Event Detection in Multilevel Secure Active Databases

Indrakshi Ray and Wei Huang

Department of Computer Science
Colorado State University
Fort Collins CO 80523-1873
{iray,hw3699}@cs.colostate.edu

Abstract. The event-condition-action paradigm (also known as *triggers* or *rules*) is a powerful technology. It gives a database "active" capabilities – the ability to react automatically to changes in the database or in the environment. One potential use of this technology is in the area of multilevel secure (MLS) data processing, such as, military, where the subjects and objects are classified into different security levels and mandatory access control rules govern who has access to what. Although a lot of research appears in MLS databases, not much work has been done in the area of MLS active databases. In this paper, we look at one very important aspect of an MLS active database – event detection.
An MLS rule, like any other object in an MLS database, is associated with a security level. Events in an MLS database are also associated with security levels. Since an MLS rule can be triggered by an event that is at a different security level than the rule, we cannot use the event detection techniques designed for non-MLS active databases. Using such techniques cause illegal information flow. Our goal is to propose new algorithms that prevent such illegal information flow. We first present an approach to detect primitive events – events that cannot be decomposed. Different types of primitive events can be combined using the event composition operators to form composite events. We also describe how to detect composite events using event graphs in an MLS database.

1 Introduction

Traditional database management systems are *passive*: the database systems execute commands when requested by the user or application program. However, there are many applications where this passive behavior is inadequate. Consider for example, a financial application: whenever the price of stock for a company falls below a given threshold, the user must sell his corresponding stocks. One solution is to add monitoring mechanisms in the application programs modifying the stock prices that will alert the user to such changes. Incorporating monitoring mechanisms in all the relevant application programs is non trivial. The alternate option is to poll periodically and check the stock prices. Polling too frequently incurs a performance penalty; polling too infrequently may result in not getting the desirable functionalities. A better solution is to use active databases.

Active databases move the reactive behavior from the application into the database. This reactive capability is provided by *triggers* also known as *event-condition-action rules* or simply *rules*. In other words, triggers give active databases the capability to monitor and react to specific circumstances that are relevant to an application. An active database system must provide trigger processing capabilities in addition to providing all the functionalities of a passive database system.

One potential use of this technology is in the area of secure data processing, such as, the military which uses an underlying multilevel secure (MLS) database. A multilevel secure database system is characterized by having a partially ordered set of security levels (the ordering relation is referred to as the dominance relation); all the database objects and the operations (transactions) on the database objects have security levels associated with them. Mandatory access control policies determine which transactions can access which objects. The idea is that information can flow from the dominated level to the dominating level but not in the other direction.

Providing reactive capabilities in an MLS database system requires event detection. We start by describing the different kinds of events in an MLS database and how to assign security levels to events. The rules in an MLS database are also associated with security levels. Since a rule can be triggered by events whose level is dominated by the level of the rule, the algorithms used for detecting events in an non-MLS database cannot be used. Using such algorithms causes illegal information flow. Our goal is to provide algorithms that do not cause such security breaches.

The first step in event detection is the detection of *primitive* events. Primitive events are atomic in nature – they cannot be decomposed into constituent events. Primitive event detection is complicated by the fact that a primitive event at the dominated level cannot automatically notify any rule at the dominating level. This requires us to have event detectors at each security level. Each level is responsible for detecting the events that are associated with the rules at that level.

The events in a rule may not be primitive events. Two or more primitive events can be combined using event composition operators to form a *composite* event. As expected, detection of composite events is non-trivial. We adapt the approach proposed by Chakravarty et al. [5] for detecting composite events. For each rule, we construct an *event tree* for detecting composite events associated with that rule. The level of this event tree is the same as the security level of the rule. Each node n_i corresponds to an event of this rule. Each directed edge (n_i, n_j) signifies that the event corresponding to node n_i is a constituent of the composite event corresponding to n_j. An event may be associated with more than one rule. Thus, an event can belong to multiple event trees. In such situations, as suggested by Chakravarty et al.[5], we can merge the event trees to form an event graph. There are two ways in which our composite event detection differs from Chakravarty's work. First, only event trees belonging to the same security level can be merged. Second, if a composite event is used by rules at different security levels, then multiple event trees need to be used – one for each security level. Finally, we provide algorithms for detecting composite events.

The rest of the paper is organized as follows. Section 2 briefly describes the underlying MLS model on which our work is based. Section 3 describes the structure of rules in an MLS active database system, how security levels are assigned to rules, and the relationship between the security levels of the different components of rules. Section 4 describes the possible architectures for an MLS active database. Section 5 focuses on event detection in an MLS active database. Section 6 describes the related work in this area. Section 7 concludes the paper with pointers to future directions.

2 Our Model of an MLS Database

A database system is composed of database objects. At any given time, the *database state* is determined by the values of the objects in the database. A change in the value of a database object changes the state. Users are responsible for submitting transactions and application programs that are to be executed on the database. A *transaction* is an operation on a database state. An *application program* may or may not perform an operation on the database state. Execution of a transaction or application program may cause the database to change state.

An MLS database is associated with a security structure that is a partial order, ($\mathbf{L}$, $<$). $\mathbf{L}$ is a set of security levels, and $<$ is the dominance relation between levels. If $L_1 < L_2$, then L_2 is said to strictly dominate L_1 and L_1 is said to be strictly dominated by L_2. If $L_1 = L_2$, then the two levels are said to be equal. $L_1 < L_2$ or $L_1 = L_2$ is denoted by $L_1 \leq L_2$. If $L_1 \leq L_2$,then L_2 is said to dominate L_1 and L_1 is said to be dominated by L_2. Two levels L_1 and L_2 are said to be incomparable if neither $L_1 \leq L_2$ nor $L_2 \leq L_1$. We assume the existence of a level U, that corresponds to the level unclassified or public knowledge. The level U is the greatest lower bound of all the levels in $\mathbf{L}$. Each database object $x \in \mathbf{D}$ is associated with exactly one security level which we denote as $L(x)$ where $L(x) \in \mathbf{L}$. (The function L maps entities to security levels.) We assume that the security level of an object remains fixed for the entire lifetime of the object. The users of the system are cleared to the different security levels. We denote the security clearance of user U_i by $L(U_i)$. Consider a military setting consisting of four security levels: Top Secret (TS), Secret (S), Confidential (C) and Unclassified (U). The user Jane Doe has the security clearance of Top Secret. That is, $L(JaneDoe) = TS$. Each user has one or more principals associated with him. The number of principals associated with the user depends on his security clearance; it equals the number of levels dominated by the user's security clearance. In our example Jane Doe has four principals: *JaneDoe.TS*, *JaneDoe.S*, *JaneDoe.C* and *JaneDoe.U*. At each session, the user logs in as one of the principals. All processes that the user initiates in that session inherit security level of the corresponding principal. Each transaction T_i is associated with exactly one security level. The level of the transaction remains fixed for the entire duration of the transaction. The security level of the transaction is the level of the principal who has submitted the transaction. For example, if Jane Doe logs in as *JaneDoe.S*, all transactions initiated by Jane Doe will have the level Secret (S).

We require a transaction T_i to obey the *simple security property* and the *restricted $\star$-property*: may read a database object x only if $L(x) \leq C$ and (2) A transaction T_i with $L(T_i) = C$ may write a database object x only if $L(x) = C$. We give the formal definition of an MLS transaction and an MLS application below.

Definition 1. [MLS Transaction] *An MLS transaction T_i is a set of read and write operations on database objects which are preceded by the command* begin *and followed by the command* abort *or* commit. *The transaction T_i is associated with security level $L(T_i)$*

where $L(T_i) \in \mathbf{L}$; it accesses database objects in accordance with the simple security and the restricted $\star$-property.

Definition 2. [MLS Application Program] *An MLS application program A_i is a set of operations submitted by the user – the operations may or may not access the database objects. An application program A_i that accesses the database objects is associated with security level $L(A_i)$, where $L(A_i) \in \mathbf{L}$; it accesses database objects according to the rules specified by the simple security and the restricted $\star$-property.*

3 Rules in an MLS Active Database

In addition to transactions and application programs, an active database system also has *rules*. A rule is specified by three components: *event*, *condition* and *action*. An event causes a rule to be triggered. Active database systems have mechanisms that monitors the database to check whether an event has occurred. If an event associated with a rule occurs, the rule's condition is evaluated. If the rule's condition evaluates to true, then the rule's action is scheduled for execution. The details of rule execution are considerably more complex and have been omitted from this paper.

A rule is a database object on which we allow the following operations: (1) Create – this operation allows a new rule to be created. (2) Delete – this operation allows an existing rule to be deleted. (3) Update – this operation allows an existing rule to be modified. (4) Enable – this operation allows an existing rule to be enabled. Only enabled rules can be triggered. (5) Disable – this operation allows an existing rule to be disabled. A disabled rule cannot be triggered. (6) View – this operation allows an existing rule to be viewed. Like other database objects in an MLS database, rules are also associated with security levels. Each rule R_j is created by some principal, say P, and it inherits the security level of the principal that created it, that is, $L(R_j) = L(P)$. Creation, deletion, modification, disabling, enabling of the rule corresponds to writing of the rule object. Hence, by the restricted $\star$-property these operations can be performed only by transactions or applications whose security level is the same as that of the rule. Viewing the rule corresponds to a read operation of the rule object. Thus, the view operation can be performed by transactions or applications whose security levels dominate the level of the rule.

Definition 3. [MLS Rule] *An MLS rule R_j is defined as a triple $< e_j, c_j, a_j >$ where e_j is the event that causes the rule to be triggered, c_j is the condition that is checked when the rule is triggered and a_j is the action that is executed when the rule is triggered. The MLS rule R_j is associated with exactly one security level which we denote by $L(R_j)$ where $L(R_j) \in \mathbf{L}$. The operations allowed on rules obey the mandatory restrictions specified by the simple security and the restricted $\star$-property.*

3.1 Events in an MLS Active Database Systems

Event specifies what causes the rule to be triggered. Possible events that can be supported in an MLS active database system are as follows. (1) Data modification/retrieval event – the event is raised by an operation (insert, update, delete, access) on some database object. (2) Transaction event – the event is raised by some transaction command (e.g. begin, abort, commit etc.). (3) Application-defined event – the application program may signal the occurrence of an event. (4) Temporal events – events are raised at some point in time. Temporal events may be absolute (e.g., 25th December, 2002) or relative (e.g. 15 minutes after x occurs). (5) External events – the event is occurring outside the database (e.g. the sensor recording temperature goes above 100 degrees Celsius).

Events can further be classified into primitive and composite events. A *primitive event* cannot be divided into subparts. A *composite events* is raised by some combination of primitive events. For example, inserting a tuple in *Employee* relation is a primitive event. Two hours after a tuple has been inserted in *Employee* relation is a composite event. A composite event is constructed using two or more primitive events connected by an event operator. Any composite event e can be denoted as: $e = e_1\ op_1\ e_2\ op_2 \ldots\ e_n$ where $e_1, e_2, \ldots, e_n$ are the primitive events making up the composite event E, and $op_1, op_2, \ldots op_{n-1}$ are the event operators. Event operators can be logical event operators ($\vee$, $\wedge$, etc.), sequence operators (;), or temporal composition operators ($after$, *between*, etc.). In this paper, we consider only two most common event operators: $\vee$ – disjunction and ; – sequence. The composite event $e_1 \vee e_2$ is said to occur when either an instance of e_1 or e_2 occurs. The composite event $e_1; e_2$ is said to occur when an instance of e_2 occurs after an instance of e_1. We next describe how to assign security levels to events.

Security Level associated with Data Modification/Retrieval Event: The event e has the same security level as the operation O that caused it, that is, $L(e) = L(O)$. If this operation O is performed by some transaction T, then the level of O is the same as the level of T.

Security level associated with the Transaction Event: The event E has the same security level of the transaction T that caused it, that is, $L(e) = L(T)$.

Security Level associated with Application-Defined Event: The event e has the same security level as the level at which the application A that generated it is executing, that is, $L(e) = L(A)$.

Security Level associated with Temporal Event: An absolute temporal event e is observable by any body and so its security level is public, that is, $L(e) = U$. A relative temporal event is a composite event. The manner in which the level of composite event is calculated is given below.

Security Level associated with External Event: The level of the event e is the greatest lower bound of the security clearances of the users U_1, U_2, ..., U_n who can observe this external event e, that is, $L(e) = glb(L(U_1), L(U_2), \ldots, L(U_n))$ (where $L(U_i)$ denotes the security clearance of user U_i).
An event like the outside air temperature is 110 degrees Fahrenheit, is observable by all users and so its level is public. Whereas, an event like the sensor reading from a military satellite that can be observed only by Top Secret personnel, will have a security level of Top Secret.

Security Levels associated with Composite Event: Consider the composite event E given by, $e = e_1\ op_1\ e_2\ op_2 \ldots\ e_n$, where e is composed of primitive events e_1, e_2, ..., e_n. The security level of the composite event e is the least upper bound of the levels of the primitive events e_1, e_2, ..., e_n composing it, that is, $L(e) = lub(L(e_1), L(e_2), \ldots, L(e_n))$.

3.2 Conditions in an MLS Active Database System

In an active database, when a rule has been triggered *condition* specifies the additional conditions that must be checked before the action can be executed. If the condition part of the rule evaluates to true, then the action is executed. Possible conditions are as follows. (1) Database predicates – the condition might be a predicate on the database state (average salary of employees greater than 50000). (2) Database queries – the condition might be a query on the database state. If the query returns some results, the condition is said to be satisfied. If the query fails to return any result, the condition is not satisfied. (3) Application procedures – the condition may be a specified as a call to an application procedure (example, *max_exceeded()*) which may or may not access the database.

We now describe how to assign security levels to conditions. The level of a condition c, denoted by $L(c)$, is the least upper bound of all the data that is accessed by the condition. That is, $L(c) = lub(L(D_1), L(D_2), L(D_3), \ldots, L(D_n))$ where $D_1, D_2, \ldots, D_n$ are the data objects accessed by condition c.

3.3 Actions in an MLS Active Database System

When the rule is triggered and its condition evaluates to true, the action of the rule must be executed. Possible actions in an MLS active database include the following. (1) Data modification/retrieval operation – the action of the rule causes a data operation (insert, update, delete, access). (2) Transaction operation – the action of the rule causes a transaction operation (e.g. abort). (3) Application-defined operation – the action causes some procedure in an application to be executed. (4) External operation – the action causes some external operations (e.g. informing the user).

Some active database languages allow a rule to specify multiple actions. Usually these actions are ordered which allows them to be executed sequentially. This is how we assign security levels for the actions.

Security Level associated with Data Modification/Retrieval Action: The action has the same level as the operation it causes, that is, $L(a) = L(O)$.

Security Level associated with Transaction Operation: The action a has the same level as the transaction T, that is, $L(a) = L(T)$.

Security Level associated with Application-defined operation: The action a has the same level as the application process A, that is $L(a) = L(A)$.

Security Level associated with External operation: The level of action a is the greatest lower bound of the security clearances of the users U_1, U_2, ..., U_n who can observe this operation, that is, $L(a) = glb(L(U_1), L(U_2), \ldots, L(U_n))$ where $L(U_1)$, $L(U_2)$, ..., $L(U_n)$ are the security clearances associated with users U_1, U_2, ..., U_n respectively.

Security Level of Action composed of Multiple Constituents: Consider a rule $R = < e, c, a >$ where the action a is composed of multiple actions, $a_1, a_2, \ldots, a_k$. The level of all the actions must be the same. That is, $L(a_1) = L(a_2) = \ldots = L(a_k)$.

3.4 Relationship of Security Levels associated with a Rule

The following illustrates the relationship of the level of the rule R_j with the levels of the constituent event e_j, the condition c_j and the action a_j: (1) $L(e_j) \leq L(R_j)$, (2) $L(c_j) \leq L(R_j)$, and (3) $L(a_j) = L(R_j)$. Item (1) states that a rule may be triggered by an event whose level is dominated by the level of the rule. Item (2) states that a rule may require checking conditions at the dominated level before it can be fired. Item (3) states that a rule can take an action only at its own level.

In a secure environment it might be necessary for dominating levels to monitor suspicious events taking place at some dominated level and take some precautionary action; hence the need for $L(e_j) \leq L(R_j)$. Moreover, $L(e_j) \not> L(R_j)$ ensures that a dominating event does not trigger a dominated rule and create illegal information flow. The same reasoning applies for condition c_j; thus, the rule R_j might check conditions involving dominated level data (that is, $L(c_j) \leq L(R_j)$), but not data at the dominated levels ($L(c_j) \not> L(R_j)$). The level of the action is the same as the level of the rule, that is, $L(a_j) = L(R_j)$. Since $L(a_j) \not< L(R_j)$, a rule at the dominating level cannot result in an action at the dominated level and create illegal flow of information. Also, since $L(a_j) \not> L(R_j)$, a rule at the dominated level while executing its action cannot corrupt data at the dominating level.

4 Architecture of an MLS Active Database System

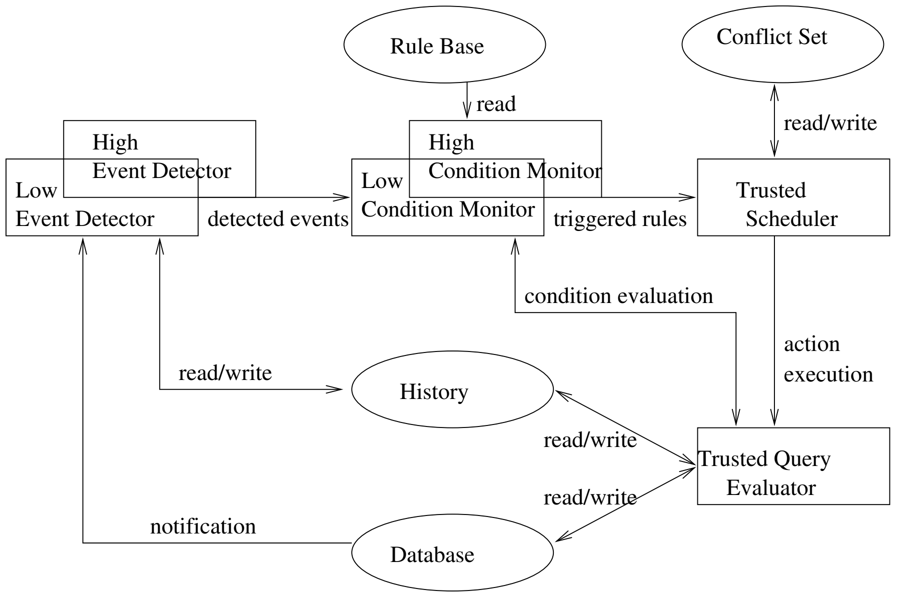

Fig. 1. A High-Level Architecture for Processing MLS Rules

In this section we describe an architecture at a very abstract level of an MLS active database system (refer to figure 1). This will give an idea of the extra components that must be supported for processing MLS rules. The principal components are depicted by rectangles and the data stores are depicted by ellipses. These are described next. **Event Detector at Level** l_i: This is responsible for detecting events at level l_i. Note that, event detector at level l_i can detect events at all levels that are dominated by l_i. Composite events are constructed using the knowledge about incoming primitive events and past events obtained from the history. **Condition monitor at Level** l_i: This is responsible for evaluating the conditions of rules that are triggered by the events detected by the event detector l_i. The condition monitor sends a request to the query evaluator to evaluate the condition. On the basis of the response from the query evaluator, the condition monitor decides which rules are triggered. These rules are then sent to the scheduler. **Trusted Scheduler**: This is responsible for scheduling which rule is to be executed. Since the scheduler accesses rules at different levels, it must be a trusted component. **Trusted Query Evaluator**: This is responsible for executing database actions and queries. We have shown the query evaluator to be a trusted component because it is responsible for executing database operations at all levels. In real world, the query evaluator will have several components, some of which are trusted and others which are not.

5 Event Detection

The process of event detection has been investigated by other researchers. In this section, we show how event detection can be adapted for MLS databases. One significant difference is that we have multiple event detectors that are responsible for detecting events that are of interest to the different security levels. The process in which the events get detected is different as well. Before describing these, we need to say a few words about event detection in non-MLS systems.

Chakravarthy et al. [5] describe event detection as the process of recognizing the occurrence of the event, collecting and recording its parameters. Examples of event parameters are the event type and time of occurrence. Event detection involves detecting primitive as well as composite events. Traditionally system clock and interrupts are used to detect primitive events. Detecting primitive events in non-MLS active database is achieved by using lower-level programmable interrupts or by embedding code in the system component that is responsible for reading and writing the data on the disk.

In an MLS environment, detecting primitive database events is non-trivial because an event at the dominated level may trigger a dominating rule. The dominated level in such a case should not be aware of the rule at the dominating level. Thus no actions can be taken at the dominated level when an event occurs. For instance, the dominated level cannot generate an interrupt to alert the dominating level about the occurrence of the event. Neither can code be embedded in the component that is responsible for reading and writing data on the disk.

In an MLS active database, if the level of the event is dominated by the level of the rule, the dominating level must monitor the dominated level to check for the occurrence of the event. For each type of primitive database event, we have a table which we call the *incident table* that records the occurrence of these events. The security level of the table is the same as the level of the event. The problem is that this solution is insecure – the dominated level can infer which events are of interest to the dominating level. To overcome this problem, we can have incident tables for all events and for some non-events. The non-events appear to be like events but they are not part of any rule. In short, we introduce some noise in the inference channel. The dominated level no longer knows for certain whether an incident table belongs to an event or not.

Since an event may be associated with rules at one or more dominating levels, the issue is how long should we maintain information about event occurrence in the incident tables. For security reasons, a dominating level cannot inform the dominated level when it has completed event detection. We propose that the incident occurrences be recorded for a fixed duration of time in the incident tables. The duration is fixed for all incidents. Within this time frame, the event detector at the level of the rule associated with the event must read the incident table in order to detect the event.

Sometimes a primitive event may not directly trigger any rules, but it may be a component of a composite event. The composite event may not occur until a later time. In such cases, the event detector at the level of the rule associated with the event must read the incident table and copy this information into a table at its own level. The table onto which this information is copied is referred to as an *event table*. Since an event can be associated with rules at different security levels, we can have multiple event tables at different security levels for the same event.

Event tables for primitive events are needed when they are part of a composite event. Sometimes a composite event is made up of other composite events. For each event that is part of a composite event, we have an event table that records its parameters. Another data structure that is needed for detecting composite events is an *event tree*. An example of an event tree for the composite event $(E_1;E_2) \vee E_3$ is given in Figure 2.

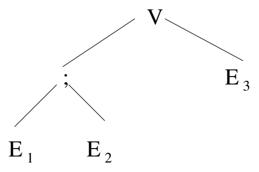

Fig. 2. An Event Tree for a Composite Event

Definition 4. [Event Tree] *An* event tree $\mathcal{ET}_e = (N,E)$ *corresponding to a composite event e is a directed tree where each node e_i represents either event e or one of its constituent. The root node corresponds to event e, and the leaf nodes correspond to the primitive events that make up node e. The edge (e_i,e_j) signifies that node e_i is a constituent of the composite event e_j. e_i in this case is referred to as the child node and e_j as the parent.*

An event can trigger multiple rules. Thus, different event trees can have nodes corresponding to the same event. In such cases, to save storage space, the event trees can be merged to form an event graph. One points need to be mentioned. Only event trees at the same security level can be merged. An event graph may contain event belonging to different rules. The nodes corresponding to events which fire one or more rules are labeled with the rule-ids of the corresponding rule.

Definition 5. [Event Graph] *An* event graph $\mathcal{EG} = (N, E)$ *is a directed graph where node e_i represents an event e_i and edge (e_i, e_j) signifies that the event corresponding to node e_i is a constituent of the composite event corresponding to node e_j. Each node e_i is associated with a label $label_{e_i}$. $label_{e_i}$ is a set of rule-ids (possibly empty) that indicate the rules that will fire when the event corresponding to node e_i happens.*

We now give the algorithms for event detection. The first one focuses on primitive event detection at security level L. The input to this algorithm is the event graph $\mathcal{EG}$ for level L, the set of incident and event tables corresponding to primitive events of $\mathcal{EG}$, and the set of incident tables for level L. The output is the updated event tables and updated incident tables at its own level. The algorithm checks for incidents occurring at level L. This information gets stored in the incident tables at level L. The algorithm also checks whether new rows have been added to the incident tables that correspond to primitive events of $\mathcal{EG}$. If so, it implies new events that are of interest to level L have occurred and it calls the procedure *ProcessEvent* to process these events.

Algorithm 1
Detecting Primitive Events for Rules at Security Level L
Input: (i) $\mathcal{EG}$ – event graph for security level L, (ii) **ET** – Set of event tables where each table corresponds to a primitive event in $\mathcal{EG}$, (iii) **IT** – Set of incident tables for each primitive event in $\mathcal{EG}$ and also for other incidents at level L
Output: (i) **ET** – Set of updated event tables for each primitive event in $\mathcal{EG}$ and (ii) **IT** – Set of updated incident tables.

```
Procedure $DetectPrimitiveEvents(\mathcal{EG}, \mathbf{ET}, \mathbf{IT})$
begin
    while true do
        if an incident $e_x$ occurs at level $L$
            store its parameters in a new row in incident table $it_{e_x}$
        for each primitive event $e_i$ corresponding to leaf nodes in $\mathcal{EG}$
            if $L$ dominates $L(e_i)$
                if there is any new entry in incident table $it_{e_i}$
                begin
                    copy the new row in $it_{e_i}$ to event table $et_{e_i}$
                    $ProcessEvent(e_i, \mathcal{EG}, \mathbf{ET})$
                end
    end while
end
```

Algorithm 2
Processing Event e_i for Rules at Security Level L
Input: (i) $\mathcal{EG}$ – event graph for security level L and (ii) **ET** – Set of event tables where

each table corresponds to an event in $\mathcal{EG}$
Output: **ET** – Set of updated event tables for each event in $\mathcal{EG}$.

```
Procedure $ProcessEvent(e_i, \mathcal{EG}, \mathbf{ET})$
begin
    for each rule-id $r \in label_{e_i}$
    begin
        send event parameters to condition evaluator at level $L$
        mark node $e_i$ as occurred
    end
    for each node $e_j$ such that $(e_i, e_j)$ is an edge of $\mathcal{EG}$
    begin
        /* propagate parameters from node $e_i$ to node $e_j$ and detect composite event */
        $DetectCompositeEvent(\mathcal{EG}, \mathbf{ET}, e_i, param_i, e_j)$
    end
    Delete row containing parameters of instance of $e_i$ from table $et_{e_i}$
end
```

The above algorithm *ProcessEvent* describes the actions taken when an event e_i has been detected. Recall that $label_{e_i}$ contains the set of rules that are triggered by e_i. The parameters of e_i are then passed on to the condition evaluator which determines whether the rules listed in $label_{e_i}$ can be triggered or not. The node e_i is marked as *occurred* indicating that this event has taken place. The parameters of event e_i are then passed onto its parents and the *DetectCompositeEvent* procedure is called. The parameters of event e_i are then removed from the event table et_{e_i}.

Algorithm 3
Detecting Composite Event e_i for Rules at Security Level L
Input: (i) $\mathcal{EG}$ – event graph for security level L, (ii) **ET** – Set of event tables where each table corresponds to an event in $\mathcal{EG}$, (iii) e_i – child event that has occurred, (iv) $param_i$ – parameters of event e_i, and (v) e_j – parent event
Output: **ET** – Set of updated event tables for each event in $\mathcal{EG}$.

```
Procedure $DetectCompositeEvent(\mathcal{EG}, ET, e_i, param_i, e_j)$
begin
    case node $e_j$
        $\vee$: store $param_i$ in a new row in the table corresponding to the composite event
            $ProcessEvent(e_j, \mathcal{EG}, \mathbf{ET})$
        ;: if $e_i$ = left child of $e_j$
            insert a new row for this event instance and store $param_i$ in the composite event table
           if $e_i$ = right child of $e_j$
              if left child of $e_j$ is marked as occurred
              begin
                  store $param_i$ in the row for this event instance in the composite event table
                  $ProcessEvent(e_j, \mathcal{EG}, \mathbf{ET})$
              end
    end case
end
```

The above algorithm shows how composite events are detected for two kinds of event composition operators – $\vee$ and ;. When the parent node $e_j = \vee$, then occurrence of child e_i signals the occurrence of the composite event. In such a case, a new row is inserted in the event table corresponding to the composite event. The values inserted in this row correspond to parameters of the child e_i. The composite event is now processed by calling the procedure $ProcessEvent$. When the parent node $e_j =;$, the event detection is a little more complex. If e_i is the left child, then a new row is inserted in this composite event table. The parameters of e_i are recorded in this row. The event corresponding to the right child must occur before this composite event can take place. If e_i is the right child and the left child has already occurred, then the parameters of e_i are inserted in the corresponding row created by the left child, and the composite event is signaled. The composite event is then processed by calling $ProcessEvent$.

6 Related Work

Related Work in Multilevel Secure Active Databases: Very little work appears in the area of multilevel secure active databases. The major work in this area is by Smith and Winslett [24]. The authors show how an MLS relational model can be extended to incorporate active capabilities. The underlying MLS relational model supports polyinstantiation: that is, all MLS entities in this model can exist at multiple security levels simultaneously. An MLS rule being an MLS entity can also exist at multiple security levels. Unlike our work is that this work is based on the polyinstantiation model. Moreover, this work also does not discuss the details of event detection. In one of our earlier works [20] we discussed the structure of MLS rules, and what impact such rules have on the execution model of an active database. However, event detection was not discussed in details.

Related Work in Expert Systems: Morgenstern [19] considers the problem of covert channels in deductive databases that are subject to mandatory security requirements. Berson and Lunt [2] describe the problems that must be solved in incorporating mandatory security requirements in a product rule system. Garvey and Lunt [10] extend an MLS object-oriented database system with productions rules. Expert system rules differ from active database rules. Expert system rules are executed upon an explicit request for information; active database rules are executed as side effects. Expert system rules are used for inferencing – the order of rule execution is not important. This is not so with active databases.

Related Work in Active Databases: Many work has been performed in the area of active databases. Most of these work differ in the knowledge model and execution model. Some of these active databases use a relational model as an underlying database and others use an object-oriented one. Some of the prototypes based on the relational model are Starburst [27], Ariel [13], POSTGRES [26]. Notable among the object-oriented models are HiPAC [17,21], NAOS [7], Chimera [5], Ode [1] , SAMOS [11], Sentinel [6] and REACH [4].

Related Work in Multilevel Secure Databases: A large number of work also appears in multilevel secure database system. Majority of these work [8,9,12,14,15,22,23, 25] are in the area of relational database systems and some [3, 16, 18] are in the area of object oriented database systems.

7 Conclusion and Future Work

The absence of a multilevel secure active database limits the applicability of active technologies to applications that use an underlying MLS database system. Our work makes a small step towards fulfilling this gap. We have identified what kinds of rules can be supported and how these rules can be classified into security levels. An important component of rule processing is event detection. Towards this end, we have shown how events can be detected in an MLS active database without causing illegal information flow. We have not based our work on any relational or object-oriented models; our observations, therefore, will be useful to developers of any MLS active database system. In future, we plan to provide more details about the rule processing mechanism and the other components of an active database system. Our eventual goal is to implement an MLS active database system.

References

1. R. Agarwal and N. Gehani. Ode (Object database and environment): The language and the data model. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 36–45, Portland, OR, May 1989.
2. T. A. Berson and T. F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 235–242, Oakland, CA, April 1987.
3. N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Virtual View Model to Design a Secure Object-Oriented Database. In *Proceedings of the National Computer Security Conference*, pages 66–76, Baltimore, MD, October 1994.
4. A.P. Buchman, H. Branding, T. Kundrass, and J. Zimmermann. REACH: A REal-time ACtive and Heterogeneous Mediator System. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4), December 1992.
5. S. Ceri and R. Manthey. Consolidated specification of Chimera, the conceptual interface of idea. Technical Report IDEA.DD.2P.004, Politecnico di Milano, Milan, Italy, June 1993.
6. S. Chakravarthy, E. Hanson, and S.Y.W. Su. Active data/knowledge base research at the University of Florida. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4):35–39, December 1992.
7. C. Collet, T. Coupaye, and T. Svensen. NAOS– efficient and modular reactive capabilities in an object-oriented database system. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 132–143, Santiago, Chile, 1994.
8. D. Denning and T. F. Lunt. A multilevel relational data model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 220–234, Oakland, CA, May 1987.
9. P. A. Dwyer, G. D. Gelatis, and M. B. Thuraisingham. Multilevel security in database management systems. *Computers and Security*, 6(3):252–260, June 1987.
10. T. D. Garvey and T. F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings of the Sixth Computer Security Applications Conference*, pages 148–159, Tucson, AZ, December 1990.

11. S. Gatziu, A. Geppert, and K. R. Dittrich. Integrating active concepts into an object-oriented database system. In *Proceedings of the Third International Workshop on Database Programming Languages*, Nafplion, Greece, August 1991.
12. J. T. Haigh, R. C. O'Brien, and D. J. Thomsen. The LDV Secure Relational DBMS Model. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 265–279. Elsevier Science Publishers B.V. (North-Holland), 1991.
13. E. Hanson. Rule condition testing and action execution in Ariel. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–58, San Diego, CA, June 1992.
14. D. K. Hsiao, M. J. Kohler, and S. W. Stround. Query Modifications as Means of Controlling Access to Multilevel Secure Databases. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 221–240. Elsevier Science Publishers B.V. (North-Holland), 1991.
15. S. Jajodia and R. Sandhu. Toward a Multilevel Relational Data Model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 50–59, Denver, CO, 1991.
16. T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham. A Multilevel Security Model for Object-Oriented Systems. In *Proceedings of the National Computer Security Conference*, pages 1–9, Baltimore, MD, October 1988.
17. D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 215–224, Portland, OR, May 1989.
18. J. K. Millen and T.F. Lunt. Security for Object-Oriented Database Systems . In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 260–272, Oakland, CA, May 1992.
19. M. Morgenstern. Security and Inference in Multilevel Database and Knowledge-Base Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 357–373, San Francisco, CA, May 1987.
20. I. Ray. Multi-level secure active database rules and its impact on the design of active databases. In *Proceedings of the Twentieth British National Conference On Databases*, Coventry, U.K., July 2003.
21. A. Rosenthal, S. Chakravarthy, B. Blaustein, and J. Blakeley. Situation monitoring for active databases. In *Proceedings of the Fifteenth International Conference On Very Large Databases*, pages 455–464, Amsterdam, The Netherlands, August 1989.
22. R. Sandhu and S. Jajodia. Referential Integrity in Multilevel Secure Databases. In *Proceedings of the National Computer Security Conference*, pages 39–52, Baltimore, MD, September 1993.
23. L. M. Schlipper, J. Filsinger, and V. M. Doshi. A Multilevel Secure Database Management System Benchmark. In *Proceedings of the National Computer Security Conference*, pages 399–408, Baltimore, MD, October 1992.
24. K. Smith and M. Winslett. Multilevel secure rules: Integrating the multilevel and the active data model. Technical Report UIUCDCS-R-92-1732, University of Illinois, Urbana-Champaign, IL, March 1992.
25. P. D. Stachour and M. B. Thuraisingham. Design of LDV: A Multilevel Secure Relational Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):190–209, June 1990.
26. M. Stonebraker and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92, October 1991.
27. J. Widom. The Starburst Rule System: Language Design, Implementation and Application. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4):15–18, December 1992.