Security In An Object-Oriented Database

James M. Slack
Computer and Information Sciences Department
Mankato State University
Mankato, Minnesota 56002

Abstract

Much of the research in security for object-oriented databases follows the traditional lines of discretionary access control, mandatory access control, and multilevel secure database systems. In this position paper, our premise is that security and integrity can be implemented in the object-oriented database model. We propose extensions to the basic data model to incorporate security and integrity. Our secrecy/integrity mechanism is based on the idea that objects are partitioned into *protected groups*. An object in a protected group is restricted in the messages it can accept and send.

1 Introduction

Much of the research in security for object-oriented databases follows the traditional lines of discretionary access control, mandatory access control, and multilevel secure database systems. Our premise is that security and integrity can be implemented in the object-oriented database model. We propose extensions to the basic data model to incorporate security and integrity.

In [13], we proposed the notion of a *one-way protected group*. A one-way protected group is a set of *one-way protected objects*, where each one-way protected object in the group will accept messages only from a distinguished object in the group called the *interface object*. We showed that a one-way protected group can support data integrity and access integrity as well as access rights based on need-to-know.

We propose a *two-way protected group* as an extension of a one-way protected group. The additional restriction in a two-way protected group is that each object in the group can only send messages to the interface object of that group. Thus, each implementation object may only communicate with its interface object. We assume each object (including all methods and attributes) will be assigned a single security classification. Millen and Lunt have shown this assumption is flexible enough to allow the enforcement of realistic security policies [11]. The model does not depend on an underlying operating system security kernel, therefore, it assumes a trusted object-oriented database system. The database system could be divided into trusted and untrusted portions, where the trusted portion implements the protected group mechanism. Alternatively, the system could be implemented over a trusted kernel in a conventional approach (e.g., [10]).

2 Object-Oriented Database Model

We assume an object-oriented database model based on [4, 8, 12]:

1. *Object and object identifier.* Entities in the real world are modeled as objects in the database. The system assigns each object a unique object identifier.

2. *Attributes and methods.* Each object encapsulates a state and a set of behaviors. The state of an object is represented by a set of attribute values. Each attribute value may be a value from a primitive class (e.g., real, integer, string, etc.), an object identifier, or a collection. (A *collection* can be a set or list of object identifiers.) The behavior of an object is defined by a set of methods. The methods of an object are externally visible; attributes are not. Therefore, the only way to access or manipulate an attribute in an object is to invoke one of the object's methods.

3. *Messages.* To invoke a method in an object, a message must be sent to the object requesting invocation of the method. A message is an object; specifically, each message is an instance of class *message*.

4. *Classes and instances.* A class groups a collection of objects which have the same set of methods and attributes, but which may differ in the values of those attributes. Each object in such a collection is an instance of the class.

5. *Class hierarchy and inheritance.* Each class may inherit the methods and attributes of other classes. The resulting structure is restricted to be a directed acyclic graph.

We define a *class object* as 4-tuple consisting of an identifier for the class, a set of attribute-class pairs, a set of methods, and a set of class object identifiers from which this class inherits additional methods and attributes. We define an *instance object* as a 3-tuple consisting of an identifier for the instance, a set of attribute-value pairs, and a class object identifier. Figure 1 shows an example of the class *teachingAssistant* and the instance *joe*.

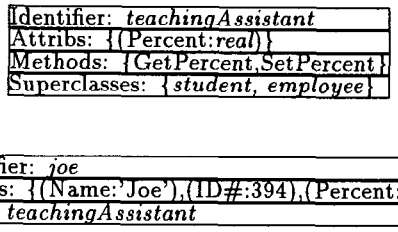

Figure 1: The class object *teachingAssistant* and the instance object *joe*.

A *subject* is an instance object which has the ability to start a message spontaneously. Each user is represented by a subject in the database. There may be other subjects in addition to users, e.g., *triggers* [6].

A message is an instance of the class *message* where the attributes of a message include:

- FromObj: the object identifier of the source of the message,
- ToObj: the object identifier of the target of the message,
- ACI: access control information for the subject which started the *message chain*, i.e., the sequence of messages which resulted in this message being sent from FromObj,
- MethodName: the name of the method in ToObj to be invoked,
- Parameters: a list of parameters to send to MethodName.

The methods in class *message* include SendMessage, ReceiveMessage, SetACI, and GetACI. The attribute ACI contains access control information for the originating subject. The particular access control information in use depends on the security and integrity model, e.g., the user identifier for discretionary access control or Clark-Wilson integrity, security level for mandatory access control. Methods SetACI and GetACI are privileged operations; only certain objects that are registered with the system may invoke them. A spontaneous message (i.e., sent by a subject) does not contain access control information. ACI is set to *null* in this case. Any other message contains the access control information of the message which invoked the associated method.

A *return message* is a message which contains a return value (i.e., an object identifier) as a parameter. The parameter list may also contain other values which identify the message as a return message. It is up to the method language to distinguish return messages from other messages, based on the parameter list.

3 Protected Groups

Our secrecy/integrity mechanism is based on the idea that objects are partitioned into *protected groups*. A one-way protected group is a set of *one-way protected objects*, where each one-way protected object in the group will accept messages only from a distinguished object in the group called the *interface object*. In a *two-way protected group*, each object in the group can only send messages to the interface object of that group. Thus, each implementation object may only communicate with its interface object.

3.1 One-way Protected Groups

Each one-way protected group has one or more *interface objects* which accept messages from any source. All other objects in the group are *implementation objects*. An implementation object is hidden from external view and only accepts messages from an interface object of the same group.

Each object is augmented with the interface object identifier from which that object will accept messages. If this field is *null*, then the object will accept messages from any source. This additional information is inherited by subclasses and instances. The approach assumes that the system guarantees the integrity and secrecy of messages.

3.2 Two-way Protected Groups

A *two-way protected group* is a one-way protected group in which each object may only communicate with an interface object of the same group. An interface object of one group may be an implementation object in another group. A one-way protected object restricts incoming messages; a two-way protected object restricts incoming and outgoing messages.

Each object is augmented with the set of identifiers of objects with which it can communicate. This set of identifiers is the *communications set* of the object. If the communications set is *null*, then the object can communicate with any object. Each two-way protected group consists of one or more interface objects and a set of implementation objects. Each implementation object is allowed to communicate only with the group's interface object(s). Each interface object is able to communicate with any implementation object in the group.

The interface object can communicate with any object in the group if its communications set includes an identifier for each implementation object, or if its communications set is *null*.

In either a one-way or two-way protected group, an implementation object will only accept messages from its interface object. A two-way protected group is stronger than a one-way protected group: an implementation object must send all messages to its interface object; there is no such restriction in a one-way protected group.

Note that an object can be an interface object of one group and an implementation object of another. This allows the construction of a lattice of protected groups. Such a lattice could be the basis for a mandatory access control mechanism.

Figure 2 shows a conceptual example of a pair two-way protected groups. The arrows represent messages

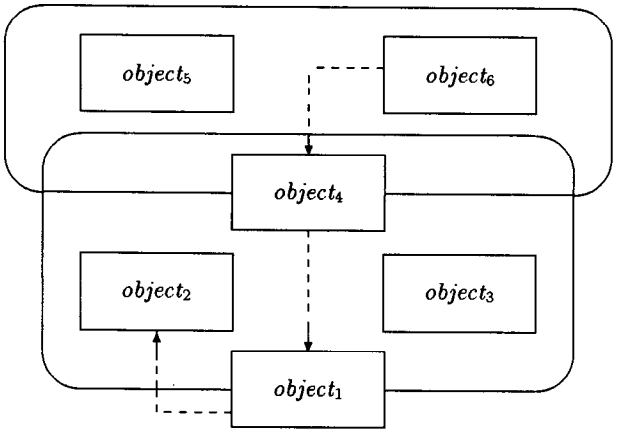

Figure 2: A conceptual view of a pair of two-way protected groups.

sent between objects. In this example, $object_1$ is an interface object for the bottom group, and $object_4$ is an interface object for the top group. Notice that $object_4$ is also an implementation object in the bottom group. A message sent from an object in one two-way protected group to an object in the other two-way protected group must pass through the interface objects of both groups.

4 Secrecy/Integrity Mechanism

A combined secrecy/integrity mechanism can be constructed based on the notion of protected groups. We assume that the system identifies and authenticates subjects, that the system can hide the existence of any object (e.g., classes, instances, subjects, messages) from any other object, and that a method can return a value that is indistinguishable from the "object not found" return value from the system.

The secrecy and integrity of a protected group of objects is based in the interface object for that group. The interface object is the only object in the group which is allowed to invoke the methods GetACI and SetACI in method *message*, i.e., it is registered with the system for this privilege.

When the interface object receives a message from some other object, that message either contains the access control information of the originating subject, or it contains no access control information, but does contain the object identifier of the source of the message.

In the first case, the access control information is known. In the second case, the interface object can obtain the access control information based on the object identifier of the source of the message. The interface object can then set all further messages in this message chain to contain the access control information of the originating subject. If the source of the message is not a valid subject, the interface object can reject the message.

The interface object also has access to the access control information of the target object of the message. This access control information may be stored in the object as additional attributes or in a separate object within the protected group. Using the access control information of the subject and the target object, the interface object can use the following general outline for each of its methods:

```
METHOD MethodName (Target, OtherParameters)
BEGIN
   IF GetACI is_null THEN
      SetACI (access control information of source object)
   ENDIF
   IF GetACI compares_favorably_with Target.ACI THEN
      Invoke Target.MethodName (OtherParameters)
   ELSE
      RETURN ('Object not found')
   ENDIF
END
```

In this approach, secrecy is a precondition that must be satisfied before access is allowed. The implementation of the comparison operator *compares_favorably_with* depends on the secrecy mechanism and the type of access control information.

The implementation of traditional discretionary access control is straightforward in this setting. Only one-way protected groups are required. A class *auth* is included in the protected group. This class is responsible for checking whether a subject s_i is authorized to invoke a method m. On receipt of a message, the interface object uses *auth* to determine whether the source of the message is authorized to invoke the given method. Both grant/give-grant and cascading revocations can be implemented by incorporating more information into *auth* instances.

Clark-Wilson [5] integrity can be enforced with the one-way protected group approach in the following way. Methods in the interface object are the TPs and objects in the protected group are the CDIs. Access triples are stored in class *auth*. This is admittedly simplistic; more work needs to be done to further develop this approach. Another way to enforce Clark-Wilson style integrity is the Generalized Framework for Access Control;[3, 1, 2, 9] this can be applied to the object-oriented data model based on protected groups.

Mandatory access control can be enforced in the following way. Let $L(g)$ be the security level of g. Form a two-way protected group for each security level. For any pair of two-way protected groups g_1, g_2 such that $L(g_1) > L(g_2)$, create an interface object I for g_1 and make I an implementation object in g_2. Object I may be protected by discretionary access control using the interface object in group g_2. Figure 3*(b)* shows how this approach works for the security lattice in Figure 3*(a)*. Each small square represents an interface object. A message may be sent from $(U, \{\})$ to $(C, \{\text{Spy}\})$, but a message may not be sent directly from $(C, \{\})$ to $(U, \{\text{Spy}\})$ or vice versa.

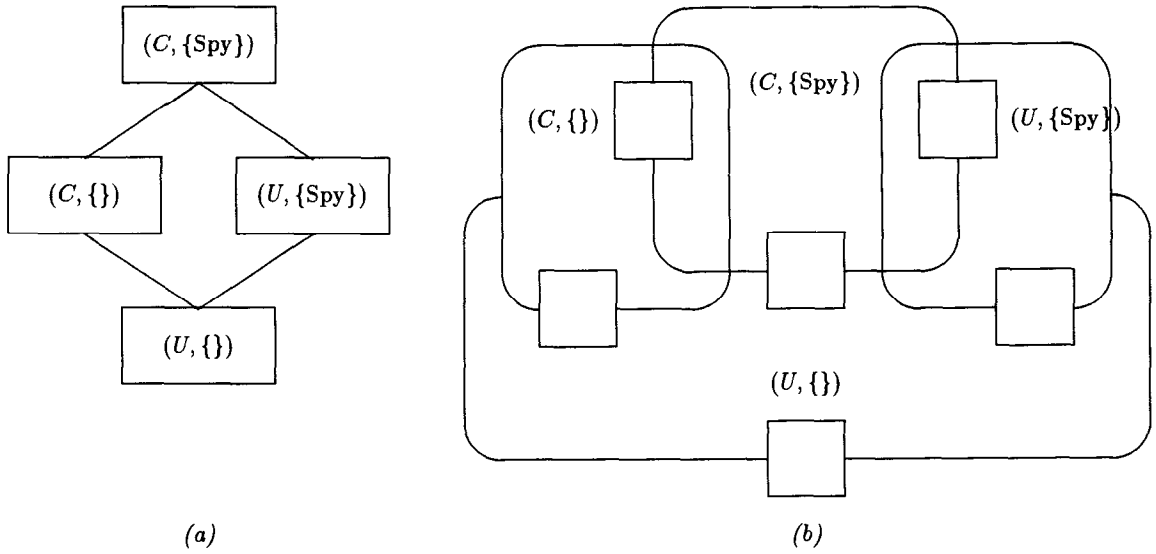

Figure 3: Mandatory access control using two-way protected groups. *(a)* A sensitivity level lattice. *(b)* Implementation using two-way protected groups.

This approach does not prohibit "write down" or "read up", although this can be prevented in any given implementation. It is advantageous to allow certain "write down" and "read up" operations, depending on the subject and the method invoked. One common application is the degrading of classified material. Unlike the traditional Bell-LaPadula approach, there is no need to step outside the model for these operations. (This is similar to the approach taken by Abrams, et al. in [1].)

In an object-oriented database model, access integrity is simply a special case of secrecy where the controlled method is state-changing. With access integrity, the subject's access control information must compare favorably with the target object's access control information before the method is invoked.

5 Future Work

In this paper, we have shown one way security and integrity can be implemented in the object-oriented database model. We proposed extensions to the basic data model to incorporate security and integrity. These extensions partition objects partitioned into protected groups. An object in a protected group is restricted in the messages it can accept and send.

Additional work needs to be done to make this approach viable. In particular, the trusted computing base must be specified. This is important because assurance is spread out rather than isolated within a kernel. An anonymous reviewer stated the opinion that interface objects must be part of the Reference Validation Mechanism, so the flexibility in setting the ACI is removed. This is certainly true with the current certification and validation mechanism. However, we take the view that this mechanism is inflexible and should be modified. Hosmer's multipolicy model makes a good case for a new certification mechanism [7].

We also need to show in more detail how security and integrity policies can be implemented with protected groups. For example, enforcement of Clark-Wilson integrity and the Generalized Framework for Access Control were sketched out in this paper; they need fleshing out.

One of the anonymous reviewers mentioned the "gazillion problem:" that the number of security classifications can become extremely large. In our approach, a protected group is necessary for each classification. We must deal with this potential proliferation of protected groups somehow.

Acknowledgments

This work was partially supported by Department of Defense grant #5-30296 and a Faculty Research Grant from the Graduate School at Mankato State University. The author wishes to thank John Campbell and Howard Stainer for their support and encouragement. This does not necessarily reflect the views of these people.

References

[1] Marshall D. Abrams, Kenneth W. Eggers, Leonard J. LaPadula, and Ingred M. Olson. A generalized framework for access control: An informal description. In *13th National Computer Security Conference*, October 1990.

[2] Marshall D. Abrams, Jody Heaney, Osborne King, Leonard J. LaPadula, Manette Lazear, and Ingred M. Olson. Generalized framework for access control: Towards prototyping the ORGCON

policy. In *14th National Computer Security Conference*, October 1991.

[3] M.D. Abrams, A.B. Jeng, and I.M. Olson. Unified access control: An informal description. Technical Report MTR-89W00230, MITRE Corporation, September 1989.

[4] R.G.G. Catell. Next generation database systems: Introduction. *Communications of the ACM*, 34(10):30–33, October 1991.

[5] D.D. Clark and D.R. Wilson. A comparison of commercial and military/computer security policies. In *IEEE Proceedings of 1987 Symposium on Security and Privacy*, April 1987.

[6] K. P. Eswaran. Specifications, implementations, and interactions of a trigger subsystem in an integrated database system. Technical Report Research Report RJ1820, IBM, San Jose, California, August 1986.

[7] Hillary H. Hosmer. The multipolicy paradigm. In *Proceedings of the 15th National Computer Security Conference*, pages 409–422. NIST/NCSC, October 1991.

[8] Won Kim. Object-oriented databases: Definition and research directions. *Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[9] Leonard J. LaPadula. Formal modeling in a generalized framework for access control. In *The Computer Security Foundations Workshop, III*, June 1990.

[10] Teresa Lunt, Dorothy Denning, Roger Schell, and William Shockley. The SeaView formal security policy model. Technical Report SRI Project 1143, A007: Final Report, Volume 2, SRI International, Menlo Park, CA, February 1989.

[11] Jonathan K. Millen and Teresa F. Lunt. Security for object-oriented database systems. In *Symposium on Research in Security and Privacy*. IEEE, May 1992.

[12] James M. Slack and Elizabeth A. Unger. A formal model of object structure and inheritance for object-oriented database systems. In *Proceedings of Great Lakes Computer Science Conference*, Kalamazoo, Michigan, October 1991.

[13] James M. Slack and Elizabeth A. Unger. Protected groups: An approach to integrity and secrecy in an object-oriented database. In *15th National Computer Security Conference*, Baltimore, October 1992. NIST/NCSC.