models permitted(U,read,H) \text{ or } D* \not\models H]$$

This equivalence is consistent with our declarative reading of a read protected clause. That is, U cannot read H from $D*$ if U is not authorized to read H in $D*$ by $S*$ or if H is not true (provable) in $D*$.

Assuming that the 5-ary *holds* predicate is used with respect to a given instance of $D* \cup S*$ the read protected form of the *IDB* predicate H may be expressed in clausal form thus:

$$\begin{aligned} holds(U,read,H) \leftarrow & permitted(U,read,H), \\ & holds(U,read,A1),\ldots,holds(U,read,Am), \\ & not\ holds(U,read,B1),\ldots,not\ holds(U,read,Bn) \end{aligned}$$

That is, U reads H from $D*$ (i.e., U knows that H is true in $D*$) iff U is permitted to read H and for all Ai literals $(i \in \{1,..,m\})$, U is authorized to know that Ai is true in $D*$ and Ai is true in U's view of $D*$ and for all Bj $(j \in \{1,..,n\})$ Bj is either not in U's authorized view of $D*$ or Bj is false in $D*$.

The read protection for each of the predicates $A1,...,Am,B1,...,Bn$ that appears in the body of $holds(U,read,H)$ is defined in essentially the same way as that for H.

In the case of an n-ary *IDB* predicate I appearing in the body of $holds(U,read,H)$, the head of the read protected form of I in $D*$ will be $holds(U,read,I(t_1,t_2,...,t_n))$ (where tj $(j=(1,..,n))$ is a term) and the body of the protected form of I will include a $permitted(U,read,I(t_1,t_2,...,t_n))$ condition. The rest of the body of I in $D*$ will be a conjunction of $holds(U,read,Ak)$ or $not\ holds(U,read,Bl)$ conditions where Ak (Bl) is a positive (negative) literal appearing in the body of I in D.

In the case where the predicate to be protected in the body of *holds(U, read,H)* is an *n*-ary *EDB* relation, *E*, the head of the protected clause for *E* is *holds(U,read,E(X_1,..,X_n))* and the body of this clause includes a *permitted(U,read,E(X_1,...,X_n))* condition together with *E(X_1,...,X_n)* itself. That is, for every *n*-ary *EDB* relation *E* the read protected form of *E* is:

$$holds(U,read,E(X_1,..,X_n)) \leftarrow permitted(U,read,E(X_1,..,X_n)),E(X_1,..,X_n)$$

The set of assertions that appear in the *EDB* of *D* do not change when *D* is rewritten in its protected form, $D*$. As such, to simplify the examples of protected databases that follow, we will omit assertions in $D*$.

Example 1 (The representation of a read protected databases)
Consider the database, $D_1=\{p(a,Y,Z)\leftarrow r(a,Y),s(Y,Z);\ r(a,Y)\leftarrow t(a,Y);\ r(b,Y)\leftarrow t(b,Y);\ t(a,b)\leftarrow;\ t(b,b)\leftarrow;\ s(b,10)\leftarrow\}$, where *s* and *t* are *EDB* relations and ';' separates clauses. The protected form of D_1, D_1*, is:

$$holds(U,read,p(a,Y,Z))\leftarrow permitted(U,read,p(a,Y,Z)),\ holds(U,read,r(a,Y)),holds(U,read,s(Y,Z))$$
$$holds(U,read,r(a,Y))\leftarrow permitted(U,read,r(a,Y)),holds(U,read,t(a,Y))$$
$$holds(U,read,r(b,Y))\leftarrow permitted(U,read,r(b,Y)),holds(U,read,t(b,Y))$$
$$holds(U,read,t(X,Y))\leftarrow permitted(U,read,t(X,Y)),t(X,Y)$$
$$holds(U,read,s(Y,Z))\leftarrow permitted(U,read,s(Y,Z)),s(Y,Z)$$

An example of an $RBAC_1$ security theory applicable to D_1* is:

$$S_1*=\{ura(bob,r1)\leftarrow;\ d\text{-}s(r1,r2)\leftarrow;\ rpa(r1,read,p(a,Y,Z))\leftarrow Z<20;\ rpa(r2,read,r(a,Y))\leftarrow;\ rpa(r1,read,s(Y,Z))\leftarrow;\ rpa(r1,read,t(X,Y))\leftarrow\}$$

5. COMPUTATIONAL ISSUES

Since $D* \cup S*$ is a function-free normal clause theory it follows that SLG-resolution may be used with $D* \cup S*$ to decide whether a user *U* has the read permission on an instance of an *n*-ary predicate *p(t1,t2,...,tn)* in *D* (where t_j *(j=(1..n))* is a term). To simplify the ensuing discussion we will assume that $D*$ is locally stratified. In this case, answer clauses generated by SLG-resolution will have an empty set of delayed literals. Our approach naturally extends to the case where $D*$ is not locally stratified and where answer clauses may not have a non-empty set of delayed literals.

In our approach, a query Q that includes *holds(U,read,p(t1,t2,...,tn))* (*not holds(U,read,p(t1,t2,...,tn))* literals may be directly posed by U and will be evaluated on $\boldsymbol{D^* \cup S^*}$ iff U is authenticated to be the specifier of the *holds* (*not holds*) subgoals in Q. It follows from the discussion above that *holds(U,read,p(t1,t2,...,tn))* will be true in $\boldsymbol{D^*}$ iff *p(t1,t2,...,tn)* is true in D and U is authorized to know that *p(t1,t2,...,tn)* is true in D. If an SLG-tree with the root $\boldsymbol{Q \leftarrow Q}$ where Q is *holds(U,read, p(t1,t2,...,tn))* terminates with $\boldsymbol{Q\theta \leftarrow}$ as an answer clause and $\boldsymbol{\theta}$ is an answer substitution then $\boldsymbol{p(t1,t2,...,tn)\theta}$ is true in D and U is authorized to know that $\boldsymbol{p(t1,t2,...,tn)\theta}$ is true in D by $\boldsymbol{S^*}$. If Q is *not holds(U,read,p(c1,c2,...,cn))* (where *ci* (i=(*1..n*)) is a constant to ensure *safe evaluation* [10]) and $\boldsymbol{holds(U,read,p(c1,c2,...,cn)) \leftarrow}$ is an answer clause by SLG-resolution on $\boldsymbol{D^* \cup S^*}$ then U is authorized to know that *p(c1,c2,..,cn)* is true (provable) from $\boldsymbol{D^* \cup S^*}$, $\boldsymbol{p(c1,c2,..,cn)}$ is true (provable) in D and *not* $\boldsymbol{holds(U,read,p(c1,c2,...,cn))}$ is *failed* by SLG-resolution. That is, $\boldsymbol{\neg p(c1,c2,...,cn)}$ is false in U's view of D since *p(c1,c2,...,cn)* is true in U's authorized view of $\boldsymbol{D^*}$. Conversely, if Q is *holds(U,read,p(t1, t2,...,tn))* and $\boldsymbol{Q \leftarrow Q}$ is failed by SLG-resolution on $\boldsymbol{D^* \cup S^*}$ then either *p(t1,t2,...,tn)* is false in D or U is not permitted to know that *p(t1,t2,...,tn)* is true in D; either way, *p(t1,t2,...,tn)* is false in D from U's perspective. If the evaluation of *not holds(U,read,p(c1,c2,...,cn))* succeeds by SLG-resolution then U is either not authorized to know that *p(c1,c2,...,cn)* is true in D from $\boldsymbol{D^* \cup S^*}$ or *p(c1,c2,...,cn)* is false in D. In either case, $\boldsymbol{\neg p(c1,c2,...,cn)}$ is true in U's view of D.

Example 2 (Read Protection)
Suppose that Bob poses the query to retrieve all instances of p from D_1 in Example 1 and that $\boldsymbol{S_1^*}$ (also from Example 1) is the $RBAC_1$ theory for $\boldsymbol{D_1^*}$. In this case, SLG-resolution produces the following set of answers: {*X=a, Y=b, Z=10*}. That is, only *p(a,b,10)* may be disclosed to Bob. Though $\boldsymbol{WFM(D_1) \models p(b,b,10)}$, $\boldsymbol{holds(bob,read,p(b,b,10)) \leftarrow}$ is not an answer clause by SLG-resolution since Bob is not authorized to know that *p(b,b,10)* is true in D_1. □

Query evaluation on $\boldsymbol{D^* \cup S^*}$ may appear to be more expensive than on D since the clauses in $\boldsymbol{D^*}$ include additional *permitted* conditions. Moreover, $\boldsymbol{S^*}$ is part of the theory on which query evaluation is performed. However, the security information in $\boldsymbol{D^* \cup S^*}$ has the effect of partitioning D into a number of subsets. An individual user can retrieve information from the one subset of $\boldsymbol{D^*}$ that $\boldsymbol{S^*}$ authorizes them to retrieve from. Hence, a user's queries are evaluated with respect to "their"

subset of D, and since this subset is typically a smaller theory than D, a user's queries may be *more* efficiently performed on $D*$ than D. From a computational perspective, constants or restricting conditions (i.e., those involving comparison operators) from $S*$ may be introduced at an early stage in the evaluation of a user's query to constrain the search space of authorized solutions. Moreover, $S*$ may cause computations to fail early if a user has insufficient read permission to evaluate a query.

In all cases of failure of a goal clause, G, the user, U, receives a "no" response. From this, U cannot infer that $\neg G$ is true in the database D; G may have failed because U is not permitted to know that G is true in D.

Example 3 (Queries involving negation)
Consider the database, $D_2=\{p(X)\leftarrow not\ q(X);\ q(b)\leftarrow\}$, where q is an *EBD* relation, protected by the security theory, S_2*, where:

$$S_2*=\{ura(sue,r1)\leftarrow;\ rpa(r1,read,p(X))\leftarrow;\ rpa(r2,read,q(a))\leftarrow;\ d\text{-}s(r1,r2)\leftarrow\}$$

For D_2* we have:

$$holds(U,read,p(X))\leftarrow permitted(U,read,p(X)),not\ holds(U,read,q(X))$$
$$holds(U,read,q(X))\leftarrow permitted(U,read,q(X)),q(X)$$

Now, suppose that Sue wishes to know whether $p(a)$ is true in D_2. Since $holds(sue,read,p(a))\leftarrow$ may be generated from $D_2* \cup S_2*$ by SLG-resolution, Sue is permitted to know that $p(a)$ is true in D_2. □

The next example shows that recursive databases require no special treatment to protect them from unauthorized disclosures of information.

Example 4 (Recursive query evaluation)
Consider the database, $D_3=\{q(X,Y)\leftarrow r(X,Y);\ q(X,Y)\leftarrow r(X,Z),q(Z,Y);\ r(a,b)\leftarrow;\ r(b,c)\leftarrow\}$, where r is an *EDB* relation, and suppose that the security theory, S_3*, applies to D_3 where:

$$S_3*=\{ura(jim,r1)\leftarrow;\ rpa(r1,read,q(a,Y))\leftarrow;\ rpa(r2,read,r(X,Y))\leftarrow;\ d\text{-}s(r1,r2)\leftarrow\}$$

For D_3* we have:

$$holds(U,read,q(X,Y)) \leftarrow permitted(U,read,q(X,Y)), holds(U,read,r(X,Y))$$
$$holds(U,read,q(X,Y)) \leftarrow permitted(U,read,q(X,Y)), holds(U,read,r(X,Z)), holds(U,read,q(Z,Y))$$
$$holds(U,read,r(X,Z)) \leftarrow permitted(U,read,r(X,Z)), r(X,Z)$$

Now, suppose that Jim poses the query to list all instances of $q(X, Y)$ such that $X=a$. By SLG-resolution, with respect to $D_3^* \cup S_3^*$, the answer $Y=b$ is generated. It follows that only the answer $Y=b$ is revealed to Jim. Although $WFM(D_3) \models q(a,c)$ and Jim is permitted to see any instance of $q(X,Y)$ if $X=a$, for $q(a,c)$ to be disclosed to Jim, Jim must be permitted to know that both $WFM(D_3) \models q(a,b)$ and $WFM(D_3) \models q(b,c)$. However, since Jim does not have read access on $q(X, Y)$ where $X=b$ the fact that $WFM(D_3) \models q(a,c)$ cannot be divulged to him. □

6. CONCLUSIONS AND FURTHER WORK

By adopting the approach that we have described, it is possible to protect any normal deductive database from unauthorized reads of its data by specifying a user's access permissions in the same language that is ordinarily used to describe deductive databases. Moreover, no special methods of computation are required for query evaluation on read protected databases.

Our approach may be used to protect the information in databases defined in subsets of normal clause logic (e.g., relational databases), and may be used to protect the information contained in other types of "logic-based" databases (e.g., abductive databases). What is more, the approach enables *RBAC* to be directly represented in candidate deductive DBMSs (e.g., *XSB* [15]) without requiring any security-specific features.

In future work, we intend to investigate how our recent work on temporal *RBAC* [3] may be combined with the ideas presented here.

References

[1] Abiteboul, S., Hull, R., and Vianu, V., *Foundations of Databases*, Addison-Wesley, 1995.

[2] Apt, K., and Bezem, M., Acyclic Programs, *New Generation Computing*, 1990.

[3] Barker, S., Data Protection by Logic Programming, *1st International Conference on Computational Logic*, LNAI 1861, Springer, 2000.

[4] Barker, S., Secure Deductive Databases, *3rd International Workshop on Practical Applications of Declarative Languages (PADL'01)*, 2001.

[5] Barker, S., Access Control by Constraint Satisfaction, *To Appear.*

[6] Bonatti, P., Kraus, S., and Subrahmanian, V., Foundations of Secure Deductive Databases, *IEEE TKDE*, 7(3), 1995.

[7] Castano, S., Fugini, M., Martella, G., and Samarati, P., *Database Security*, Addison Wesley, 1995.

[8] Chen, W., and Warren, D., Tabled Evaluation with Delaying for General Logic Programs, *J. ACM*, 43(1), 1996.

[9] Cuppens, F., and Demolombe, R., A Modal Logical Framework for Security Policies, *ISMIS'97*, 1997.

[10] LLoyd, J., *Foundations of Logic Programming*, Springer, 1987.

[11] Minker, J., Logic and Databases: A 20 Year Retrospective, *1st I'nat. Workshop on Logic in Databases*, LNCS 1154, Springer, 1996.

[12] Przymusinski, T., Perfect Model Semantics, *Proc. 5th ICLP*, 1988.

[13] Ramakrishnan, R., *Applications of Logic Databases*, Kluwer, 1995.

[14] Sandhu, R., Coyne, E., Feinstein, H., and Youman, C., Role-Based Access Control Models, *IEEE Computer*, 1996.

[15] Sagonas, K., Swift, T., Warren, D., Freire, J., Rao. P., The XSB System, Version 2.0, Programmer's Manual, 1999.

[16] Van Gelder, A., Ross, K., and Schlipf, J., The Well-Founded Semantics for General Logic Programs, *J. ACM*, 38(3), 1991.

[17] Vardi, M., The Complexity of Query Languages, *ACM Symp. on the Theory of Computing*, May, 1982.