Teaching System Access Control

Juan R. Bauer Mengelberg Colegio de Postgraduados Montecillo, Edo. de Mexico, Mexico

jbauer@colpos.mx

Abstract

Many specialists believe that more emphasis on system infrastructure is needed, including the protection of the system's data against unauthorized use or updates. This paper proposes a way to teach future information system specialists the main concepts involved, using a powerful yet very flexible method to implement access control. Constraints, widely known as context constraints, which depend on the value of a variable or data item, are included. We discovered it was extremely difficult for students to understand, let alone adopt, a model as general as the one we use to explain the necessary ingredients of an Access Control model. However, the order in which the concepts were introduced produced a very significant increase in the degree of understanding on the part of the students. Rather than just provide another method, the paper's objective is to promote discussion as well as further study of the subject.

Keywords: teaching, access control, information systems, security, context constraints

Introduction

An information system has a number of components which complement and support its main functions, those which update and use its data. We could include security features, good and fast backup procedures, items which make the system easier or safer to use and certain attributes which individual designers may include to decrease several types of risks associated with madeto-order systems. Many specialists consider that such components of information systems, referred to as their infrastructure, need to be strengthened and have proposed ways to achieve this goal. We have approached this matter in two different ways: teaching specialists the concepts and needs, but also designing tools for system developers to include powerful and robust ready to use components in their systems. These tools turned out to be useful teaching aids. Though the method we present is based on its concepts, we did not include a detailed description of the toolset, which contains the data structures and routines to provide a system with access controls such as we describe here.

Even though this paper is addressed to teachers, we do not present it as didactical material or even reference for students in a course, nor will we try to persuade anybody that this is some kind of a

solution to one of the difficult problems of teaching these types of concepts. Its purpose is to show yet another example of how changing the way in which we explain a topic – in this case altering the order in which we introduce certain concepts – can produce surprising increases in comprehension on the part of the students.

Material published as part of this journal, either on-line or in print, is copyrighted by Informing Science. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission from the publisher at <u>Publisher@InformingScience.org</u>

Though we have only used it with graduate students, there is no reason why it should not be useful with others. We would like to make a comment which will sound a bit out of place in a paper such as this one. We have often felt that exposing students to an oversimplified version of a topic tends to *vaccinate* them againt the real thing. Thus we prefer to present rather complete models or explanations of the topics even though they might seem much too complicated for most applications. We are sure the reader will agree that the model we present here falls within this category. We might quote our students complaints that we tend to add complexity to a situation: our response usually is that we are not adding it, but reflecting the nature of the phenomenon.

We have used our method teaching "non-IT" students as well as IT students, the main topics of what has been called Access Control (AC) of an information system (IS). We separate the design of a system's protection functions from the technical aspects of implementing them. We think of the latter as being the competence of IT students, a category which we define so as to include anybody who wishes to participate in such activities. The division is made in order to offer students the possibility of not having to deal with purely technical aspects such as database design, programming interfaces and processes, studying and implementing data scramblers and other concepts which we could roughly label as pertaining to the Computer Sciences. Though of course our method includes some ways to implement solutions, we will hardly mention them here. However, due to several suggestions in this sense, in the Appendix we have included a relational database schema to show how the model's entities could be implemented.

We use a brief description of system security focusing mainly on its objectives, to point out the scope of the presentation. A sequence of entities involved in the protection schemes follows. The whole point of our paper is to relate how a *necessity and solution* type of description greatly increased the very limited success we had previously achieved, when we tried to explain the model to our students. We have tried to limit remarks to the really helpful ones, a technique which might produce a sense of incompleteness in some readers. Since we think of these as colleagues – teachers – we assume that anybody interested enough in the matter will easily provide all the additional, or missing, detail using his own experience or most of all, thought.

We have omitted some of the main related topics for several reasons. We will not mention the topic of authentication, since we feel all students understand the need for it and know quite a bit about the ways to implement or achieve it. And we will only state that some type of data must be scrambled appropriately, meaning at a sufficient level of security, but shall not go into details even though we think of it is a fascinating subject.

A diagrammed non-technical description of our data model presents the main entities necessary to store information connected with the data protection devices. No database concepts are assumed or used, since no type of storage and access methods is implied or recommended. Finally, after presenting an example meant to remove some very likely doubts, brief mentions on some ways to implement such protection in the implementation of a system were included.

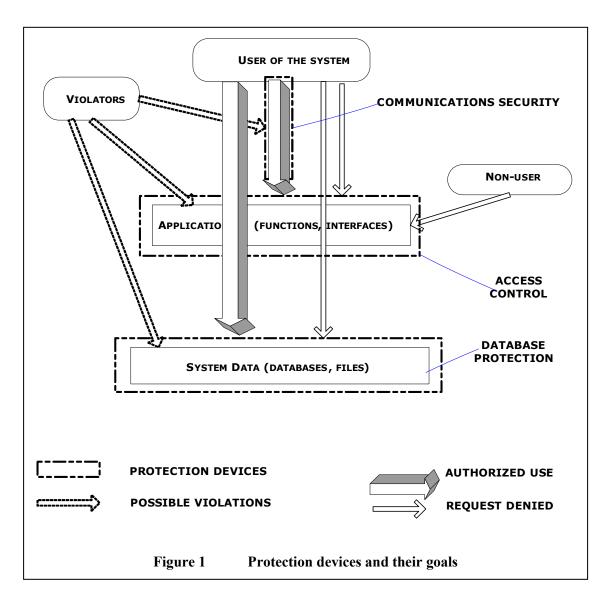
System Security

We define system security as everything that can be done to protect our data from unauthorized updates and use, including the detection of violations before it is "too late". We divide security into 2 main groups:

- restricting the use of the system's functions, meaning any use of the data which is performed through the application programs
- protecting the system's data from use by other programs or utilities.

In Figure 1 we show these 2 types of protection. In the paper we only discuss the devices built around the "application" itself, which are components of what is often called Access Control. Database security and protection of communications are of course just as important, but will not be discussed here. In other words, we will focus on a user's access to the functions of the system, which we could identify with routines of programs or objects through which certain functions are invoked.

We might point out that we have avoided using more powerful or general terms such as referring to subjects and objects. This was part of the strategy chosen for our teaching model: to stay away from other types of classifications which, though important or even essential, could divert attention from the main concepts we attempt to transmit. The reader will discover that throughout the paper we will not label our models or methods according to the increasingly used AC terminology, especially referring to RBAC (Role Based Access Control), though of course our model is just another example of that type of models. However, we feel that in many such models, the emphasis is on the "role" part of the model, whereas in our case this is not as important as the other concepts involved.



Access Control Components of a System

We have used successive layers, which we call levels, to present a working model which has proved to be adequate in many systems. Only the most significant data items of each entity - besides those which define it - are mentioned. The diagrams of the data models in the following sections should shed additional light on them. Additionally, the Tables illustrate our approach, in which we state a *necessity* and provide the *solution* or the entity defined to respond or address it. The sequence of concepts is crucial: presenting them in a different order repeatedly led to confusion and ultimate rejection of some of the main ideas by all our students. This is only partly due to the fact that the model differs from all others encountered before 2002. However, a recent paper which presents some of our concepts as part of their models is quoted as part of the comment on our DVDC terminology. When we say our model is different, we include published works, SW packages, developers tools and especially, the study of access control components found in many actual information systems.

As far as determining the cause of our difficulties on the part of our students, we blame previous knowledge of the subject, since they have been in touch with systems where they have a *profile*, in other words, they can do some things but are barred from others. When a new component is introduced at the wrong moment, the openness is at most very small, if not null.

Table 1 shows the first limitations we wish to impose as part of our access control and what we include in a system to address these needs.

NECESSITY	SOLUTION
Not everybody should be able to use the system	We introduce USERS (authentication)
Distinctions are needed in order to be able to limit users to certain functions	We define AC-FUNCTIONS: they are the ones we authorize or prohibit
A user may or may not be authorized for a function	We specify USER+AC-FUNCTION: User may use this function (true/false)

Table 1 Access control entities of a system – level l

A list of AC-functions must be prepared for every system. We have used the prefix to differentiate them from other uses of the term *function* in a system. Of course we could have used the more traditional designations used in the Access Control literature: subjects and objects. We wanted to distinguish our model clearly from others, since we thought these might have added confusion. Naturally in a class we could call attention to this fact or even provide both sets of terms. We found a very concise note on classifications and widely used terminology (Hiemstra, 2004).

An AC-function may, but does not have to, coincide with a function of the system or of the programs or procedures of the application. An AC-function can be very broad (use an entire subsystem) or quite particular (change the shipping date of today's orders.) Determining these functions should be based on their need, which we could describe by saying that a function is necessary if there is are at least two users of the system of which only one of them is authorized to use the function. Later on you may discover this specification might not be sufficient, but we think at this point it serves its purpose.

A user will have some privileges or permits, and as a consequence will be prohibited to use the other functions. This might be implemented the other way around, that is, including his constraints or prohibitions in the AC-data. In practice, often specifications for every one of the func-

tions are included, typically with a True-False value, the value of a numeric variable or even an old-fashioned bit indicator.

Introducing Constraints which Depend on the Value of a Data Item or Variable

A situation which often arises is that a user may be authorized for certain functions, but only "sometimes". This might depend on some external data, or the value of a data item of the system itself. For example, he might be able to use ac-function #23

Only in the morning or from a given terminal

Only for data concerning his department

Only for students of his courses

Only for his specialty (his topic, expertise)

In table 2 we indicate our solution to this type of need: we introduce a new type of constraints.

NECESSITY	SOLUTION
For certain combinations of a USER and an AC-function, privileges or prohibitions may depend on a data value	We introduce the entity: USER-ID + AC-FUNCTION + a particular DATA VALUE. For this function the user will be assigned either an Au- thorization or a P rohibition, which applies only to that data value.

Table 2Access control entities of a system - level 2

Remarks

We have called these constraints Data Value Dependent Constraints (DVDC). We cling to this terminology even though these are now widely referred to as *context constraints*. When we first formulated them and included them in our model, around 1986, we found no literature on the subject. Further research led to the same results in 1998, and finally when we first drafted this paper in 2002. However in 2003 a significant paper (Neumann & Strembeck, 2003) appeared which dealt with context constraints in RBAC models, though they mention another work dating further back (Nitsche, Holbein, Morger, & Teufel, 1998). A partial justification could be that we think of these constraints in the context of an Information System, hence related to values of variables which are related to system data, whilst context constraints include these – data-dependent ones - in the larger set of circumstances, environment, time period or other constraints.

In the models, every DVDC has an identifier, which we call its key, indicating to what it applies, and an indication which specifies "A" for authorized or "P" for prohibited (since either way of imposing these constraints are supported), and a data value. The key will have additional components after we allow for groups and roles. Though we explain this quite thoroughly below, an example at this point might be helpful. The "key" U25 + F37 + "P" + "OAK" would indicate that our user 25 cannot (note the "P") use AC-function number 37 if the value of the variable is "OAK". In the technical model, we solve uniqueness requirements arising from the fact that several values could share the same key. Applicable value descriptions including wild characters might be useful in some cases.

These constraints seem necessary, even essential, in some situations. The topic is central to our model, but constitutes the greatest challenge as far as explaining it, and therefore will be discussed further in the rest of the paper.

Later discussion will deal with the identification of the type of data involved. For example "OAK" will mean something only to the persons who defined the constraints, not the AC-programs: these will simply check values. Thus furnishing the correct value to be compared with a DVDC is usually included in the application programs, where this term pretends to separate them from the infrastructure components.

Several values for the same user-function combination

Often instead of just one value being allowed, there might be a list of such values. Of course the dual situation will also arise - several values have to be prohibited. Thus a DVDC may have a set of values which apply to the same key. We might think of a list of values to be allowed or prohibited for a given key. Again the use of wild characters may sometimes shorten such lists.

A criterion will be necessary to deal with the possibility of several different values applying to the same instance. We introduce the rule:

If a "P"-type constraint is present for a combination of a user and an AC-function, any value not also present as a "P" constraint will be ALLOWED (i.e. not prohibited). Conversely, if an "A" DVDC is present, any value not included with an "A" will be prohibited for that combination of user and function.

This implies another rule – the reader probably has formulated it for himself: If a "P"-type constraint is included for a user-function combination, no "A"-type constraints can be added for the same combination.

We shall see below that DVDC's can be introduced in such a way as to apply to several cases: many functions or several users.

Groups of AC-functions

In a large system and especially in modern totally integrated systems, the number of such ACfunctions might be very large. Many users will only be associated to certain parts of the system, so we introduce a subdivision of the functions into subsets, which we shall call groups. We define a **group of functions** as a subset of AC-functions. Groups constitute a partition of the functions, and are mutually exclusive. We like to number the groups, but of course they may have a label or a name. The functions of a group are numbered within the group. When using groups, it will be meaningless to refer to a function number without specifying the group to which it belongs.

In Table 3 we describe the entities added to our model to include groups of functions.

NECESSITY	SOLUTION
It might prove useful to subdivide the functions	We introduce GROUPS of AC-FUNCTIONS
We want to limit users to certain groups	We introduce the USER-GROUP entity
We want to limit the users to use only certain functions of each group	We introduce USER-GROUP-FUNCTION: may use (True/False)

Table 3	Access control entities of a system - level 3
---------	---

Remarks

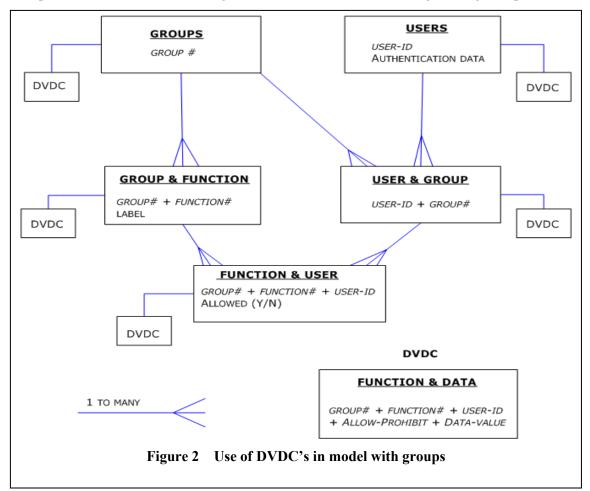
The groups do not increase the functionality of our AC model in any way: they will only make it more manageable in many ways.

We emphasize that a user may be associated to many groups, meaning he may be authorized to use functions of several groups.

Groups have an unfortunate by-product: it is easy to talk about a user being a member of a group, thus implying that a group is a subset of users. Actually it is quite difficult to stay away from this terminology, which fortunately is only confusing when studying the subject, not in applications. We could say that a group is both a subset of the functions and a subset of the users, but this kind of statement does not contribute to understanding and actually acts against a correct application of the concept. In many references the term *domain* is used to signify these groups, though not necessarily in exactly the same sense in which we use it, a fact that led us not to adopt that terminology.

The model at this stage

Of course the introduction of groups of functions has an effect on our DVDC's. The function will now be qualified by the group number. Thus, the key is now user-id + group # + function #. This in turn makes it possible to introduce a constraint without a function number, that is, it will apply to every function of the group. Thus, user-id + group 5 + function 0 would apply to any function of the group. The diagram in Figure 2 shows the AC-entities included in our model, and how they relate to each other. Neither this diagram nor the one presented later in Figure 3 implies a particular data model. It might help the reader to think of some of these entities as tables of a relational database, though the files in a given application may be different from those indicated or implied here. Notations in the diagram follow no standards; we thought it might help to indi-



cate a type "1 to many" relationship, for example, to show that 1 user will (or can) be in several groups. Every rectangle simply represents an entity of the access control method; for each entity, we have italicized a potential primary index should relational database tables be used. A word about the DVDC boxes might be useful. We have just included them every place where they could be appropriate, but this association is logical, meaning you probably will not have a table in a database which contains "user and group DVDC's". Instead, you might have some of these constraints which may apply to one particular combination of a user and GROUP, meaning to all functions of the group.

Introducing roles within the groups

Since many users of a system may need similar privileges, roles have become popular to assign them on a collective basis. As we have said before, we will not discuss the many advantages and attributes of such Role Based Access Control models, though of course we have the same reasons to use roles. We must warn the readers that we use this term in a different context than many RBAC models. We define a ROLE as a subset of users within a particular group. Some groups may use these roles while others may not need them. In Table 4 we add the entities which will allow us to use such roles.

NECESSITY	SOLUTION
For each group, are there many users with the same privileges?	We introduce Roles of a group
We must authorize "roles" instead of individual users for the functions of that group	We introduce authorizations for GROUP- ROLE-AC-FUNCTION May use(True/False)
Every user of a group which uses roles must "belong" to one and only one of the roles	We use the USER-GROUP entity to specify the role of a user in a group

Table 4Access control entities of a system - level 4

Remarks

If a group uses roles, no authorization at the user-group-function level can be assigned. The user inherits the privileges from his role in the group. We repeat that you may use groups which do not use or need roles. Of course this not only generalizes the model, but also complicates it a bit.

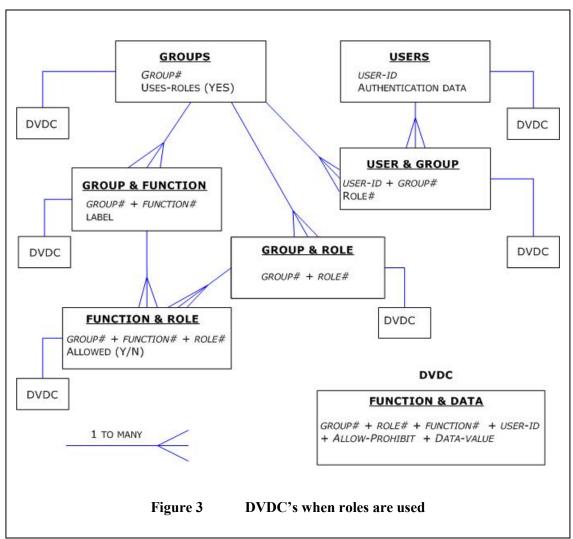
We number the roles within groups. As usual, labels or names are recommended for them as well.

Remember that it does not make sense to talk about a role without mentioning the corresponding group.

Figure 3 reflects the impact of the use of roles on our model.

Roles not only will save work when assigning authorizations or allow for consistent updates - the latter of course being one of the main reasons to use such roles. They may also constitute a valuable way to constrain certain privileges via DVDC's associated to these roles, as will be seen in the final model, which we present below, where the key to a DVDC assumes its final form:

user-id + group # + role # + function-#



The role number will be zero if the group does not use roles. Of course it will also be zero if the user is specified. But now we could prepare a DVDC for an entire role of a group:

No user specified + group 5 + role 2 + function 5

would apply to any user of that role (#2) of that group (#5). The function could also be zero. Note that even if a group uses roles, a DVDC can still be formulated for one of its functions for only one particular user.

The following diagram shows the impact of the use of roles on our model. Groups which do not use roles were not shown, but of course can occur.

AC Designer Activities for a System

There are two basic approaches to the design of the AC components for a particular system using our suggested model and procedures. The designer may see how he applies the model, and then decide which components he will use, and assign the particular values for the system. Or he may design his protection scheme and then see how he uses the model to implement it. Of course in this case he might not be able to do so, if he uses entities or relations which are not supported by the model.

We recommend the first way, since we think that adherence to the model will simplify the task, thus improving the chances of producing a good AC-system, even without a lot of experience in the matter. However, it is neither easy nor practical to try to define things in the order in which we introduced them, since this ordering was especially conceived to explain them. First define your groups of AC-functions – of course you can think of it as a first version, since you might change your mind. Next you can tackle roles, where it makes sense to decide initially for which groups you will need them, based primarily on the number of users associated with each group. Even though we have not included it in our topics, these roles may be especially useful in conjunction with DDL "grants" or "revokes" implemented via the database software.

Finally, those constraints which involve the value of a data item or variable should be studied. Of course there are many ways to introduce such constraints. A straightforward alternative to our DVDC's is to subdivide the AC-functions into others, depending on the value of some data item. For example "update credit limits of customers" might be replaced by "update credit limits of customers of Division A" and another similar one for Division B. But this would not be a good alternative if you had to provide a function for every city in a nationwide business enterprise.

Another way to introduce such constraints is to include them in the programs. Thus the authorization to use the function would be added to the validation of the data value associated to the transaction. We separate this type of validation from the functions of our access control model, since we would like to implement the latter without intervention, even knowledge, of the programmers who develop the application itself.

In the example below we shed additional light on these possibilities. Furnishing several examples to illustrate different situations and the consequent changes in the approach to the designs was not only impossible due to the format of the paper, but also considered unnecessary.

We add a comment about the explanations provided in the example. We feel that many additional explanations of the somewhat cryptic indications would make the example easier to read, but not necessarily contribute to understanding it. Since we have directed this paper to potential teachers or perhaps some system designers who could use the model outlined here, we have left the burden of understanding some pretty *concise* indications to the reader, though we have not assumed any specific knowledge except the concepts contained in our paper. Thus, it might be useful to be able to refer to a diagram of the model when reading some descriptions or examples, instead of just trusting one's memory.

An Example

We used a system where we could show examples of most of the entities without having to include lengthy, detailed or complete explanations of concepts and relations involved. Let us point out that we use quotes, whenever a term is used with the precise meaning it has in the system. We are to design the AC-components of a system for a large bus company, which has many routes and each of these offers a number of "trips". Besides the assignment of drivers and buses to every trip, and the scheduling of the trips, the system allows all personnel involved to report any abnormal or modified circumstance concerning the availability of any of the resources, but also about driving conditions, street or highway repairs or temporary noteworthy information – such as large holes or depressions, slippery conditions and others. Of course anything regarding the buses' mechanical condition is also reported through the same system.

We start thinking about the *actors* of the system: in the AC context, they will probably constitute types of users. BUS DRIVERS will notify their availability or lack of such due to holidays, illness or other factors. They may tentatively schedule themselves for a particular set of trips,

though the official schedulers can change these assignments. And they will report anything interesting they have to say.

SCHEDULERS introduce routes and trips, assign vehicles (buses) and drivers, and handle all kinds of exceptions, besides reading and processing all the reports either by taking some actions or by forwarding them to the appropriate department of the company.

PERSONNEL department employees will update driver data, and the EQUIPMENT division will indicate the availability of the buses.

Finally, SYSTEM ADMINISTRATORS and some other "users" will be in charge of the AC components themselves, typically assigning or updating privileges and prohibitions.

Anybody belonging to one or more of these "actor categories" will have to be a USER of the system in the AC sense of the word. Bus drivers, besides being such users, are also entities of the system itself: there will be a table of bus drivers. This allows us to include some AC functions as part of the programs themselves, a delicate matter which should not be used too often. Of course we will have to establish a connection between the data of the AC user and the bus driver, since being the same individual obviously does not furnish this connection. For example, in the USER file we might include a data item indicating the identification of the bus driver, as such, in the system, and upon sign-on we would make this data available to the application programs.

Some of the AC-function groups could be (we usually number them for easy reference and use):

- 1. Route and trip data
- 2. Bus driver data
- 3. Equipment (buses) data
- 4. Driver assignments to trips
- 5. Assigning buses to trips
- 6. Observations about trips, equipment
- 7. Updating AC data (administration)
- 8. other groups not used in the example.

Group 2 could contain the following functions (they will not be used in the example, but are provided precisely as an example of such functions.)

- 2.1 Introduce new drivers
- 2.2 Eliminate drivers from the system
- 2.3 Update general information (address, phone)
- 2.4 Update availability
- 2.5 Assign holiday periods

We could use ROLES in groups 2 through 6, since there might be a considerable number of users who will have to be granted permission for the functions of that group. This is what we mean by being associated with the group. The roles seem attractive if only to save a bit or work. For example, in group 6 (Observations about trips, equipment) we could formulate the following ROLES:

schedulers

bus drivers

mechanics and other equipment inspectors

special task force to follow up tips and needs.

Note that these roles sometimes will, but do not have to, coincide with the descriptions of the functions. Remember that the roles are just a shortcut to individual user-group-function authorization: we assign privileges to the roles of the group and then indicate to which role the user belongs as a member of that group. Please avoid the confusion resulting from different uses of the term roles in other models, where a user often is assigned a unique role. Here he will have a role in every group with which he is associated, but of course only if the group uses roles.

We could constrain the bus drivers to their own data in two ways. We could ask the developers of the system to include constraints in their programs, through which any bus driver would be prohibited to use any function if it involves any other bus driver, i.e. he can only see and update his own data. Actually, in many systems these validations are performed by the functions of the system itself, rather than through AC-prohibitions. This is probably the reason why few people have used what we have called DVDC's in their AC-devices and they have taken so long to appear in the literature. The need to plan for changes or additional constraints if obvious, and is precisely the reason for the inclusion of a DVDC:

User: the user-id of the bus driver (his AC user id)

Group: 0 this indicates the constraint applies to all groups

Role 0 all roles of the group or "not applicable". If a user-id is specified for a DVDC, no role number is used, and vice-versa.

Function 0 it has to be 0, since the function number is nested within the groups. 0 means all functions.

"A" meaning: authorize when the data value coincides with the requested one

data value the bus driver's id as a system data item, that is, his bus driver number or whatever the system uses.

The programs would include the data value in any request associated to a bus driver made to the authorization function, which is described in the next section.

Similarly, in order to limit a scheduler's privileges to trips of certain routes we could include 3 DVDC's, one for each group:

User	the AC user-id of the scheduler
Group	1 or 4 or 5 (only 1 value in 1 DVDC)
Role:	0 (all roles or the group uses no roles)
Function	0 (all roles of the group)
"A"	

data value: route1, route2 the routes he manages. We can supply a list of values.

Whenever the application programs would have to validate the user's privilege to schedule a trip, the route of the trip would be added to the validation request. Once again, please refer to the validation function, since we only included it to allow such references to clarify our descriptions.

The Authorization Function

To make sure that the main concepts have been interpreted in their proper context, an access validation function is described. We may suppose here that somebody furnished this routine to the developers of the application, together with the file structure to store the AC data and the programs used to update such data, or it might be part of a program library.

FUNCTION MAY_HE_DO_THIS (user-id, group, function, Optional: data_value)

RESULT: an integer according to the following table

0	=	permission granted (meaning, of course, he is AC authorized)
1	=	function is prohibited for the user
2	=	the data value not included, but other values appear as "A" value
3	=	the data value appears as a "P" value applicable to the user-function

The function returns the cause of denial to the calling program because often it might be incorrect to tell a user he may not use a function, when he is only limited to certain data values.

Note that the first three parameters are required. On the other hand, the presence of a non-null value of the (optional) data value parameter will indicate that the DVDC's are to be checked. Though probably unnecessary, let us remark that the applicable role is not included in the parameter list, since it is determined by the combination of the user-id and group number.

The function will first check if the user is associated to that group (of course, if he is not, the user may not use the function.) If he is, there are 2 cases. If the group does not use roles, the value of the user's authorization for that particular function number of the group is found. Should the group use roles, the user will have a "role number" in that group, which is used to find the YES-NO value of the group-role-function number. If the result is 1 (denied) or the data value parameter is null, the function ends.

If the result is 0 (user is authorized) we must check the data value parameter. If the function is invoked including a particular data value, it will look for applicable DVDC's, where of course this applicability means: any DVDC none of whose key items violate the ones furnished as parameters of the function. We ask you to note the inverse formulation, which is inherent to the model.

There are 3 possibilities. Here we use the term *applicable* to indicate that none of the items of the key of the DVDC differs from the ones in the request.

No applicable DVDC exists for the combination of user, group and function: this means we return the value of 0 (granted), since no contrary evidence was found. Observe that "user" will be replaced by role if appropriate (for this combination of user and group).

An applicable DVDC of the "A" type was found. We look at all such constraints (only the ones which apply, naturally) until we find one with the same data value as the one invoked. The result will be to grant permission if we find it (result = 0), and to prohibit the value (result = 2) if we could not come up with the correct constraint. The underlying principle is: if a set of values is authorized, all others are – implicitly - prohibited.

An applicable DVDC of the "P" type was found. Now we look for a constraint with the same data value, except that now we will PROHIBIT (value 3) if we find a matching constraint, but authorize (result 0) if we cannot find one. Values not expressly prohibited are allowed implicitly. Reminder: we will never find a "P" and an "A" type constraint both applicable to the same usergroup-function combination. We will mention a connected issue in the section regarding the administrator of the AC components and data.

Impact of the DVDC's on Related Activities

Introducing constraints that depend on the value of a variable or data item modifies the way we invoke AC-devices in an application program. Though some of the situations are mentioned, details are left to the interested reader.

Two ways are frequently used to include prohibitions in a transaction:

- check for permission when the user tries to invoke a certain function of the system (and inform the user of a denial)
- only offer choices for which the current user has permission.

We add two different situations which arise when the choices are offered (a menu, options, buttons or even selection of a certain value):

- the system knows the value which will be checked for a DVDC, and thus the constraint can be taken into account
- this value will be updated or obtained later.

This affects the design of the transactions and probably, the amount of programming and attention necessary to invoke the protection devices. The very popular process of disabling disallowed options - for example menu items or other objects of a form - at load time, would now have to check for additional prohibitions. Another way is to postpone enable-disable decisions until the value which will affect the choice is known, or alternatively determine that value before the decisions have to be made. In some cases this is easy or even natural, as in our example when a bus driver signs on and thus provides the system with the data it needs about him, typically his driver number. But this might not be the case with the supervisors, who may have several routes and thus authorization can only be checked after the system knows which route is involved in the particular transaction.

The other rather significant effect of the use of DVDC's is regarding activities which will be performed by the administrators of the system. During the design we should take into account that updating the data used by the AC procedures might be quite a task, especially if frequent changes occur or the model requires complicated privileges. During this design, we like to remember that the system administrators (let's call them that)

- are not experts on the subject
- have many other tasks and assignments, and therefore will not remember everything you ask them to know
- make mistakes, just as we would
- handle this very delicate data
- and might be led into temptation...

The usual "that's what he gets paid for" argument is not a very productive alternative. The interface programs should, of course, be flexible and provide ways to avoid errors wherever possible. Presenting a screen we encountered recently, where the user has to indicate TRUE or FALSE for a set of 154 items, is a nasty thing likely to irritate anybody. Even if each item has a description or tool-tip, it is not a good idea. Our division in groups was initially motivated by a (very large) system which, even after trimming, ended up with more than 200 functions to be authorized or not. Labeling groups, roles and functions might be quite a task, but it is very worthwhile, especially when we compare it to the use of a printed (or screen-displayable) catalogue.

The biggest difficulties arise in conjunction with the awkward but necessary DVDC's. The programs do not know the meaning of the data values: they don't have to. But the administrators better know what they mean, or at least some other person who requests the changes. A good AC system will include descriptions of the type of variable or data item associated to them. Since another problem consists in preventing invalid values, especially contradictory values of DVDC's – a matter that poses some interesting program design questions – the SW provided for such updates has to be perfect in this sense. For these and several other reasons it might be a good idea to use existing SW instead of designing, coding and debugging your own. A good AC system for a large IS is not only delicate, it's a lot of work!

Conclusions

The method proposed may arouse the interest of students in this topic, which traditionally is only covered very slightly. Teachers know that some ideas are hard to transmit, mainly due to different attitudes toward the subject. We hope that other researchers and teachers will add to our suggestions. Many topics were not included in the paper though they have been the object of a lot of research. How to implement some of these security devices in modern systems, where users have access to the data coding their own inquiries and updates, and with the ever increasing flexibility and mobility of systems and the access to its data, should be a fascinating subject to many students in the IT field. We do not provide many references since this is not the purpose of this paper.

One of the main topics of our paper is the inclusion of privileges or prohibitions which depend on some value of a data item, or perhaps a variable (it could be the name of a terminal, the time of day or even the weather conditions.) The term context constraints has recently been used in this sense. One could easily say that in most instances, the need for this type of constraint is very closely associated to the functionality of the system, rather than to security considerations. We heartily agree with this argument, but also recognize the danger of allowing the application to handle all such situations, since we are always concerned about the power or potential ways the developers of a system may have or plan to violate protection devices or validations of the system in their benefit.

Let us state this from a different point of view: the owners of the system – as we like to call those who will suffer directly from malfunctions or violations – should protect themselves as much as possible from insiders and IT specialists. By increasing the functions performed by an AC sub-system and controlling the quality and confidentiality of this component they may achieve a significant reduction of certain risks. But who will protect the system which protects the AC components of a system? In the typical database protection, another database is used to store the privileges or constraints; this has to be protected against unauthorized updates, of course, and usually the same DBA (database administrator) will perform these functions, which sheds some light on our concern about it.

We recommend using third party SW for the security aspects of a system. For example, our own design of such components is not only totally secret, meaning nobody knows the entire scheme, much less the details of the algorithms used to scramble texts, but also personalized so that owning a copy of the SW will not enable you to attempt – even with great computational effort – to violate the system somewhere else.

References

- Hiemstra, J. (2004). Access control models. Retrieved November 4, 2004 from http://www.techexams.net/technotes/securityplus/mac_dac_rbac.shtml
- Maciaszek, L. A. & Owoc, M. L. (2001). Designing application authorizations. Proceedings of the 2001 Informing Science Conference, June 19-22, 2001.
- Neumann, G. & Strembeck, M. (2003). An approach to engineer and enforce context constraints in an RBAC environment. Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT), Como, Italy, June, 2003
- Nitsche, U., Holbein, R., Morger, O., & Teufel, S. (1998). Realization of a context-dependent access control mechanism on a commercial platform. *Proceedings of the 14th International Information Security Conference* (IFIP/Sec'98, part of the 15th IFIP World Computer Congress) (Vienna/Budapest, Austria/Hungary, 1998), Austrian Computer Society, Vienna, pp. 160-170.
- Osborn, S., Sandhu, R. & Munawer, Q. (2000). Configuring role based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, *3*(2), 85-106.

Appendix

In order to provide a guideline towards the implementation of the model, we provide the design of the tables. This in no way conveys an acceptance of this as a good way to do things, since we never use such structures when some amount of secrecy is involved. The model we actually include in our toolset will be available in August 2005 on our "Mopsite" (now in the process of relocation, where MOP stands for "My Own Publishing site", is a computer software we developed). The model is based on a mixture of database tables and random files, which we consider better for such purposes though they imply more programming and design loads. Interested readers should get in touch with the author who will provide the exact name of the site.

We feel we should warn the reader that the diagrams of Figures 2 and 3 in the paper – which were kept simple for the benefit of making them less awkward – do not correspond to this model. This should be obvious since the "entity" DVDC appears several times in the diagrams, as we pointed out in Figure 2.

An Implementation of the Model Using a Relational Database

Figure 4 shows a possible set of relational database tables which might be considered for an implementation of the model.

Just in case, we will comment briefly on each table, and shall seize the opportunity to indicate other data items (fields, columns) which must or might be included.

USERS: every user will be associated to a unique "id" (this may or may not be a numerical item) and we like to provide him with a "Nickname". Thus he can sign in either with his id or keying in the nickname, which he picked out himself. Additionally, we will want to store some information about the way the user will prove he is the one who is starting a session, so we add a field (or more than one) to store whatever information we will need. Of the course the best known would be a password, but questions and answers, the name of a file which contains a voice-print, finger-print or other biometrical data may be stored. Naturally all these items should be stored in some very adequately scrambled manner. We usually add what we call an AUDIT field, where we include a number which is obtained by performing some calculations (or other algorithms) on sensitive data so as to detect any non-authorized changes.

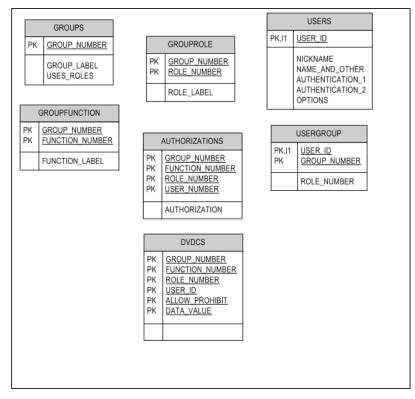


Figure 4 A relational database implementation of the model

Optionally, we may add options (the language he prefers, if he wants to be treated on a first name basis or not, if he wants to use pull down menus or not, whatever the system designers choose to include.) The user's name, address or place of work and many other information items may or should be added, but they play no real part in our access control functions.

GROUPS: we store the group number (remember the AC-functions are introduced as members of one of the groups.) If a system should not need such groups, the group number 1 (the only one, alas) is used. A label is useful, of course, in order to guide those that will assign functions, and later the administrators or whoever assigns privileges to users and avoid having to refer to notes.

As we pointed out, the use of roles associated with every group is optional, so we include a field "Uses_roles" (typically a True-False field) to indicate precisely that. As you already know, a value of FALSE (indicating no roles will be used in this group) means that users will be assigned their privileges for this group one by one (every user individually.)

GROUPFUNCTION. Remember function numbers are nested within their groups. Again we add a label or function name to serve as a reference when assigning authorizations to users for the function.

GROUPROLE: For each group, we define ROLES. We just store a label or role's name, reminding you that this role is used only for this particular group. Please see USERGROUP for more detail.

USERGROUP: we will include a record (row of the table) for every group for which the user is authorized to use at least one of the group's functions. If that group "uses roles", we will indicate the user's ROLE in that particular group. At this point things start getting confused, and we rec-

ommend referring to the diagram presented in Figure 3 (do not use Figure 2, since it has NO ROLES!) This should also help you realize a user "plays" a unique role in every group.

AUTHORIZATIONS: at last we arrive at this one, which in many access control models seems to be the only table (when combined with the user.) It will specify the functions for which a user is authorized, but in a somewhat awkward way. First we provide the function, preceded by its group since the function number does not make sense by itself. Then we say: if there are roles in the group, the permit is assigned to the role (and therefore, to every user of that role in that group). If not, we will include a user-id and assign the permit to that user.

In the past we used the "prohibition by absence" way of indicating authorizations, thus saving a field. We have long ago adopted the criterion of including an indicator (yes-no) to indicate that the user (or role) can use the particular function. Of course everybody will implement this the way he likes better, or more frequently, in a manner with which he is familiar as well as asking that it works.

DVDC's: here we include all the Data Value Dependent Constraints. The large key has an advantage (there are no other fields!) but of course requires explanation, especially since we never expect people – much less students - to remember what we have said or written.

This way of storing the constraints allows all kinds of combinations. You could for example include a constraint that one particular user can only use a function of a group (any one of them) if the value of a particular variable is equal to his NICKNAME. Or you could bar a role from a function if the value of another variable were not "TODAY".

The only mandatory data, besides the ALLOW_PROHIBIT field is the GROUP NUMBER. This was imposed by the model's creator, since we could actually allow "0" for that field, meaning any. But this in turn would imply assigning a particular user, since roles do not make sense without confining them to one group (mind you, in OUR model, not in other ones you might encounter elsewhere.)

Look at the key, and imagine that Function, Role and user-id may be "null" (we use a zero or blanks, but you can adapt this to your DBMS.) Function 0 of course means "all functions of the group".

Now let's consider the other fields of the key. ALLOW_PROHIBIT is the type of data item which has made this author extremely unpopular amongst his programmers, since they tend to have difficulty working this way, though its use is extensive in modern computing, for example in the many include-or-exclude facilities of software products. If we provide a DATA VALUE and indicate a "P" (prohibit, watch the initial, since you might think of permission) then the user (or users of the role) will not be able to use the function or functions referred to by this constraint for that data value. On the other hand, if this field has the value "A", the opposite is true: he has been granted specific permission for this data value. At first glance, or even at some subsequent superficial ones, this constraint seems to be "obviously redundant" since should it be absent, the user could use the function anyway. However, the presence of an "A" (authorize) constraint has exactly the opposite effect: it will exclude the user's ability to use the function for other values.

We have mentioned in the paper that you can include several data values for the same key (meaning the other items of the main key). You may "of course" NOT include a "P" and "A" constraint applicable to the same user-function combination, even though may not be immediately obvious by the keys. The reason you may not do this is left to the reader.

Finally, we may use "patterns" instead of data-values. Suppose you are working in a store, and the owner does not wish you (user-id number 12345) to mess around with his copper pipes. All pipes have a code that starts with "PI" followed by a number, and then end with the MATERIAL

used to manufacture them (copper, plastic, whatever.) The copper ones all end with "CU". We would include a constraint (please suppose there are 3 groups, but you only have to do with functions of group 2).

GROUP: 2 FUNCTION: 0 ROLE: 0 USER-ID: 12345 ALLOW PERMIT: "P"

Data-value: "PI*CU" where * stands for a string of characters of any length.

Should he include plastic bags as well, he would add another constraint where the only difference would be data-value "PLABAG" supposing this is what they call them.

Note that this means that for any function you "have" (before checking for a DVDC, the system will find out if you can use a particular function of the system) you will be allowed to use it for any other data value, provided there is not *another such constraint* active.

We say this to make you think a bit. You may *not* include an "A" type constraint for the user, since doing so will allow all other values, which will be inconsistent with the prohibition you just introduced via the "P" constraints. Let me point out that this transforms the interfaces needed to update (include) the DVDC's into quite interesting design problems.

A comment on the database design: we have already stated that we do not like this schema very much, not only because it does not conform to any standards at all. Actually we use "vectors" to completely eliminate any semblance of normalization. Let us describe but one such string of values. Instead of creating a record or row for each combination of USER and GROUP (as in our model shown above) we use a field which contains a string of digits, or if necessary, of numbers. Each one represents the ROLE which the user has in that group. We actually include this string in the USER entity. Let me give you an example since after reading this I discovered a) I did not understand what it meant, and b) I could not explain it better.

Suppose your system uses 18 groups, and the greatest role number you use is 14. (I don't like that many groups myself, but YOU created it.) Then the user-group-role string will look something like that:

User-group-role = "030600001300000707030200000402050000"

Refers to group 1 2 3 4 5 6 7 8

You may separate this into 18 2-digit numbers. The meaning will be group 1: role 3, group 2: role 6, groups 3 and 4, not authorized, Group 5: role 13.

Then group number to which the roles refer are pointed out below for our ease. Of course this is not part of the data we would store.

In the specific problem of storing DVDC's, we usually use vectors or we concatenate several values with a separator. Neither is a good idea if you want to avoid programming and confusion, but both are very good ways to avoid huge tables if you have a very large system, with lots of constraints. The first time I needed such a model was in an Accounting System where almost every user had all kinds of constraints, and even the use of patterns did not avoid the presence of several tens of thousands of constraints. Oh, we forgot to mention, the users (actually a set of administrators of the accounting data) introduced these constraints themselves. Believe us they did not know or care how we stored them, but the machine did, and thanked us profusely via excellent performance.

Biography



John Bauer Mengelberg, after obtaining a degree in Mathematics at the Universidad de Buenos Aires, Argentina, got a PhD in Statistics and Operations Research at the University of Wisconsin, at Madison, where he also taught courses in the area of Stochastic Programming. He has since worked in Mexico, where besides teaching at the Colegio de Postgraduados, a school primarily involved in the field of Agronomy but which has departments of Statistics and of Applied Computing, he has held several positions, always connected with the field of Information Systems, in which he has also been a consultant all his professional life. He is primarily conerned with the subject of "systems that work", a concept he has extended to signify that they work even in abnormal circumstances. He has created and

implemented many computer packages, and is currently working on several software products involved with publishing papers or books in the electronic media, and what he calls subsetting in very large data collections. He has often complained he has to work by himself, and his main interest in attending conferences seems to consist in finding ways to change this.