NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR

Security In Object-oriented DBMS

Pratik Pandey 19111041
Database Management System
6th Semester Biomedical Department

Guided by :
Saurabh Gupta Sir

Abstract
The paper discuss a traditional essential security kernel that can be used to construct a hierarchical object-oriented system. A single security level is assigned to each object, which applies to all of its contents (variables and methods). Compatibility of security level assignments with the class hierarchy is one of the features of the informal security policy model. The illustration depicts the representation of integrity and classification limitations.

TABLE OF CONTENT

1 Overview of Object-oriented paradigm

A database system that can cope with data objects, such as those used in object-oriented programming languages, is known as an object-oriented database (OOD).
Everything is an object in object-oriented programming, and many objects are highly complicated, with many different characteristics and actions. An object-oriented database management system collaborates with an object-oriented programming language to make object-oriented data storage and retrieval easier.

- Object: Objects are the first things that come to mind while building a programme in object-oriented programming (OOP), and they are also the units of code that emerge from the process. Each object is an instance of a certain class or subclass, with its own set of methods and data fields.
- Hierarchy: A hierarchy is a directed rooted tree with the root at the top, as is customary. The instance relation is the relationship between an object and the things immediately below it. The class relation is the inverse relationship, which goes upward. A class object (also known as a class) is simply an object that is a class for

some other object, i.e. it has objects beneath it. In most cases, a class object represents organisational entities at a higher conceptual level than logical records. For example, if the database contains employee records, there may be a class object called "Employee" that has the common properties of all workers and whose instances are individual employee records.

- Inheritance: A hierarchy is a directed rooted tree with the root at the top, as is customary. The instance relation is the relationship between an object and the things immediately below it. The class relation is the inverse relationship, which goes upward. A class object (also known as a class) is simply an object that is a class for some other object, i.e. it has objects beneath it. In most cases, a class object represents organisational entities at a higher conceptual level than logical records. For example, if the database contains employee records, there may be a class object called "Employee" that has the common properties of all workers and whose instances are individual employee records.

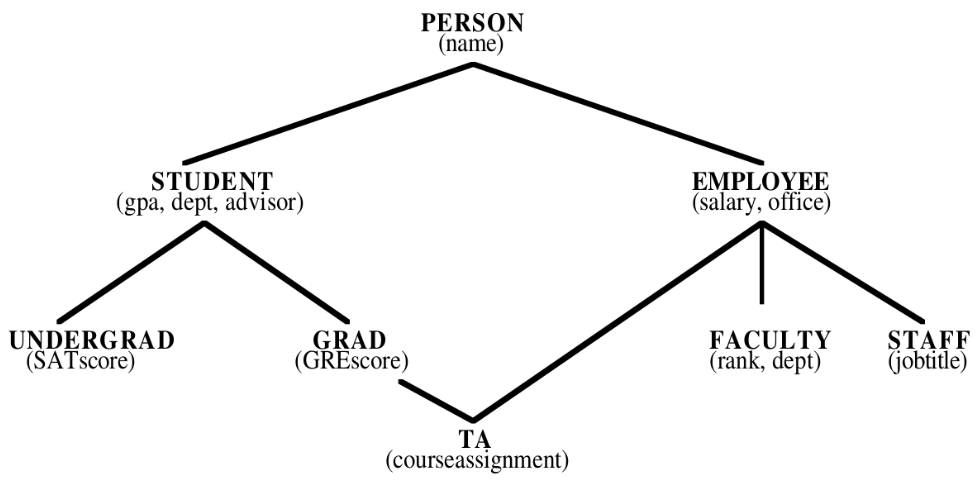

Figure 1: Inheritance Hierarchy Modeling University Staff

It uses generic axioms rather than specific transition rules to explain and constrain the behaviour of the application interface to an object-oriented system. The fact that each object has a single security level that applies to everything within it is a key feature of our approach. The different functions are bound by the property and other restrictions enforced by any necessary security kernel in order to construct an object-oriented interface layer. In the context of the entities to which these derived restrictions apply, six security attributes are presented below.

- **Property 1**: This property is known as hierarchial property. This states that the level of an object must dominate its class objects. This attribute is required for the object to inherit its parent's methods and variables. Any effort to read or execute an object by reading one of its variables may automatically read the object's parent class, and maybe the parent's parent class, and so on, until a default value or acceptable function is discovered.
- **Property 2**:Subject level property. The security level of subject dominates the level of invoking subject and its also dominates the home subject. This property is required for the subject that was formed to handle the message to be able to read the message as well as variables and methods in the object where it is situated. It is sufficient for the subject to be formed at a level equal to the invoker and object levels' lowest upper bound.

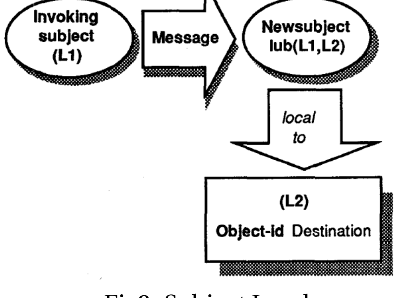

Fig2: Subject Level

- **Property 3**:Object Locality Property. A subject can execute method or read or write method only in its home class.
- **Property 4**:A subject may write into its home object only if its security level is equal to that of the object. The combination of the Object Locality and Subject Level Properties prevents writing and reading up.
- **Property 5**: Return value property. A subject can return value to its invoking subject only if it is at the same security level. The invoking subject will have to get something instead of a return value in the implementation, such as a standard null value or a system confirmation that a value from the addressee is unavailable.
- **Property 6**: Object creation property. The security level of newly created object dominates the level of subject that request creation.

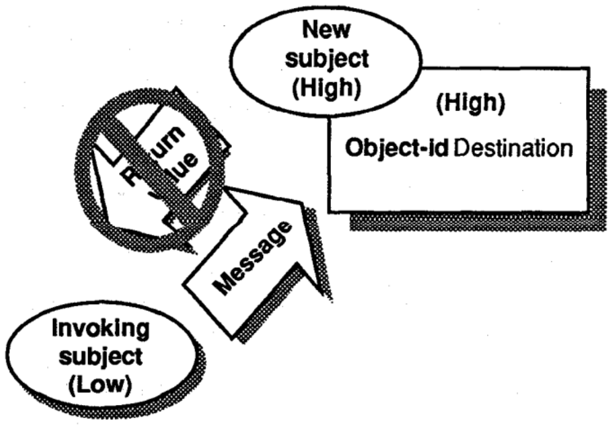

Fig3: Higher subject level do not return value

3 Object System Structure

3.1 Layered Object System

We think that an object system built in accordance with Section 2 Security policy and the architecture outlined below can be used as an untrusted layer over the kernel. The user application interface to the object layer is made up of system calls and system methods, which are

separated into two categories.
The primitive operations will be explained in the following sections. The primitives will do their own security cheeking in accordance with the policy given in Section 2, but the underlying kernel will perform its own independent enforcement, thus the primitives are not required to be trusted for this purpose. Their job is to keep the object system abstract and keep the object system user safe from kernel error messages.

3.2 Object system primitives

Begin by picturing a "empty" object system, one that is just comprised of system software. This is an object system "shell" that may be used as a starting point for object-oriented applications. To implement database management functionality, another layer would be required on top.

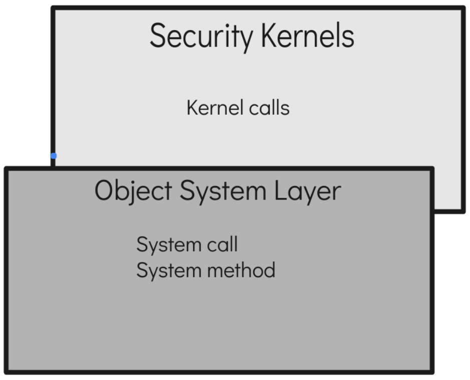

Fig 3: A layered secure object system

3.2.1 Root Object

The empty system isn't completely empty; it requires at least one object, the universal class object, sometimes known as the root object. This object has a subject, the root subject, executing on it. The capacity

to deliver messages is a basic primitive activity required by the root subject.
We propose a system call. Send an object-id and a message with two parameters. A Send call creates a new topic in the addressed object and assigns it to work on the message. We can't merely consider "Deliver" as a message since we'd need a mechanism to send it, which would lead to an infinite loop. As a result, Send has to be a system call. Return, which is used to terminate a method, and a command to upgrade a subject's sensitivity level (discovered to be useful during prototyping trials), both need to be system calls for similar reasons.
Five messages have built-in methods in the root object. The Create message creates an instance of a class object and contains the names of any additional variables desired in the instance as arguments. AddMessage is a method that adds to an object by specifying the command and parameters of a new or changed object. Delete deletes the instances. For reading and writing variables, we provide built-in methods for the messages Getvar and Setvar.

System calls	**Arguments**
Send	Object-id, message
Upgrade	Level
Return	Value

Table 1: Table to show system calls.

System Messages	**Arguments**
Create	level, new-variable
Delete	object-id
AddMessage	command, args, methods
Getvar	variable
Setvar	variable, value

Table 2: Table to show system messages.

```
Method system calls can be written as:

    <system call> <argument>,<argument>
    Example: Send Self.employee_id, "abort!!!"

Message call are written using syntax:

    <command> (<argument>,<argument>)
    Example: Delete (Emp_id)

Hence to use system call send to send message, we use:

    Send Self.employee, Getvar(Class)
    Return (value=employee)

A message returns a value by storing it in the "value" variable,
which is a special reserved variable. The following syntax can
be used to access variables in the local object:

    Self.Var=Self.Var2
```

4 SEMANTICS OF SECURITY

In this part, we'll look at how the single-level-object technique may be used to solve multilevel database demands efficiently and correctly. Understanding how categorization labels are applied is the first step. It's critical to clarify exactly what it means to classify something. This includes defining exactly what information or associations an object's categorization label is meant to secure.
Three dimensions of classification is as follow:

- the data itself can be classified
- the existence of data may be classified
- the reason, for classifying data may be classified

4.1 CLASSIFYING DATA

Some key facts are more correctly categorised at the higher, aggregate classification for many apparent aggregation concerns, as is generally the case when the aggregate indicates a relationship among multiple entities. Such issues are easily overcome by categorising the relationships between entities separately and at a level strictly higher than the entities themselves.
To begin, imagine that employee names and salaries are both unclassified, but that the link between a certain wage and a specific worker is classified as secret. Keeping the pay in distinct objects at the secret level would not only safeguard the association, but it would also classify the salaries. Take a look at Figure 5 for an example of a data design. This approach categorises the association, but leaves the set of wages unclassified for statistical purposes.

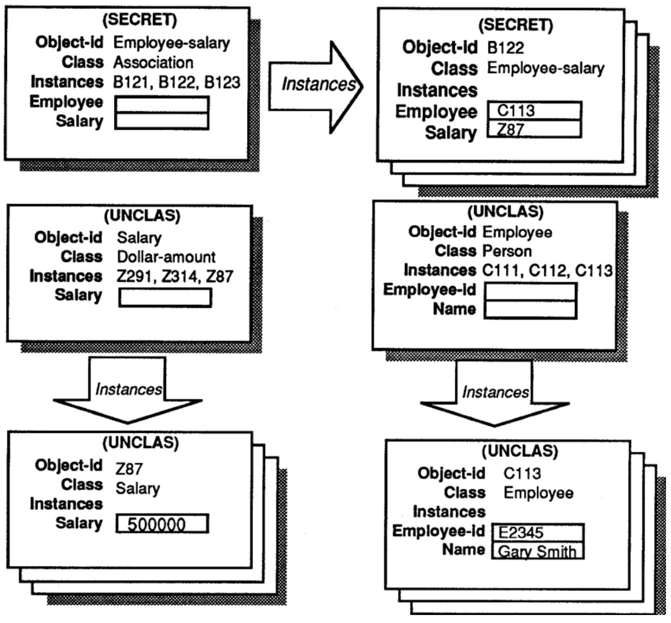

Fig5: Classification of Data

4.2 CLASSIFYING OBJECT EXISTENCE

If an object was made by a high subject, it should be kept hidden from low subjects. Otherwise, a high subject might indicate a lower one by generating and removing new items, which is a covert channek. The appeamnce of an object's object-id in the value of a variable in another object can be used to infer its existence. As a result, we safeguard an object's existence by guaranteeing that its object-id is not kept in a lower-level object.

4.3 Object-Ids and polyinstantiation

Polyinstantiation does not occur in our model since we utilise globally unique object-ids instead of user-defined object names to identify objects. It is, however, difficult to ensure object-id uniqueness. The untrustworthy object-layer process will not be aware of object-ids established by higher-level subjects when a low-level subject generates an object.

One option is to employ a pseudo-random number generator to ensure that object-ids are not duplicated. To produce more meaningful object-ids, a user might give a freely selected mnemonic prefix to the pseudo-random suffix, and unclassified objects could, by convention, dispense with the pseudo-random suffix.

5 Conclusion

A multilevel database system can be implemented using an object-oriented approach. An object system layer may be superimposed on top of a traditional obligatory security kernel, and its characteristics can be used to establish a data base and the related computing equipment. The object system layer can be untrustworthy to the kernel if each object has a single security level.

The object system layer provides a user interface that consists of numerous primitive actions, such as system calls and system messages, that methods can employ. As long as the user cooperates, these primitives will respect and enforce a set of security characteristics that will safeguard the user against kernel violations. When a user application escapes the object layer, it loses access to object-oriented services while remaining subject to the kernel's security constraints.

6 References

1. Security for object-oriented database, Researchgate, by Jonathen Milen and Teresa Lunt
2. Gajnak, G. E., "Some Results From The Entity/Relationship Multilevel Secure DBMS
3. Kim, W. and F. H. Lochovsky, Object- Oriented Concepts, Databases, and Applications, Reading, MA Addison-Wesley
4. MongoDB.com