18

TEMPORAL AUTHORIZATION IN THE SIMPLIFIED EVENT CALCULUS

Steve Barker

Abstract This paper shows how the *simplified event calculus (SEC)* may be used to represent security models for discretionary access control when access rights may be expressed as holding for limited periods of time.

The approach involves formulating a set of axioms to represent a specific security model with time-constrained authorizations. These model-specific axioms are combined with a set of rules which represent the core axiom of the SEC and a set of ground atomic assertions which record a history of security events which affect a database. In this framework, a subject's request to exercise an access right on a database object is allowed only if authorization can be proved from the axiomatization.

An example security model is presented to demonstrate the approach, extensions to this model are outlined and implementation issues are discussed.

Keywords: Event calculus, temporal authorization model

1. INTRODUCTION

The need to protect a database against unauthorized access has long been recognized as important [4]. However, while languages for expressing access rights and methods for deciding questions of authorization have been supported in commercial database management systems for some time, more powerful means for expressing security information are increasingly being demanded. One such demand is for access control methods which permit privileges on database objects to be granted to subjects for limited periods of time [14].

When access rights are only allowed for a certain duration they are automatically removed on the expiration of the interval of time for which the rights were initially permitted. The facility which enables access to be specified as lasting for a restricted interval of time gives the grantor of a right a good deal

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: 10.1007/978-0-387-35508-5_22

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

of control over the permissions they give to grantees and restricts the scope the latter have for compromising the security of the data contained in a database.

In this paper a method is presented which may be used for discretionary access control when access rights may be restricted by time. As such, our work addresses similar issues to those considered in [2]. Our approach is, however, quite different and has certain advantages.

In [2], a language is described which enables users to specify *temporal authorizations* and *derivation rules*. The former are used to explicitly record the rights individuals have been granted on objects by whom and for how long. The latter are used to derive the privileges which are implied by the temporal authorizations. The temporal authorizations and derivation rules may be expressed as holding for a particular interval of time.

Rather than providing a specific language for users to express their security information, in our treatment a security administrator chooses a security model to implement and formulates this model in the *simplified event calculus (SEC)* [10]. Moreover, the notion of an event, rather than an interval of time, is central in our approach.

The expressive power of the SEC provides a security officer with considerable scope for representing various security models which support the specification of time-constrained privileges. This scope is achieved by treating security actions as constants in rules which are written in *clausal form logic* [5] to describe the consequences of a security event being performed on a database. The use of clausal form logic, on which the SEC and our approach is based, makes it possible for users to express their security requirements in a clear and concise way. Moreover, this formalization may be regarded as an executable specification which may be verified, informally with prospective end-users and formally by proving properties of the specification, prior to implementation.

The rest of this paper is organized in the following way. In Section 2, a brief overview of the simplified event calculus is presented. In Section 3, the details of the event-based specification of security information are given. In Section 4, our use of the SEC to represent a particular security model with temporal authorizations is outlined. In Section 5, theoretical and practical issues are addressed and, in Section 6, various extensions to our example security model are discussed. In Section 7, some conclusions are drawn and suggestions are made for further work.

2. THE SIMPLIFIED EVENT CALCULUS

The original *event calculus* [7] consists of a number of general axioms, expressed in terms of Horn clauses augmented with *negation as failure* [3], which

are intended to be used for knowledge representation in situations where reasoning about time and events is required.

The basic idea is that the general axioms of the event calculus be combined with a set of domain-specific axioms, which define the initiation and termination of relationships, and a description of events which have occurred in the world a database describes. From this set of axioms and a description of events the consequences of those events may be derived together with the periods of time for which they hold.

The simplified event calculus, as its name suggests, is a restricted form of the event calculus which nonetheless has proved to be useful for treating a number of practical problems (see, for example, [6]).

The SEC only permits forward persistence and is based on the simplifying assumption that complete information exists about events, including the times at which they occur. Under this assumption, a single persistence axiom is all that is required to specify that an initiated relationship, R, continues to persist until an event occurs to terminate it. The core axiom of the SEC, *C0* (say), which captures this notion may be expressed thus (where $\neg$ denotes a chosen form of negation and $<$ is an "earlier than" relation):

$$holdsat(R,T) \leftarrow happens(E1,T1), initiates(E1,R), T1 < T,$$
$$\neg \exists E2,T2[happens(E2,T2), terminates(E2,R), T1 < T2, T < T2]$$

More fully, *C0* expresses that a relationship R holds at a time point T if an event *E1* happened which initiated R (i.e., made R true) at an earlier point in time *T1* and no intervening event *E2* has terminated R (i.e., made R false) subsequent to its initiation.

Notice that for *C0*, and henceforth, terms which appear in the upper case denote variables.

An example of the type of domain-specific rules which are used to define the *initiates* and *terminates* predicates appearing in *C0* is given in Section 4. Before presenting these rules, however, we consider the representation of a history of security events performed on a database.

3. EVENT DESCRIPTIONS AND AUTHORIZATION HISTORIES

To record the fact that a security event has taken place, the notion of a *security event description* is used. A security event description consists of a set of ground assertions which, as the name implies, describes the occurrence of an event which has taken place and which is relevant to the security of a database.

To see what is involved in representing these events, suppose that Bob creates a database object o1 on 1/1/99 and takes read and write access rights on o1. This event, *e0* (say), may be represented by the following event description:

{happens(e0,1/1/99),act(e0,create),creator(e0,bob),object(e0,o1), mode(e0,read),mode(e0,write)}

Any constant which appears in a security event description is one of: an event, a time, a subject, a group identifier, an object or an action which is permissible in the security model.

In the event *e0*, *create* is the action performed. Now, while a subject clearly needs to be able to do more than just create objects, the precise set of actions they may perform depends on the choice of security model to be implemented.

The actions upon which the example security model given in Section 4 is based are those which are included in the set, **A**={*create,grant,grantgroup, revoke,revokegroup,destroy*}; the access modes considered are those in the set, **M**={*read,write*}. That is, subjects may create and destroy objects and may grant and revoke some or all of the read and write privileges individuals or groups of individuals may have on database objects.

Notice that security event descriptions for any of the actions in **A** will be similar in form to the event description for *e0*. Whilst the predicate names and constants will differ, a security event description will always be represented using a set of ground binary relations. A simple front-end may be used to assist users in describing security events and a validation procedure is used to ensure that security event descriptions are syntactically and semantically meaningful.

To see more fully what is involved in describing a history of events affecting database security, suppose that Bob has created the database object o1 and consider the following narrative:

> On 2/1/99 Bob grants John write access on o1 until 5/1/99 and read access until 20/6/99. On 15/4/99, Bob grants Sue read and write privileges on o1 indefinitely into the future. On 25/4/99, Bob grants every member of the Sales department read access to o1 until 1/6/99. Sue's write privilege on o1 is removed, by Bob, on 20/5/99.

The following set of event descriptions (in which *ei*, where *i* is a natural number, denotes an event) describes this scenario:

{happens(e1,2/1/99),act(e1,grant),grantee(e1,john),object(e1,o1), mode(e1,write),stop(e1,5/1/99)}

{happens(e2,2/1/99),act(e2,grant),grantee(e2,john),object(e2,o1), mode(e2,read),stop(e2,20/6/99)}

{happens(e3,15/4/99),act(e3,grant),grantee(e3,sue),object(e3,o1), mode(e3,read),mode(e3,write)}

{happens(e4,25/4/99),act(e4,grantgroup),grantee(e4,sales),object(e4,o1), mode(e4,read),stop(e4,1/6/99)}

{happens(e5,20/5/99),act(e5,revoke),revokee(e5,sue),object(e5,o1), mode(e5,write)}

A set of security event descriptions is recorded as part of the information which is maintained in the database to be used for access control. In the example security model which we consider, only the creator of a database object, O, can grant and remove access rights and decide the period of time for which rights are allowed on the object. As such, an event description is only permissible for O if it is asserted by the creator of O. The validation procedure for security event descriptions ensures that the specifier of such a description for O is always O's creator.

Henceforth, we will refer to a set of security event descriptions, like the one above, as an *authorization history*.

In our example security model, it is the *grant* and *grantgroup* operations which have a temporal component. That is, rights may be specified as being granted for a limited period of time. Not surprisingly, the removal of a privilege, P, from a subject, S, on an object, O, at a time, T, prevents S gaining P access on O unless and until S is granted the privilege P at some time point which is subsequent to T.

An access right is assumed to hold from the point in time at which the event which grants the access actually happens. The difference between the time in a *happens(e,t)* fact and a *stop(e,t1)* fact (where $t < t1$) is the interval of time for which the right, granted in the event e, is permitted to hold for the subject(s) and object specified in the security event description for e. As our example authorization history demonstrates, if a *stop* time is not associated with a *grant* operation then the right which is granted is assumed to be permitted indefinitely into the future. Conversely, if a *stop* time is associated with a *grant* then that specifies the maximum future time point up until which a privilege is to hold. The grantor of a privilege has the option of terminating this privilege earlier than the maximal time by using a *revoke* operation.

4. A SEC-BASED SECURITY MODEL

As we have said, there are two sets of rules which are important in our approach. One set of rules is needed to represent the core axiom, *C0*, of the SEC; the other set is needed to define the initiation and termination of the properties

of interest in the specific security model, with time-constrained access rights, to be implemented.

To simulate the *C0* axiom the following three rules may be used:

(C1) holds(access(S,P,O))←happens(E,T),initiates(E,access(S,P,O)),
not ended(E,access(S,P,O),T)

(C2) ended(E,access(S,P,O),T)←happens(E1,T1),T < T1,
terminates(E1,access(S,P,O))

(C3) ended(E,access(S,P,O),T)←stop(E,T1),current-time(T2),T1 < T2

The reading of this set of rules is similar to that for *C0*: if an event, E, happens at time, T, which causes a subject, S, to be given the access right, P, on a database object, O, and the period of time for which this right was granted has not expired and no subsequent event is known to have occurred to have ended the right P which S has to access O at time T then S holds the privilege P on O. The variable, T2, in the *current-time* atom is instantiated with the time, taken from the "system clock", at which the *current-time* atom in *C3* is selected in the process of deciding whether a privilege has expired.

Notice that the *C2* rule deals with the termination of a privilege as a consequence of the occurrence of an event whereas *C3* deals with the termination of a privilege as a consequence of the ending of a period of time for which the privilege was granted. Note also that the $T < T1$ condition in the *C2* rule assumes that properties hold only after their initiating event happens. The $T1 < T2$ condition in *C3* is based on the assumption that event descriptions do not have identical times for the *happens* and *stop* predicates (any attempt to include this type of meaningless security event description in an authorization history would be rejected in the process of validating event descriptions).

To see what is involved in writing the domain-specific *initiates* and *terminates* rules for the example security model based on **A**, we consider first the *initiates* rule, *P1*, required to treat object creation. For that, suppose that an event, E, happens in which a subject, S, creates an object, O, and takes the right, P, on O. This event has the consequence of initiating (i.e., making true) the fact that S has the right P on O viz.:

(P1) initiates(E,access(S,P,O))← act(E,create),creator(E,S),
object(E,O),mode(E,P)

Similarly, if the creator of object O grants subject S the right P on O then the initiation of a period of time during which S may exercise the privilege P

on O may be represented by the following rule:

(P2) initiates(E,access(S,P,O))←act(E,grant),grantee(E,S),
object(E,O),mode(E,P)

To deal with the granting of rights to groups of individuals, the body of the rule *P2* needs to be slightly modified to include a condition for testing for group membership. Specifically, we need:

(P3) initiates(E,access(S,P,O))←act(E,grantgroup),grantee(E,W),
memberof(S,W),object(E,O),mode(E,P)

Notice that a set of *memberof* facts (e.g., *memberof(bill,sales)*) may be recorded in the database to permit group authorizations to be specified and subjects may be members of several different groups simultaneously (without conflicting privileges arising).

When combined with an authorization history and the core axioms of the SEC, the *initiates* rules are used to infer the start of an interval of time for which access rights are held by subjects on database objects.

As previously discussed, the *C3* axiom is applicable when the interval of time for which an access right holds is ended by a *stop* time in an event description. For the cases where an action is performed which ends a subject's right to access an object, a set of *terminates* rules is required. These *terminates* rules are used in conjunction with the core axiom *C2* and enable the owner of an object to remove privileges from subjects.

The following pair of *terminates* rules may be used for dealing with revocations in our example model:

(P4) terminates(E,access(S,P,O))←act(E,revoke),revokee(E,S),
object(E,O),mode(E,P)

(P5) terminates(E,access(S,P,O))←act(E,revokegroup),revokee(E,W),
memberof(S,W),object(E,O),mode(E,P)

Finally, to deal with the termination of rights when an object is destroyed by its creator we need:

(P6) terminates(E,access(,*,O))←act(E,destroy),object(E,O)*

In *P6* the variable '*' denotes any subject and any privilege.

As we have said, the set of access rules, *Pi* ($i \in \{1,..,6\}$), represents the particular security model which we have chosen to use to demonstrate our approach. Different security models may be represented by choosing different security actions to support and then writing the model-specific *initiates* and *terminates* axioms to represent the consequences of performing these actions. Similarly, any number of auxiliary rules may be included in a security model to simplify the specification of an authorization history (e.g., a "write permission implies read permission" rule may be used).

In contrast to [2], the rules defining the security theory above do not have a time interval associated with them to represent the periods during which they apply to a particular subject exercising a right to access an object. In our approach, the intervals of time for which the rules are applicable to a *(subject,privilege,object)* triple are implicit from the times included in an authorization history. This simplifies the specification of time-constrained privileges but without compromising the scope a security officer has for representing security theories.

5. THE ACCESS CONTROL PROCEDURE

Given an authorization history, to decide whether a subject, S, should be permitted to exercise the privilege, P, on object, O, the approach we adopt involves attempting to prove that the access request is a theorem of the authorization history together with the axiomatization which defines a chosen security model with time-constrained privileges. The existence of a proof permits S to perform P on O at the time at which the access is requested; otherwise the access request is denied.

Henceforth, we will denote, by Σ, the set of core axioms, *Ci* ($i \in \{1,2,3\}$), together with the *Pj* rules, ($j \in \{1,..,6\}$), which define our example security model. Similarly, we will denote the authorization history given in Section 3 by **H**.

Since Σ can be described as a set of function-free *stratified clauses* [1] it follows that access rights may be determined by using *safe SLDNF-resolution* [8]. That is, to decide whether a subject, S, can exercise privilege, P, on a object, O, an *SLDNF-derivation* [8] for the goal clause $\leftarrow$*holds(access(S,P,O))* may be attempted on the security theory $\Sigma \cup \mathbf{H}$. In this context, if $\Sigma \cup \mathbf{H} \cup \{\leftarrow holds(access(S,P,O))\}$ has an *SLDNF-refutation* [8] then subject S has the privilege P on O. Conversely, if $\Sigma \cup \mathbf{H} \cup \{\leftarrow holds(access(S,P,O))\}$ has a *finitely-failed SLDNF-tree* [8] then S does not have the privilege P on O.

Notice that it is impossible for inconsistent access rights to arise in Σ for any authorization history. For inconsistency, *holds(access(S,P,O))* would need to be simultaneously true and false (for some substitution for variables). How-

ever, since there is only one definition of *holds* in Σ and SLDNF-resolution is consistent, it should be clear that inconsistent access rights do not arise.

Before giving an example of the use of SLDNF-resolution to decide questions of access, we need to mention an important practical issue. For efficiency reasons, ground instances of the *initiates*, *terminates* and *ended* predicates may be *memoized* [9], by *dynamic assertion* [13], when a *holds(access(S,P,O))* request is first evaluated. This approach, increases the size of the security theory but has the advantage of permitting proofs of authorizations to be efficiently performed since ground assertions may be used, almost entirely, in the process. The existence of previously generated *initiates*, *terminates* and *ended* facts is assumed in the example which follows.

Example
The following (abbreviated) SLDNF-derivation for $\Sigma \cup \mathbf{H} \cup$ $\leftarrow$*holds(access(john,write,o1))*, performed on 25/1/99 (say), reveals that John is not permitted to write o1:

$\leftarrow$*holds(access(john,write,o1))*
|
$\leftarrow$*happens(E,T),initiates(E,access(john,write,o1)),*
not ended(E,access(john,write,o1),T)
|
$\leftarrow$*initiates(e1,access(john,write,o1)),*
not ended(e1,access(john,write,o1),2/1/99)
|
$\leftarrow$*not ended(e1,access(john,write,o1),2/1/99)*
fail

For $\Sigma \cup \mathbf{H} \cup \{\leftarrow$*holds(access(John,read,o1))*$\}$, on 25/1/99, the SLDNF-derivation is very similar to the one above. However, whereas *ended* succeeds by *C3* in the *holds(access(john,write,o1))* case, in the test for John's possession of a read right on o1 the *ended* subgoal finitely fails since John's read privilege over o1 has been neither terminated by an event nor stopped as a consequence of the inclusion of a *stop* atom in an event description. Thus, an SLDNF-refutation exists for $\Sigma \cup \mathbf{H} \cup \{\leftarrow$*holds(access(John,read,o1))*$\}$ and so, on 25/1/99, John is empowered to read o1. $\diamond$

An issue which is sometimes raised in the context of temporal authorization models relates to the ending of a period of time for which a privilege has been granted *during* the test to determine whether a subject's access to a database object is permitted or not. In our view this is a matter of practical concern

rather than a security model issue and is not specific to temporal authorization models since, unless steps are taken to prevent it, the revocation of a privilege can take place whilst a question of access is being decided in a non-temporal environment.

This problem does not, however, arise in our approach. To see why notice that, since we assume that a *leftmost selection rule* [8] is used in the SLDNF-derivations performed on Σ, in the *C3* rule, the current time is used immediately prior to the test for the expiration of a time-constrained access right. Hence, even if a time-constrained privilege does expire after the access control procedure has been invoked, it is not until the $T1 < T2$ condition is evaluated in *C3* that the expiration of this privilege is checked and at this point in the computation the current time will have been extracted from the system clock immediately prior to the test for $T1 < T2$.

In implementations of our approach periodic garbage removal may be performed on the memoized assertions and on an authorization history to remove redundant security event descriptions. For instance, if the current time is subsequent to the *stop* time included in a security event description then the event description may be physically deleted from the authorization history together with any facts which were initiated by the event. For instance, the event description for *e1* could be removed from **H** after 5/1/99 since John does not have write access on o1 after this point in time. If this deletion were performed then, in any subsequent test of John's permission to write o1, the access control procedure may establish more quickly that John does not have this privilege since the early failure of the *initiates(E,access(john,write,o1))* condition will cause *holds(access(john,write,o1))* to fail early too. Similarly, when an object is destroyed the event description which created that object may be deleted together with any dynamically asserted *initiates*, *ended* or *terminates* facts relating to the object. Several other optimizations are possible including changing the order of the conditions appearing in the rules of Σ to exploit the subgoal selection strategy used in an SLDNF-derivation. For example, by resolving an *initiates(E,access(S,P,O))* subgoal with an appropriate memoized ground instance of the same form, it is possible to delay selecting the *happens(E,T)* condition in *C1* until substitutions for *E* and *T* have been found.

Another practical issue which needs to be considered relates to the *soundness* and *correctness* results applicable to the proof methods which may be used to decide questions of access. For Σ (together with any authorization history), Clark's *2-valued completion* [3] is a candidate declarative semantics for our choice of access control mechanism. More specifically, since Σ is always an *allowed* [11] and *hierarchical* [3] normal clause theory and *holds(access(S,P,O))* is an *allowed query*, it follows, from [12], that, for Σ, (safe) SLDNF-resolution is sound and complete with respect to Clark's se-

mantics. In this context, soundness implies that SLDNF-resolution correctly identifies which subjects have which access rights on which database objects; completeness means that all the access rights which are implied by an authorization history, together with Σ, can be derived by using SLDNF-resolution.

6. EXTENDING Σ

A significant attraction of our approach is that it permits any number of security theories with temporal authorizations to be expressed in a simple and completely uniform way. For a different security theory to Σ, a different set of actions to **A** is chosen and the rules which define the consequences of performing these actions are formulated.

In this section, we consider some possible extensions to Σ to demonstrate the flexibility afforded by our approach. More specifically, we consider how negative authorizations may be expressed, how proactive authorizations can be supported, how shared privileges may be specified and how rules for default authorizations may be represented.

To permit negative authorizations to be specified as holding for a restricted interval of time, a *not denied(access(S,P,O))* condition may be added to *C1* while the following rule is added to the core axioms:

denied(access(S,P,O))$\leftarrow$*happens(E,T1),act(E,denying),*
subject(E,S),object(E,O),mode(E,P),
stop(E,T2),currenttime(T3),T3 < T2, T1 < T3

It follows from this rule that, in addition to access being prohibited as a result of the occurrence of an event which terminates a privilege or the expiration of a time interval for which a right was granted, a subject S may also be prevented from exercising the privilege P on O if S has been denied access to O at time T1 and the period of time for which the denial applies has not expired.

Notice that a *stop* time must be specified in a security event description which relates to the act of denying and that a *denials take precedence* conflict resolution strategy is effectively enforced by our formulation of the core axioms of the SEC.

Until now, we have assumed that an access right holds from the point in time at which the event which grants it happens. However, the proactive initiation of authorizations can also be supported in our approach. This makes it possible to specify, on 1/6/99 (say), that, for example, Steve will be authorized to read o1 from 1/10/99. To achieve this, an event description is included in an authorization history with an instance of a *happens(E,T1)* assertion such that if *T* is

the current time then $T < T1$. The core axiom *C1* also needs to be extended to include a *current-time(T)* condition and a $T1 < T$ condition.

To specify that privileges may be shared for a specified interval of time, the following rule may be used:

initiates(E,access(S2,P,O))←act(E,sharing),subject(E,S1),sharer(E,S2),
S1≠S2,object(E,O),mode(E,P),holds(access(S1,P,O))

This rule expresses that the subject S2 may exercise the privilege P on object O for the period of time during which subject S1 (S1≠S2) has the right, P, on O.

Authorizations may also be derived as a consequence of the absence of others (as in [2]). For example, in [2] a *whenevernot* operator is defined and used to express the mutual exclusivity constraint that a subject S1 is permitted to exercise privilege P on O at all time points at which subject S2 does not have this right. To represent this in a SEC-based security theory, the following axiom may be included:

initiates(E,access(S1,P,O))←act(E,defaultpermit),permitted(E,S1),
excluded(E,S2),object(E,O),S1≠S2,
mode(E,P),not holds(access(S2,P,O))

The next rule enables the specifier of a security model to express that at all points in time at which subject S1 holds a privilege P on object O, subject S2 is prevented from exercising the P privilege over O:

terminates(E,access(S2,P,O))←act(E,exclusion),permitted(E,S1),
excluded(E,S2),S1≠S2,object(E,O),
mode(E,P),holds(access(S1,P,O))

The examples above illustrate a number of direct extensions to Σ. However, a number of less direct extensions are also possible. For instance, *initiates* rules may be defined in terms of *initiates* or *terminates* conditions and *terminates* rules can be defined in terms of *initiates* or *terminates* conditions, a *with grant* option can be added to enable authorizations to be granted by subjects other than owners of objects and temporal authorizations may be expressed as being conditional upon the occurrence of events. The definitions of *initiates* and *terminates* relationships can also be extended to include a time argument. This makes it possible to specify temporal authorizations which are based on an authorization history.

The key point which arises from this discussion is that our approach provides a security administrator with a good deal of flexibility when it comes to specifying security theories with time-constrained access rights. In principle, an implementor can choose to support any set of security actions and any set of privileges; all that is required is that appropriate *initiates* and *terminates* rules be written to define the consequences of performing these actions on a database. In contrast, pre-specifying a set of security operators precludes the formulation of security models which do not include these operators and, hence, may limit the scope a security administrator has for protecting a database.

The only constraints on a security administrator who uses our approach are those which apply to the methods used to decide whether a subject S has privilege P over O; whether, for instance, the methods available for deciding questions of access are sound, complete and guaranteed to terminate.

7. SUMMARY AND CONCLUSIONS

By using the simplified event calculus it is possible to represent a range of security models for discretionary access control where privileges on database objects may be granted to subjects for limited periods of time. These security models may be formulated in subsets of clausal form logic for which efficient, sound and complete theorem provers are known to exist.

The "high-level" nature of the language on which the event calculus is based and the fact that users provide security information by writing ground atomic assertions makes the suggested approach especially easy to use. Moreover, a simple front-end may be developed to further ease the burden of expressing security information.

Security models which are directly based on the theoretical framework outlined above have been implemented (in PROLOG) and have been successfully employed to manage access to relational databases. However, since no assumptions have been made about the underlying data model, the approach may also be used with other types of database.

In future work we will show how the SEC may be used to formulate a RBAC security model with time-constrained assignments of permissions and users to roles and how our current work on security in deductive databases can be extended to include temporal authorizations.

References

[1] Apt, K., Blair, H., and Walker, A. (1988). Towards a theory of declarative knowledge. *Foundations of Deductive Databases and Logic Programming*

(ed. J. Minker), Morgan-Kaufmann.

[2] Bertino, E., Bettini, C., Ferrari, E., and Samarati, P. (1996). A temporal access control mechanism for database systems, *TKDE*, **8(1)**.

[3] Clark, K. (1978). Negation as failure. *Logic and Databases* (eds. H.Gallaire and J. Minker), Plenum.

[4] Griffiths, P., and Wade, B. (1976). An authorization mechanism for relational database systems. *ACM TODS*, **1(3)**.

[5] Kowalski, R. (1979). *Logic for Problem Solving*, Elsevier.

[6] Kowalski, R. (1992). Database updates in the event calculus. *Journal of Logic Programming*, **12**.

[7] Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, **4(1)**.

[8] LLoyd, J. (1987). *Foundations of Logic Programming*, Springer-Verlag.

[9] Michie, D. (1968). Memo functions and machine learning. *Nature*, **218**.

[10] Sadri, F. and Kowalski, R. (1995). Variants of the event calculus, *Proceedings of ICLP*, MIT Press.

[11] Shepherdson, J. (1984). Negation as failure. *Journal of Logic Programming*, **1**.

[12] Shepherdson, J. (1997). Negation as failure, completion and stratification. *Handbook of Logic in AI and Logic Programming, Volume 5, Logic Programming* (eds. D. Gabbay, *et al.*, Oxford.

[13] Stirling, L. and Shapiro, E. (1994). *The Art of PROLOG*, MIT Press.

[14] Thomas, R. and Sandhu, R. (1993). Discretionary access control in objected-oriented databases: Issues and research directions. *Proceedings of the Sixteenth National Computer Security Conference.*