POLYINSTANTIATION AND INTEGRITY IN MULTILEVEL SECURITY: A Survey

Md. Rafiqul Islam, Harihodin bin Selamat and Mohd. Noor Md. Sap.

Faculty of Computer Science and Information system
University Technology Malaysia
Jalan Semarak, 54100 Kuala Lumpur, Malaysia, Tel: 03-2904957.

Abstract

Polyinstantiation is used to solve the data availability problem in a multilevel secure system. But to solve availability problem there arises another type of problem that belongs to integrity.
In multilevel system there are another type of integrity problems also. This paper presents the concepts of polyinstantiation and integrity in multilevel system. The advantages and disadvantages of polyinstantiation and various kinds of integrity problems are also presented.

Keywords: Polyinstantiation, integrity, multilevel, security.

1. Introduction

The term *security* is used in a database context to mean the protection of the database against unauthorized disclosure, alteration, or destruction [2]. Database security is the study of secure databases and secure database systems. As a subject matter, it has had increasing emphasis and focus of study in recent years.

The disclosure problem is the problem of protecting against unauthorized disclosure of data under the control of a database management system. The integrity problem is the problem of protecting the data from unauthorized modification. The database consistency problem is subsumed by the integrity problem insofar as any modification of the data in violation of consistency constraints is an unauthorized modification of the database. The combination of these two problems is the database security problem. A security policy ought to address both problems.

Many civilians, defense, and commercial application require a *multilevel database system* that supports data having different *access classes* (security markings) and users with different authorizations, or *clearances*. This paper represents a review article. In this paper we discuss about polyinstantiation and integrity in multilevel secure database system, that means the security policy that uses multilevel database system. The advantages and disadvantages of polyinstantiation and integrity problems in multilevel system are also described.

2. Multilevel Security

The concern for multilevel security arises when a computer system contains information with a variety of classifications and has some users who are not cleared for the highest classification of data contained in the system. Multilevel security model is developed by Bell and LaPadula [1]. This model introduces the concepts of *level* and *category*. Each

subject is assigned a *clearance level* and each object a *classification level.* A subject generally represents a process executing on behalf of a user and having the same clearance level as the user. The objects can be area of storage, program variables, files, I/O devices, users, or anything else that can hold information. A *Security level* represents by a pair (*A, C*), where *A* denotes classification level and C a set of categories. For the military environment there are four classification levels :

0 - *Unclassified*
1 - *Confidential*
2 - *Secret*
3 - *Top Secret*

Each subject and each object also has a set of categories such as *Atomic* and *Nuclear.* One security level is said to *dominate* another if and only if :

1. its classification or clearance level ≥ the other, and
2. its category set contains the other.

That means given classes (A, C) and (A′, C′), (A, C) ≤ (A′, C′) if and only if A ≤ A′ and C ⊆ C′. For example, transmissions from (2, {Atomic}) to (2, {Atomic, Nuclear}) or to (3, Atomic}) are permitted, but those from (2, {Atomic}) to (1, {Atomic}) or to (3, {Nuclear}) are not.

The DoD (Department of Defense) policies [6] restricting access to classified information to cleared personnel are called *mandatory security.* Mandatory security requires that classified data be protected not only from direct access by unauthorized users, but also from disclosure through indirect means, such as covert signaling channels. Covert channels are information channels that were not designed to be used for information flow but can nevertheless be exploited by malicious software to signal data to low users. Here for simplicity the terms "high" and "low" are used to refer to any two access classes when the second does not dominates the first. For example, a high process (i.e., a program instance having a high clearance because it is acting on behalf of a high user) may use read and write locks observable to a low process over time to encode high information(e.g., locked = 1, unlocked = 0). Mandatory security requires that no information can flow from high classes to low.

Other access controls may be imposed in addition to mandatory security; these enforce *discretionary security.* The concepts of discretionary security have long been introduced and frequently discussed in the security research and development literature. Discretionary access controls typically govern a richer set of access modes that are specific to the particular types or categories of information to those individuals with a *need-to-know* for the information. The permissible discretionary accesses can be specified and changed by the users of the systems. In contrast, mandatory access controls which govern the reading and writing of data by individuals based on their authorized security clearance level, can be changed only by a specifically authorized security officer and are altered relatively infrequently. The access controls commonly found in most database systems are examples of discretionary access controls. The mandatory security policy is generally used in multilevel security.

3. Polyinstantiation

Polyinstantiation refers to the simultaneous existence of multiple data objects with the same name, where the multiple instantiations are distinguished by their access classes [6]. Polyinstantiation is necessary in order to hide the actions of high subjects from low subjects, thereby preventing signaling channels.

Polyinstantiated tuples are tuples identified by a primary key and associated key class, so that the same multilevel relation may contain several tuple instances for a primary key value corresponding to different access classes. *Polyinstantiated elements* are elements identified by a primary key, key class, and element class (in addition to the attribute name), so that there may be multiple elements for an attribute that have different access classes, but are associated with the same (primary key, key class) pair.

A Polyinstantiated tuple arises whenever a subject inserts a tuple that has the same primary key value as an existing but invisible (more highly classified) tuple. The effect of the operation is to add a *second* tuple to the relation, whose primary key is distinguishable from the first by its access class. Although the polyinstantiation is invisible to this subject, subjects at the higher access class can see both tuples. For example, if an unclassified subject adds tuple for flight number 1125 to the multilevel relation whose unclassified instance is shown in Table 1 and whose secret instance is shown in Table 2, then the outcome, as seen by a secret subject, is shown in Table 3. The tables are taken from [8].

Table 1. Secret relation instance

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	Chicago	U	U
75	U	1400	U	Berlin	U	U
1125	S	1730	S	San Salvador	S	S

Table 2. Unclassified relation instance

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	Chicago	U	U
75	U	1400	U	Null	U	U

A Polyinstantiated element arises whenever a subject updates what appears to be a null element in a tuple, but which actually hides data with a higher access class. A polyinstantiated element can also arise when a high subject updates a low element, although such polyinstantiation can be avoided by returning an error message to the high subject.

For example, if an unclassified subject replaces the perceived null value for the destination for flight 75 in Table 1 with the value "pair" the outcome, as seen by a SECRET

subject, is as shown in Table 4. Unclassified subject will see the result shown in Table 5. Tables are taken form [8].

Table 3. A polyinstantiated tuple

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	Chicago	U	U
75	U	1400	U	Berlin	S	S
1125	S	1730	S	San Salvador	S	S
1125	U	1925	U	San Francisco	U	U

Table 4. A polyinstantiated element

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	Chicago	U	U
75	U	1400	U	Berlin	S	S
75	U	1400	U	Paris	U	U
1125	S	1730	S	San Salvador	S	S
1125	U	1925	U	San Francisco	U	S

Table 5. Unclassified instance

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	Chicago	U	U
75	U	1400	U	Paris	U	U
1125	U	1925	U	San Francisco	U	U

4. Integrity

The term integrity is used in database contexts with meaning of *accuracy, correctness* or *validity* [2]. The problem of integrity is the problem of ensuring that the data in the database is accurate - that is the problem of guarding the database against invalid updates. Invalid updates may be caused by errors in data entry, by mistakes on the part of the operator or the application programmer, by system failures, even by deliberate falsification.

As defined in [7] the database integrity has three properties. These are: *Consistency, Correctness* and *availability*.

Consistency

A database is *consistent* if, whenever two different methods exits of deriving a piece of information, a request for that information always yields the same response no matter what method is used.

Correctness

A database is *correct* if all data satisfy all known constraints.

Availability

A database is *available* if the data in it can be made available to any authorized user.

4.1 Basic Integrity Rules

In this subsection we present the basic criteria for database correctness as in [2].

I. *Key integrity*: Every tuple in a relation must have a unique key.

II. *Entity Integrity*: Every tuple must have a non-null key.

III. *Referential Integrity*: If an attribute in a relation is designed as a foreign key for another relation, then any tuple appearing in that relation either has a null value as its entry in that attribute, or there is a tuple in that other relation with that entry as the key.

IV. *Domain Integrity*: Any entry in a column must belong to the domain of elements which that column is specified to column.

4.2 Multilevel Entity Integrity

As mentioned above entity integrity states that no tuple in a relation can have null values for any of the primary key attributes. If this constraint is to be satisfied with respect to the data visible at each access class, then in any given tuple, all the elements forming the primary key must all have the same access class. Otherwise, a subject whose access class is lower than that of the highest key element would see null values for some of the elements forming the key. In addition, the access class for the primary key must be dominated by the access classes of all other elements in the tuple. If the primary key class were not dominated by the class of some element in the tuple, then that element could not be uniquely selected by a subject operating at the element's access class.

4.3 Multilevel Referential Integrity

Referential integrity states that every secondary key must reference a tuple that exists in some other relation where the key is primary. In a multilevel database, this means that a secondary key element cannot reference a tuple with a high or noncomparable access class because the referenced tuple would appear to be nonexistent at the access class of the reference. Multilevel referential integrity requires that if a foreign key is visible at a given access class, then a tuple containing the referenced primary key must also be visible at that access class, and that the class of the foreign key element must equal the class of the referenced primary key.

4.4 Polyinstantiation Integrity

Polyinstantiation integrity controls the effects of polyinstantiation by specifying that there must never be two tuples with the same primary key unless they represent polyinstantiated tuples or elements. SeaView's polyinstantiation integrity property has two parts. The first is a functional dependency condition and the second is a multivalued dependency condition. The property is stated as follows.

A multilevel relation instance $R(s, c)$ in state s at access class $c \geq$ class(R) (Where class(R) is the access class at which the relation R is defined) satisfies *polyinstantiation integrity* if and
only if

1. For each non-key attribute A_i, there is a functional dependency from the primary key (including the key class) and the classification attribute C_i to A_i:

   $K, C_k, C_i \rightarrow A_i$.

2. For each non-key attribute A_i, there is a multivalued dependency from the primary key (including their key class) to the A_i and C_i:

   $K, C_k \rightarrow\rightarrow A_i, C_i$.

Where K represents the primary key attribute(s) and C_k represents the classification attribute for the primary key.

5. Advantages and disadvantages of polyinstantiation

The simplest form of polyinstantiation, *tuple polyinstantiation* requires that an entire tuple be stored at one level and then allows tuples with the same key to exist at different levels. This approach has a number of advantages:

- The level at which the tuple is stored provides an easy way of extending the key. By considering the level as part of the key, the requirement that each tuple have a unique key is maintained by requiring that at any level each tuple have a unique key.

- At any level, the view that a user has of the database can easily construct and reflects the actual intent of the data as entered in the database. The view consists of all tuples that are stored at the level of the user or below.

- If writing up in level is not allowed, then it is easy to maintain the consistency of the data at a given level. A user is only allowed to modify tuples at the level of the user. For modification purposes the database can be viewed as a collection of single level databases. This prevents a lower level user from writing over data that was inserted at a higher level.
  The real disadvantages of the approach arise when a tuple contains data elements that are classified at different levels. These disadvantages include :

- A tuple that contains individual data elements that are classified at different levels cannot be stored across the levels. This implies that the entire tuple must be stored at

the higher level of the data elements in the tuple. The level is sometimes called the tuple class of the tuple.

- If lower level data elements that are part of the tuple are to be viewed at their own level, then a separate tuple that only contains the lower level data elements must be inserted at the lower level. This introduces two new problems. The first is how to guarantee that the higher level tuple and lower level tuple are updated consistently. The second problem is the more fundamental question of knowing that the higher level tuple and lower level tuple are to be associated so that an update to the lower level tuple results in an appropriate update to the higher level tuple.

One method of handing tuples that contain data elements that are classified at different levels is element polyinstantiation. In this approach, elements in a tuple can be stored separately at different levels. For each key and associated key level, there may then be several tuples associated with that key and level. These tuples are constructed by recombining elements stored at different levels that have the same key and key level. The primary advantage of the approach is:

- The data elements in a tuple can be stored across several levels. This means that separate tuple does not need to be created at the high water mark of the data elements in the tuple. Each data element is stored at the appropriate level. Lower level updates to the tuple are reflected in the entire tuple when it is reconstructed and viewed at the higher level.

Therefore the added granularity of this approach provides a method of handing tuples that are not uniformly classified. However, it introduces some problems that are not present in tuple polyinstantiation. The main of these is:

- How can the tuples that are stored across several levels be reconstructed in a manner that is consistent with the intent of the database users who entered the tuples? Different methods have proposed for handing this problem. One approach is to join all of the possible data elements having the same key and key level. The result is that a number of tuples may be constructed and presented in the user's view that donot reflect tuples as intended by the user.

5. Integrity problem

Now we discuss the integrity problems that arise in multilevel secure system. If a user at a lower level attempts to enter a tuple into the databases, and a tuple with the same key already exists at the higher level, then either:

- The insert at the lower level is rejected and the lower level user receives the information that a higher level tuple with the same key exists, or
- The insert at the lower level is accepted and there are now two tuples in the database with the same key. Since the first action leads to a downward flow of information, it is not acceptable in a multilevel secure system. As we know, the second solution, called polyinstantiation. However, polyinstantiation at different security levels does not remove all ambiguity. For example, suppose that a high level user asks to view a

tuple instantiated at both a high and a low security class. Which tuple should be returned? If the low tuple is the most recent, it may contain the most accurate information. However, the creator of the high tuple may have had information not available to the creator of the low tuple. Moreover, both situation may occur in the same relation, and even in the same attribute or tuple. Consider the following instance as in [7]:

Table 6. Secret relation instance

Label	Name	Destination	Engine
U	wombat	Norfolk	diesel
S	wombat	Persian Gulf	nuclear

It is clear that the entry "diesel" in the unclassified tuple is merely a cover entry for the real, classified, entry "nuclear" in the secret tuple. However, the entry "Norfolk" in the unclassified tuple could either be a cover entry for the entry "Persian Gulf" in the secret tuple, or it could represent the fact that the unclassified tuple has been updated more recently.

If data are classified at the column or element level, problems can arise if the security class of the key is higher than or incomparable with the security classes of other elements in the relation. Thus it is necessary to require that the security class of any field in the key be dominated by the security classes of all data in the relation. Depending on the way relations are defined, this approach may reduce data availability. For instance, consider the following example, Let R be the relation $ABCD$ with key AB. Suppose that A and D are highly sensitive, that B and C are not, and that C depends only upon B. In order to maintain entity integrity, C must be classified at a level as least as high as A. This problem can be avoided by breaking R into two relations, ABC and BC. But here the data redundancy is occurred.

In the case of referential integrity, database integrity can be violated when any of the four approaches to data classification is taken. For example, suppose that data are classified at the relational level, and that a foreign key in a relation with a low security class refers to a tuple in a relation with a higher security class. To the user with a lower clearance, the key would appear to be dangling. Similar problems can occur when data are classified at the tuple, column, or element level.

The method of avoiding dangling keys that is recommended as in [10] is to require that, if an element A appearing in relation R is designated as a foreign key for relation R', then the security class of A in R must dominate the security class of A in R'. However, care must be taken as to the method by which referential integrity is enforced, since it is possible to introduce a covert signaling channel by using the wrong method. Suppose that an element A appearing in tuple T relation R is designated as a foreign key for R', that T' is the tuple containing A in R'.

There are two ways of enforcing referential integrity when a user attempts to delete T'. One way is to delete A from T automatically. The other is to prevent the user from deleting T' without at first deleting A from T. If T is classified at a higher level than T', the second method opens a covert channel, since, by repeatedly removing and inserting a tuple T, a secret process could signal information to an unclassified process that repeatedly attempts to remove and insert T'. The first method of automatically deleting A from T when T' is deleted does not open any covert channel, however, since only data at the higher security level is effected when this approach is taken.

6. Conclusion

In this paper the fundamental concepts of multilevel secure system, polyinstantiation and integrity in the system are given. As a paper it may serve as a tutorial of the state of the art with respect to the above mentioned subjects. This article will be helpful to the researchers who are interested in multilevel security, specially polyinstantiation and integrity in the multilevel secure system.

Reference

1. E. B. Fernandez, R.C. Summers, C. Wood; *Database Security and Integrity,* Addison-Wesley, Reading, MA, 1981.
2. C. J. Date; *An introduction to Database Systems,* Volume II, Addison-Wesley, Reading, MA, 1985.
3. D. E. Denning; *Cryptography and Data Security,* Addison-Wesley, Reading, Ma, 1983.
4. T. C. Ting; *A user-role based data security approach,* in Database Security: Status and Prospects, Editor Landwehr, North-Holland, 1988.
5. D. E. Denning and et. al.; *Views for Multilevel Database Security,* Reading, IEEE transactions on software engineering, Vol. SE-13, No.2, February 1987.
6. T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman and W. Schockley; *The SeaView Security Model,* IEEE transactions on software engineering, Vol.16 No.6, June 1990.
7. C. Meadows, S. Jajodia; *Integrity Versus Security in Multilevel Secure Databases,* In Database Security : Status and Prospect, Editor, C. E. Landwehr, North-H-land, 1991.
8. T. F. Lunt and D. Hsieh; *Update Semantics for a Multilevel relational Database System,* in Database Security, Editors, S. Jajodia and C. E. Landwehr, North-Holland, 1991.
9. J. T. Haigh, R. C. O'Brein, and D. J. Thomson, *The LDV Secure Relational DBMS Mode,* in Database Security, Editors, S. Jajodia and C. E. Landwehr, North-Holland, 1991.
10. J. T. Haigh; *Modeling Database Security Requirements,* In Database Security: Status and prospects, Editor, Landwehr, North-Holland, 1988.