Entity Modeling in the MLS Relational Model

Ken Smith Marianne Winslett
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801 USA
{ksmith, winslett}@cs.uiuc.edu

Abstract

Previous proposals for a multilevel secure relational model have utilized syntactic integrity properties to control problems such as polyinstantiation, pervasive ambiguity, and proliferation of tuples due to updates. Although successive versions of these models have shown steady improvement, most thorny problems have been mitigated but not resolved. We believe that the major roadblock to progress has been that no effort to date has shown what a multilevel secure database *means* semantically; instead the focus has been on making syntactic adjustments to avoid problems. In this paper, we introduce a belief-based semantics for multilevel secure databases that supports the description of semantic *multilevel secure entities*, and argue for the generality of this semantics. We also present our syntax for multilevel secure databases, and show its relationship to the semantics. Our syntax is free of most problems of previous models, and is also simpler without sacrificing security or expressiveness.

Proceedings of the 18th VLDB Conference
Vancouver, British Columbia, Canada 1992

1 Introduction

The proposals for a multilevel secure relational model [JS91, HOT91, DLS+87] implement the policy of *mandatory* protection defined in [Dep85] and interpreted for computerized systems by Bell and LaPadula [BL74]. Under mandatory protection, objects (data items) are assigned a security classification and subjects (active processes, users) are assigned a security clearance. Classifications and clearances are both taken from a common domain of *access classes*, or *levels*, which are partially ordered in a lattice. For example, levels Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U) are widely used. For two levels c_1 and c_2, if $c_1 > c_2$ in the lattice partial order, we say c_1 is 'higher than' or 'above' c_2. If $c_1 \geq c_2$, we say c_1 *dominates* c_2.

The Bell-LaPadula model imposes the 'no read up, no write down' restrictions on accesses by subjects. Subjects are only permitted to read from a level dominated by their own; subjects are only permitted to write to a level that dominates their own. This is sufficient to prevent subjects from directly passing information downward through the security lattice, as required by the mandatory policy. These two restrictions are known as the *simple* property and the $\star$-property (pronounced 'star property') respectively.

Multilevel secure (MLS) relational models have implemented these restrictions by associating access classes with the elements of a relation, such as tuples and fields. The simple property affects a user's view of a relation: only tuples in levels dominated by that of the user are visible. Therefore different users see different versions of a relation, depending on their access class. A subject may only update, delete, or insert items at his or her own level. Relations in which tuples are visible based on their access class are called MLS relations.

Indirect means of downward information flow, called covert channels [Lam73], must also be prevented. For

example, if a service of the database is denied to a subject based on the presence of a tuple at a higher level, the subject can infer the existence of that tuple, resulting in downward information flow. With respect to security, more is at stake than inferring the presence of a particular tuple. The success or failure of the service request can be used repeatedly to communicate a bit of information to the lower level. Therefore, *any* information visible at the high level can be sent through the channel.

The problem of *polyinstantiation* arises through the avoidance of a covert channel. If a user inserts a tuple with key k, a user from a lower level cannot be prevented from inserting a different tuple with key k at a later time, as refusing the later insertion would open a covert channel. As a result, MLS relations can contain multiple tuples with the same key value, known as polyinstantiated tuples. This problem has been addressed in previous models by means of syntactic integrity properties, which control the extent and form of polyinstantiation.

The recent Jajodia-Sandhu model [JS91] improves on prior models. However, several problems still remain to be solved.

- *Semantic ambiguity.* An MLS relation does not always have a single semantic interpretation. For example, Figure 1 shows an instance of the relation scheme SOD(Starship, Objective, Destination), where Starship is the key, as an MLS relation under the model of [JS91]. Each attribute value is followed by its security classification; TC is the classification of the entire tuple. Consider

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Vulcan U	U
Enterprise C	Diplomacy C	Romulus C	C

Figure 1: A Starship Database

  the two polyinstantiated tuples sharing the attribute value 'Enterprise'. These tuples can be interpreted in at least two ways:

  - They represent two different levels of secure understanding of a single starship named Enterprise: the first tuple gives the perspective of an unclassified subject, and the second tuple gives the perspective of a confidential subject.
  - Two entirely different starships exist, both (unfortunately) with the name Enterprise; one tuple refers to each ship.

- *Query ambiguity.* In Figure 1, when a confidential subject asks:

  SELECT Destination
  FROM SOD
  WHERE Starship = 'Enterprise',
  what should be the answer? One possibility is 'Romulus', because that is the value associated with that user's security level. Another answer could be the set {'Romulus', 'Vulcan'}, because that is the exhaustive list of all values associated with destinations of Enterprises in the database. Current MLS proposals do not address this point.

  In the case of a more complicated query, the ambiguity becomes more acute: current proposals will join together two tuples from different security levels, even in the common case where it is clear that no subject at any level would believe that the joined tuple correctly reflects the state of the world. For example, no one believes that the Enterprise is conducting exploration on Romulus, yet a series of projections and joins could produce that tuple as part of a query answer. Should these dubious joined tuples contribute to a query answer?

- *Proliferation of tuples due to updates.* In the Denning-Lunt model [DLS+87], updates can introduce a number of new polyinstantiatied tuples that is exponential in the number of updated non-key attributes in the relation [JS90]. The Jajodia-Sandhu model [JS90] eliminates much but not all of this proliferation, as in the following update to Figure 1 by a TS subject:
  UPDATE SOD
  SET Destination = 'Pluto'
  WHERE Starship = 'Enterprise'.
  As Figure 2 shows, the number of Enterprise tuples doubles after this update is performed. If

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Vulcan U	U
Enterprise C	Diplomacy C	Romulus C	C
Enterprise U	Exploration U	Pluto TS	TS
Enterprise C	Diplomacy C	Pluto TS	TS

Figure 2: Tuple Proliferation Upon Update

  the two Enterprise tuples refer to different secure understandings of *one* ship, this update can be interpreted to mean the TS-user is adding a still-more-secure understanding about the destination of the Enterprise. Doubling the number of tuples should not be necessary to add this understanding, as a semantics can be conceived in which this information can be represented in one tuple.

The common theme in these problems is the lack of

any semantics underlying the MLS relational model. Without a semantics, syntactic issues cannot be resolved through connection to the semantics. For example, polyinstantiation has posed such a thorny issue because it is not clear what polyinstantiation *means*. In this paper, we contribute a semantics for MLS relational databases based on database interpretations containing *multilevel secure entities*, entities which span multiple levels of security and offer the ability to model multiple secure understandings of attribute values. We then present a new syntactic MLS relational model based directly on our semantics. We are able to map directly and unambiguously from syntactic MLS relations to our semantics, and back again. Our model is both syntactically simpler and semantically clearer than previous models. In addition, we are able to resolve difficult research issues in MLS relational databases in the context of our model: syntactic and semantic ambiguity, the appropriate granularity for security labels, the meaning of nulls, the meaning of queries, the proliferation of tuples under updates, and the profitable utilization of polyinstantiation.

Our lever for attacking these problems is our assumption that databases are used to describe the existence and properties of entities, as is true of relations derived from entities in an entity-relationship diagram. In general, databases will also contain relations derived from the relationships in an ER diagram. The treatment of relationships is a natural extension of our handling of entities; we omit its presentation in this paper due to space constraints.

The remainder of this paper is organized as follows. In Section 2, we present our belief-based semantics and the modeling concept of MLS entities. Section 3 describes a syntax for MLS relational databases based on our semantics. Section 4 discusses the relationship of our model to the the Jajodia-Sandhu (and other) models. Section 5 defines select, insert, update, and delete operations for MLS relational databases under our syntax and semantics. Finally, in Section 6, we summarize our results and describe future work.

2 A Semantics for MLS Relational Databases

In our model, we distinguish between the semantic and syntactic aspects of MLS databases; this section describes our semantics for MLS databases that model entities. First, we define an *interpretation* (in the logic sense) of an MLS database, which draws on Kripke models with a simple, non-logic-based presentation. The interpretation of an MLS database captures the concept of multiple levels of belief about a shared set of entities. Second, we define *multilevel secure (MLS) entities*, entities which may span multiple levels of the interpretation. Also, we show that every interpretation of an MLS database corresponds to a set of MLS entities.

2.1 Belief-based Interpretations

Under our semantics, the interpretation of an MLS database is a set of ordinary relational databases, one database for each level in the security lattice.[1] The databases all share the same schema[2], and each database is labeled with its level. The database instances may contain null values in non-key attributes. In addition, there is a binary relationship between databases in the interpretation, which holds exactly when the label of the first database dominates the second, according to the security lattice. From the properties of the security lattice, it follows immediately that the binary relationship is reflexive, antisymmetric, and transitive. Finally, the schemas and instances of the databases must satisfy certain constraints, which we motivate and present as Property 1 in Section 2.2.

The database for a level contains the total *beliefs* of the subjects of that level about the state of the world reflected in the schema.[3] We use the term 'belief' because different levels may make different statements about the value of the same attribute for the same entity. A null value in a tuple means that the subjects at that level believe that a value exists for that attribute, but do not have a belief about what that value is.

The binary relationships between databases in an interpretation are motivated by Figure 3, which shows subjects and databases for three linearly ordered levels: S, C, and U. A subject *believes* the contents of the database at its own level, as represented by the thick arrow in the figure from a circle (a set of subjects) to a box (a database) at the same level. The subjects of each level *see* what they and the subjects of each lower level believe, as represented by the thin arrows between subject groups in the figure. A subject may see many tuples that it does not itself believe.

[1] There is the potential for confusion between the syntactic MLS database and its interpretation as a set of databases. In this section, the term 'database' will refer to a database for a level of the interpretation. In Section 3 we concentrate on syntactic databases.

[2] If different schemas are desirable at different levels, this can be accomplished using techniques similar to those used in the SeaView project [DLS+87]. The issue of different schemas at different levels is orthogonal to our proposal for semantics.

[3] We use the term 'belief' in an intuitive manner here, not in the technical sense of modal logic models of belief, which are not necessary in this simple application.

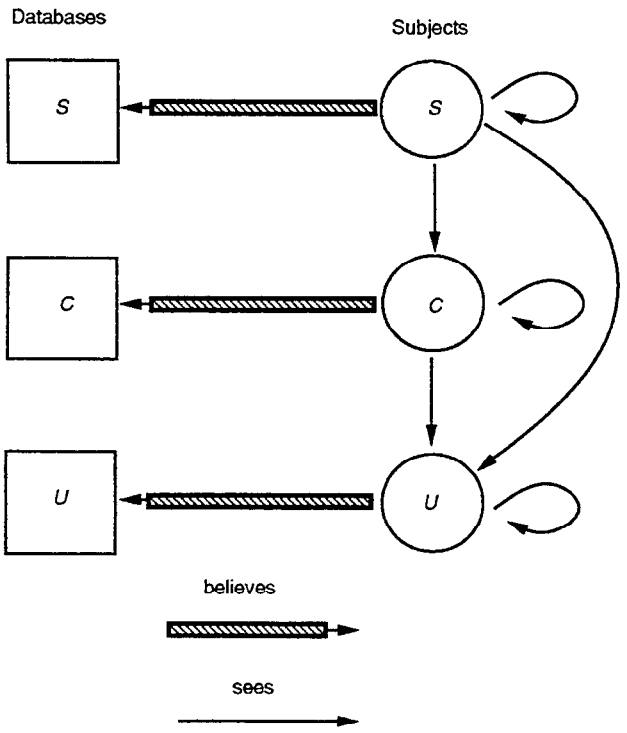

Figure 3: Relationships Between Subjects and Databases of Different Levels

Database access privileges for subjects are summarized below:

MLS Database Access.

1. **Update Access.** A database update request (insert, delete, update) from a subject can only alter data in the interpretation at the subject's own level. Data at a particular level can only be altered by subjects at that level.

2. **Read Access.** A query from a subject at level l can access data from exactly those databases whose label is dominated by l.

The read access rule corresponds to the Bell-LaPadula simple property: a subject can retrieve exactly what it can see that someone believes. The update access rule states that a subject can change its own beliefs, but no one else's. The update access property is stronger than the Bell-LaPadula $\star$-property; the latter is semantically tailored to message-passing systems, where it would not violate security to allow a lower level to append a message to a higher level's message queue. We will consider queries and updates in more detail in Section 5, including the ability to specify in a query just whose beliefs the query is to be evaluated against.

We believe that this simple model of separate-level databases, related to one another by the visibility criterion in the security lattice, should underlie every MLS database syntax proposal. Surely from any syntactic MLS database one should be able to extract an ordinary database that describes exactly what the subjects at a particular level believe to be the state of the world; yet it is not at all obvious how to perform such a mapping under previous proposals.

In the next section, we show that each interpretation corresponds to a set of MLS entities. In Section 5, it will be easy to define the semantics of queries and updates by explaining the effect of the operations on the levels of the interpretation of the MLS database, or on its MLS entities. Previous proposals had no such model to fall back upon, and so their answers to queries and updates differ sharply from our own.

2.2 Multilevel Secure Entities

Let us consider the need for additional constraints on the schema of the interpretation of an MLS database. Consider an ordinary (non-MLS) relation R with key K (K may be a set of attributes), plus m non-key attributes: $R(K, A_1, \ldots, A_m)$. When R is used in an MLS schema, K will not function properly as an entity key. For example, if subject s inserts a tuple with key k, s does not know if a tuple with key k already exists in some database at a higher level. If key k has already been used at a higher level to identify a different entity, then reuse of the key at a lower level will make it impossible to unambiguously refer to these entities. Rejecting the insertion request would introduce a covert channel. Property 1 addresses this facet of the polyinstantiation problem.

> **Property 1 [Key Classifications].** The schema for every relation R in the interpretation must include a 'key classification' attribute KC, representing the security level of the information in K: $R(K, KC, A_1, \ldots, A_m)$. The domain of KC in a database at level l is the set of lattice levels dominated by l. Furthermore, K concatenated with KC (written $K \mid KC$, and called the entity identifier (eid)) functionally determines all the attributes of R within the database at level l.

The inclusion of KC allows us to differentiate between the entities believed to exist by different levels. For example, level S can include information about the existence of a secret starship named Enterprise. Later, level U can innocently name a new starship 'Enterprise', and record information about it in the

database at level U. There is no confusion between the starship at level U and that at level S: they are different entities with different eids. If, on the other hand, the starship newly described at level U is in fact the same as the starship described at level S, level S can record the fact that the two entities are the same by changing KC to 'U' in its records on the Enterprise. The S-level information recorded about the Enterprise can still be quite different from that recorded at level U, corresponding to a better understanding at the higher security level regarding the true nature of the Enterprise. Finally, level S can record its belief in *both* starships, by including in the database at level S a tuple with eid 'Enterprise U' and a tuple with eid 'Enterprise S'. The tuples refer to distinct real-world entities.

The attribute(s) K that the user originally designated as a key for R may of course fail to be a key in the interpretation of R at a level. The above case of level S believing in both Enterprise starships is an example. For this reason, we call K the *apparent key* of R.

Because the eid functionally determines the other attributes of R at level l, l's database cannot contain two different beliefs about the value of an attribute of a entity; the beliefs of a level about an MLS entity are unique. As an example, the two tuples of Figure 4 violate Property 1, since they differ on the secret objective of the starship Voyager.

Starship	KC	Objective	Destination	TC
Voyager	S	Shipping	Mars	S
Voyager	S	Spying	Mars	S

Figure 4: Violation of Unique Belief Requirement

In including KC in the key for an entity, we have altered the original schema for R and diverged from the standard definition of an entity. In the ER model [Che76], entities are real world objects which exist, have associated attribute values, and are distinguishable from other entities [KS91]. MLS relational databases differ from the standard relational model in that entity existence and attributes are not modeled by a single tuple. In the MLS relational model, a single real-world entity can be described in several tuples, each at a different level, as in the Enterprise example. We call these *multilevel secure (MLS) entities*; they correspond to real world entities for which a *range* of different beliefs, or levels of secure understanding, can be represented.[4]

[4]Note that due to the lattice structure, there is no means of recording the fact that two lower incomparable levels are in fact describing the same real-world entity.

More precisely, the MLS entity from relation R with entity identifier *eid* is obtained as follows: we append an additional attribute TC (*tuple class*) to R, such that in each database the value of TC is the level label of that database. The entity is the set of tuples

$$\sigma_{K|KC=eid}(R),$$

unioned over all database levels. In addition, to make the entity self-describing, it is tagged with the schema information for R and a copy of the security lattice. Every interpretation can be transformed into exactly one set of MLS entities by applying this derivation to all relations and all eids.

One can also show that this mapping is invertible, by giving a direct definition of MLS entities that includes all applicable constraints (no nulls in keys, same schema for every relation, tuple class must dominate key class, etc.). We will omit the direct definition, since it is not needed for our purposes.

The tuples of an MLS entity record whether each level believes in the existence of the entity, and, if existence is believed, then possibly additional information about the values believed to hold for attributes of that entity. A tuple at level l that contains all null values except for the entity's eid means that the subjects at level l believe that the entity exists, but have no beliefs about what its values are. It is immediate that if no tuple for a particular eid appears at level l, then subjects at that level do not believe that that entity exists. As a mechanism for uniquely identifying MLS entities, at the time an MLS entity first appears in the database, it must have a *base tuple*, a tuple whose key class and tuple class are equal. A base tuple is a lowest-level database tuple where the existence of an entity is asserted.

For example, consider a starship relation in our simple three-level database, where the existence of the starship *Enterprise* is asserted by level U. The MLS entity with eid *Enterprise* U is shown in Figure 5. The base tuple of *Enterprise* U comes from the

Starship	KC	Objective	Destination	TC
Enterprise	U	Exploration	Vulcan	U
Enterprise	U	Exploration	Romulus	S

Figure 5: The MLS Entity *Enterprise* U

database at level U. Level S agrees that *Enterprise* U (the *Enterprise* believed in by level U) exists, and agrees with level U regarding the starship's objective, but has a different belief about its destination. Note that the beliefs of U about the *Enterprise* U are seen by level C, but C does not believe that *Enterprise* U exists, since no C tuple belongs to *Enterprise* U.

If the level U tuple describing the Enterprise were deleted, then the remaining S-level tuple would record S-level beliefs about the Enterprise entity formerly believed to exist by level U. Non-base tuples can provide a mechanism for users with a higher security clearance to represent their (possibly conflicting) beliefs about an entity in an MLS database.

MLS entities composed of base and non-base tuples as a meaning for polyinstantiation is an important feature for security applications, and is not present in standard relational databases. MLS entities motivate the semantics of the UPDATE operation of Section 5.

3 The Syntax of MLS Relational Databases

Section 2 gave a semantics for MLS relational databases, by assigning one single-level relational database to each security level, providing a visibility relation between appropriate levels, and defining a set of MLS entities which spans those levels. In this section we present a simple syntax for MLS relational databases that merges the information from different levels into a single multilevel database, show the unambiguous mapping between syntax and semantics, and argue that the attribute-level classifications used in most other models do not model MLS entities any more powerfully. In this section, we will be careful to distinguish between the syntactic representation of information, which we will call the database, relation, tuple, etc., and the semantics of that information, which we will refer to as its interpretation.

Suppose that, as in Section 2, we have a single-level relation R with scheme $R(K, A_1, \ldots, A_m)$, where K is a set of attributes that form the key for R. As in Section 2, each non-key attribute may assume either a value from its underlying domain or a null value meaning 'value exists but no belief given.' To convert R into an MLS relation, we add new attributes KC and TC, giving scheme $R(K, KC, A_1, \ldots, A_m, TC)$. The purpose of KC and TC was explained in Section 2, and their properties can be summarized in an analog of Property 1:

> **Property 1′ [Key Classifications].** The schema for every relation R must include a 'key classification' attribute KC, representing the security level of the information in K, and a 'tuple classification' attribute TC, representing the security level of the remainder of the tuple: $R(K, KC, A_1, \ldots, A_m, TC)$. The domain of TC and KC is the set of security levels, with the restriction that in a particular tuple, the attribute value for TC must dominate that for KC. Furthermore, $K \mid KC \mid TC$ must functionally determine all the attributes of R, and together form the key of the syntactic relation R.

An MLS entity from relation R consists, as before, of all the tuples of R having the same eid $(K \mid KC)$.

As an example of an MLS relation, consider the familiar SOD relation in Figure 6. The interpretation

Starship	KC	Objective	Destination	TC
Voyager	U	Shipping	Mars	U
Enterprise	U	Exploration	Vulcan	U
Enterprise	U	Diplomacy	Romulus	C
Zardor	S	Warfare	Romulus	S

Figure 6: SOD: An MLS Relation for Starships

of SOD (formalized below) contains three databases. In the level U database, the relation interpretation for SOD contains the first two tuples of SOD (those having a TC of U), with the TC attribute removed. Levels C and S of the SOD interpretation contain the third and fourth tuples, respectively, truncated.

We now show the unambiguous mapping between our syntax and our semantics. Every syntactic MLS database has one interpretation, constructed from the syntactic database as follows. The interpretation contains one database for each level in the security lattice. The database at level l is related to the one at level l' iff l dominates l'. The relation schemas at each level are the same as those in the MLS database, except that the TC attribute of each relation is removed in the interpretation. The instance of R in the interpretation at level l is obtained by selecting all those tuples of R in the syntactic database that have value l for attribute TC, and then projecting out TC.

Each interpretation so obtained is a legal interpretation. First, the interpretation has the same schema at each level. Second, it has no nulls in non-key attributes, because the syntax for R forbids nulls in K, KC, and TC. Third, it satisfies Property 1: Every relation schema at level l contains a key classification, whose domain will not include any values that are higher than l, by Property 1′. In addition, Property 1′ guarantees that $K \mid KC \mid TC$ functionally determines all the attributes in the syntactic R. Therefore if all tuples of R are guaranteed to have the same value for TC, $K \mid KC$ will functionally determine all attributes of R. Therefore $K \mid KC$ functionally determines all the attributes in the interpretation of R at a particular level.

Conversely, every MLS interpretation can be expressed as an MLS database by appending a new TC

attribute to each relation scheme in the interpretation, giving each tuple at level l the value l for attribute TC, and then taking the union of the tuples at different levels for each relation. It is immediate that the composition of the mappings from syntax to semantics and back again is the identity function.

3.1 Properties of this Syntax

Semantic ambiguity exists when one MLS relation can have more than one interpretation. Previous MLS proposals have not included any definition of semantics; the example of Figure 2 show that it is easy to think of multiple interpretations for those syntactic proposals, in which the same database corresponds to several distinct sets of MLS entities. Conversely, it is easy to think of several ways to represent the same MLS entity under the syntax of previous examples, which is *syntactic ambiguity*. The one-to-one correspondence between syntactic databases and their semantic interpretations in our proposal prevents both syntactic and semantic ambiguity.

We also argue that strict attribute-level security classifications (one security attribute for every data attribute) do not model MLS entities any more powerfully. Most models use an attribute level security scheme in an effort to provide maximum modeling power [JS91, DLS+87]. One way to define modeling power is the ability of a particular syntax to express a particular semantics, such as we have defined in Section 2. Since other proposals do not present a semantics for MLS databases, it is difficult to define 'modeling power' in their cases. However, we argued in Section 2 that any MLS syntax should be reducible to a semantics that includes a set of single-level databases, showing what is believed at each level, and which can be used to capture the concept of MLS entities. The current section has shown that it is easy to capture such a semantics using a syntax in which non-key data attributes are uniformly classified. Only two security attributes per relation are needed to model MLS entities under this approach [5].

4 Comparisons

Several of the concepts introduced in Section 2 have been proposed in other MLS relational models. For example, key classifications are used in all MLS relational proposals to date. A functional dependency equivalent to Property 1 was proposed and called 'Polyinstantiation Integrity' in [JS91] and [DLS+87]. The semantic concept of eids has not been proposed before. A concept of MLS entities can be detected in several papers [JS91] [HOT91] as well. One author goes so far as to state that tuples with the same apparent key but different key classifications refer to different entities, and tuples with the same apparent key and key classification refer to the same entity [Lun91].

Previous authors have not made use of MLS entities to give meaning to relational operations as we do in Section 5. They have also not considered the questions of relationships between MLS entities, or the question of what exactly users at each level believe to be the state of the world, as we do in Section 2. None of them have introduced a formal semantics for their data model or operations.

In the following, we consider the work of other database security researchers, as it relates to the work of this paper. We compare our model with other existing MLS relational models. Since no other MLS relational model develops a semantics, no comparison is made at that level, although we note stated intentions. Instead, we summarize the syntactic properties of each and make observations and comparisons with our own. We focus on the recent Jajodia-Sandhu model in our comments.

4.1 The Jajodia-Sandhu Model

In the Jajodia-Sandhu (JS) model, an MLS relational database is a set of multilevel relations, whose definition is different than that in our proposal. The schema for a single-level relation $R(A_1, \ldots, A_n)$ becomes

$$R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$$

under the JS model, where each A_i is a *data attribute* over a domain D_i, each C_i is a *classification attribute* for A_i, and TC is the *tuple-class* attribute. The domain of each C_i is specified by a range $[L_i, H_i]$ which defines a sub-lattice of access classes ranging from L_i up to H_i. The domain of TC is $[lub\{L_i : 1 \leq i \leq n\}, lub\{H_i : 1 \leq i \leq n\}]$ (where *lub* denotes least upper bound).

A *relation instance* is a set of tuples, each of the form

$$(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc),$$

where each $a_i \in D_i$ or $a_i = \text{null}$, $c_i \in [L_i, H_i]$, and $tc = lub\{c_i : 1 \leq i \leq n\}$.

A subset of the attributes of an MLS relation is designated as its user-specified key K. As in our proposal, K is not a key in practice, due to polyinstantiation; therefore K is also called the *apparent key*. In the JS model, K is restricted in two ways, both of which are included in our own proposal: tuples cannot contain nulls in K, and all attributes of K receive the same

[5] An MLS relationship [Smi92] can require more than two security attributes per tuple, however less than one per data attribute is required in general.

classification in a tuple. KC, the classification of the key, is therefore spoken of as though it were a single attribute in the JS model. The value of KC must dominate the classification of any attribute in R: if an attribute of R is visible to a user, then its key is too.

Additionally, the JS model requires that all data attributes receiving a null value be given the same classification as KC. This means subjects cannot state that, in contrast to what might be believed at lower levels, they are uncertain about the value of a particular attribute.

The JS model constrains schemas in additional ways. First, the JS model introduced the concept of *Polyinstantation Integrity*, from which the functional dependency in our Property 1 is drawn:

[Polyinstantiation Integrity].

$\{K, KC, C_i\} \rightarrow A_i$ must be satisfied in an MLS relation R for all $1 \leq i \leq n$.

With a security label on each field, in the extreme case a single tuple can contain a different security class for each data attribute label. Therefore, there is no direct and simple correspondence between a syntactic tuple in a JS MLS relation and the set of beliefs of a security class about an MLS entity, in the style of Section 2.

In the JS model, if l dominates all the security levels present in a tuple, then the subject can see that tuple. If all the security levels present in a tuple dominate l, then the subject cannot see that tuple. However, if l dominates some security levels in a tuple but not others, then during query processing the JS model uses a *filter function* σ to replace by nulls all attribute values that the user does not have the clearance to see. Given a tuple t, and level l, for which $KC \leq l \leq lub\{c_i : 1 \leq i \leq n\}$, $\sigma(t, l)$ returns a tuple t' using the following procedure. For each non-key attribute A_i in t,

$$t'[A_i, C_i] = \begin{cases} t[A_i, C_i] & \text{if } t[C_i] \leq l \\ < null, KC > & \text{otherwise.} \end{cases}$$

Unfortunately, σ introduces an additional semantics for nulls: a null value can now mean 'information hidden.' As 'value exists but is unknown' nulls can also be present in the database, it is hard for a subject to tell what is believed by its own level, as information hidden by higher layers may look like uncertainty present at the subject's own level.

Another source of semantic and syntactic ambiguity is that there is no means of determining whether tuples with the same key but different key classifications refer to the same or to different underlying entities. For instance, no semantic distinction is made between the two tuples of Figure 7, which might have the same meaning in the JS model, despite their syntactic difference. In Figure 7, C_i immediately follows A_i.

Starship	Objective	Destination	TC
Enterprise U	Diplomacy C	Romulus C	C
Enterprise C	Diplomacy C	Romulus C	C

Figure 7: SOD: Possible Tuples

In the JS model, it is difficult to determine exactly what a subject at level l believes and thinks that others believe, due in part to the problems with nulls and with entity identity described above. It is unclear whether higher levels that make no statement about an entity inherit the beliefs of lower levels or not, or how conflicting beliefs from lower levels could be resolved. There is no way for a level to state that it does not believe in the existence of an entity that lower levels believe in. In addition, queries are evaluated over all tuples seen by the subject, even though the answers to such a query might not be believable by any level in the database.

4.2 The Denning-Lunt Model

The Denning-Lunt (DL) model [DLS+87] is being used in the SeaView project, a joint effort between SRI International and Gemini Computers to produce an MLS relational database. This project uses attribute level security and was the first to point out the important issue of polyinstantiation. The DL model is similar to the JS model: each attribute of a tuple has an individual security level, and key attributes are uniformly classified and cannot contain null values. The most notable difference is in the treatment of polyinstantiation. It was shown in [JS90] that the DL model can produce a number of tuples exponential in the number of non-key attributes in a relation. [JS90] argues that these tuples are spurious. This proliferation of tuples on updates is traced to the DL requirement that, in addition to Polyinstantiation Integrity, the key and its classification must determine each attribute and its classification via a multivalued dependency. This latter requirement is dropped in the JS model.

Other MLS database work done by Teresa Lunt includes work on an object-oriented model [Lun90] which proposes multilevel objects and defines constraints for their secure classification. This work has been extended to object-oriented databases with the expert systems functionality of forward and backward chaining rules to implement, among other things, classification constraints on data [GL90] [LM89].

In a recent paper [Lun91], Lunt discussed the need for understanding the affect of semantics on polyinstantiation, and proposed a solution in harmony with the approach developed in this paper. She distinguishes between *polyinstantiated elements* and *polyinstantiated tuples*. Polyinstantiated elements correspond closely to MLS entities (although in an attribute-level classification scheme). The operations of [DLS+87] are being adjusted to reflect this new semantic understanding.

4.3 The LDV Model

The LDV model, proposed in [HOT91], is also the base of a research protoype. This model eschews attribute level security, seeking instead to obtain modeling power by augmenting the model with syntactic tools such as the *derive* option. The derive option permits a user to require that the value of an attribute at a lower tuple is automatically copied up into that attribute of its own higher-level tuple. The LDV model makes mention of the existence of 'multilevel entities'; however, the concept is not developed.

5 Operations

In this section, we present simple MLS versions of four SQL relational operations: select, insert, update, and delete, defined on the MLS relational syntax presented in Section 3. We assume the MLS relations involved contain MLS entities, as described in Section 2.2. The syntactic effect of each operation on the tuples of the relation is defined by its effect on the semantic interpretation of the relation. For the update operation, the semantics are motivated by operation's effect on the MLS entities involved. In contrast, the operations defined in [JS91, DLS+87] are based purely on syntactic manipulations.

Recall from Section 2 that subjects can only change beliefs at their own level. The INSERT operation asserts the existence of new MLS entities, and the UPDATE operation alters a level's beliefs regarding preexisting MLS entities. In the following notation, the rectangular brackets denote optional items, and '*' denotes zero or more repetitions.

5.1 Select

The SELECT statement has the following general form:

SELECT	A_i [, A_j]*
FROM	R_k [, R_l]*
WHERE	P
[BELIEVED BY	L] ;

In this operation, as in those described in subsequent subsections, the SELECT, FROM, and WHERE clauses are ordinary SQL syntax with the ordinary SQL syntactic restrictions. Nested SELECTs cannot have a BELIEVED BY clause, for reasons explained below. TC may not appear in the SELECT clause; it is implicitly part of the answer to every query.

The clause BELIEVED BY tells which database interpretation levels the query is to be evaluated against. L is a list of one or more security levels or security variables, described below. If the BELIEVED BY clause is absent, L is set to the security variable SELF.

Before query execution, security variables in L are replaced by explicit security levels. *SELF* is replaced by the level of the query issuer. *ANYONE* is replaced by a list of all levels that the query issuer dominates. If the level of the subject issuing the query does not dominate some security level in L, then that level is eliminated from L. The use of security variables permits the same query to be used in different levels of the security lattice without being rewritten.

The result of the SELECT operation is defined with respect to the database interpretation, as follows: The entire query including subqueries (but minus the BELIEVED BY clause) is evaluated separately at each level of the database interpretation in L, using ordinary SQL semantics, producing an answer at each level. The answers can be converted to syntactic form by appending a TC attribute to each answer tuple, and unioning the answers from all levels. A cascade of queries is also meaningful, and is defined in the obvious way.

To list the beliefs of all visible levels about the destination of starship entity *Enterprise U* of Figure 6, we use

SELECT	Destination
FROM	SOD
WHERE	Starship = "Enterprise"
	and KC = U
BELIEVED BY	ANYONE;

which returns $\{Vulcan, U; Romulus, C\}$ for a user classified C or higher. With an empty WHERE clause, this query would return all visible beliefs about destinations of all starships in SOD, which is $\{Mars, U; Vulcan, U; Romulus, C; Romulus, S\}$ for users classified S or higher.

The semantic definition of SELECT can be used to evaluate SELECT directly against a syntactic MLS database, by modifying the query before execution as follows: For each level l_1 through l_m in L, a new query expression is formed by conjoining the WHERE clause

in the original query (and each of its subqueries) with a clause making each new query specific to one semantic database. For level l_i having relations r_1 through r_n in its FROM clause, the new query is formed by conjoining its WHERE clauses with:

$$(r_1.tc = l_i \text{ and} \cdots \text{and } r_n.tc = l_i)$$

In this new query, the SELECT clause is augmented with $r_1.tc$, and the BELIEVED BY clause is removed. Finally, the union of the m new query expressions is formed. This resulting SQL query is evaluated against the syntactic MLS database in the standard way.

We prohibit BELIEVED BY clauses in subqueries, thus preventing joins between tuples from different levels. Cross-level joins can be useful as metaqueries, such as, "give me all eids for which there exist different beliefs at different levels." Although metaqueries are easy to define syntactically, we have not included them here because we do not have a good semantic definition for them.

5.2 Insert

The INSERT statement has the following general form:

INSERT
INTO $R\,[(A_i\ [,A_j]^*\ )\,]$
VALUES$(a_i\ [,a_j]^*)$;

INSERT places a new MLS entity into the database by inserting a base tuple in the database interpretation at the level of the requestor. The base level's belief in the existence of the new entity, and in its attribute values, are asserted in the inserted tuple. In the inserted tuple, attribute A_i is given the constant value a_i. KC is automatically set to the level of the subject; KC and TC cannot appear in the VALUES clause. The remaining key attributes must be assigned non-null values, else the request is rejected. Any omitted non-key attributes of R are assigned a null.

Due to Property 1, if a base tuple already exists with eid matching that of the inserted tuple, the insertion is rejected. This rejection does not open a covert channel, since the tuple causing the rejection is already visible to the issuer.

We set one strong restriction on INSERT: that users at a particular level not reuse their old entity keys for new entities. For example, if the *Enterprise U* is blown up, a new starship could be named *Enterprise-II U* but not just *Enterprise U*. This restriction guarantees that after the deletion of an entity at a lower level, higher levels can continue to believe in that same entity, with no possibility of confusion between the old entity and any new entities introduced later on at lower levels.

5.3 Update

The UPDATE statement has the following general form, in which the SET clause cannot contain TC, KC, or other key attributes.

UPDATE	R
SET	$(A_i = a_i\ [,A_j = a_j]^*\ )$
WHERE	P
[BELIEVED BY	L];

The semantics of UPDATE are a bit subtle. The WHERE and BELIEVED BY clauses select tuples of R in the database interpretation at any level specified in L, just as for SELECT. Due to the update access property, only tuples at the level of the issuer can be altered. Therefore, unlike the traditional UPDATE, the set of tuples selected is not always the same as those updated.

The response to this mismatch is motivated by MLS entities. The WHERE and BELIEVED BY clauses are used to describe attributes of a set of MLS entities in R. The set contains those entities which match the attribute values in the WHERE clause at any of the levels of L (duplicate entities are removed from the set). The SET clause alters a level's beliefs about these entities, setting attribute $A_i = a_i$, where a_i is an expression involving constants, relational attributes and appropriate operators (such as the arithmetic operators).

Imagine an extra *Speed* attribute in relation SOD. We can increase *Speed* for the *Voyager U* by 10% with the following update issued by a U subject:

UPDATE	SOD
SET	Speed = Speed * 1.1
WHERE	Starship = "Voyager" and KC = "U".

This query selects the *Voyager U* entity, and increases the speed given for it at the U level by 10%.

It may be the case that the updator's level l has expressed no beliefs yet about a particular entity qualified by the WHERE and BELIEVED BY clauses. In this case, a new tuple for that entity is formed and inserted in the level l database interpretation to contain l's beliefs expressed in the SET clause. (The value a_i must be a constant in this case.) The eid of the tuple is set to the entity's; any non-key values not given in SET are set to null.

As an example of a new tuple being formed as the result of UPDATE, consider relation SOD of Figure 6 again. Suppose an S subject issues the command

UPDATE	SOD
SET	Destination = "Earth"
WHERE	Destination = "Romulus"

BELIEVED BY ANYONE;

meaning 'Every starship anyone (whose beliefs I can see) believes is headed to Romulus, I now believe is headed to Earth'. Figure 8 shows the updated relation. The MLS entity *Enterprise U* satisfies the WHERE

Starship	KC	Objective	Destination	TC
Voyager	U	Shipping	Mars	U
Enterprise	U	Exploration	Vulcan	U
Enterprise	U	Diplomacy	Romulus	C
Enterprise	U	null	Earth	S
Zardor	S	Warfare	Earth	S

Figure 8: *SOD*: Update Rerouting to Earth

and BELIEVED BY clauses. Level S had not previously expressed any beliefs about *Enterprise U*, so the update results in a new tuple for *Enterprise U* at level S.

This semantics permits the scope of an update to be narrowed to a single entity by specifying a WHERE clause which qualifies only one eid. This is a semantic solution to the problem of tuple proliferation: when one eid is qualified in p, at most one tuple will be added to the database and to its interpretation. For example, if an S subject issues the command

```
UPDATE        SOD
SET           Destination = "Earth"
WHERE         Starship = "Enterprise"
BELIEVED BY   ANYONE;
```

against Figure 6, the result is the same as Figure 8, except that the Zardor tuple is not updated. The update causes the database to grow by only one additional tuple. Under the JS model (which uses attribute classification), the same update doubles the number of *Enterprise U* tuples, to four. This occurs because the effect of an update in the JS model is determined by considering each tuple separately, without consideration of MLS entities.

Note that UPDATE cannot be used to change an eid. That change can be accomplished by a combination of other DML operations, or by introduction of a new command to express that a higher-level entity is in fact the same entity as one introduced at a lower level.

5.4 Delete

The DELETE statement has the following form:

```
DELETE
FROM   R
WHERE  p
```

A DELETE request says that the issuer no longer believes in the existence of the entities qualified by p. Tuples are selected from the database interpretation at the level of the issuer, using the WHERE clause as in an ordinary MLS query. Then the selected tuples are removed from the database interpretation at that level. No BELIEVED BY clause is needed for DELETE, because a subject can only retract belief in entities that exist at the subject's own level. If a deleted tuple was a base tuple, higher level beliefs about that same entity will persist.

6 Conclusions

In this paper, we have presented syntax and semantics for an MLS relational database, with an unambiguous relation between the two. Our semantics is based on Kripke-like database interpretations, which express the beliefs held at different security levels about a set of shared entities. These *multilevel secure entities* expand on the traditional definition of an entity to permit expression of multiple levels of secure understanding about the value of an attribute. Syntactically, we present a simple representation of MLS relational databases which we show expresses the full meaning of our semantics without ambiguity (semantic or syntactic) and without the need for attribute level security. Other problems present in previous models, such as ambiguous null values and tuple proliferation under updates, are resolved in our model. We believe that our semantics is useful and general, and that it can be a useful retrofit for other models with existing implementations.

We define the select, insert, update, and delete operations for our model, deriving the effect of each operation from its effect on the semantic interpretation of the relations involved. Queries are unambiguously defined by clearly expressing the particular set of beliefs in the database interpretation the SELECT statement is to retrieve, using a BELIEVED BY clause. The particularly difficult UPDATE operation is defined through its effect on preexisting MLS entities.

We have recently generalized our model to include *MLS relationships*, enabling standard ER modeling techniques to be utilized in the design of a multilevel secure databases [Smi92]. Also, we have extended our model to allow a level to believe by default what a lower level believes, unless explicitly overridden, preventing the generation of potentially many tuples to state agreement with a lower belief.

As future work, we plan to develop the semantics of meta-queries, involving cross-level joins into our

model. Also, levels may wish to respond automatically to certain kinds of changes that take place at lower levels. Database-integrated production rules, such as those implemented within Starburst [WF90] and Postgres [SHP88], have recently been integrated into the MLS database framework [SW92]. We believe that the greater flexiblity offered by rules permits the expression of contextually dependent responsive policies which will solve such problems within the framework of mandatory security. We plan to develop a framework for extensible secure policies, which may be implemented by rules.

Acknowledgement
The authors wish to express their appreciation to Sushil Jajodia for his extensive comments, useful discussions, and encouragement in this work.

References

[BL74] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical report, MITRE Corporation, 1974.

[Che76] P. P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM TODS*, 1(1):9–36, January 1976.

[Dep85] Department of Defense, National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, 1985. DOD 5200.28-STD.

[DLS+87] Dorothy Denning, Teresa Lunt, Roger Schell, et al. A Multilevel Relational Data Model. In *Proceedings: 1987 IEEE Symposium on Security and Privacy*, pages 220–234, Oakland, CA, April 1987.

[GL90] Thomas D. Garvey and Teresa F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings: 6th Computer Security Applications Conference*, pages 148–159, Tuscon, December 1990.

[HOT91] J. Haigh, R. O'Brien, and D. Thomsen. The LDV Secure Relational DBMS Model. In S. Jajodia and C. Landwehr, editors, *Database Security, IV*, pages 265–279. North-Holland, Amsterdam, 1991.

[JS90] S. Jajodia and R. Sandhu. Polyinstantiation Integrity in Multilevel Relations. In *Proceedings: IEEE Symposium on Research in Security and Privacy*, pages 104–115, Oakland, CA, May 1990.

[JS91] S. Jajodia and R. Sandhu. Toward A Multilevel Secure Relational Data Model. In *Proceedings: ACM SIGMOD*, pages 50–59, Denver, Colorado, May 1991.

[KS91] Henry F. Korth and Abraham Silbershatz. *Database System Concepts.* McGraw-Hill, New York, 1991.

[Lam73] B. W. Lampson. A Note on the Confinement Problem. *CACM*, 16(10):613–615, October 1973.

[LM89] Teresa F. Lunt and Jonathan K. Millen. Secure Knowledge-Based Systems. Technical Report SRI-CSL-90-04, SRI International, August 1989.

[Lun90] Teresa Lunt. Multilevel Security for Object-Oriented Databases. In D.L. Spooner and C. Landwehr, editors, *Database Security, III*, pages 199–209. North-Holland, Amsterdam, 1990.

[Lun91] Teresa F. Lunt. Security in Database Systems: A Researcher's View. In *Proceedings: Second German Conference on Computer Security*, June 1991.

[SHP88] Michael Stonebraker, Eric N. Hanson, and Spyros Potamianos. The POSTGRES Rule Manager. *IEEE Transactions on Software Engineering*, 14(7):897–907, July 1988.

[Smi92] Ken Smith. *Managing Rules in Active Databases.* PhD thesis, University of Illinois at Urbana-Champaign, June 1992.

[SW92] Ken Smith and Marianne Winslett. Multilevel Secure Rules: Integrating the Multilevel and Active Data Models. Technical Report UIUCDCS-R-92-1732, University of Illinois at Urbana-Champaign, March 1992.

[WF90] Jennifer Widom and Sheldon J. Finkelstein. A Syntax and Semantics for Set-Oriented Production Rules in Relational Database Systems. In *Proceedings: ACM SIGMOD*, pages 259–270, Atlantic City, N. J., May 1990.