A Security Language For The Card: The S-Shell.

Jean-Marie PLACE and Patrick TRANE
Recherche et Développement Dossier Portable (RD2P)
1, rue du Professeur Jules Leclerc
59037 LILLE Cédex
Tel: (33) 20.44.60.46
e-mail: patrick@rd2p.lifl.fr or jmplace@rd2p.lifl.fr

Abstract

This article draws up the project S-Shell. The goal of this project is to create an environment of conception and realization of a global security model of an information system. After a first analysis of the different paradigms brought into play on several systems (Unix, database system and microprocessor card), it appeared that each system presents some weakness.

We have on the one hand the lack of a great enough capacity of expression for the security system which only takes into account several basic operations. Most of the time, this lack imposes the existence of a superuser who can correct the scheme according to the circumstances. On the other hand, the scattering of the scheme according to the different entities to be protected does not allow neither a complete and easily handable description nor the proof of the scheme.

So as to answer all these comments we thought of a way we could solve these problems. We have designed a language that allows a user to describe a security scheme in a given context of an operating system. We conceived a tool so as to test and to validate the language on an example relating to the context of an operating system of a card we will describe. In conclusion, we high lighted some precise points for further research.

Key Words: security scheme, access matrix, operating system, database system, microprocessor card operating system, smart cards, capacity of expression, superuser.

Preliminary

As the term «security» [A90] is used as a generic term for a lot of works connected with confidentiality, integrity, dependability, cryptography, networks, operating system, information technology, hardware and so on, we would like to stress what this paper is about or, more exactly what this paper is *not* about.

This project is an overall reflexion on the way people designing an information system think of security, how they don't proove it,, how they describe it and what tools they can use for that purpose. An outcometo this reflexion is a language or a tool which permits the expression of what we call a security scheme[1].

Even though this language is brought into play by the mean of a compiler and/or an interpreter, we know that all this work is based on assumptions which are not yet realized or may even never be realized. These assumptions can be the following ones:

- The hardware is resistant to all attacks

- One day, somebody will write a perfect operating system software

In fact, the important point for us is: what should be done to make security description simpler and then more effective?

Introduction

In the process that leads from the conspicuous position of a need to its automatic or semiautomatic implementation, it is commonly admitted that there exists a phase of analysis of the information system (organic analysis, functional analysis) followed by a phase of implementation.

That widespread practice leads to separate the analysis of the problem and its implementation. That independance of logic and physics enabled to elaborate database management systems around models brought to the fore. That is the way most of the difficulties linked to the description of the data are nowadays solved.

Should we now consider problems linked to constraints of integrity and security, it would appear that the process used is much more pragmatic. There is no common technique to reveal that the cashier of a bank should not be able to realize the same kind of operations as the head of the agency.

The implementation of these constraints (that we group together under the term of "security scheme") depends on the intelligence of the one who brings into play that solution and on his knowledge of the system used and of its mechanisms of security. Therefore, there is really no standard that entitles somebody to express in an easy and direct way a security scheme. Moreover, actually there is no method that would help to produce this scheme [Tr92].

1. The actual answer is to set a stock of bits inside a memory.

In this paper we first explain how the access matrix model has to be extended in order to answer these needs. Then we analyse current systems and see how they can answer the problem. The next step is the description of a language designed to allow the expression of a complex security scheme. Then, we describe the interpreter we designed for this language and give an example of its use. At last, we describe how such a tool can be integrated in an operating system.

The access matrix

The overall problem is to relate objects that are to be protected to the different actors who want to make an attempt at executing a number of operations upon these objects. The access matrix model is an answer to this need. This model regards the security scheme as a matrix having a column for each protected object and a row for each user. The cell at the crossing of a particular object and a particular user gives what kind of operations the user is allowed to do upon this object. Usually, the security system handles only few operations (Read, Update, Execute,...) [D82].

From this starting point, we will add some extensions in order to increase the power of expression of the scheme. Let us consider the following definitions:

- The objects are the groups of information that are protected. Usually they are files, tables, fields etc...

- Actor (user) is a generic name for whoever can be identified by the system having its own rights over objects to be manipulated. It can be a real user - with a login and a password -, a process or a terminal fulfilling authentication procedures

- The operations are not necessarily restricted to a small set like Read and Write. For instance, SQL states Insert and Update operations. According to the access matrix model, it means that a cell may state a lot of operations including operations defined at the application level.
  One criterion of our analysis is to show how new operations are defined and taken into account in the existing systems

- By considering our experience, we found that, in a great number of circumstances, the superuser of a system has to correct the security environment of this system to enable some kind of manipulation - like disabling virus protection to allow a software installation. Each of these operations weaken the exactness of the mechanism in the way that the security of a system relies mainly upon the rigour of the system administrator. Moreover, the security of a smart card can not involve any external component.
  Thus, rather than be depending on a human being, we prefer to give the designer of the system a tool powerful enough to express every circumstances handled by the system.
  The result of these extensions for the access matrix is that an operation should be conditionaly stated, based on some trusted data. For instance, this allows to grant a Read permission from 8 am to 5 pm for example (in this case the current time is considered as a trusted data)

We illustrate this proposition with three examples which don't come under management applications but which make use of an information system under well identified elements form.

For each of these we see how security is described and how needs stated above are handled. We also analyse how the security scheme is described. We focus on the ease of description (we think that one mean to achieve security is to have a way of describing the intended scheme as simply as possible). We do not consider the effectiveness of implementation.

1. An operating system.
2. A database.
3. A card with microprocessor.

Three examples

An operating system

One of the roles of an operating system is to manage permanent objects usually stored in memory [Ta84, Ta89]. As an actual system is asked to deal with several users, a system has to protect its objects from illegal accesses. To do so, the users - identified by a login and a password [MT78] - may be defined by the system administrator. Each user may create objects - like files - in the mass storage. And the rights that the users have on objects are associated to the object itself.

A look at a particular system: Unix shows up that within the properties of the files there are the name of the owner of the file, the group (a symbolic name) of the file and the rights described with three classes of three rights. Each class lists the rights that a user or a class of users has upon the file. The classes are :

- The owner of the file

- The users who belong to the same class as the group of the file

- Anybody else

For each class, we can read three flags, one for each operation on the file. The three operations are Read, Write and Execute. This is basically the idea of the security scheme on a Unix system (We do not consider directories for which the meanings of the three operations are slightly differents).

According to this mechanism, one can enforce the confidentiality of the information within its files by reserving the Read access to the users he trusts in, and logical integrity by denying Write access for his files from the rest of the world. The Execute right can give a restricted access to protected informations but requires the writing of some software.

This security system is not sufficient enough for an everyday use. We are going to show a few problems that can not be well solved and then require either some manipulations from the superuser or the writing of software.

The separation of users into three classes is too rough to handle properly usual problems. You may need to give some rights to a group of students, other rights to a smaller group of students and others again to the teachers. The whole on the same file. That means that one has to design very cleverly the groups of users that are generally mutually exclusives. In fact, many Unix systems have included the ability to describe rights for a file to one or more identified users or groups.

Moreover, the semantic of each operation is just sufficent for a global use but fails if we have to answer a specific problem. When one obtains the Write access for a file one may indiscriminately increment, change or even destroy the data in it. In the case of a bank account this is a crippling default. By way of example, a proper system ought to allow somebody to increment but not to decrement a value.

A similar problem is encountered while dealing with rights that depend on a global environment. For example, you may need to allow an access to a data only during working hours, from 8am to 5 pm. The standard Unix system does not permit to define such conditions. As a solution, you may either write a piece of software that checks the time of execution and grants execution rights to it, or ask the superuser to change manually the rights in the morning and in the afternoon. Both solutions are bad, for errors or frauds can be introduced -thus weakening the security - in the process.

A database system

The process of designing an information system is prettier in the databases domain. First, the design is effectively taken apart from the logical description which is in turn separated from the physical aspect. In other terms, the designer of the information system does not take into account the tool used for the final realization [BM84].

The security itself is better treated in this context. This is explained by two remarks:

The final users are more concerned with security since the information system may contain crucial information. Thus the need for realizing and expressing global security scheme is a priority in this approach.

One more argument is that an ideal database system completely encloses the information it manages. A user or a program can access the data only through the database system. This is a great advantage. Furthermore, the set of operations that are available are well known [Gr90]. So the security scheme is simpler to describe than in other cases.

All these considerations make the description of security in the context of a database very simple and widely used. But a few drawbacks remain:

The set of operations that are treated are not atomic enough to permit a low level description. Once again, one could not give the ability to grant an increment right on an integer while denying the decrement right. This is usually - but partially - solved by designing forms that check the validity of the data.

The user identification relies completely upon the operating system. That means that you can specify user but you can not create or even state your own groups of users.

On the other hand, we appreciate the ability to dintinguish rights over tables, or views which permits to have a scheme close to the data description and a granularity [Y90] as tiny as needed.

Then the database approach seems to us to be a good one but we think that it should be extended to increase the facility and powerness of expression.

A Microprocessor card operating system

We take an instance of the MCOS [Ge90] card which is a multi-applicative card. Inside the card [F86, CP93], the data are described as a collection of files which are themselves grouped in directories. A user is identified by a pin code. Each user is granted some rights which are represented by bits in an authorisation register. A file may ask for specific bits to be raised for successfully complete a Read, a Write or an Update operation. There are also complex rules for managing the operation of creating a file, a directory or a code.

The way that security is designed and realized is even worse in card operating systems. It is roughly a translation of classical operating systems concepts, reduced to fit inside a small ROM code. We therefore can see that a card which is meant to be a definitively secure system is in fact difficult to use in a real case where the rights we want to implement are nor just Read and Write access. The ultimate solution is to include a specific software in the system code, if possible.

As a result, the designer is envolved with bit descriptions of users, directories and files. Though we may create small groups of users who share similar rights, we are limited to a small number of users.

A new generation of smart card - like the MCOS - are designed to support several applications. This makes the problem much more complex because the applications do not trust each other. The card emitter may not trust any application. The general rule applied then is "one application is one directory". This rule makes the sharing of data between applications very hard to realize safely.

Propositions.

We think that one thing to do is to make the level of security description higher than it is now. Rather than dealing with authorisation bits, the designer should use a language of security description as natural as possible. This language has to be linked to the description of the objects of the operational environment (database, operating system,...). The language must consider the commands available at run time.

The intrinsic form of the access matrix which is a table having three entries (commands, objects and users) makes its text description difficult to read and to analyse. We first wanted to experiment the ease and adequation between such a description and a real use. The problem is that a tool able to handle this kind of description is much faster to realize. In fact, the ultimate form of description uses an interactive tool presenting many different points of view for the scheme. In this choice, we did not consider the - binary - effective form (the one to be evaluated) of the scheme.

However, we were also concerned with the feasibility of the evaluator - which should eventually be a part of the operating or database system.

The purpose of a security language

The aim of our security language is to create a system of protection capable to reject incorrect commands sent. That functionality is done from a predefined security scheme.

Following the security scheme described in a text file created outside the environment, the security language is able to validate or reject a command.

If the command is accepted by the 'evaluator' of the language, it will immediatly be executed, and the security language will be transparent for the user.

On the other hand, in case of rejection, the security language prevents the execution of the command and supplies the user with an error diagnosis (which can weaken the system for it can give useful informations).

To be more precise about that description, we must add the fact that to increase the power of calculation and to answer to some typical problems (like problems of chains of events) the notion of variables. This involves the problem of types of variables and of course the problem of control of variables.

We insist on the fact that we do not want to create a new programming language but only give a user the tools so as to easily describe his own scheme. The only thing we want to do is to create a link between a command (Unix, Dos, 7816-3,...), an action and conditions without writing a single line of software.

The scheme is done once and for all, and is not modifiable. The acceptance or refusal of a command at a certain time and a certain date does not depend on the user but only on the scheme of protection of the system.

Possible outcome

We stated before that the only interest of our software tool was the opportunity to test how the language designed could fit a real application. This tool is obviously not designed to be used as it is as a protection software. Many of the hypothesis done are not real and no more feasible in the framework of a usual operating system. However, there is still a possible outcome for this work in the smartcards domain. The operating systems of microprocessorcards are small and secure. One can imagine that the mechanisms of protection within the card could be described as a high level with a language similar to the one we are working on, instead of the bit level usually in effect. This can be allowed by the higher performance of the embedded hardware.

The designed language

All we want to do is describe in a text file our security file corresponding to the situation we are in. This text file is broken down into three distinct parts repeated as many time as necessary.

1. A title of command

2. A list of actions associated to the command
3. A list of conditions associated to the command. That list can be of three kinds: Always, Never or If followed by the condition(s)

Exception: The title of the command can be a specific action designed by the keyword: 'default'. This means that the group of commands associated to the 'default' action will not be taken into account during the description of the scheme.The list of actions associated, in that precise case will be called a list of initial actions.

To that description we must add that we reserve spaces to both declare the *variables* , their type and their initial value and the *events* to be verified or/and to be executed before any operations (this can be reinitialization of one variable every week for example).

For a 'n' command text file, we can describe the general skeleton of its content.

We obtain a description file of that type:

[declaration]

[list of (variables, types, values)].

[events]

[list of events to verify or to execute].

[description]

[default] [{ list of associated actions }] :

[always, never or if followed by condition(s)] ;

[command 1] [{ list of associated actions }] :

[always, never or if followed by condition(s)] ;

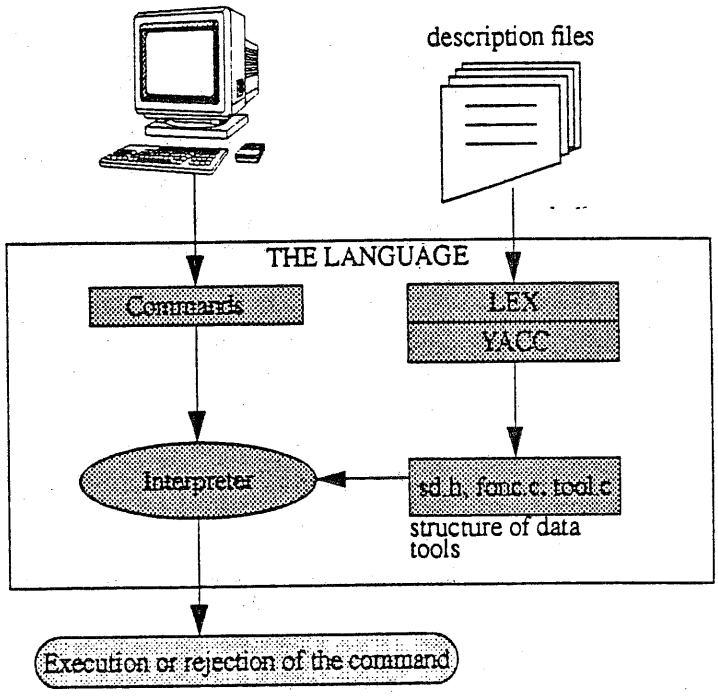
...

[command n] [{ list of associated actions }] :

[always, never or if followed by condition(s)] .

To be complete in the description and to make it more powerful we divided the description file into sub-files specific to an aspect of the description of the security scheme. This gives a structuration of the problem. We obtain a sequence of files very easy to write only considering that a specific keyword of a file 'i' will become a command title in the file of description 'i+1'.

The sofware tool



To understand how this scheme works, we have to explain this graphic above. According to a simple and well-defined language, we can define our own security scheme. It must respect a strict vocabulary developed thanks to the lexical analyser Lex (all the programming has been done under a Sun Unix) and a flexible grammar described thanks to the syntax analyser Yacc (Yet Another Compiler-Compiler) [LM90]. Moreover it has at its disposal a whole set of tools to simplify at the most the ease of description of its scheme. This set is included in the language and enlarges the user-friendliness of the tool. For example one can write: (25/12/93 + 2 days) or (10:00 between 09:30 and 12:00) and so on... An internal evaluator produces a result to these writings (27/12/93 or True for the two examples given here).

A Case of study

We are going to describe, by the way of example, the case of a lending department of an association of families in a school [PP91]. The principle is very simple. The families can buy books that will be used by their children during their year of study. These books can be taken back by that association the following year (or later if the student held back a year).

A below par rating can be proceeded according to the condition of the book (there are three conditions: new, used, spoilt). A list of the books asked for each child has been given to the parents according to the grades followed.

One deposit per family and per year is asked whose total amount is 20 ECUs. Unfurnished books can be ordered through the medium of the association.

To get a book there is a certain number of steps to do correctly:

1- A station of recovery: the family has to give back the books of the previous year. At this step, the condition of the book will be evaluated and the book will be stored.

2- A station of purchase: according to the list of books and their condition, a bill will be produced. It may happen that the book desired is not available, it must then be ordered. Finally, the family leaves with the books to take away, the list of the books to be payed and/or to be ordered.

3- A station of payment: in that station, the family has to pay everything. Ordered books are payed in advance.

4- A station of collection: Parents will come to that station to collect books already ordered and payed.

Objectives

From the description of the problem, we can identify two goals for security.

The first goal is to prevent a smuggler to read or modify data in the card. This is done thanks to the rights embedded in the card itself. The related authentication uses classical methods of pin code or passwd. These mechanisms are out of the scope of our works because it is part of our hypothesis.

The second goal is to supervise the actions of the legal users to prevent them from changing data in the card so as to gain illegal benefits. As we always said, the lowest level of mechanisms which aim that goal is the read or write protection for a file in the card. For instance, this is a way to prevent the station of recovery from making books be considered as payed (that is a privilege of the station of payment).

The work claim is to allow a more detailled and a more natural description of what can be done and also what has to be done. In this example, a perfect description should

clearly say that when a book is marked recovered, the state of this book should also be filled. This introduces two remarks.

The first remark is the following: The further we describe the security scheme, the closer we will be from the description of the integrity constraints. Ultimately, we will have to describe the whole rules of our information system.

The second remark is that usual mechanisms allow only the forbidding of a command according to the circumstances (code presentation, etc). The designer of the information system cannot easily give obligations.

Let us take the example of an information system managing two accounts. The system is such that the sum of the two accounts should remain constant. The goal of the security system is to prevent a cashier from increasing one account and not decreasing the other. Usual mechanisms allow to declare that the cashier alone is able to change the data in the files. But this is not enough for what we want to do.

The classical solution for this problem is to write a piece of code which makes both the increasing of one account and the decreasing of the other one by the same amount). It implies a work tied to the implementation target (the machine, the system or the smart-card) and subject to mistakes and errors.

The language we describe allows an account to be increased by an amount only if the other account has just been decreased by the same amount. Although we think that it is a real progress, we are lacking some tools to make things smarter. In fact, as we are approaching integrity considerations, we do need the concept of atomic transaction taken from the database domain.

The case of study is a good example of that. The exact translation of the actual protection mechanism is very easy if not automatic. But when we want to do more, the language does not permit to define a transaction such as payment or recovery. The description is tied to lowest actions (read and write). The description of the security turns out to be tedious and somehow artificial. Instead of that, our second proposition uses the actions of payment, recovery considered as transactions.

Presentation

We first want to organize such a department with an operating system like MCOS (Multiple Card Operating System, [Ge]). Later, we will do the same thing using our security language to proove that:

1- We can in a concrete example do at least as much as MCOS. It means administrating the security (in terms of access rights) of this lending department. We do not take care of the personalization phase.

2- If MCOS can garantee the integrity of access it can not garantee the correct organization of the different stations. We are going to show how we can do it with our security language in a very few lines of description.

The description (MCOS)

Description of the protection.

For security reasons, each field will be manipulated by different stations with different rights. Rights can only be granted at the file level. We must then split our data structure into groups of fields according to the rights.

Methodology:

1. We first establish a list of users. In our case we have five different stations which are "Recovery", "Purchase", "Payment", "Collection" and "Refunding". We consider that the "Personalization" step as a special step for it takes place at the very begining of the life of the card [CP93 , MP91] before being given to the family.
2. Then, we establish the rights associated to each user on each field. This boils down to that picture.

STATIONS	RIGHTS					
1. Recovery	R		R	R	R	W
2. Purchase	R		W	W		
3. Payment	R	U	R	R	W	R
4. Collection	R		R	W	R	
5. Refunding	R	U	U	U	U	U
FILES	1	2	3	4	5	6
FIELDS	- Family - Number	- Year	-Book - Condition	- Furnished	- Payed	- Back - Condition

Remark: The goal of this part of the paper is not to produce a precise elaboration of a MCOS implementation. But to be more understandable, we must underline that normally we should translate that drawing in term of values of rights to each file.

How we can handle it with our security language.

The purpose of this chapter is to proove that it is not difficult to define with our security language a description of a state of protection of MCOS data. We insist on the fact that we do not want to optimize this picture above. A text file written for our language would give in that precise case:

Read :	if (file = 1) or ((file = 2) and (user = 'Payment') or (user = 'Refunding')) or (file = 3) or (file = 4) or ((file = 5) and (user <> 'Purchase')) or ((file = 6) and (user = 'Recovery') or (user = 'Payment') or (user = 'Refunding')) ;
Write :	if (((file =2) or (file = 5) and (user = 'Payment') or (user = 'Refunding')) or ((file = 3) and (user = 'Purchase') or (user = 'Refunding')) or ((file = 4) and (user = 'Purchase') or (user = 'Collection') or (user = 'Refunding')) or ((file = 6) and (user = 'Recovery') or (user = 'Refunding'))) ;
Update :	if (((file<> 1) and (user = 'Refunding')) or ((file = 2) and (user = 'Payment'))) ;

As shown above, the text file proposed is very simple and allows a user to have at least the same possibilities as with MCOS.

Additional benefits.

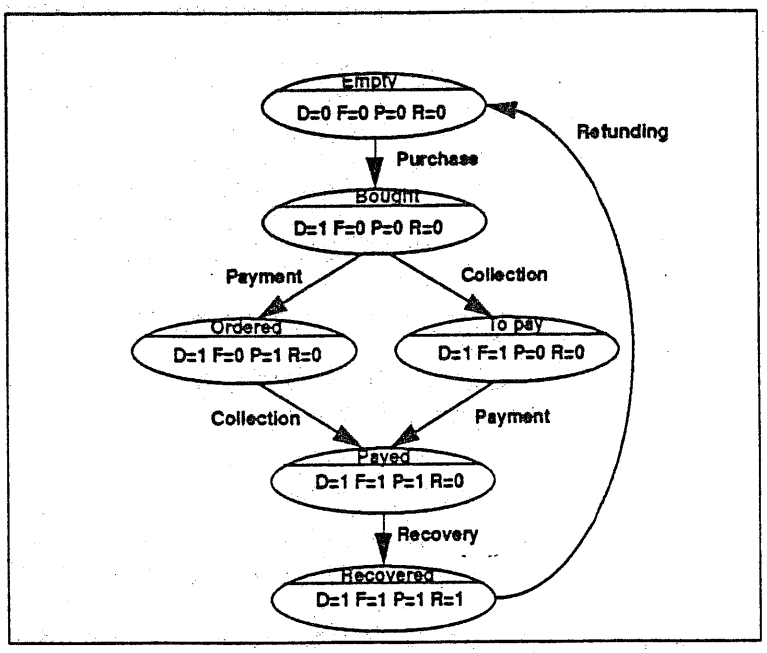
Now then, if we consider the way this lending department works, we notice that there is a sequentiality in the actions. For instance we can not collect a book which is not yet bought. This means that if we really want to control both the rights associated to the files and the sequences of actions, we must write a software outside MCOS facilities.

With our security description and thanks to the introduction of variables we can achieve it.

This automaton (see scheme next page) shows that we can describe the state we are in by four variables D(defined), F(furnished), P(payed) and R(recovered) associated to each book. The transitions from a state to another are Payment, Collection, Recovery and Purchase. The aim of what we would like to do is to validate or to prevent a transition according to a certain value of the group of variables (D,F,P,R).

We can take by way of example (D,F,P,R)= (1,1,1,0). The user wants to pay at this level. Our language will now look at the command 'Payment' and will note that to do such an action we must have either (1,0,0,0) or (1,1,0,0). Therefore it will not accept this command and will furnish a diagnosis of error. That functionality can not be directly done with MCOS.

To make it easier to understand we can draw a scheme of the actions available for a single book, here is a drawing of it.



We have now:

Recovery :	if ((Book.D = 1) and (Book.F = 1) and (Book.P = 1) and (Book.R = 0)) { Book.R = 1 ; } ;
Payment :	if ((Book.D = 1) and (Book.P = 0) and (Book.R = 0)) { Book.P = 1 ; } ;
Refunding:	if ((Book.D = 1) and (Book.P = 1) and (Book.F = 1) and (Book.R = 1)) { Book.D = 0 ; Book.P = 0 ; Book.F = 0 ; Book.R = 0 ; } ;
Collection :	if ((Book.D = 1) and (Book.F = 0) and (Book.R = 0)) { Book.F = 1 ; } ;
Purchase :	if ((Book.F = 0) and (Book.P = 0) and (Book.R = 0) and (Book.D = 0)) { Book.D = 1 ; } ;

Thus, with the same group of values, by asking for a recovery, the security language will verify that the values of the variables are correct and will turn Book.R to one. This will produce a new state (D,F,P,R) = (1,1,1,1).

In conclusion we can say that by well dividing the actions, the objects and the commands, we can easily write the textfile(s) corresponding to a defined situation. Thanks to our first file, we have produced a simulation of what MCOS can do. The second file will give a user the sequentiality that fails in the operating system of the card. Of course we have not taken in consideration the fact that a family could take more than one book and

pay everything at once. This problem is fairly different but does not affect our way of thinking.

So as to do it properly, one of the most important thing to do is to isolate commands, objects and actions. For instance in our first file the commands were Read, Write and Update. There was no specific action. What is interesting to be noticed is that the users we consider in the first file (Payment,...) are in the second the commands. This means that we can divide a general security problem into subproblems easier to solve.

Conclusions

We propose to replace classical low level descriptions of security by an implementation independant language. We believe that this can bring a natural, easily handable and translatable tool for information system designers.

By practising our tool on academic problems, we found that such a tool should use broader concepts than only data, action and users. We explained the need for transaction concepts and sub-domains as well as the showed up connections between protection and integrity constraints.

This approach is complementary to other works that tend to improve physical and logical security. It only tries to fill the gap between the conceptual level of description of an information system and the actual realization of it by the use of hard coded rights.

References

[A90] AFCET, European Symposium On Research In Computer Security, 1990

[BM84] JM. Busta, S. Miranda, "Introduction Aux Bases De Données", Ed. Eyrolles, pp 11 to 22, 1984

[CP93] V. Cordonnier, T. Peltier, "Taxonomy Of Smartcards", Publication LIFL, 1993

[D82] D. Denning, "Cryptography And Data Security", Addison-Wesley, pp 191 208, 1982

[F86] R.C. Ferreira, "On The Utilisation Of Smart Card Technology In High Security Applications; Perspectives For The Future, IFIP/Sec 86, pp487 to 503, Dec 1986.

[Ge90] Gemplus, "MCOS Reference Manual", 1990

[Gr] R. Graubart, "A Integration Of Security Considerations With DBMS Development", IFIP WG 11.3 North-Holland, pp 167 to 189, 1990.

[LM90] J. Levine, T. Mason, "Lex & Yacc", chapters 2 and 3, O'Reilly & Associates, Inc, 1990

[MT78] R. Morris, K. Thompson, "Password Security: A Case History", Bell Laboratory, pp 595 to 601, 1978.

[MP91] C. Macon, A. Pillot, "Téléinformatique Et Systèmes D'exploitation", Tome 2, Nathan, ch 6-4, pp 243-244, 1991.

[PP91] J.M Place, T. Peltier, "Les Cartes A Microprocesseur, TP D'application", 1991, Publication LIFL.

[R91] C. Rolland, "Conception Des Systèmes D'information", 1991

[Ta84] A. Tannenbaum, "Structured Computer Organization", chapter 6, InterEditions, 1984.

[Ta89] A. Tannenbaum, "Les Systèmes D'exploitation", 1989

[Th90] M.B. Thuraisingham, "Security In Object-Oriented Database System" , 1990

[Tr92] P. Trane, "Protection Et Système D'information", Mémoire de DEA, Publication LIFL, 1992

[Y90] K. Yazdanian, "Relational Database Granularity", AFCET, Esorics, pp 15 to 19, 1990