26

MoFAC: A Model for Fine-grained Access Control

Johan S von Solms, Martin S Olivier and Sebastiaan H von Solms
Department of Computer Science, Rand Afrikaans University
PO Box 524, Auckland Park, Johannesburg, 2006 South Africa
Email: {jvs,molivier,basie}@rkw.rau.ac.za

Abstract

Access control in Information Technology (IT) systems, also known as Authorization, is one of the cornerstones of any Information Security Policy. The granularity of such access control can be on different levels, for example on volume (disc pack) level, database level, table level, and even on individual record (or tuple) and data field level. Although very fine-grained access control, for example on record level, is often required, in most systems access control on table level is used. The reason is that the management process is significantly easier and simpler the courser the level of control becomes.

MoFAC presents a model in which access control is finer than table level, but where the increase in complexity and management stays within acceptable limits.

Keywords: Authorization, Access control, Distributed systems, Role-based security

1 INTRODUCTION

Security protection in a computer system is based on a security model or a security architecture (Kay, 1994). A security model is a protective control measure—an action, a device, a procedure, or a technique—that reduces the vulnerabilities. A security model should be applicable for a wide range of systems and applications, and consequently it is intended that it should include a wide range of security services that can be used and combined in different ways to meet the different security policies (Varadharajan, 1995).

Security is the totality of mechanisms and techniques used to create trust dependencies and minimize the risk of exposure of assets and resources to various vulnerabilities, thereby protecting the system (Muftic, 1991).

The two main security services are authentication and authorization. Most security products and research are in the field of authentication rather than authorization. For example the Single Sign-On Requirements paper by the greater New York ISSA Chapter gives little attention to authorization, the reason being resource authorization rules and practices vary far more than user authentication across hardware and operating platforms, and any product which attempted to handle all or most of these variations are seen to

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

be too complex (ISSA, 1994). These views on the complexities cause that research in authorization is lagging research in authentication.

Authorization is, however, also one of the cornerstones of Information Security (Kay, 1994). Controlling who has access to your electronic resources (objects), and what they may do with such resources is just as essential for proper information security. The level of granularity of access control is also very important—the courser the level, for example on disc volume level, the easier it is to manage the system, but the less specific the access itself is. Very fine-grained control, for example on the record level, or even on field level, increases management function significantly, and also makes it more complex (Pfleeger, 1989).

In some systems, for example military systems, access can be controlled very specifically but in most commercial systems today, a midway is followed by implementing access on table level, although in many cases a finer-grained level would be preferred (Baker, 1991).

MoFAC presents a model in which access control is on a sub-table level. Although the management effort is increased, it is less than with some other implementations providing access on levels finer than data file (Sandhu, 1993), for example an extended access control matrix (see section 2).

This paper is structured as follows: Section 2 defines authorization and gives a background on access control mechanisms; Section 3 formulates an example to be used in the rest of the paper; Section 4 introduces MoFAC; Section 5 evaluates MoFAC; Section 6 does a 'mapping' exercise between the MoFAC concepts and the OO concepts; and Section 7 gives a summary, and also indicates the context of MoFAC and discusses the future of this work.

2 BACKGROUND

Access control determines whether access to a resource is permitted. Permissions and authorization of users or processes are defined according to the policies of the business. An *access control policy* basically specifies a set of rules that describe the methods in which a client can access a server.

An *access control matrix* is a simple mechanism for the storage of access control information (Pfleeger, 1989). It is a table in which each row represents a subject, each column represents an object (the object can be a file or a record etc.) and each entry is the set of access rights for that subject to that object. In general the access control matrix will be sparse, because most subjects will not have access rights to most objects. Every subject will, however, be mapped with every object (subject, object, rights).

This approach can provide very fine grained security control. The problem is the more fine grained the control becomes the more entries are required in the table. In a big system the table can quickly become very big and difficult to manage and slow to search.

A simple approach that reduces the size of the access control matrix, is the concept of groups, for example used in UNIX systems (UNIX International, 1990). The number of rows can be reduced by grouping together similar users into user groups. The number of columns can also be reduced by grouping together similar objects into object groups. Extra tables are created to store the group definition information. Figure 1 illustrates a table for user group definition as well as object group definition.

The access control table can then be reduced by using the user group definition for

Users	Groups
johan	manager
rene, jack	teller
elmarie	accountant

Objects	Groups
object_1, object_2	objectgroup_X
object_3 ... object_5	objectgroup_Y

Figure 1 Users / Groups table (left) and Objects / Groups table (right)

the rows and object group definition for the columns. An example of this is illustrated in figure 2. It functions as follows: the user `elmarie` belongs to the `accountant` group, she can therefore read, write and execute `object_1` and `object_2` in `objectgroup_X` and read and write `object_3 ... object_5` in `objectgroup_Y`.

The group approach goes a far way in reducing the size of the access control matrix. It is, however, a bit restrictive and enforces **course grained security**. A user belonging to a group is restricted to the access rights of that group. An easy solution is to specify the individual user's access rights in a row in the access control matrix (for example the user `ilse`). The problem is then, if you want to implement finer-grained security control many users will be specified individually and the group approach fails. The access control matrix becomes bigger and bigger until it represents the original access control table with its problems. Furthermore two extra tables must be stored and managed.

A different approach is to use capabilities (UNIX International, 1990) and access control lists (Farrow, 1991). The first method is to specify only the objects that a user may access. This approach is called a capability. It can be seen as a token giving the possessor certain rights to an object. The capability can be stored with the subject. An example of a capability is:

```
[john, file_1(rw-), record_2(r-x), ... , object_n(-w-)]
```

It means john can only access `file_1`, `record_2` and `object_n` with the corresponding access privileges.

A second method is to create a list that specifies which subjects can access an object, including their access rights (Farrow, 1991). This approach is called an access control list (ACL). The ACL can be stored with the object or the resource. An example of an ACL is:

```
[file_1, john(r--), pete(rwx), jack(r-x)]
```

	objectgroup_X	objectgroup_Y
manager	rw-	--x
teller	r--	r--
accountant	rwx	rw-
ilse	-w-	--x

Figure 2 Reduced Access control matrix using user and object groups

It means that `object_1` can only be accessed by `john`, `pete` and `jack` with their corresponding access rights.

These methods also have some problems: Firstly, it is very restrictive, because a user can only access the objects explicitly named in its capability, and a object can only be accessed by a specified set of users in its ACL. Secondly, the management and administration of these approaches become impractical and near impossible; for example, in a big system with many objects it becomes impossible to update and check all the object's ACLs when one subject leaves the system.

In this section we examined several mechanisms to implement authorization and described some shortcomings. In all these mechanisms the achievement of finer-grained security control meant the management function became more difficult and complex. Our goal is to create a model that provides finer grained security control while keeping the management process easy and simple.

3 CASE STUDY

In this section we present a specific environment, with specific security requirements. We then investigate how these requirements can be implemented in the traditional IT way, and identify the implementation problems. We will then use these insights as the basis for the development of MoFAC.

Description of Bank Environment

Suppose we have a bank running its own IT system. Customer records exist for every customer, depending on the services used by a customer. Let there be a table for cheque accounts called CHQ, containing records for clients $C_1, \ldots, C_{vip}, \ldots, C_n$.

Transactions $T_1, \ldots, T_m$ are used to manipulate CHQ, where a transaction can simply be seen as a piece of code doing some manipulation on a record. Let transaction T1 be the QUERY transaction, and T2 the OVERDRAFT transaction, without specifying the functionality of these transactions. We call QUERY and OVERDRAFT 'job actions', because they are the actions performed and understood within the specific business role. A user will not know of transaction T1. The user performs the job action QUERY, which 'kicks-off' the transaction T1, totally transparent to the user.

Different business roles (Sandhu, 1993) exist within the bank—say a teller, a manager, an accountant etc. Every business role has a strictly specified set of job actions, implemented in terms of transactions, which can be executed on different electronic resources, in this case table CHQ.

Business roles have levels, and the set of actions between levels differ, for example a level 1 teller can not query a certain group of customer records including customer C_{vip}, because these are specific sensitive customers. These customer records can only be queried by tellers on level 2. Another group of customer records including C_1's record, can be queried by level 1 and level 2 tellers. A teller on level 2 can grant certain overdrafts which a level 1 teller cannot do.

Finally we have employees who occupy certain roles in the bank—for example John may be a teller and Jack a manager.

The challenge is now to implement an access control mechanism to control this situation (security policy):

1. A table for cheque accounts called CHQ, containing records for clients must be protected.
2. Two transactions (job actions), QUERY and OVERDRAFT are defined to manipulate CHQ.
3. Three business roles (teller, manager, accountant) exist with strictly specified transactions.
4. The different business roles are divided into security levels, and the set of actions between the levels differ.
5. Employees occupy certain roles in the bank.

Implementation of the Access Control policy

Suppose we define a Role Profile (RP) (Holbein and Teufel, 1995) as a data structure describing the job actions allowed by a specific business role.

We now assign a specific employee, say John, to an RP. John is a teller on level 1, but because we have only one teller RP, John is necessarily assigned to the RP for `Teller`. This means that John can only perform job actions QUERY and OVERDRAFT, i.e. indirectly perform transaction T1 and T2.

Note that we are already contravening our security policy, because John, a teller on level 1, can now also perform the job action OVERDRAFT, which is forbidden by the security policy.

With the CHQ, we now associate an Access Control List (ACL), containing the role profile of `Teller`, and other possible role profiles which may access CHQ through job actions.

If John now wants to query the balance of a specific customer, say customer C_1, he performs the job action QUERY. The system checks the Role Profile he is attached to, say `Teller`.

The systems then determines that a `Teller` may execute the job action, and requests the system to execute T1 against CHQ. The system checks that `Teller` may access CHQ, by determining that `Teller` is in the ACL of CHQ. In this case it is, and John can query the balance, but if John queried the account of customer C_{vip}, it would also have been successful, although according to the security policy stated above, it should have been denied. One problem is that the implementation can not distinguish between tellers on level 1 and level 2. There is no method to specify that John can only execute level 1 operations.

The obvious solution is to create more roles namely `Teller_1` (QUERY operation) and `Teller_2` (QUERY and OVERDRAFT operations), and assign John to `Teller_1`. This will mean that `Teller_1` and `Teller_2` must be included in the ACL of CHQ. This will not solve the problem, as the reader can check, John will still be able to query the record of customer C_{vip}, and managing more role profiles increases the overall management.

Another solution is to create two versions of the QUERY transaction, say T4:QUERY1 and T5:QUERY2 where T4 is programmatically limited to non-sensitive customers, and T5 is usable on all customers. We can then put T4:QUERY1 in the RP of `Teller_1`, and T5:QUERY2 in the RP of `Teller_2`. Again we have more role profiles, more programs

to maintain, and a training problem, because tellers on different levels must remember different type of job actions.

The most serious problem, however, is that to prevent T4:QUERY1 to access customer C_{vip}, the transaction will have to have record numbers hard coded in the program, or will have to have access to a special file containing the numbers of the special customers to which access is denied. All these aspects complicates management and maintenance.

Yet another solution is to create another separate data file for the sensitive customers, resulting in two record data files with the associated extra management and synchronization problems.

One of the main reasons for all these problems is that we are controlling access on the table level, making it very difficult to distinguish between records in the same table whose access must be handled differently. Finer access control, i.e. on sub-table level, and eventually even on individual record level, would support stricter security policies, but would make management of the implementation unrealistically complex and complicated.

For all these reasons, most companies do not implement security policies specifying fine-grained access control, and accept the risks involved. In very special cases, some controls may be coded in programs itself (Pfleeger, 1989), but as stated above, this increases the maintenance and management, and is done in only very special cases.

All these aspects lead us to the requirement for a finer-grained access method, without a corresponding significant increase in management effort.

This is precisely what MoFAC tries to do.

4 MOFAC

In this section we introduce some new concepts needed for MoFAC, and then show how the model can address the problems encountered above.

4.1 Record Groups

The first new concept we need is that of a Record Group.

A Record Group (RG) is just a collection of record IDs. In our example above, we will for example define two record groups RG1 containing all the record IDs (record numbers) of the 'normal' customer records, and RG2 all the record IDs of the VIP customer records.

Note that a RG is not another data file. In our example, CHQ is still the table containing the cheque records. As will become clear later, an RG is a structure containing the IDs of records belonging to the RG, as well as certain other information relevant to records belonging to that RG. The other information may be aspects like:

- What transactions may be executed on records belonging to this RG;
- What business roles may access the records belonging to the RG etc.

An RG will therefore contain a list of record IDs, and the roles authorized to access the records in this specific RG. If a role operates on more than one level, like our `Teller` role above, the specific level required for access is also indicated.

RGs for our CHQ table used above, may look as follows:

```
RG1 : Table : CHQ
List of all 'normal' customer record-IDs
Transactions : QUERY/Teller=Level1, level2/Manager=Level*/
               OVERDRAFT/Teller=Level2/Manager=Level*/

RG2 : Table : CHQ
List of all 'VIP' customers record IDs
Transactions : QUERY/Teller=Level2/Manager=Level*/
               OVERDRAFT/Teller=Level2/Manager=Level*/
```

An easy method to generate the different lists (sets) used in the RGs is to use set operations. For example let:

A = set of all the customer records ($C_1..C_n$) in CHQ;
VIP = set of all explicitly defined customer records (C_{vip}) in CHQ; then
N = set of all normal customer records in CHQ = A - VIP

The only set that must be managed actively is the 'VIP' set. All the other sets are automatically generated, when a client record is used, during runtime using set operations. The set generation process during runtime is slow but reliable; research on set generation during client record modification is being done (Von Solms, Olivier and Von Solms, 1995).

Note that we can simplify (consolidate) these RGs, but we leave it as above for clarity.

4.2 Role Profile Assignment record

In the previous section, we assigned John to the role profile `Teller`, but `Teller` is divided in two levels. We have to distinguish between these two levels, and we did not want to create separate role profiles for the two levels. Nevertheless, to strengthen the Record Groups defined above, we must in some way indicate that John is a `Teller` on level 1.

We do this by, assignment to a role. We create a Role Profile Assignment (RPA) record for the employee. If the role operates on different levels, as in our `Teller` case, the specific level of assignment is also indicated. In John's case, we will create the following RPA record for John:

```
RPA : John
Teller : Level = 1
```

Note, that in section 2, we also needed an assignment record for John, but that would only have contained the roles assigned to him, without the level indications where relevant.

4.3 Operation

Let us now examine how MoFAC, using all the data structures defined above, operates.

Suppose we have the structures as depicted in figure 3. John wants to access the record of the customer C_{vip}, as in section 2. There he succeeded to access this record, contrary to the security policy. This situation, with MoFAC in place, is depicted in figure 4. The

Role	Transactions
Teller	Query:T1 Overdraft:T2

RPA: John	
Role(s)	Level(s)
Teller	1

The Record Groups (RG) for CHQ	
RG1	List of all 'normal' customer record-ids
	Transactions: QUERY/Teller=Level1,level2/Manager=Level*/
	OVERDRAFT/Teller=Level2/Manager=Level*/
RG2	List of all 'vip' customers
	Transactions: QUERY/Teller=Level2/Manager=Level*/
	OVERDRAFT/Teller=Level2/Manager=Level*/

Figure 3 Role / Transaction Relationship, Role Profile Assignment record for user John and Record Groups for the CHQ table

tasks that are performed when John attempts to execute the QUERY operation on the C_{vip} record are the following:

1. John attempts to log on and chooses to work in the Teller role.
2. The Security System (MoFAC) checks John's RPA, and determines that John is allowed to log on to the role of Teller, on level 1. This information is stored for later use.
3. John executes the QUERY operation on the customer C_{vip}'s record.
4. MoFAC checks whether the QUERY job action (transaction T1) can be executed by Teller, and finds that it can.
5. The system now investigates the RGs of CHQ to see in which RG the record of customer C_{vip} appears, establishing that it is RG2.
6. From RG2, the system determines whether the job action QUERY can be performed on this set of records, and if so, whether a Teller on level 1 can perform this job action on this set.

Because John is on level 1 (from his RPA record), access is denied. Note that the specific record itself is not even retrieved. The reader can check that John will be denied to grant an overdraft.

5 EVALUATION

It should be clear that in achieving access control on a finer level, as MoFAC illustrated in the previous section, extra management is still needed. The RGs, RPs and RPAs are all new structures that must be managed. Nevertheless, we claim that it is more elegant than the 'brute force' method of having an ACL for individual records with employee IDs. Furthermore, it is easier to assign a new employee the access rights needed for his job.

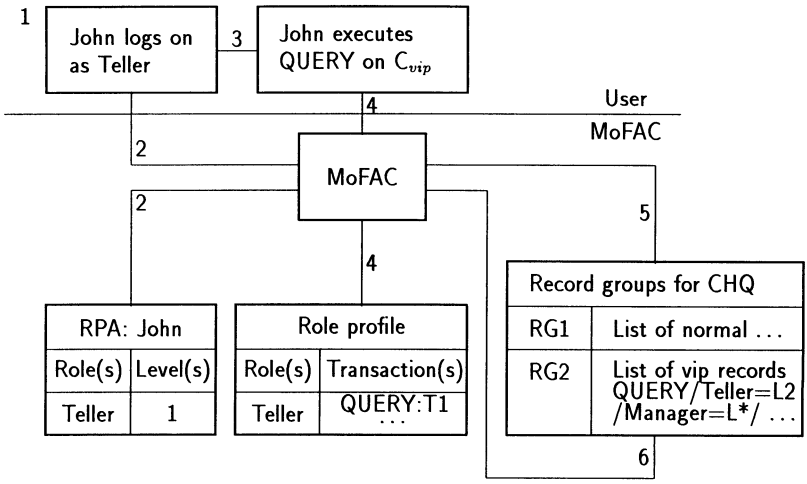

Figure 4 Mechanics of MoFAC

All that is necessary is to assign him to a role profile, i.e. create an RPA record for him. Deleting access rights when he leaves the company is just as easy.

It should also be obvious that the MoFAC model can be extended to cater for other types of rights users may have on resources, resulting in another dimension of granularity. Rights like read, write and execute can also be handled individually by MoFAC, as well as time constraints placed on members of an RG.

The basic idea behind MoFAC, and its power, lies in the fact that access to different records in the same file can be treated differently through a subject wanting access to record possessing (offering) certain rights, and the resource (object) to which access is required, demanding certain rights. Only if the offered rights satisfy the demanded rights, is the access granted.

6 MOFAC AND THE OBJECT-ORIENTED ENVIRONMENT

Many of the concepts in MoFAC can easily be mapped directly onto OO concepts. This is investigated in more detail by Von Solms, Olivier and Von Solms (1995) where it is shown that the actual power of MoFAC lies in the OO paradigm. In this section we will briefly illustrate that a correlation exists between the introduced MoFAC concepts and the concepts used in the OO environment and demonstrate how the concepts map onto one another.

Record Groups and Object Classes A Record Group 'represents' a set of records, and contains certain information and rules relevant to all the records belonging to that

RG. A Record Group can be seen as a basic form of an Object class, although an Object class entails much more than our RG.

Customer records and Objects In the OO environment, our customer records can be equated to objects, also referred to as an instantiation of an Object class.

Transactions and Methods Transactions are operations which can be performed on the records in the Record Group, and the same transaction will behave differently, depending on the RG on which it is operating. Transactions as used in MoFAC, can be seen as a form of the methods which operates on objects.

The reader conversant with 'overloading' and 'inheritance', should recognize them in the two versions of QUERY we defined above.

7 SUMMARY

MoFAC provides a way to control access on a finer level than normally implemented.

The use of MoFAC can further be extended for a distributed environment: through the use of smartcards, where the user carries his role profile on his card and through integration with the Kerberos protocol (Kay, 1994) that implements authentication.

These issues and others are currently investigated in more detail (Von Solms, Olivier and Von Solms, 1995).

REFERENCES

Baker, R (1991) *Computer Security Handbook*, TAB Reference Books.

Bull, JA, Gong, L and Sollins, KR (1992) Towards Security in an Open Systems Foundation, 3–20 in *Computer Security—ESORICS 92. Second European Symposium on Research in Computer Security* (eds Y Deswarte, G Eizenberg and J-J Quisquater), Springer-Verlag, Amsterdam.

Farrow, R (1991) *UNIX System Security*, Addison-Wesley.

Holbein, R and Teufel, S (1995) A Context Authentication Service for Role Based Access Control in Distributed Systems, 270–275 in *Information Security—the Next Decade* (eds JHP Eloff and SH von Solms), Chapman & Hall.

ISSA (1994) *Single Sign-On Requirements*, Greater New York ISSA Chapter Subcommittee on SSO.

Kay, R, (1994) Distributed and Secure, *Byte*, **19**, 6, 165–180.

Muftic, S (1991) *Security Mechanisms for Computer Networks*, Ellis Horwood.

Pfleeger, CP (1989) *Security in Computing*, Prentice-Hall.

Sandhu, RS (1993) Lattice-based Access Control Models, *IEEE Computer*, 9–19.

UNIX International (1990) *UNIX System V Security*, UNIX International.

Varadharajan, V (1995) Distributed Object System Security, 305–321 in *Information Security—the Next Decade* (eds JHP Eloff and SH von Solms), Chapman & Hall.

Von Solms, JS, Olivier, MS and Von Solms, SH (1995) Authorization in the Distributed Object Environment MOdel for Fine-grained Access Control (MoFAC), Poster, IT Sicherheit '95, Graz, Austria.

8 BIOGRAPHY

Johan von Solms is currently an MSc student at the Department of Computer Science of the Rand Afrikaans University in Johannesburg, South Africa. The work reported in this paper forms part of his MSc studies.

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.

Sebastiaan (Basie) von Solms also holds a Ph.D. in Computer Science and chairs the Department of Computer Science at the Rand Afrikaans University. His research interests include all aspects of information security. He also chairs IFIP TC11.