27

A Discretionary Security Model for Object-oriented Environments

Wilna Jansen van Rensburg and Martin S Olivier
Department of Computer Science, Rand Afrikaans University
PO Box 524, Auckland Park, Johannesburg, 2006 South Africa
Email: molivier@rkw.rau.ac.za

Abstract

This paper describes a discretionary security model, DISMOO. The model is designed for object-oriented environments. The model itself is therefore based on object-oriented concepts. The purpose of DISMOO is to provide a security model that provides a fine grain of protection, as well as a more advanced, enhanced and flexible discretionary security model. The security model is driven by capabilities, which is an adaptation of the traditional capability concept, used in access control mechanisms. The capability in DISMOO has much more flexibility and functionality. The model will enable all users of the system to protect their entities according to their own discretion.

Keywords: Security models, discretionary security, object-orientation

1 INTRODUCTION

This paper focuses on the use of discretionary security in object-oriented environments by introducing a discretionary security model built on an object-oriented base. DISMOO (Discretionary security model for object-oriented environments) is a capability-based discretionary security model that aims to provide a very flexible security system that can be used by all the users of the environment to its full.

The new model will be described after a brief overview of discretionary security has been given.

2 DISCRETIONARY SECURITY

Firstly the term discretionary security needs to be described in more detail. The meaning of discretionary security can be summarised by the following three definitions (Longley, Shain and Caelli, 1987):

Discretionary security: *In computer security, discretionary security is measures that are initiated by the entities themselves. It is also those aspects of a security policy that*

S. K. Katsikas et al. (eds.), *Information Systems Security*

involve the provision of security services as a result of a request by an entity requiring an instance of communication.

Discretionary protection: *Discretionary protection is access control that identifies individual users and their need-to-know's and limits users to the information that they are allowed to see. It is used on systems that process information with the same level of sensitivity.*

Discretionary access control: *Discretionary access control is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (granting permission—perhaps indirectly) on to any other subject.*

Some of the features of discretionary security from the definitions above need to be highlighted and explained in terms of an object-oriented environment and DISMOO.

- It is highlighted that users in an environment, where discretionary protection or security is enforced, will have a certain kind of access to certain entities. Traditionally, this would have been a READ, WRITE, UPDATE, etc. mode of access. In an object-oriented environment the access method used is the sending of messages. Sending of messages usually occurs in the form of a request being forwarded (in the form of a method identifier) to entities (usually objects). Thus, passing of messages are used instead of the use of specific types of access methods.
- The second important factor in discretionary security is the fact that each entity is owned by a user who grants access to this entity to other users (or revokes it from other users). Such a user is known as the *owner* of the entity. In DISMOO we will be using a hierarchy of owners, with the system security officer (SSO) as the "super" owner of all entities.
- In DISMOO a capability is a multi-faceted object which can differ according its set-up by the granter of the rights (owner of the entity). More than one capability for the same object may be defined by the owner. This decision is usually according to the owner's (user's) discretion, but the choice is mostly made according to the needs ("need-to-know") of the receiving user.

Discretionary security principles have mostly been modelled by means of an access control matrix. An access control matrix contains names of all the users of the system, versus the elements of the system that may be accessed, as well as the access type. Such a matrix is usually implemented as access control lists or capabilities. An access control list (ACL) is a list, associated with the protected entity, containing the identities of all subjects allowed to access the entity (including the modes in which the subject may access the entity). In contrast, capabilities are associated with the subject and specify which entities may be accessed by the subject.

In DISMOO we will assume that the latter implementation technique is used: Access rights (capability objects) are associated with the subjects and are required to access the entities they protect.

DISMOO provides objects as capabilities to protect other objects, object classes, instances and other entities. The purpose of the capability being an object is to make its characteristics as flexible as possible and to enable the user to make use of methods to

implement different access methods to the entity they are protecting. The major characteristic of the capability will lie in the capability superclass, and each refinement of this capability will be placed in either a class, subclass or an instance of the latter. The refined capabilities will be used by the owner of an entity as keys to the protected entity, and will be granted to different subjects on a "need-to-know" basis.

See Kim (1995) and Bertino and Martino (1991) for more information about object-orientation and Ting, Demurjian and Hu (1992), Rabitti *et al* (1991) and Olivier and Von Solms (1994) for more information about security in object-oriented systems.

3 COMPONENTS OF DISMOO

In the building of a discretionary security model, a number of characteristics have to be decided upon as listed by Dittrich, Hartig and Pfefferle (1989). These model characteristic questions of Dittrich *et al* will be used to describe the way in which DISMOO is built and the way in which it performs all its tasks. The list of questions to be answered is as follows:

1. What are the subjects (ie what are the active elements in the system)?
2. What are the objects (ie what are the passive elements in the system that are accessed)?
3. What are the operations (ie what kinds of accesses to objects can be distinguished by the protection mechanism)?
4. How does the security system function?

The first question to be answered is: *What are the subjects of the system?*

3.1 The subject

A subject in DISMOO is any user or user group that uses entities in the system, by sending messages to the protected entities. A subject can also be an object sending a message to another object. All entities can thus also act as subjects.

In a discretionary security system, unique subjects will have the right to control specific entities existing in the protected systems. These subjects are called the *owners* of the entities. Subjects can only obtain the right of ownership of an object by creating an entity or by receiving the ownership capability (key) to a protected entity in a transfer of the particular capability from its owner. Owners of entities in DISMOO will be in possession of the following:

- Possession of the capability (right) to use the entity.
- The capability class of the capability that protects the entity. New subclasses of this class can be defined to customise capabilities to be given to other subjects.
- The right to distribute the capabilities to other subjects; this is done by instantiating a new capability for the receiving subject from either the concerned capability class, or one of its (more restrictive) subclasses.
- The owner of an entity is also in possession of the right to revoke or take back any keys or capabilities which were handed out to other subjects.

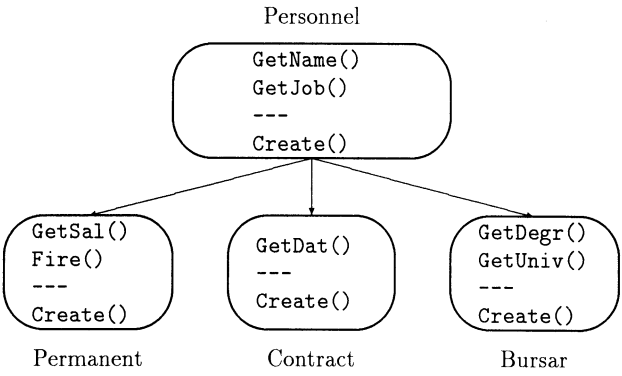

Figure 1 The personnel database

It is important to note that the onus to ensure security rests on the shoulders of the owners of the entities, since they are responsible of securing their entities according to their discretion.

The subject therefore consists mainly of users of the protected system as well as objects (programs) in the protected system. The next section will describe the passive entities of the protected system. In DISMOO the passive elements are called *entities*, while the term *object* is used in the object-oriented sense.

3.2 The entity

An entity in DISMOO is any element of the object-oriented database. Suppose, for example that we have a company with a personnel department. The department will typically own a personnel database, in which all details of the employees are stored. The personnel class will typically have permanent employee, contract worker and bursar subclasses.

Suppose the database looks as depicted in figure 1. Here the DISMOO entities that may be protected are the PERSONNEL class, the PERMANENT, CONTRACT and the BURSAR subclasses, as well as instances of these classes. Note that we assume that classes are also objects, which consist of behaviour (such as CREATE methods) and data. Each object will be protected as a whole; this includes protection of its methods.

An entity will typically look as depicted in figure 2.

The next question to answer is: *What are the operations (ie what kinds of accesses to objects can be distinguished by the protection mechanism)?*

Messaging, or the sending of messages to objects, is the only way in which objects react to external sources. DISMOO controls messaging by using capabilities as a 'gateway' or 'filter' to an entity. No entity can be used without a capability as key to that entity. A typical situation which will occur in an object-oriented situation can be seen in figure 3, where object O1 sends object O2 a message and O2 replies with an answer. Instead of having a direct link between objects, the capability will now act as intermediate gateway,

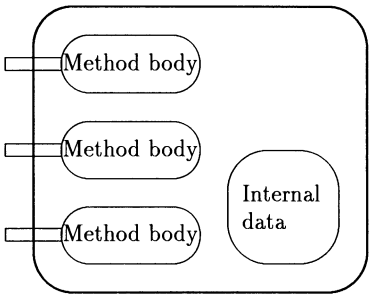

Figure 2 The entity

as well as a protection mechanism during the passing of messages. In figure 4 it is clear that a message from the sender to the target object will go through Capability1 which will route the message to the correct method in the target. If the sender were not in possession of Capability1, it could send as many messages as it wanted to, but the message will never be received by O2, because the method identifier in O2 is always a combination of the pointer in the capability to O2 and the method identifier in O2; in other words, no entities can be accessed without the proper capability.

Capabilities will now be discussed in more detail, before we discuss the operation of the protection system in more detail.

3.3 The capability

A capability is an unforgeable identifier for the object to which it serves as key. A capability also acts as a filter to the object that it is protecting.

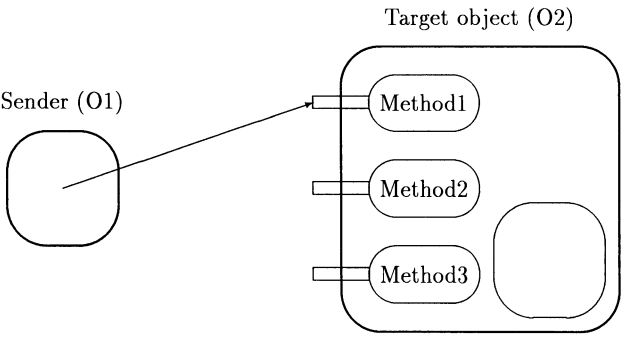

Figure 3 Sending messages in traditional environments

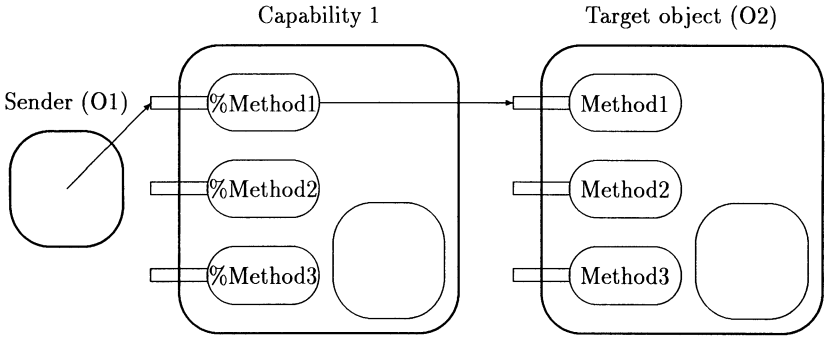

Figure 4 Sending messages in DISMOO

A capability is an object; therefore it consists of two parts, namely a state (attributes) and a behavioural part (methods). It also means that the capability is encapsulated and therefore protected against external attacks. The fact that a capability is an object gives the programmer of the capability much more flexibility in the structuring of the capabilities which will be granted to other subjects. As mentioned earlier the owner of the capability can now add intelligence to the protecting key. The composition of a capability is depicted in figure 5.

In this figure, the methods in the capability object serve as 'pointers' to the correct method in the protected entity. These capability methods refer to the method identifiers of

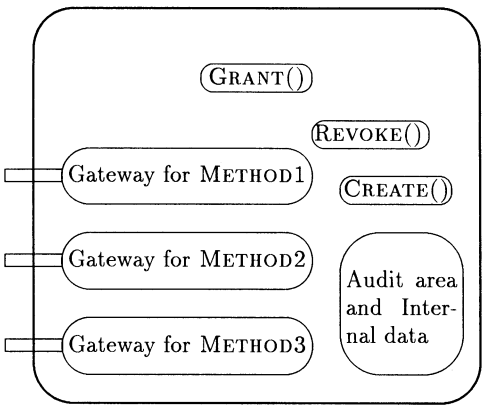

Figure 5 The capability

the called object and, when initiated, redirect the call to the receiving object or protected entity.

If there is more than one method in the protected object with the same function, but providing a different view, the capability will form the method identifier so that the correct method will be called. For example, say the SALARY object has two methods for reading a personnel member's salary, (a) METHOD1 capable of reading all members with salaries below a certain limit and (b) METHOD2, for reading all members' salaries. The owner of the entity may form one capability with a GETSALARY method that is redirected to METHOD1 and another capability with a GETSALARY method that is redirected to METHOD2. If the user sending the message is in possession of the capability that will form the method identifier for METHOD1, the user will only see salaries below the limit specified. However, if the user was in possession of the other capability the user will be able to see all salaries. The message sent to the protected entity will be the same for both users, so that they do not see the difference.

A capability can only be acquired or received when an object is created or if it is granted by an authorised subject. The method in which a capability is placed in the hands of a subject when an object is created is as follows:

1. The right to create a subclass rests with the owner of the concerned superclass, and can be granted by the owner of that superclass to any subject. We assume that the root of the class hierarchy is owned by the system security officer.
2. The CREATESUBCLASS method in the capability that protects a class automatically creates a capability class for the newly created subclass and makes the sender of the message the owner of the newly created capability class.
3. In addition to creating a new capability class, CREATESUBCLASS also instantiates a capability object for the newly created class from the newly created capability class.

Capability classes are enhanced by building different capability subclasses and instances out of this class to use as "need-to-know" keys. These capabilities can now be used to suit the particular requirements of each subject that needs to use the entity. The owner of the object is still in control, because access to the protected entity can only be granted to other users or subjects by the owner, unless the owner has given the right to grant access to other subject. In the latter case the owner is still in control, because he can revoke all granted access rights.

As an example of the above situation, suppose that the owner of the personnel database, the personnel manager, creates a new entity, the PENSION object. The following will happen : First, the personnel manager will receive a capability to access the PENSION object, and a capability class to create alternative capabilities to the PENSION object. Next the personnel manager creates a capability subclass with a 'pointer' to alternative methods for reading the salary. This pointer will now execute the alternative method each time the capability is used. Assume that the modified capability is given to another subject. Whenever the other subject makes use of the methods in the protected object, only alternative methods will be executed, without that subject noticing. Finally, the identifiers of the subject that received an access right or capability are recorded in the audit area of the capability class of the owner (see later).

Capabilities and capability classes are kept in a protected security area where changes can only be made by the permitted subjects and the system security officer. We also

assume that the system is constructed such that the only type of access which will ever be possible to an entity is via messaging. Since this is effectively controlled by capabilities, the model covers all types of accesses that can be made to entities.

The following question for a discretionary security model concerns the operation of the whole security mechanism.

3.4 Operation of DISMOO

This section describes the operation of different aspects of the security system, such as how control is enforced in the creation of an object, the granting and revoking of access rights, and the procedure in cases where the owner does not exist anymore and control has to be taken of the capabilities of the owner subject. The discussion will follow the sequence mentioned.

Controlling the creation of entities

Entities are those elements of the object-oriented database which need protection. The next aim for DISMOO was to create an environment where it would not be possible to fabricate entities that will masquerade as authentic entities. The next section describes how the prevention of fabrication is achieved.

The first mechanism that was put into place, was the rule that no entities could be created if the subject trying to create the entity was not in possession of a create right or create capability. The create right or capability is given to the subject by the system security officer or the database administrator (DBA). This has been described in an earlier section.

The fact that the system security officer or database administrator supplies a create capability to the subject, makes the SSO (or DBA) the super owner of the entity. This 'super' ownership gives the right to grant or revoke the right of using the object to the super-owner. The use of this right will only be used in exceptional cases of misuse of the entity or if the authorised owner becomes invalid. An example of a situation in which this right needs to be used is when the owner disappears without transferring the right for whichever reason. The system security officer has to take charge in this situation to be able to transfer the unowned and therefore unprotected capabilities.

We will apply some of the above mentioned situations in an example, which will also be used in the further discussion. Assume that a personnel manager asks the system security officer or database administrator for a create capability to create a personnel object database. The system security officer will, after the investigation, give him this right. The CREATE capability is now created as an object class of the security system. The personnel manager will use this capability to create the database by sending the CREATE message. The protecting capability and capability classes have been created during the execution of the appropriate CREATE method, and the personnel manager's capability lists have been updated accordingly. The personnel manager will next use the capability of the object to create subclasses or objects. Capabilities and capability classes for such new parts of the old entity or object are now part of the owner's capability class and therefore the capabilities start forming a hierarchy. Methods of the superclasses can now be used in the sub-entity capabilities, because all subclasses and instances will inherit methods and characteristics from their parent classes.

This example closes the description of creation of an entity and the control of the

situation, from where we will go onto the discussion of control in the granting and revoking of capabilities.

3.5 Control in granting of capabilities

One of the distinguishing characteristics of discretionary security is that a subject with the ownership right of an entity has the right to grant or revoke the right of usage of an entity to other subjects. The capability class of the owner of an entity contains a GRANT function, which is used to grant a capability to other users or subjects to enable usage of the entity, or even to grant the right to grant capabilities to an entity. In cases where a grant right is given, the GRANT function can be changed for the appropriate subject to which it is given in such a way that it will, for example, only grant capabilities giving access to certain methods in the entity.

Control can be carried out in the granting of capabilities by defining the granting function to limit granting of capabilities to certain user-roles (Ting, Demurjian and Hu, 1992) or identifiers. Granting methods can be changed to accept only granting to users with the appropriate role-classification or identifiers.

There is an audit area in every capability class in which the identities of subjects to which the capability has been granted. Stated differently, the audit area contains the sequence of subjects that granted the capability to one another. An integral part of the granting function is the updating of the audit area. The granting function will update the audit area with the identifiers of the subjects to which capabilities have been granted. The audit area cannot be changed in any other way, except by the granting and revoking function. The audit area is primarily used for cascading revokes. See Jansen van Rensburg (1995) for details.

3.6 Control in the revoking of capabilities

The revoking of capabilities can happen in the following four ways:

1. In the event or situation where a trigger is built into a capability, in which case the capability will destruct itself. That is, the capability is not revoked in the normal way by a message from the owner of the protected entity, but is in fact revoked by the destroying of the capability, by the capability itself. Since the capability is an object containing code this can be accomplished easily.
2. Revoking by the sending of a REVOKE message by the owner of the protected entity. This REVOKE message can only be sent by the owner of the capability class of the protected entity.
3. Revoking by sending a REVOKE message by any of the 'super' owners of an entity. Remember that any object that has been created, has been created by permission of the owner of the class from which it was instantiated, and every subclass that has been created, has been created by permission of the owner of the superclass. This means that any such owner is a 'super' owner of any entities created lower in the hierarchy, and access rights can be revoked by the 'super' owner. In particular, the SSO, who is the owner of the root class, can revoke any access rights in the system.
4. Cascading revoke can occur whenever the authorisation that allowed (or could have allowed) a capability to be granted in the first place, is revoked. Note that this can

also happen when the right to create an object or a subclass is revoked—the creator of the object or subclass will loose all rights to objects created. The recommended action here, if existing rights are to remain in place, but the create right not, is to grant the creator a capability to access the created object or subclass and then revoke the right to create new objects or classes.

The control of granting, revoking and creating capabilities can be described in much more detail, but the detail mentioned above will suffice for this paper—see Jansen van Rensburg (1995) for details. This covers the operation of DISMOO.

4 CONCLUSION

The use of object-orientation in a discretionary security system enabled us to introduce the idea of an intelligent key mechanism. Capabilities, unforgeable keys to protected entities, are controlled by the subjects or users of entities themselves. Owners of entities use these capabilities to construct keys according the needs of the users of the entities and their discretion as to what is really needed and what they want to give. A capability will filter a message according to the identity of the user of the capabilities and will then send the appropriate message to the protected entity. Capabilities can be granted and revoked to or from other subjects by the owner of the entity, or by a subject with a *give-grant* capability.

The aim of this method of using object-orientation in the building of DISMOO is to give a finer grain of control in the security system, as well as to build a more flexible security system. Further research possibilities include the use of mandatory security in combination of the discretionary security.

REFERENCES

Bertino, E and Martino, L (1991) Object-oriented Database Management Systems: Concepts and Issues, *Computer*, 33–41.

Dittrich, KR, Hartig, M and Pfefferle, H (1989) Discretionary Access Control in Structurally Object-oriented Database Systems, pp 105–122 in *Database Security II: Status and Prospects*, (ed CE Landwehr), Elsevier, Amsterdam.

Jansen van Rensburg, PW (1995) *Diskresionêre Sekerheid in Objek Georiënteerde Omgewings*, MSc-verhandeling, Randse Afrikaanse Universiteit, Johannesburg.

Kim, W (ed) (1995) *Modern Database Systems: The Object Model, Interoperability and Beyond*, ACM, New York.

Longley, D, Shain, M and Caelli, W (1987) *Data and Computer Security: Dictionary of Standards, Concepts and Terms*, Stockton Press, 1987, New York.

Olivier, MS and Von Solms, SH (1994) A Taxonomy for Secure Object-oriented Databases, *ACM Transactions on Database Systems*, **19**, 1, 3–46.

Rabitti, F, Bertino, E, Kim, W and Woelk, D (1991) A Model of Authorization for Next-Generation Database Systems, *ACM Transactions on Database Systems*, **16**, 1, 88–131.

Ting, TC, Demurjian, SA and Hu, M-Y (1992) Requirements, Capabilities and Functionalities of User-role Based Security for an Object-oriented Design Model, pp 275–296 in *Database Security V: Status and Prospects*, (eds CE Landwehr and S Jajodia), Elsevier, Amsterdam.

5 BIOGRAPHY

Wilna Jansen van Rensburg completed her MSc degree in Computer Science at the Rand Afrikaans University in Johannesburg, South Africa in 1995. The work reported in this paper formed part of her MSc dissertation.

Martin Olivier holds a Ph.D. in Computer Science and is currently a senior lecturer in Computer Science at the Rand Afrikaans University. Current research interests include database security, especially for object-oriented and distributed databases.