Formal Specification and Verification of an Extended Security Policy Model for Database Systems

Zhu Hong, Zhu Yi, Fu Ge

Database and Multimedia Technology Research Institute
Huazhong University of Science and Technology, Wuhan, Hubei, 430074, P.R China

zhuhong@public.wh.hb.cn

1 The work presented in this paper is supported by 863 hi-tech research and development program of China, granted number: 2006AA01Z430.

Abstract

In order to develop highly secure database systems to meet the requirements for class B2, an extended formal security policy model based on the BLP model is presented in this paper. A method for verifying security model for database systems is proposed. According to this method, the development of a formal specification and verification to ensure the security of the extended model is introduced. During the process of the verification, a number of mistakes have been identified and corrections have been made. Both the specification and verification are developed in Coq Proof Assistant. Our formal security model was improved and has been verified secure. This work demonstrates that our verification method is effective and sufficient and illustrates the necessity for formal verification of the extended model by using tools.

1. Introduction

In many database applications (such as e-business, military and governmental information systems), sensitive information needs to be processed. Database management systems are required to provide high levels of security.

Before a high secure database system is designed to meet the security requirements for class B2 [5], a formal security policy model is normally needed. The Bell & LaPadula model (BLP model) [1][2] is the security model recommended by the TCSEC [5]. It can be used to design a secure system to meet the criteria of Class A1. However when we apply the BLP model to modern databases, a problem arises: in the BLP model objects form a tree, but a tree would not be able to describe all the relationships among objects because of the complex relationships among objects in database systems. Therefore, objects and their structures in the BLP model need to be extended. Because of the extension, the descriptions of the security properties as well as state transition rules in the BLP model need to be modified or extended. In this study, we extended the BLP model to support complex relationships between objects in modern databases.

Once the BLP model is extended, the security of the extended model should be verified. In this paper, we mainly introduce our effort on specifying and verifying the security of our extended model by using proof assistant tools.

The contributions of this paper are in the following:
1) A formal security model for highly secure database systems is presented. The structures of objects in the security model are extended into two categories: tree and graph. Meanwhile, the security properties and state transition rules are also extended.
2) A method for verifying formal security model for database systems is developed by using proof assistant tools. During the process of the verification of the security model, a number of mistakes have been identified by our method and corrections have been made. Our formal security model was improved and has been verified secure. Both the specification and verification of the formal security model were developed in Coq Proof Assistant [15]. This experience illustrates the necessity of verification of the model by using tools.

The rest of this paper is organized as follows. First, section 2 is the related work. The extended model is illustrated in section 3. Our method for verifying the extended security model is proposed in section 4. The specification and verification of our security model in Coq is presented in section 5. We analyse the results of
2. Related Work

In this section, we will illustrate the related work for the security models for different secure systems as well as their verification.

The BLP model is widely applied in secure systems. It describes the well-known “no read up” and “no write down” rules (or simple security and “*-properties) for preserving mandatory security. “No read up” indicates that no subject is allowed to read from an object with a classification higher than the subject’s clearance. “No write down” requires that information at a classification cannot be written to an object at a lower classification. The BLP model also adds discretionary access controls which control the modes of access by subjects to objects.

Many researchers have formally analyzed the security of the BLP model and its extended models for operating systems.

Liping Li and Sihan Qin proposed a formal method to specify models by using Z. They took BLP model as an example in [11] and illustrated that the proving by using tools was more rigorous than proving manually. In addition, the BLP model was extended and formally analyzed when the extended model was applied in operation systems. Jianbo He analyzed an extended BLP model and specified the model in Z language. After they analyzed the model by using tools, they made a conclusion that their extended model was not secure in [7]. The paper illustrated the necessity of formal verification of the extended models. Moreover, verifying the security of a model by using tools is a more convincible way.

C. Maximiliano proposed an extension of a UNIX file system. They formalized their extension and analyzed that the file system operations satisfied a set of security properties by using the Coq Proof Assistant. In their paper, a method for specifying the policy of a system specified in Coq was presented in [13].

Other extended models based on the BLP model had also been formally analyzed.

The formal security policy model for the NATO Air Command and Control System was specified in Z language in [3]. An approach for specifying the model was presented in the paper, but their verification was finished manually because of the limitation of Z language.

Boniface Hicks has modeled the SELinux MLS policy and specified the model in the Prolog language in [8]. In addition, they developed an analyser which is able to automatically find information flows for SELinux in XSB Prolog. In addition, a method was proposed to determine policy compliance in different systems. According to the method, they proved the accordance of the SELinux reference policy with the simple security property and the *–property defined by Bell and LaPadula model.

The BLP model has been extensively applied in secure database systems and several security models have been proposed for secure databases. The first model was proposed by Hinkehchaever [9]. It supports the label of the objects at the granularity of attribute level, and it relies on the underlying trusted operating system for enforcement of both mandatory and discretionary security.

The TRW model [10] was developed to support the label of the objects at the granularity of tuple level. It enforces both mandatory and discretionary security within the database system, and it requires more trusted code and duplication of some of the security functionality of the operating system. Therefore the overload of verification was heavy.

The SeaView Project specified a formal security model to meet the security requirements for multilevel secure databases [12]. In addition, a general description of the database operations and the formal verification of the SQL operations in EHDM are presented in [16]. However, more objects, such as stored procedures and triggers in modern database systems, are not included in the SeaView model. Moreover, because the labels of the objects are at the granularity of element label (every element has a security level), more storage space was needed and the performance of the system would be decreased.

Wanjun Cheng extended the object hierarchy in a secure model for a database system and verified the transition rules manually [4], but objects such as stored procedures and triggers are not mentioned in their model.

3. Security Model

A security model is a set of states and rules which describe the security policy precisely and unambiguously. In this section we will briefly illustrate our extended model.

3.1 The Preliminary of the Model

We describe the state of a database system as a five-tuple \( v = (B, M, F, S, O) \). Every element of a state is extended or modified based on the BLP model in the following.
Because the structures of the objects in the BLP model would not be able to describe all the relationships among objects in database systems, we extended the objects and their structure in our model.

\( O \) is the set of all objects in database systems. We extended the structures of objects and divided the types of the objects into two categories: stored objects and derived objects. Suppose \( SO \) denotes the set of stored objects and \( O-SO \) denotes the set of derived objects in the following discussion.

There are three types of stored objects which are really stored in a database system. These objects include tuples, tables and databases. There are also three types of derived objects including views, stored procedures and triggers.

There are two categories of structures for objects in a database system: tree and graph. All the stored objects form a tree. For example, the database is the root, the tables are the children of the database, and tuples in a table are the children of the table. Every tuple or table has only one parent (the table or the database). Empty table has no child. A derived object is derived from stored objects or other derived objects. The relationships between derived objects and deriving objects form a graph. A derived object is a child of the objects from which it is derived. The object who derives an object is the parent of the derived object. For example, the view \( va \) is derived from table \( ta \) and \( tb \), and view \( vb \) is derived from \( ta \) and \( va \). Therefore, \( va \) is the son of \( ta \) and \( tb \), and \( va \) is the parent of \( vb \) and \( ta \). A derived object has at least one parent and may have no child if no derived objects are derived from it.

\( S \) is the set of all subjects, which are active sessions accessing objects in a database system under a state \( v \). In our model subjects in set \( S \) are all untrusted subjects.

\( A \) is a set of access mode. \( A = \{ \text{read(r)}, \text{append(a)}, \text{write(w)}, \text{control(c)}, \text{execute(e)} \} \).

\( B \) is an accessible set, consisting of triplets, \((s, o, x)\). Every element in set \( B \) denotes that subject \( s \) accesses object \( o \) in mode \( x \) under a state \( v \) in the database system.

\( M \) is an authorization set, consisting of triplets \((s, o, x)\). Every element in set \( M \) denotes subject \( s \) has privilege to access object \( o \) in mode \( x \).

\( F \) is a set of security level function, consisting of four functions \( f_s, f_e, f_{ol}, f_{oh} \). They returns the highest security level of a subject, the current security level of a subject, the lowest security level of an object, and the highest security level of an object respectively.

For any \( s \in S \), \( f_s(s) \geq f_e(s) \), the highest security level of \( s \) dominates its current security level.

After we have analyzed the performance and storage cost in SeaView model which labels the objects at the minimal granularity of element level, we label the objects at the minimal label granularity of tuple level. Namely, every tuple in our secure database system is labeled a security level.

If \( o \in (\text{Tuple} \cup \text{(O-SO)}) \) then \( f_{ol}(o) = f_{oh}(o) \). Tuple denotes the set of tuples. If the object \( o \) is a tuple or a derived object, then the security levels of object \( o \), \( f_{ol}(o) \) and \( f_{oh}(o) \) are the same.

If \( o \in (\text{Table} \cup \text{Database}) \) then \( f_{oh}(o) \geq f_{ol}(o) \). Table denotes the set of tables. Database denotes the set of databases. If the object \( o \) is a table or a database, the security level of \( o \) is between ranges of security levels \([f_{ol}(o), f_{oh}(o)]\). Such an object includes other objects which have different security levels. When an object \( o \) is created, \( f_{ol}(o) \) is the security level of the creator of \( o \), no matter what type of \( o \) is, and \( f_{oh}(o) \) is the highest security level of system if \( o \) is a table or a database.

Assume that \( \rho \) is a state transition rule, \( \rho: RE \times V- \rightarrow D \times V \). A database system has a set of state transition rules, denoted by \( \omega = \{ \rho_1, \rho_2, ..., \rho_d \} \).

\( RE \) is the set of requests.

\( V \) is the set of the states of the system.

\( D = \{ \text{yes, no} \} \), is the set of response decision.

A database system \( Sys \) is abstracted as a state machine and described as a triple \( Sys = (V, \omega, D) \).

Because of the extension of the objects, the descriptions of the integrity constraints, the security properties as well as state transition rules all need to be extended or modified.

### 3.2 The Security Properties

In our model the security properties are three extended integrity constraints and security axioms.

#### 3.2.1. Integrity Constraints

In the following, we will first present the object-compatibility property. Object-compatibility property is extended because the objects are extended in our model.

**object-compatibility property**: A state \( v = (B, M, F, S, O) \) satisfies the Object-Compatibility Property, if and only if for the state \( v \), an object \( o_2 \) is the parent of the object \( o_1 \), and one of the conditions below should be satisfied:

i) if \( o_1 \) is a stored object, \( f_{oh}(o_2) \geq f_{oh}(o_1) \), and \( f_{ol}(o_1) \geq f_{ol}(o_2) \).

ii) if \( o_1 \) is a derived object, \( f_{oh}(o_2) \geq f_{oh}(o_1) \), because \( f_{oh}(o_2) = f_{oh}(o_1) \).

When a subject accesses a table in a database system, tuples are usually in the final results. When a subject \( s \) accesses a stored object, if the current security level of \( s \) could not dominate the lowest security level of the object, \( s \) could not access any objects contained in the
object. Therefore, we have i) in above object-compatibility property.

We also label a derived object whose security level dominates the lowest security level of its parents. Therefore, the derived objects could only be accessed by subjects with higher security level. No information (even the descriptive information of objects) in the higher security level would be disclosed to a subject with lower security level. With the property above, we would prevent the covert channels by using data dictionary in database systems.

entity-integrity property: A state \( v=(B, M, F, S, O) \) satisfies the Entity-Integrity Property, if and only if in the state \( v \), for all tuples in any table, the primary key and the security level of the tuple together identify the tuple uniquely. The primary key should not be empty.

Assume only a primary key identifies a tuple uniquely. When a subject \( s \) inserts a new tuple \( t_1 \) at its security level \( f_o(s) \) into a table, if the primary key of \( t_1 \) is the same as the primary key of \( t_2 \) in the table and the security level of \( t_2 \) dominates \( f_o(s) \), the insertion of \( t_1 \) would not be allowed. Then the subject \( s \) would be able to infer that there exists a tuple whose primary key is the same as the primary key of \( t_1 \). In this situation, a covert channel would be exploited to transfer sensitive information. To avoid the covert channel, we permit the insertion of tuple \( t_1 \) and the security level of \( t_1 \) is \( f_o(s) \). The \( t_1 \) and \( t_2 \) in the same table are different tuples with the same primary key, but they are at different security levels. Then the covert channel is eliminated.

reference-integrity property: State \( v=(B, M, F, S, O) \) satisfies the Reference-Integrity Property, if and only if for the state \( v \), and there exist objects \( o_1 \) and \( o_2 \), the condition below should be satisfied:

If \( o_1 \), \( o_2 \) are tuples and the table which contains \( o_1 \) is referenced by the table which contains \( o_2 \), then the value of the primary key of \( o_1 \) equals to the value of the foreign key of \( o_2 \) and \( fol(o_1) = fol(o_2) \).

In the property above, we specified that the security level of tuple \( o_1 \) and \( o_2 \) should be equal. Otherwise, if a subject deletes the tuple \( o_1 \) referenced by a tuple \( o_2 \) in a higher security level, the delete operation would be rejected. Therefore, the subject at lower security level could infer that there is a tuple at higher security level referencing \( o_1 \). That would cause information leakage and covert channels.

Although there may be several foreign keys in a table, the notations of the property for several foreign keys are similar. Therefore, we only denote one foreign key without loss of generalization.

Entity-integrity property and reference-integrity property above are modified based on the SeaView Model. Integrity constraints are important in database systems. They define a correct database state after an operation is complete in database systems.

3.2.2. Security Axioms. The BLP model defines a security state and security system by simple-security property, star-property, and discretionary property. We modified the Star-Property, and Discretionary Property according to different types of objects. Our model should conform to these properties.

simple-security property: the simple-security property in our model is the same as the simple security property in BLP model.

The simple-security property indicates that no subject is allowed to read from an object with a classification higher than the subject’s clearance. Therefore, information flows from high security level to low security level are controlled and no illegal information flow would occur.

star-property: State \( v=(B, M, F, S, O) \) satisfies the Star-Property, if and only if for the state \( v \) and any subject \( s \), one of the conditions below should be satisfied:

i) when \( s \) accesses \( o \) in mode \( r \), \( f_o(s) \geq fol(o) \); ii) when \( s \) accesses \( o \) in mode \( a \), \( fol(o) \geq f_o(s) \geq fol(o) \); iii) when \( s \) accesses \( o \) in mode \( w \), \( f_o(s) = fol(o) \). iv) when \( s \) accesses \( o \) in mode \( c \) or \( e \), state \( v \) satisfies the Star-Property.

We use star-property to control the updated-involved access to objects in database systems.

discretionary property: State \( v=(B, M, F, S, O) \) satisfies the Discretionary Property, if and only if for the state \( v \) and any subject \( s \) accessing an object \( o \) in mode \( x \), either of the conditions below should be satisfied:

i) if the object \( o \) is a tuple, \( s \) has the privilege of the table which contains the tuple \( o \); ii) if the object \( o \) is not a tuple, \( s \) has the privilege to access object \( o \) in mode \( x \).

3.3 State transition rules

\[
re_i = \text{query};
\]

If \( [o_i \in O \land (s_i, o_i, r_i) \in M \land f_o(s_i) \geq fol(o_i) \land o_i' \in InvolvedO(o_i) \land f_o(s_i) \geq fol(o_i')] \]

Then \( \rho_i(re_i = \text{query}, v_j) = (yes, v_j = (b \cup (s_i, o_i', r_i), M, f, H)) \)

Else \( \rho_i(re_i = \text{query}, v_j) = (no, v_j) \)

Figure 1. The formal description of query rule

We defined nine state transition rules according to nine important operations in databases, including: query, insert, update, alter, execute, delete, drop, grant, and revoke. We take the query as an typical example.
In the following query rule, only one object (table) is concerned because the access to several objects is actually the composition of several accesses. Therefore, the access to one object is the basis and we only analyze the query rule for one object in the following.

In Figure 1, InvolvedO(o_j) denotes the set of tuples involved in o_j and o'_j ∈ InvolvedO(o_j) denotes o'_j is one of the tuples.

The query rule: Under state v_i, a subject s_i can query an object o_j in a database system, when the following conditions are satisfied:

i) o_j exists in the database;
ii) s_i has the privilege to access o_j in mode r;
iii) \text{\textit{InvolvedO}}(o_j) ≥ \text{\textit{fol}}(o_j).

If the response is “yes”, v_i is changed into v_j. For any tuple o'_j involved in o_j, s_i is added into the accessible set B in the state v_j to obtain a new B in v_j. The other four elements of v_j, M, S, F, O are the same as the elements in v_i and are not changed by the query operation.

However, if any condition in i), ii) and iii) above is not satisfied, the decision would be “no” and the state of the database system is still v_i.

4. The Method for Verification

4.1 The Extended Security Theorems

Before we verify the extended model, we should define the security invariants. The security invariants should describe the security policy accurately and could be preserved during the state transitions. If the security invariants are not consistent with the security policy, the system is not secure even if the invariants could be preserved during the state transition. In our extended model, the security policy is that no illegal information flow between subjects with different security levels. Namely, no information in high security level would be transferred to the subjects with low security level. Therefore, the security invariants could partly be insured by the three security axioms in our model.

The difference between database systems and other systems is the requirements for data integrity. For example, the ignorance of the entity integrity constraint would cause covert channels. Therefore, in order to forbid the illegal information flow in our model, we should preserve the integrity constraints in the database systems. In addition, during our proof when we specify the security invariants, the security invariants should be extended to include integrity constraints in our model. Namely, the integrity constraints should be preserved in every reachable state in a database system. Otherwise, the operations from a subject would not return the correct results and the security would be violated. Therefore, when we verify the security of a state in the system, we should verify the integrity of the state first.

The extended security theorems for verification are defined as follows. \textbf{Theorem-1}, \textbf{Theorem-2} and \textbf{Theorem-5} are extended theorems, \textbf{Theorem-3}, \textbf{Theorem-4}, \textbf{Inference-1} and \textbf{Theorem-6} are modified from BLP model.

\textbf{Theorem-1:} A state v_i satisfies the integrity constraints if and only if the state satisfies the Object-Compatibility Property, the Entity-Integrity Property and the Reference-Integrity Property.

\textbf{Theorem-2:} A state v_i is secure if and only if the state satisfies the integrity constraints, the Simple-Security property, the Star Property and the Discretionary Property.

\textbf{Theorem-3:} A state v_i is a reachable state if and only if it is the initial state v_0 or it is transferred from the initial state v_0 by a sequence of state transition rules \{\rho_1, \rho_2, \ldots, \rho_m\}.

\textbf{Theorem-4:} A state transition rule \rho_i \in \omega is secure if for any secure state v_i, a state v_{i+1}, which is changed from v_i by a request re under the state transition rule \rho_i, is also secure.

\textbf{Theorem-5:} Suppose state v_{i+1} is changed from v_i by a request re, and \rho_i is secure,

i) If v_i satisfies the integrity constraints, then v_{i+1} satisfies the integrity constraints,

ii) If v_i satisfies the Simple-Security Property and v_{i+1}satisfies the integrity constraints, then v_{i+1} satisfies the Simple-Security Property,

iii) If v_i satisfies the Star-Property and v_{i+1}satisfies the integrity constraints, then v_{i+1} satisfies the Star Property.

iv) If v_i satisfies the Discretionary Property and v_{i+1}satisfies the integrity constraints, then v_{i+1} satisfies the Discretionary Property.

\textbf{Inference-1:} A sequence of state transition rules is secure if and only if every rule in the sequence is secure.

\textbf{Theorem-6:} A system Sys is secure if and only if all the reachable states in the system are secure.

4.2 The Verification Method

According to \textbf{Theorem-6}, in order to verify the system is secure, we should verify that every state of the system is secure. The first step is verifying that the initial state is secure.

Different systems have different definitions of the initiate state. In our model for secure database system,
we assume that there are only untrusted subjects in the system. So both the set \( B \) and set \( S \) are empty in the initiate state \( v_0 = (B, M, F, S, O) \). Also there is no object created by untrusted subjects in set \( O \) in \( v_0 \). Therefore, we should verify that the initiate state satisfies the integrity constraints as well as the security axioms first. Then we should prove that any state \( v_{i+1} \), which is changed from a secure state \( v_i \) by a state transition rule \( \rho_i \), is also secure. Therefore, the state transition rule \( \rho_i \) is secure. From the steps above, we can obtain that any reachable state is secure if it is transferred from the initial state \( v_0 \) by a sequence of state transition rules \( \rho_{i1}, \rho_{i2}, ..., \rho_{in} \).

We have following steps when we verify that the model is secure.

**Step 1** Verify \( v_0 \) is secure.

All the other states are transferred from \( v_0 \) according to a sequence of state transition rules. After verifying that \( v_0 \) is secure, we should verify all the other states \( v_1, \ldots, v_n \), which are transferred from \( v_0 \), are also secure by any state transition rule \( \rho_i \).

Then we should prove that any state \( v_{i+1} \), which is changed from a secure state \( v_i \) by a state transition rule \( \rho_i \), is also secure. Therefore, the state transition rule \( \rho_i \) is secure.

**Step 2** Verify \( v_{i+1} \) satisfies the integrity constraints according to Theorem-1:

**Step 3** Verify \( v_{i+1} \) satisfies the Simple-Security property, the Star Property and the Discretionary Property;

**Step 4** Verify \( v_{i+1} \) is secure according to Theorem-2;

**Step 5** Verify any \( \rho_i \) is secure according to Theorem-4

**Step 6** Verify any sequence of the state rules is secure according to Inference-1.

**Step 7** Verify that any reachable state is secure if it is transferred from the initial state \( v_0 \) by a sequence of state transition rules \( \rho_{i1}, \rho_{i2}, ..., \rho_{in} \).

**Step 8** Verify the system is secure according to Theorem-6.

We applied the steps above to verify our extended model by using Coq in section 5. In addition, this method is able to be applied in verifying other security models for secure database systems.

5. **Formal Specification and Verification**

5.1 **The Coq Proof Assistant**

Coq Proof Assistant is a flexible theorem proving tool. It allows interactively constructing formal proofs and supports specification of static data, functions and definitions which can be developed using the basic specification language Gallina in Coq. Conveniently, the reasoning is at the same platform with the specification. What’s more, Coq provide various tactics to prove the goals interactively. All above strong points of Coq make us to use it to finish our specification and verification work.

5.2 **The Formal Specification of the Security Model in Coq**

We applied the method and the steps presented in section 5 to verify our model in Coq. Before we verified the model, we specify our model in the Gallina language in Coq.

A formal specification in Gallina consists of a sequence of declarations and definitions. There are basically three kinds of specifications in Coq: logical propositions, mathematical collections, and abstract types. They are classified by the three basic sorts of the system, called \( \text{Prop} \), \( \text{Set} \), and \( \text{Type} \) respectively. Every valid expression \( e \) in Gallina is associated with a valid expression, called type \( E \). We write \( e : E \) for the judgment that \( e \) is type \( E \). Arrow in Coq has two usages: i) function constructor \((\lambda \rightarrow B)\) means a function expecting one arguments of type \( A \) in order to get type \( B \); ii) the arrow symbol can be used as well as propositional connective in implication.

5.2.1 **Definition of Elements’ Type.** In Coq an element with type \( U \) is defined by keyword \text{Variable} \. A set consisting of elements with type \( U \) is defined by using \text{Sets} \ in Coq library (shown in Figure 2. line (1)). Moreover, \( S, O, B, M \) in our model are all sets in mathematics. In order to define such sets we should first define the type of the element in these sets.
5.2.2. Definitions of Proposition. We use predicate to specify the security invariants and integrity constrains of a system state.
is true indicates that state \( v \) satisfied the Reference-Integrity Property.

\[
\begin{align*}
(1) & \text{Definition Object-Compatibility}(v: \text{State}) := \\
(2) & \quad \forall (o_1, o_2: \text{Object_element}), \\
(3) & \quad o_1 \in v.O \land o_2 \in v.O \rightarrow \text{Parentof}(o_1) = \text{Parentof}(o_2) \\
(4) & \quad \forall (a_1 \in v.Table \lor a_2 \in v.Tuple) \\
(5) & \quad \forall (\text{Fol}(o_1) \geq v \land \text{Fol}(o_2) \geq v) \\
(6) & \quad \forall (o_1 \in v.View \lor a_1 \in v.Procedure_trigger) \\
(7) & \quad \forall (\text{Fol}(o_1) \geq v \land \text{Fol}(o_2)).
\end{align*}
\]

(8) Definition Entity_Integrity \((v: \text{State}) := \)

(9) Definition Reference_Integrity \((v: \text{State}) := \)

(10) Definition Security_State \((v: \text{State}) := \)

(11) Definition Integrity_State \((v: \text{State}) := \)

(12) Definition AddB \((v: \text{State}) := \)

Figure 6. Definitions of integrity properties in Coq

5.3 The Formal Verification

We verified the model following the steps presented in section 4.2. We take one state transition rule \( \text{query} \) as a typical example to illustrate our formal analysis of the model, which is accordant with the definition in section 3.3. As the key procedure above for verification is verifying that any rule in our extended model is secure, we describe the important procedure in detail.

Firstly, the rule \( \text{query} \) was defined as a predicate with two parameters: subject and object (shown in Figure 8). If the proposition \((\text{query} s o)\) is true, \( s \) and \( o \) query is satisfied and the state is unchanged.

In the \( \text{query} \) rule shown in Figure 8, if \( o \) exists and \( s \) has the privilege to query \( o \), a query request is permitted. The returned objects for the \( \text{query} \) rule are tuples no matter what type of \( o \) is. \( \text{InvolvedTuple}(o) \) is a function which returns a set of tuples involved in \( o \). For every tuple \( t \) in \( o \), if \( f(s) \geq \text{fol}(t) \) and \( t \) is in the results of the \( \text{query} \) rule, then \( (s, t, r) \) should be added into the set \( B \). Other elements of \( v_j, M, F, S \) and \( O \) are the same as that of \( v_i \) since they are not changed by the \( \text{query} \) rule.

\( \text{AddB} \) \((v: \text{State})\) describes the relationship between \( v_i \) and \( v_j \), because new elements are added to \( v_j, B \).

\[
\begin{align*}
\text{Definition Query} (s: \text{Subject_element}) \\
(o: \text{Object_element}): \text{Prop} := \\
(\forall t: \text{Object_element}) \\
(t \in v_j.O \land (s, o, t) \in v_j.M \\
\land (\text{Parentof}(o) = \text{Parentof}(t)) \\
\land (\text{KeyLevelof}(o) = \text{KeyLevelof}(t)) \\
\land (\text{Fkeyof}(o) = \text{Fkeyof}(t)) \\
\land (\text{Pkeyof}(o) = \text{Pkeyof}(t)) \\
\land (\text{sel}(s, t, r) \in \text{AddB}(v_i, s, t, r))).
\end{align*}
\]

Figure 8. The definition of query rule in Coq

Secondly we define the propositions needed to be verified by using the keyword \text{Lemma} in Coq. We should verify that the state \( v_i \) satisfies the object-compatibility property, the entity-Integrity property and the reference-integrity property respectively. Then we can conclude that \( v_j \) satisfies the integrity constraints according to \text{Theorem-1}. In the following, we should prove that \( v_j \) satisfies the simple-security Property, star property and discretionary property respectively in order to prove state \( v_j \) is secure. We take the verification of discretionary property in Figure 9 as a typical example.

\[
\begin{align*}
\text{Lemma Query_Discrentionary}: \\
(\forall (s: \text{Subject_element}, o: \text{Object_element}), \\
\text{Secure}(v_i) \rightarrow \text{Security_Discretionary}(v_j). \\
\end{align*}
\]
The state describes the table(s) deriving the view. As a result, we should have the privilege to read the parent of the returned objects are also the tuples in the results for the query s o. However, we don’t have this condition from the definition of the extended objects. The parent-and-son relationships may be something wrong in the state transition rule made us verify the Lemma Query_Discretionary unsuccessfully. Then we modified the definition of the query rule in Figure 2 and obtained the modified query rule in Figure 4. From the verification, mistakes and imprecise description which seems quite right intuitively in our mind could be found out. Because the Coq tool operates on the current proving goal by attempting to construct a proof of the current goal from corresponding conditions (Secure vi, Integrity_State vj and Query s o), if one condition was missing, the current goal would not be proved. This proving logic makes our verification sure stricter and more precise. The analogous mistakes could be corrected and our model could be improved.

The query rule in Figure 1 in section 3.3 should be modified in the following:

Definition Query (s: Subject_element)
(o:Object_element):Prop :=
((o ∈ vi.Table ∧ o ∈ vi.O ∧ (s,o,r) ∈ vi,M ∧ vj,Fc (s) ≥vj,Fol(o)) ∧ (forall t : Object_element, t ∈ vi, Tuple/o=Parentof (t) ∧ vj,Fc (s) ≥vj,Fol(t)) ∧ vj, Tuple/(Parentof t) ∈ DerivedSO ∧ vj,Fc (s) ≥vj,Fol(t)) ∧ (vj=AddB vi (s, t, r))).
In Figure 11, the formal description of modified query rule is illustrated. \( o_j = \text{parentof} \ (o'_j) \) represents that \( o'_j \) is the son of \( o_j \), if \( o_j \) is a derived object, \( o_i \in \text{DerivedSO} (o) \) represents that \( o_i \) is a stored object from which \( o_j \) is derived.

The verification procedure that verifies the state \( v_j \) satisfies simple-security property and star property are almost the same as the proof of the Lemma Query_Discetionary.

Finally, the state \( v_j \) is secure according to Theorem-2 and the query rule is secure according to Theorem-4.

Other eight state transition rules in our model could be also formally verified secure analogously. According to inference-1, any sequence of the state transition rules is secure.

Suppose a sequence is \( \text{Seq} = \rho_{i1}, \rho_{i2}, ..., \rho_{im} \), and \( v_0 \) is secure. If any state \( v_i \) is secure, the successive state \( v_{i+1} \) of \( v_i \), which is transferred by transition rule \( \rho_{ij} \), satisfies the integrity constraints according to Theorem-5 and the state \( v_{i+1} \) satisfies the Simple-Security property, the Star Property and the Discretionary Property according to Theorem-5. Therefore, the state \( v_{i+1} \) is secure according to Theorem-2. Analogically, the successive state of \( v_{i+1} \) transferred by transition rule \( \rho_{ij} \) is secure. Therefore, a reachable state \( v_j \) transferred from \( v_i \) by \( \text{Seq} \) is also secure.

Since \( v_0 \) is secure, any state \( v_i \) is secure, state \( v_j \) transferred from \( v_i \) by \( \text{Seq} \) is also secure. Finally, the system is secure according to Theorem-6.

6. The Result of the Work

Both the formal specification and verification were developed in Coq proof assistant. We spent two months to finish the formal proof, and 72 lemmas and 2100 lines Coq code were written to verify the nine state transition rules. The verification of the extended security model costs 20 minutes. The whole verification of our model is executed in the computer with Intel Core 2 Duo processor (1.86GHz), 2G memory and Microsoft Windows xp operating system.

During the formal analysing, four ambiguities and imprecise description for the model have been found out and modified. Our model has been improved. Moreover, Coq proofing assistant has been used in a new field for specifying and verifying the security model for database systems.

7. Conclusion and Future Work

A formal security model is vital for high secure DBMS development. In this study, an extended BLP model is presented to model the complex relationships between objects in modern database systems. It is based on the BLP model, the security properties and state transition rules are also extended.

Also, a method for verifying a formal security model for database systems is proposed. The integrity constraints are the premises of security axioms. Several mistakes have been identified during the verification process and have been corrected accordingly. This work demonstrated that our verification method is effective and sufficient.

In the future, we plan to extend our security model into a more specific level and closer to the implementation of database systems. We also plan to develop techniques to verify that all the SQL operations in database systems are accordant with our security model.

References


