

Last modified on September 18, 2003

Chapter 4. Security models

Security models are a more precise and detailed expression of policies and are used as guidelines to build and evaluate systems. Models can be discretionary or mandatory. In a *discretionary* model, holders of rights can be allowed to transfer them at their discretion. In a *mandatory* model only designated roles are allowed to grant rights and users cannot transfer them. An orthogonal classification divides models into those based on the access matrix, Role-Based Access Control, and multilevel models. The first two of these models control access while the last one attempts to control information flow.

The access matrix

Although it was introduced as a model for operating systems security [Lam71], the access matrix (AM) is a security model that can be applied to any system. In its original form it defines a discretionary model but it can be restricted to make it a mandatory model. It can control both confidentiality and integrity.

The model defines a set of *subjects* S (requesting entities), a set of *protection objects*¹ O (requested entities), and a set of *access types* T (the way in which the object can be accessed). In an operating system, the subjects are processes, the objects are system resources, and the access types are usually read, write, and execute. In a DBMS, the subjects are users, the objects are database items, and the access types are retrieve, update, insert, and delete. In object-oriented systems, the protection objects are classes or objects and the access types are class operations (methods). A combination (subject, protection object, access type) or (s,o,t) is an *authorization rule*. A pattern to describe authorization rules is given in Figure 4.1.

An extended access model may include also: a predicate, a condition, a copy flag, an authorizer. The predicate defines content-dependent constraints on the access, the condition establishes a condition which must be true for the rule to apply (a guard), the copy flag if 'on' authorizes the subject of the rule to grant the right to other users, and the authorizer indicated who performed this authorization. In this case, the authorization rule has the form (a, s, o, t, p, c, f). Database systems typically use a subset (s, o, t, p).

In the original matrix of Lampson [Lam71], there was the concept of owner, which as we have discussed in the last chapter, is a violation of the principle of separation of duty. There was also the concept of "controller", defining special rights of a process over another.

The Lampson matrix and its extension [Gra72], include operations to modify the matrix and to allow propagation of rights. We call these operations "administrative" operations because they would normally be used by a security administrator. These include:

transfer{r/r*} to (s,o) (transfer can be destructive or not, depending on the policy)

¹ In the literature these are called just objects, we use the name security or protection object to distinguish these from objects in object-oriented design.

grant{r/r*} to (s,o)
delete r from (s,o)
read(s,o)
create object o
delete object o
create subject s
delete subject s

Harrison et al. [Har76], extended and formalized this model (the HRU model), to prove safety properties. The main difference between this model and the access matrix described above is the way the matrix is changed. They use very much the same operations but they add a set of commands to apply these operations. A command has the structure:

Command $c(x_1, x_2, \dots, x_k)$
if r_1 in $M(s_1, o_1)$ and
if r_2 in $M(s_2, o_2)$ and
.....
if r_m in $M(s_m, o_m)$
then op_1, op_2, \dots, op_n
end

In particular they proved that the general safety problem for the access matrix is undecidable. This means that starting from an initial state where subject s does not have a right t for object o , it is not possible to decide if s can get the right in a given number of steps. Several variations of the access matrix have been proposed trying to get a model where safety is decidable. We know that in a simpler model, the take-grant model [Sny81], safety is decidable, the problem is to find something in between take-grant and access matrix where safety is still decidable. A more practical approach is to use a mandatory version of the access matrix, such as RBAC, where general users cannot transfer their rights.

An implementation of the access matrix must have a way to properly store the authorization rules, intercept user or program requests, and to compare the request to the access matrix to decide access. This interceptor is the *reference monitor*. A request (s', o', t') is compared by the reference monitor with the access rules. If there is a corresponding (s, o, t) , the access is validated and the request is completed; otherwise the request is denied. A pattern for the Reference Monitor is given at the end of the chapter.

Role-Based Access Control (RBAC)

RBAC can be seen as a variation of the access matrix, where subjects can only be roles. A role corresponds to a job or to functions within a job; for example, a professor may have the roles of teacher, researcher, advisor, thesis chairman, and others. Rights are assigned to roles, not to individuals. If users are assigned to roles and given rights only by an administrator this becomes a type of mandatory model. RBAC can conveniently implement the policies of least privilege and separation of duties. Least privilege can be

implemented by assigning to each role only the rights it needs to perform its functions. Separation of duties can be implemented through mutually exclusive roles. A pattern for RBAC is shown in Figure 4.2 [Fer01].

One way to enforce the least privilege policy is to assign rights to roles from use cases [Fer97]. The use cases for a system define all the needed accesses for its roles; the union of the rights for a role over all the use cases are its needed rights. Roles can be structured and different rights inheritance policies can then be defined [Fer94, San96]. A good amount of material on RBAC can be found in the pages of the NIST [nist] or the Laboratory for Information Security Technology at George Mason University [list]. A good early survey is [San96].

Implicit authorization

For convenience we may need to group subjects, protection objects, and some access types may imply others in some ordering structure. In this case, a rule component may be implied by another rule and this must be taken into account when evaluating access.

The concept of implicit authorization was introduced in [Fer75] and subsequently has been developed in some research projects and used in the later versions of the Windows operating system (starting with NT Version 4.0). We discuss this idea in more detail in Chapters 10 and 11

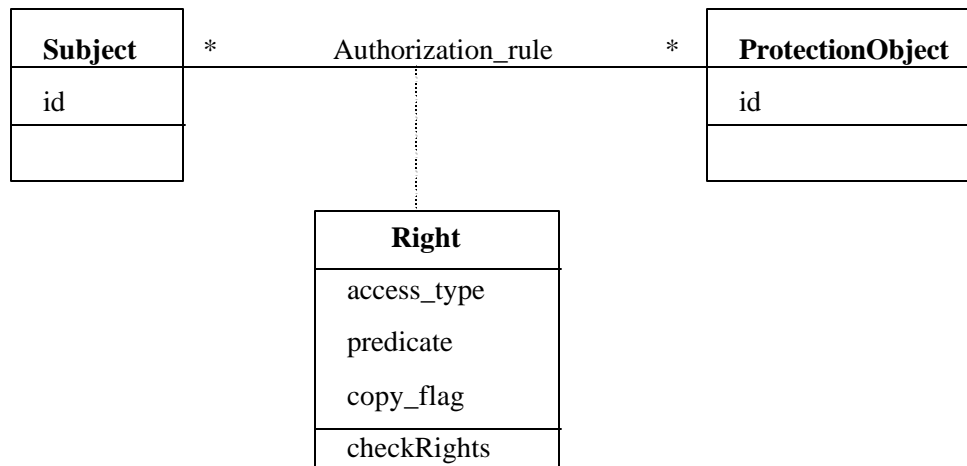


Figure 4.1. The Authorization pattern

A *security level* is defined as a pair (classification level, set of categories). A security level *dominates* another if and only if its level is greater or equal than the other level and its categories include the other categories. Two properties, known as “no read up” and “no write down” properties, define secure flow of information:

Simple security (ss) property. A subject s may read object o only if its classification dominates the object’s classification, i.e., $C(s) \Rightarrow C(o)$. This is the no read-up property.

***-Property.** A subject s that can read object o is allowed to write object p only if the classification of p dominates the classification of o , i.e., $C(p) \Rightarrow C(o)$. This is the no write-down property.

This model also includes *trusted subjects* that are allowed to violate the security model. These are necessary to perform administrative functions (e.g., declassify documents, increase a user’s clearance) but make the proof of security properties less credible. This model is complemented with the Biba integrity model below. Figure 4.3 shows a pattern to describe this model.

The Biba integrity model

Biba’s model classifies the data into integrity levels and defines two properties dual to the simple security and * properties.

This model includes:

Single integrity property

Integrity *-property

The Lattice model

A lattice is an extension of partial orders, where every pair of elements has a least upper bound and a maximum lower bound. Because lattices are not strictly hierarchical orders they can model a larger variety of systems. However, they are harder to implement and use and are rarely seen. A detailed description can be found in D. Denning’s book [Den82].

Application of multilevel models

Multilevel models are the most secure of the three basic security models. As we will see later, it is possible to build DBMSs and OSs that follow multilevel approaches. However, they are hard to implement because they require labeling of the protected objects and separate processing at each level; otherwise we have the possibility of *covert channels* (covert channels are low bandwidth channels that can be used to leak information between levels). They are also complex to use, we need at least two models, one for confidentiality and one for integrity. Further, in institutions other than the military it is hard to classify people and data into levels. As we will see later, they have value for implementing systems that need to compartmentalize their functions for security; e.g., operating systems and firewalls.

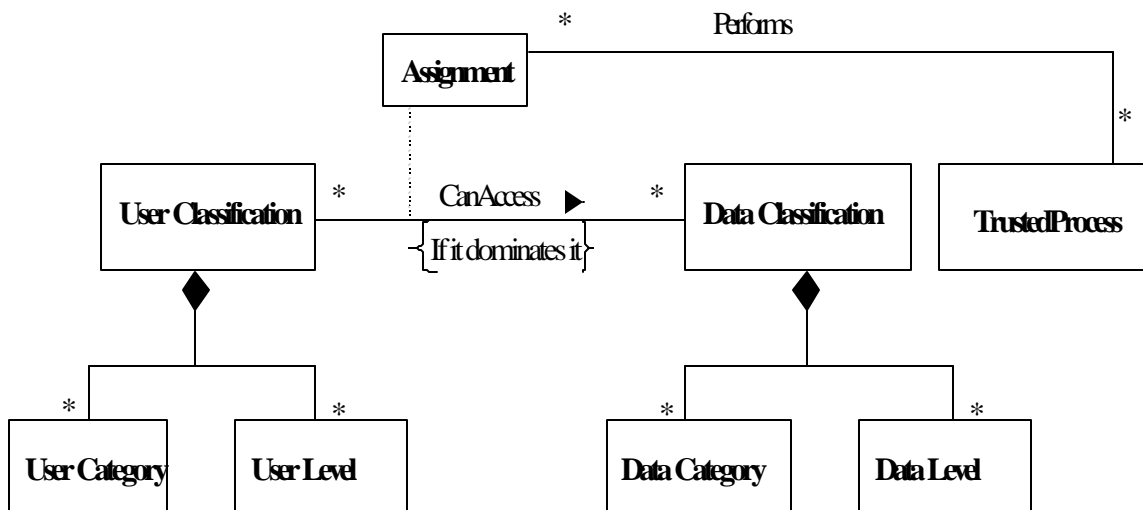


Figure 4.3. Multilevel security pattern

Models and secure system design

These models define a state machine view of a secure system, if we start from a secure initial state and all the state transitions are secure we'll always be in a secure state. This definition doesn't take into account if the initial state or the transitions are meaningful or if they contradict institution policies.

There are examples of three of the possible combinations of models. Access-matrix based discretionary models have been used in most operating systems and DBMSs until recently. RBAC is the most common model of modern systems, including operating systems, DBMSs, and web servers. Multilevel models have been used only in military systems, although as we will see later, they are useful to control attacks to different parts of a system. In particular, Joshi et al. [Jos01] discuss the suitability of these models for web-based applications. They consider RBAC as the most suitable model but think that in the future it needs to be extended to consider dynamic and task-based aspects. This is a good direction for future work.

Once we have decided about the policies we want, the next step is to convert them into models. We should try to find the model (or combination of models) that matches the policy requirements. For example, if we need a system where users should be given specific types of accesses to documents, we need some type of access matrix. We can define the subjects of this matrix according to the user functions and if the users should not grant their rights or receive rights from other users, it is clear that we need RBAC. The pattern of Figure 4.2 is then used as a reference to further define the system: Do we need the concept of session? Do we need structures of roles? Usually, this model will

cover only part of the policy requirements and complementary policies must be used. The next step is to reflect the selected model into the lower levels. Starting with Chapter 6 we consider each level in detail.

The Reference Monitor pattern

We introduce now the template we will use to describe patterns, although only a few patterns will be shown in so much detail.

Intent

Enforce authorizations when a subject requests a protection object.

Context

A multiprocessing environment where subjects request protection objects to perform their functions.

Problem

If we don't enforce the defined authorizations it is the same as not having them, subjects can perform all type of illegal actions. Any user could read any file for example. How can we control the subjects' actions?

Forces

The following forces will affect the solution:

- ?? Defining authorization rules is not enough, they must be enforced whenever a subject makes a request for a protection object.
- ?? There are many possible implementations, we need an abstract model of enforcement.

Solution

Define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations.

Figure 4.4 shows a class diagram showing a reified Reference Monitor. In this figure `Set_of_Authorization_Rules` denotes a collection of authorization rules organized in a convenient way. Figure 4.5 shows a sequence diagram showing how checking is performed.

Known uses

Most modern operating systems implement this concept, e.g., Solaris 9, Windows 2000, AIX, and others. The Java Security Manager is another example.

Consequences

Advantages include:

- ?? If all requests are intercepted we can make sure that they comply with the rules.
- ?? Implementation has not been constrained by using this abstract process.

Disadvantages are:

- ?? Specific implementations (concrete Reference Monitors) are needed for each type of resource. For example, a file manager controls requests for files.
- ?? Checking each request may result in intolerable performance loss. We may need to perform some checks at compile-time for example and not repeat them at execution time. Another possibility is to factor out checks; for example when opening a file or having some trusted processes which are not checked.

Related patterns

This pattern is a special case of the Checkpoint pattern [Yod97]. The Interceptor pattern [Sch02] can act as Reference Monitor in some situations. The Authorization pattern must be used together with this pattern to define the legal authorizations.

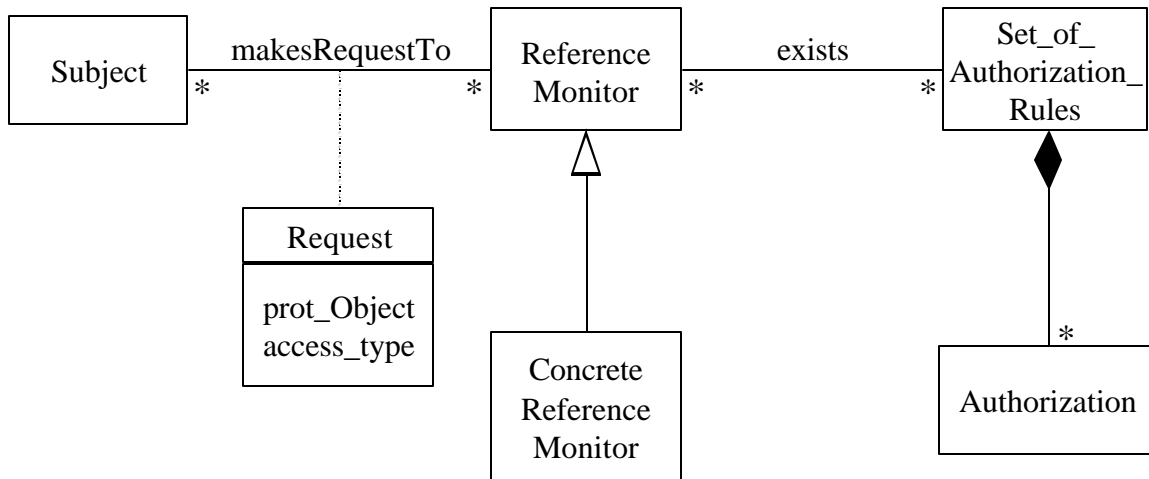


Figure 4.4 Class diagram for the Reference Monitor

References

[Den82] D.E. Denning, *Cryptography and data security*, Addison-Wesley, 1982.

[Fer75] E.B.Fernandez, R.C.Summers, and T.Lang, “definition and evaluation of access rules in data management systems”, Procs. First Int. Conf. On Very Large Databases, Boston, MA, 1975, 268-285.

[Fer94] E. B. Fernandez, J. Wu, and M. H. Fernandez, “User group structures in object-oriented databases”, Proc. 8th Annual IFIP W.G.11.3 Working Conference on Database Security, Bad Salzdetfurth, Germany, August 1994.

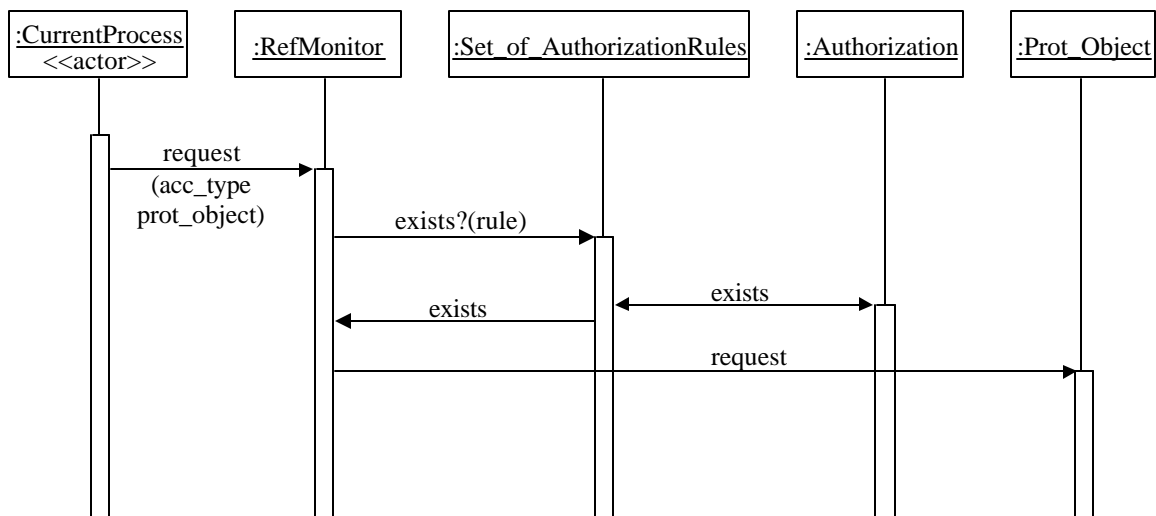


Figure 4.5 Sequence diagram for enforcing security of requests

- [Fer97] E.B.Fernandez and J.C.Hawkins, “Determining role rights from use cases”, *Procs. 2nd ACM Workshop on Role-Based Access Control*, November 1997, 121-125. <http://www.cse.fau.edu/~ed/RBAC.pdf>
- [Fer01] E.B.Fernandez and R. Pan, “A pattern language for security models”, *Procs. of PLoP 2001*, <http://jerry.cs.uiuc.edu/~plop/plop2001>
- [Gra72] G.S. Graham and P. Denning, “Protection: Principles and practice”, *AFIPS Conf. Procs.*, 40, 1972 SJCC, 417-429.
- [Har76] M.A. Harrison, W.L.Ruzzo, and J.D. Ullman, “Protection in operating systems”, *Comm. of the ACM*, 19, 8 (August 1976), 461-471.
- [Jos01] J.B.D.Joshi, W.G.Aref, A. Ghafoor, and E. H. Spafford, “Security models for web-based applications”, *Comm. of the ACM*, vol. 44, No. 2, February 2001, 38-44.
- [Lam71] B.W. Lampson, “Protection”, *Procs. 5th Annual Conf. on Info. Sciences and Sys.*, 1971, 437-443. Reprinted in *ACM Operating Sys. Review*, 8, 1 (January 1974), 18-24.
- [list] GMU Laboratory for Information Security Technology, <http://www.list.gmu.edu>
- [nist] NIST <http://hissa.ncsl.nist.gov/rbac>
- [San96] R.Sandhu et al., "Role-Based Access Control models", *Computer*, vol. 29, No2, February 1996, 38-47.

- [Sch00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-oriented software architecture*, vol. 2 , *Patterns for concurrent and networked objects*, J. Wiley, 2000.
- [Sny81] L. Snyder, "Formal models of capability-based protection systems", *IEEE Trans. on Computers*, Vol. C-30, No 3, March 1981, 172-181.
- [Yod97] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security". *Procs. PLOP'97*, <http://jerry.cs.uiuc.edu/~plop/plop97> Also Chapter 15 in *Pattern Languages of Program Design*, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Addison-Wesley, 2000