Chapter 2. Principles, policies, and models

Security principles, policies, and models are the starting points of a secure design or configuration and will guide and unify the following chapters. We show the need for guiding principles, and the use of policies to control threats and for compliance with regulations. Security design principles are derived from experience and should be reflected in the policies, models, and patterns that we use to build and operate systems. We discuss how to make policies more concrete and precise through security models.

2.1 Design principles for secure systems

The design of secure systems requires the application of a set of principles and approaches. These must be used from the start, there is no much hope to produce a secure system by retrofitting an insecure system. Most of these principles also apply to the design of highly reliable systems and it makes sense to coordinate these two aspects whenever possible. For example, a system that is not fault tolerant could allow security exposures when failing. A system that is both secure and fault tolerant is said to be *dependable*. Initially this coordination was advocated to produce critical systems, those whose compromise or failure could mean harm to humans [Neu86]. With our increased dependency on computers this definition must be extended to include systems whose compromise or failure would mean large economic losses or serious annoyance to many people, e.g., an e-commerce web site, a banking system, a travel reservation system, and similar. A dependable computer system is a machine that provides a set of dependable functions to its users.

Some principles that are accepted as conducing to secure systems are [Sal75]:

- *Closed system*. This is the policy discussed earlier. Anything not explicitly allowed is forbidden. Here the policy is applied to application rights and to process resource access during execution. This policy can be considered a special case of *fail-safe* defaults.
- *Open design*. When the mechanism is subject to public scrutiny it is easier to find flaws and with time it becomes increasingly secure. Some of the data used in the security system could be secret though, e.g., passwords, keys.
- *Least privilege*. This is just the application of a basic security policy at all design levels.
- *Economy of mechanism*. The security mechanisms should be as simple and small as possible. This makes comprehension, testing, and analysis easier.
- *Complete mediation*. Every request for access must be validated.
- *Minimal trust*. The parts of the system that must be trusted should be minimal.
- *Separation of privilege*. Critical actions may need two independent mechanisms to agree before being performed.
- *Least common mechanism*. Do not mix security with other functions.
- *Ease of use or transparency*. Users must accept the security system or they will not use it. Security functions must be transparent or at least easy to use.
- *Information hiding and encapsulation*. Implementation details should not be exposed, only a clear, functional interface should be shown. This principle comes from [Neu86] and points toward object-oriented design.

Other principles that have been found by experience include:

- *Holistic approach*--Cover all architectural levels and all units, apply security in all phases of system design
- *Highest level*—security constraints must be defined where their semantics are clear and propagated down
- *Defense in depth*—have more than one line of defense
- *Separate and compartmentalize* –isolate units that do related work at execution time
- *Safe defaults*—failures (crashes) should close everything for access, everything not explicitly granted is forbidden
- *Submarine principle*—we can accept to lose some units but we preserve essential units

We can also define principles for specific systems, we will see some principles for operating systems in Chapter 4. We can also use more general principles of good design. One of them is:
Use of object-oriented design. Its abilities for abstraction, encapsulation, and information hiding are particularly appropriate for designing secure systems. This property was noted as early as 1993 [Neu93].

Most of the design of a secure system is software design, hardware is either given or selected from standard architectures. The way software is developed (software process) and the specific methodologies used for it are very important to produce secure systems. It is clear that security principles should be applied at every development stage. We sketch now a secure software development cycle we consider necessary to build secure systems. We will use this methodology as a guideline throughout the book. Figure 2.1 shows a secure software lifecycle, indicating where security can be applied and where we can audit for compliance with security policies

Requirements stage: *Use cases* for the system are defined at this stage, where a use case describes an interaction with the system. We can enumerate threats with respect to use cases. Security constraints can also be defined in use cases (secure use cases). Application-specific regulations should also be applied here.
Analysis stage: From the use cases we can determine the rights for each actor. We can build a conceptual model where we can apply patterns to realize the rights determined from use cases. Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. This makes the job of defining rights even easier. This is discussed in detail in Chapter 2. We can verify that all the enumerated threats are controlled by some pattern.

Design stage: Interfaces can be used to enforce the security constraints defined in the analysis stage. Remember that user interfaces should correspond to use cases. Components can be secured by using rules defined according to the authorization needed for each component. Distribution may require additional security. Deployment diagrams can define secure configurations, to be used by security administrators. We can now verify that our mechanisms take care of all the types of attacks defined in the requirements stage. In order to select the appropriate mechanisms we need to look at how the attacks for each use case are actually realized through the architectural layers of the system.

Implementation stage: This stage requires reflecting in the code the security constraints defined in the design stage. Because these constraints are expressed as classes and associations, they can be implemented as the rest of the application. We may also need to select specific security packages, e.g., a firewall product, a cryptographic package.

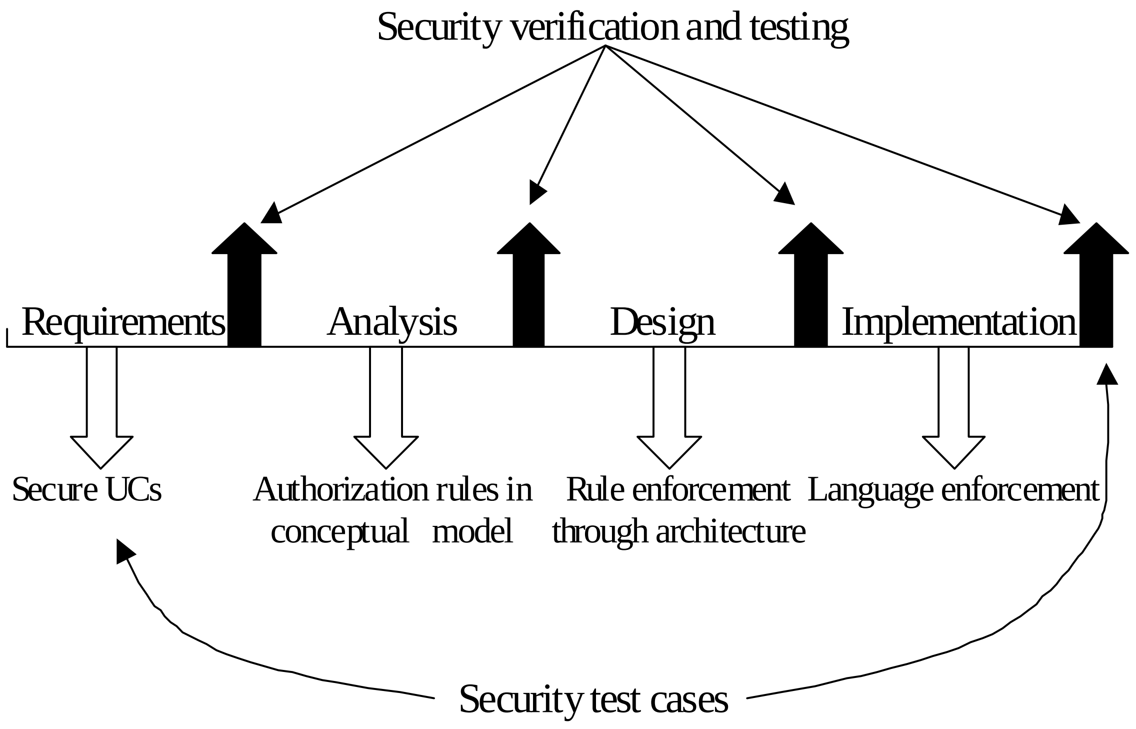

Security verification and testing
Requirements
Analysis
Design
Implementation
Secure UCs
Authorization rules in conceptual model
Rule enforcement through architecture
Language enforcement
Security test cases

Figure 2.1. Secure software lifecycle

At the end of each stage we can perform audits to verify that the institution policies are being followed and that the threats are being controlled. If necessary, the security constraints can be made more precise by using some type of formal constraints. A *domain model* can be built to support related applications; for example, Academic applications. In this model we can also apply regulations and institution policies.

2.2 Standards, consortiums, and public security services

There is a variety of standards for security, defined by organizations such as IEEE, Internet Engineering Task Force, and others [Mer03]. In 1983 the U.S. Dept. of Defense published a document that was used for many years to evaluate computer security. This was the "Orange book" [DoD], that provided a set of categories for classifying the level of security of computer systems. Because of its military origin, that book emphasizes multilevel security and has lost relevance for commercial systems. The NIST has introduced the Common Criteria for evaluating a larger variety of systems and it is now the most used standard [nis]. Until now we don't have numeric measures of security, all our measures are qualitative, usually based on how the system has been built and on tests of the code. NIST has defined seven levels of security, where EAL 7 is the highest [nis]. The highest level of security reached until now by a commercial system is an operating system kernel for avionics, certified at the EAL 6+. This level implies formal verification and code testing. This kernel certification took about 10 years, it is not clear how applicable it is to more general or bigger systems. Most commercial systems have new versions every 2-3 years and are quite dynamic. Having a systematic way to enumerate threats we can try to show that in the final product those threats are not possible, both from a design perspective (examining models), and from a code perspective (examining and testing the code). Formal proofs are only practical for relatively small and static systems. There are also more specific standards, e.g., for web services security; these will be discussed in the respective chapters.

There is a variety of groups seeking ways to improve the security of systems:
The Trusted Computing Group (TCG) (http://www.trustedcomputinggroup.org/) is an industry group that defines standards for software and hardware products [tcp]. TCG includes over 190 companies. Their first push is towards hardware-based security, a controversial approach (see Chapter 3).
The Center for Internet Security (CIS) has developed a set of security benchmarks for operating systems [cis].
The Forum of Incident Response and Security Teams (FIRST) is an international consortium of computer incident response and security teams who work together to handle and prevent incidents [fir].
The Information Technology-Information Sharing and Analysis Center (IT-ISAC) is an alliance of several companies to share information about product security [iti].
The Security Industry Association *(www.siaonline.org/). Represents manufacturers of security products and services.*

For companies that do not have expertise on security there is the possibility of outsourcing. Several companies offer services to manage all the security aspects of another company. There are advantages and disadvantages to this approach but for small companies it is an attractive option (see Exercise 1.3). Another useful security service is offered by the U.S. Post Office and similar organizations in other countries, which have online secure services to transfer sensitive documents and provide digital certificates (see Chapter 3). .

2.3 The need for policies

Policies are high-level guidelines defining the way an institution conducts its activities in its business, professional, economic, social, and legal environment. More concretely, policies are management instructions indicating a predetermined course of action, or a way to handle a problem or situation [Woo00]. Every institution has a set of policies, explicit or implicit, some of which are security policies. Without policies it is impossible to build secure systems, we don't know what we should protect and how much effort we should put on this protection. However, many institutions don't have explicit security policies; others have them but don't apply them [Ver06]. Institutions without policies risk greater exposure to lawsuits in the event they mishandle information. Effectively, a security policy divides the states of a system into authorized and non-authorized states, everything that conforms to the policies is authorized. That means that we cannot even talk about security without policies because we don't know what states we should avoid. In recent years policies have become more widely appreciated. There is even an annual conference dedicated to policies [Pol]. Policy packages [Bla00], and software for automatic policy generation are available [And03], and even whole books have been devoted to this topic, e.g. [Gre06, Woo00].

The *institution security policy* includes laws, rules, and practices that regulate how an institution uses, manages and protects resources. In a company, there are global policies that affect all aspects of the business and more specialized policies that refer to the basic division of functions into Supply Chain Management (SCM), Enterprise Resource Planning (ERP), and Customer Relationship Management (CRM). Some of these policies may be imposed or suggested from external sources; e.g., legislation, government or industry standards. There are some interesting issues about who should define the external policies [Mea00] or what standards an institution should follow. For example, all banks as required by U.S. government law, send their customers a statement of their privacy policy, indicating what personal information they collect, with whom do they share it, and similar details. If an institution has an explicit set of policies that show compliance with regulations and define clearly the use of information its customers trust it more and it provides a good defense against lawsuits in case of misuses. An example of a use policy is: In 2005 IBM adopted the policy of not using genetic information in hiring or in their benefits or healthcare plans.

Policies are mandatory in an institution and the computer systems at the institution should enforce these policies by reflecting them into their mechanisms. The Layers pattern that we saw in Chapter 1 can be used to describe how the policies are structured, going from institution policies at the highest level down to policies to allow specific accesses of users to data, encryption policies, etc.

2.4 Security policies

Through experience several policies have evolved as the most convenient to build or configure secure systems. A specific system uses a combination of policies according to its objectives and environment. We enumerate below the most common policies used in practice, more specialized policies may be used in specific environments, e.g. for medical applications.

- *Open/closed systems*—In a closed system, nothing is accessible unless explicitly authorized, in an open system or institution everything is accessible unless explicitly denied. Institutions where information security is very important, e.g., banks, should use a closed policy. Institutions whose objective is disseminating information, such as libraries, use an open policy (all books are accessible except rare books for example).
- *Least privilege (need to know)*—People or any active entity that needs to access computational resources should be authorized only to access the resources they need to perform their functions. This policy can be considered a refinement of the closed system policy. For example, a secretary should not have access to product plans in a company. It also applies to the institution, it should not collect more information about its members than necessary.
- *Authorization*—Explicit rules must be used to define who can use what resources and how. Authorizations may allow or deny access and may impose conditions for access. Authorization rules can be used to further refine a least privilege policy. For example, a team leader in a project should have access to job performance information of the people in his group but not information about their salaries.
- *Obligation*—Access to a resource is given only if some further action is executed before or after the access. An example is a medical policy that requires that any access to a medical record must be notified to the corresponding patient.
- *Separation of duty*—Critical functions should be assigned to more than one person or system. For example, the person who decides that a product is needed is not the same who approves its purchase.
- *Auditing*—An audit trail should record everything that was done at some time. This is important for accountability purposes and in case of an attack it could help preventing future attacks.
- *Authenticated transactions.* Any exchange of information should be authenticated at both ends.
- *Centralized/decentralized control*-- In a decentralized system its units or divisions have their own administrators and authority to define their own policies or enforcement mechanisms as far as they don't violate global policies.
- *Ownership and administration*-- In many systems the user that creates some data becomes its owner and has all the rights on it. An administrative policy separates the administration of the data from its use. Ownership may violate separation of duty in that the user of information is also its administrator, which for institution data is a conflict of interest but it is acceptable for personal files. Ownership is not a good policy for secure institution systems although it is commonly used in most operating systems.
- *Individual accountability*—People or processes must be uniquely identified and their actions recorded and reviewed. This policy is supported by logging.
- *Roles*—People act in different roles in their work or in their private life; a professor is at times a teacher, a researcher, a thesis advisor, or a committee member. In each role different rights are needed to perform the corresponding functions. The rights of the roles could be assigned following a least privilege policy.
- *Name- or item-dependent access control*—We control access to named data items or classes including all their instances. For example, the CEO of a company can access salary information for all employees.

- *Content-dependent access control*-- Access to data depends on the specific records requested. For example, a department manager can access salary information only for the employees in her department.
- *Context-dependent access control*-- Access to data depends on what other information is also requested; for example, one cannot get salaries and names together. Another interpretation of this policy is basing access decisions on system or workflow state.
- *History-dependent access control*—We consider all or a subset of past requests to decide access. This policy can be used to avoid user inference; for example, a user could ask for names and then for salaries and combine the two sets of results.
- *Discretionary delegation/granting*—A user who has a right may be allowed to delegate or grant this right at his discretion (keeping or losing the right).
- *Mandatory rights*—Users receive rights from the system but cannot grant them to others.
- *Multilevel rights*—Users are classified in levels and their access rights depend on their levels. Typically, military institutions use this approach.

2.5 Application-specific policies

The policies above apply to any secure system. Some applications require more specific policies; for example, the military put high emphasis on secrecy, while a legal office is more interested in integrity. We discuss below a few specialized policies.

Confidentiality policies

Document classification—Documents are classified according to the sensitivity of their information. People are given clearances. The policy defines a relationship between classification and clearances. For example, the clearances and the classifications may be hierarchical levels: Top Secret, Secret, Confidential, Public, and a user cleared for a given level can read all the documents at her level or below (an example of a multilevel rights policy).

Categories—Define vertical partitions of the levels, e.g., Army, Navy. Now, not only the classification must be appropriate to read a document but also the user category must match or include the category of the document.

Originator controlled (ORCON)—A document is released only to people or units in a list specified by the originator.

Aggregate access—Users are allowed to read only values of data aggregates, e.g., the average of salaries, the mean of student grades. These policies are particularly important when the privacy of individuals is involved.

Integrity policies

Document integrity--A document cannot be modified or can only have registered modifications.
Constrained change—Data can be changed only in prescribed ways

Task policies
Authorized actions—People can perform only actions for which they are authorized (an example of authorization policy).
Rotation of duty—A task should not always be carried out by the same person
Operation sequencing—The steps of some task should be carried out in a specific order

Conflict of interest policies
Chinese Wall policy—Information is grouped into "conflict of interest" classes and a person is allowed to access at most one set of information items in each class.
Conflicting roles—A user cannot have two roles that may imply a conflict of interest. For example, a manager cannot be also a regular employee of the same department.

2.6 System policies.
Most of these policies can also apply to low-level system aspects. For example, a process should execute with the least amount of privilege it needs to perform its functions. Other system policies define specific use or implementation of some system; for example, a user account/password policy may define aspects such as the length of passwords, what characters they may or may not have, and how often they should be changed. An example of separation of duty at the system level is separating the application of an authorization rule from the storage and maintenance of the rules. Some of the system policies come directly from similar policies at a higher level, others are peculiar to the characteristics of some architectural level.

Policies can be considered as declarative rules governing choices in a system's behavior [Slo02]. Moffett and Sloman [Mof88] classify system security policies in three levels:

- *General policies*. These apply to any institution. Examples are giving access control administration to security administrators. Regulations would be included in this group.
- *Specific policies*. These refer to specific organizations, e.g., emphasizing integrity over confidentiality in a financial institution.
- *Access rules*. Define specific user access to specific resources.

A common error is to define low-level policies without using higher-level policies as a reference. For example, Visa requires that online merchants using their cards should: install a firewall, keep security patches up-to-date, encrypt stored and transmitted data, etc. These policies are too detailed to be effective and are restrictive to the participant merchants because they are not based on higher-level policies. They are also dependent on the current technologies, if these change the policies need to be updated. In fact, policies may be erroneous or incorrectly enforced. Erroneous policies include policies that don't make sense in the context where they are applied, are overrestrictive or incomplete [McG00]. Also, the enforcement mechanism may not enforce the policy correctly.

Some policies may be applied to a variety of systems; for example:

- *Isolation or containment*. A system should be isolated from external systems, a process should be isolated from other processes.

- *Controlled sharing*. Resources or information should be shared by processes or systems in a controlled way, subject to specific authorizations.

- *Memoryless systems*. A program should not keep any traces of its past executions. For example, a program to calculate taxes should not keep any of the values it has used in some past calculation.

In general, isolation and controlled sharing are mutually exclusive when applied to a specific process but can be combined when talking about a set of processes, which may be isolated as a whole but may share resources among them.

2.7 Policy examples

Many common policies refer to authorization aspects. The specific authorizations defined must fit the needs of the application. For example, Anderson mentions how in the UK a government office tried to impose multilevel policy standards on medical systems and it did not work because it did not fit the requirements; for example, patients want to control the use of their records, something not allowed in the multilevel model [Mea00]. See Exercise 2.4.

An example of policies for medical systems is given in [And96]. The following is a possible set of policies for a university system, assuming also a closed system policy:

- An instructor can look at all the information about the course he is teaching.

- An instructor can change the grades of the students in the course he is teaching

- A student may look at her grades in a course she is taking or has taken.

- A department head can add/delete course offerings in his department.

- The department head can add/delete students from course offerings and can assign instructors to them with the agreement of the corresponding faculty.

- Faculty members can look at information about themselves.

- A registration clerk can enroll students into course sections.

- A department head can look at information about his department and can change information about faculty and courses.

- A dean can look at the information of all the departments in his college.

Some policies may be quite complex and depend on values of the variables involved. An example of a complex policy is [Sch90]: “A user may see the records of each employee he supervises if the user’s salary is greater than that of the employee”. Authorization policies are implemented as authorization rules and managed using some security administration system. Because of this, it is common in industry to refer to the rules themselves as policies.

Policies can refer to multilevel policies, for example [Sch90]:
- A flight plan may be classified if the passenger list includes specific named officials.
- A classified flight plan may become unclassified once the flight is completed.

2.8 Role-based policies
A role define a task or duty of a person and their related actions; for example, the members of a Ph.D. committee have some specific functions such as approve the topic of the dissertation. It is important to define roles with respect to the information produced or used in an institution or system. Some possible roles with respect to documents are:
- Originator—The person who issues a document
- Authorizer—The person who controls access over the document
- Custodian—The person who keeps the document and controls its use
- User—The person who reads or modifies the document
- Auditor—The person who checks the actions, results, and controls applied to documents.

We can also define appropriate roles for people according to their job functions and assign rights according to these functions; for example, manager, secretary, student, and instructor. In the university example above, the roles used are instructor, student, faculty member, registrar, department chairman, and dean. Each role may have some sub-roles; for example, a faculty member can be an instructor, a thesis advisor, a department committee member, and a researcher.

Prescriptive policies, see Uzunov 3.4.2

2.9 Policy standards
In the US, the first government institution in charge of security policies was the Department of Defense. They published a reference document defining different levels of security. This document (known as the Orange Book) lists a set of requirements for secure systems that can be considered as security evaluation policies. Later, the National Institute of Standards and Technology (NIST) developed a set of documents known as the Common Criteria [cc]. Other policies have been defined by ECMA and ISO.
- Medical information—The BMA in the UK and HIPAA in the US. Policies for specialized applications have been defined for:
- Financial information—The Sarbanes/Oxley act of the US.
- Information systems—The Electronic Communication and Transactions Act (ECT) of 2002 in South Africa, has among its objectives “to prevent the abuse of information systems”.
- US Government –[nis09]

2.10 Standards for policies and policy languages

The Policy Core Information Model (PCIM) is a policy model to extend the Common Information Model (CIM) developed by the Distributed Management Task Force (DMTF) and the IETF Policy Framework group [iet04]. The CIM defines generic objects that include systems, managed system elements, logical and physical elements, and services. They define a policy rule and its components conditions and actions. A policy rule is of the form <condition set> then do <action list>. Policy rules may be simple or groups (a Composite pattern). Conditions and actions may be part of specific rules or being stored in repositories for common use by several rules. The use of repositories is an implementation aspect, which should not have been mixed with the logical rule structure. The PCIM also provides detailed models for conditions and actions.

IBM has developed a Trust Policy Language. This uses XML to define criteria for assignment of clients to roles and resource authorization. Rules cannot be inherited and have other restrictions [Slo02]. Some policies can be represented formally using models as shown below, while others are mostly described by words. Fuzzy logic has been used to make word policies more precise [Hos95]. A well-thought set of policies is reusable, as shown by the fact that there are pre-packaged policies [Woo00].

2.11 Policy conflicts and lack of policies

All the policies applied in a system interact with each other. Ideally, they should collaborate or converge [Hos95]. Sometimes they may overlap, which may result in unnecessary redundancy. The worst situation is when policies conflict with each other, because this may lead to security vulnerabilities [Lup97]. For example, privacy may conflict with accountability. Transnational use of data often results in policy conflicts. Distributed systems require the coexistence of many policies and metapolicies are needed to coordinate them [Kue95a].

It is also possible that the objects to which a policy refers overlaps with those of another policy. For example, a policy may indicate that a given object is accessible to a user while another policy may deny it. In these cases the conflict can be resolved by policies such as “permissions have priority”, “denials have priority”, or by adding explicit priorities to each rule. Conflicts are common when we have hierarchies of policies and it is important to provide policies to resolve them before continuing with the more detailed design. In some cases, conflicts may appear at execution time and we need special policies for dynamic conflict resolution.

An example from a real case illustrates what happens when policies are not defined or applied [Vij02]. A disgruntled former employee from Global Crossing Holdings Ltd. posted numerous names, SSNs, and birth dates of company employees in his web site. The company allowed all software developers to have full read access to the employee information and the customer billing system was accessible for read and write to a large number of employees. The first problem was not applying the need-to-know policy, software developers do not need access to operational data. The second problem was similar, access to billing information should have been restricted only to those who had a need for their work functions, i.e., lack of need-to-know and role-based access. Apparently they did not have a closed system either. On the other hand, a too strict need-to-know policy could hamper productivity because employees need to ask frequently for access to information they need.

2.12 Policies and secure systems design

Policies that correspond to regulations must apply to all the applications used by an institution. They should be defined in a domain model and propagated to each application.

Once we have a list of the threats to a specific application we can decide which of these threats are important and how we can stop them using policies; that is, the policies will guide the selection of the specific mechanisms we need to stop the threats. For example, if secrecy is important we must protect against viruses or Trojan Horses that may compromise secrecy. Policies are also important for evaluating a secure system, if a system enforces its policies, it is secure for our purposes. A complex system may need to support multiple policies, including a variety of policies for access control [Kue95b].

The security policies should be reflected in the security mechanisms used in the different architectural levels. The lower level mechanisms should enforce the policies defined at the high levels. Many commercial systems do not apply the policies described earlier; for example, in Unix a file creator becomes its administrator and user, which violates the policy of separation of duty. We will discuss specific policies when we discuss specific architectural levels. It is also important to define security policies in a context, e.g., a specific system or level.

At this moment we can consider the regular use cases of the system to define the rights that the users must have to be able to perform their functions [Fer97]. Use cases can be extended with policy assertions, as shown in the example (coming later). In fact, the actors defined as involved in the use cases should be the only actors that can have these interactions. This is a basic application of the need to know policy. Judicious application of policies can prevent many threats; for example, in a recent case, a secretary spent a large amount of the company money in her own expenses. She had the right to use some money in paying for office expenses but simple policies could have avoided this misuse:

- Limit on the amount to be spent at one time and over some period of time.
- A supervisor should have checked her monthly expenses (auditing policy).
- Large expenses should have required the approval of a supervisor (separation of duty policy).

Some security policies can be made more precise through the use of formal or semi-formal models. A model allows us to analyze security properties and it is the basis for system design. Models are discussed in the next sections.

2.13 Applying security policies to stop threats

We illustrate our approach through an example. Consider a financial company that provides investment services to its customers. Customers can open and close accounts in person or through the Internet. Customers who hold accounts can send orders to the company for buying or selling commodities (stocks, bonds, real estate, art, etc.). Each customer account is in the charge of a custodian (a broker), who carries out the orders of the customers. Customers send orders to their brokers by email or by phone. A government auditor visits periodically to check for application of laws and regulations. Figure 2.2 shows the Use Case diagram for this institution.

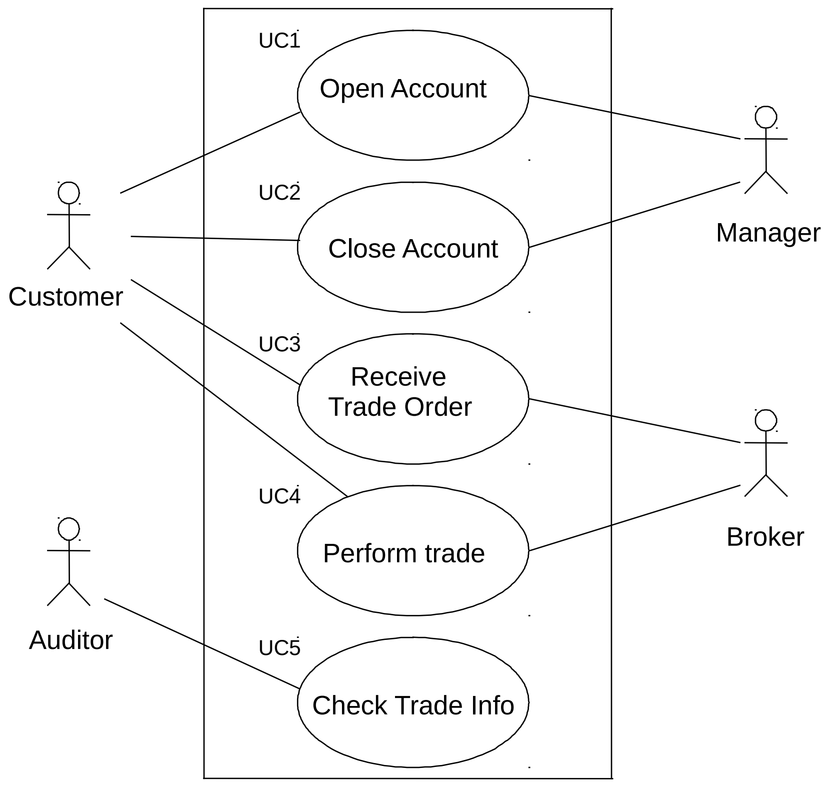


Figure 2.2 Use cases for a financial institution

Figure 2.3 shows the activity diagram for the use case “Open account” in this institution, indicating the typical actions required to open an account for a new customer. We indicate “swimlanes” for Customer and Manager, the two actors involved in this use case. These actions result in new information, including objects for the new customer, her account, and her card-based authorization

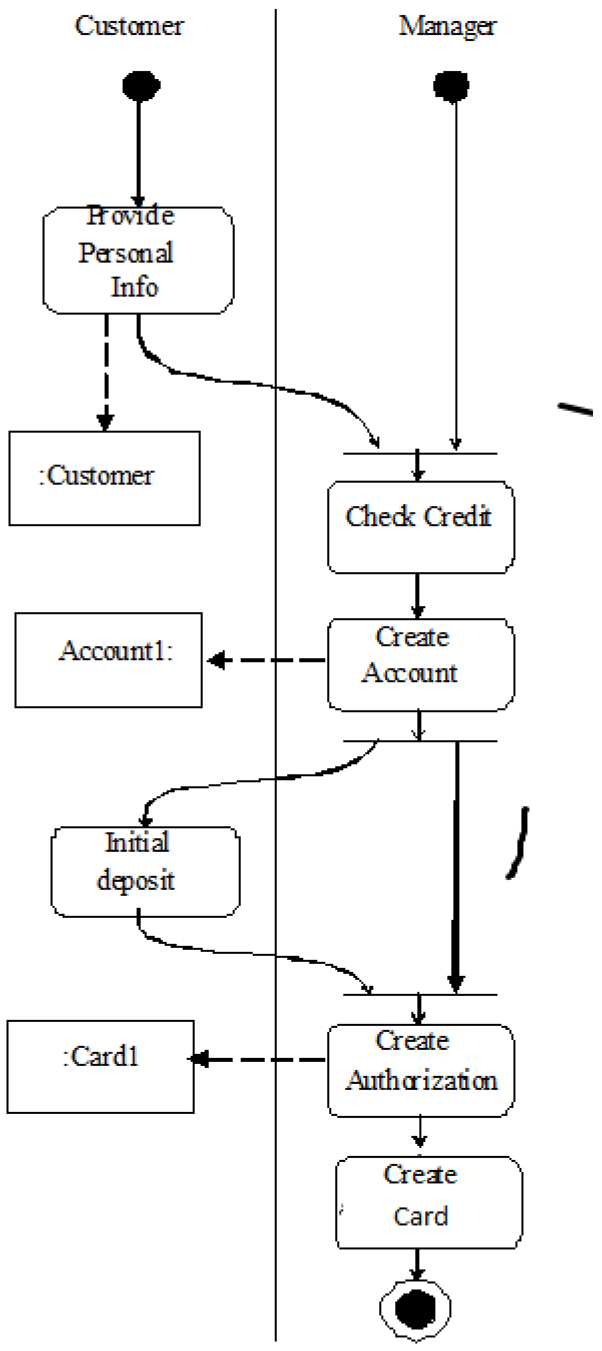


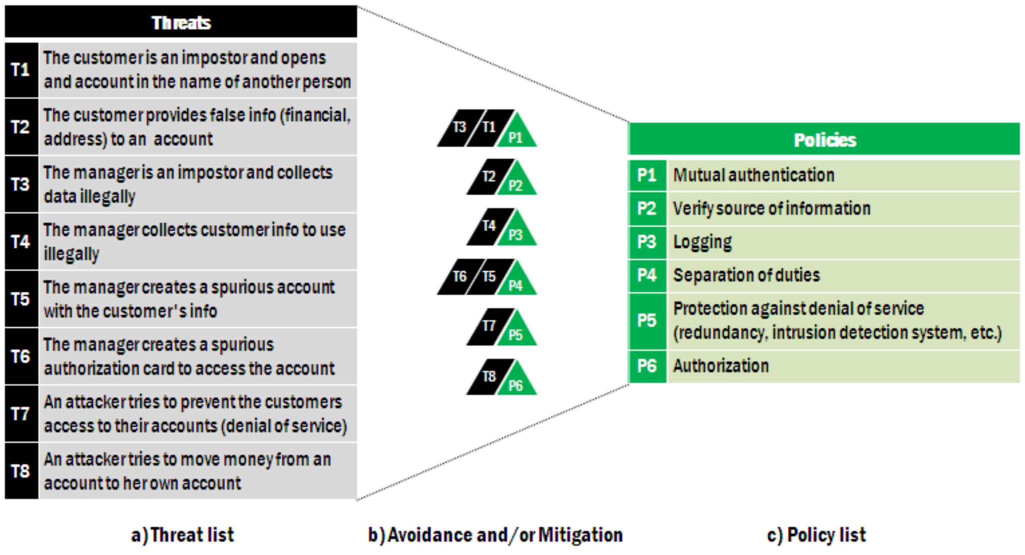
Figure 2.3 Activity diagram for use case “Open account”

The misuse activities approach [Bra08] consists of a systematic way to identify system threats, and determining policies to stop and/or mitigate their effects. To do that, two activities are employed. The first activity is an analysis of the flow of events in a use case or a group of use cases, in which each activity is analyzed to uncover related threats. This analysis should be performed for all the system uses cases. The second activity entails a selection of appropriate security policies which can stop and/or mitigate the identified threats.

The analysis of the flow of events in a use case implies a detailed investigation of each activity from each use case, aiming to figure out any possible ways to subvert it by either insiders or outsiders. In order to uncover the associated threats, three aspects will be considered when performing this analysis. The first refers to the use cases, in which it is expected to find all system scenarios. Sometimes, it is necessary to analyze a sequence de uses cases which represents a critical business workflow, therefore the activity diagram should depict not only a specific use case flow, but a business workflow. First consider the policies that can be used to handle some threats. Let's take the finance example from Figure 2.2, whose threats are shown in Figure 2.4. In this case the illegal money transferring in T8 depends on the spurious account creation, therefore both should be analyzed together. The second aspect is the set of security attributes. Every activity from the activity diagram should be analyzed according to the standard main security attributes: confidentiality, integrity, availability and accountability. All this means that we must analyze whether the activity could be subverted in terms of confidentiality: traffic analysis, covert channels, inference, and information disclosure; integrity: unauthorized data modification, transactional integrity, deception, masquerading, spoofing, impersonation; for availability: denial of service, disruption; and accountability: repudiation, track erasing.

As the third aspect, we have the source of the threat, which relates to the privileges that the threat agent has to execute the attack. Here, we can divide the attackers into few groups: the outsider, people with no authorization to access any part of the system; the authorized insider, people who can access the system and to execute the activity; and the unauthorized insider, who has system access authorization, but not to the activity being analyzed.

As before, an activity diagram is the starting point to perform the analysis, which could be done in parallel. We must scrutinize the activities keeping in mind the following statement "What misuse could be done in <*activity*> by <*source*> which compromises <*security attribute*> from the <*asset*>". In this statement the activity refers to each activity from the activity diagram, the source refers to the third aspect (source of threat), the security attribute refers to the second (type of misuse), and the asset refers to the flow object, e.g. in Figure 2.6, one of the threatened assets is the object "Account2".



Threats	
T1	The customer is an impostor and opens and account in the name of another person
T2	The customer provides false info (financial, address) to an account
T3	The manager is an impostor and collects data illegally
T4	The manager collects customer info to use illegally
T5	The manager creates a spurious account with the customer's info
T6	The manager creates a spurious authorization card to access the account
T7	An attacker tries to prevent the customers access to their accounts (denial of service)
T8	An attacker tries to move money from an account to her own account

Policies	
P1	Mutual authentication
P2	Verify source of information
P3	Logging
P4	Separation of duties
P5	Protection against denial of service (redundancy, intrusion detection system, etc.)
P6	Authorization

Figure 2.4 Threats and policies to handle them

Misuse Activities				
#	Sec. Att. CO/IN/ AV/AC	Source AIn/UIn/ Out	Description	Asset
T1	AC	AIn	Disavows having opened this account	Account
T2	AV	Out	Overwhelms application	N/A
T3	CO	Out/UIn	Eavesdropping	Customer
T4	CO	Out	Uncovers cust. relationship with institution by trying to create new account with his info	Customer
T5	IN	AIn	Provides invalid info (financial, address)	Customer
T6	IN	AIn	Provides another person's info (name, address, SSN)	Customer
T7	AC	AIn	Disavows having modified customer credit info	Customer
T8	CO	AIn	Collects customer personal info to disseminate illegally	Customer
T9	CO	Out/UIn	Eavesdropping	Customer
T10	CO	Out	Collects data illegally as an impostor	Customer
T11	IN	AIn	Changes the customer credit info to get more clients	Customer
T12	AC	AIn	Disavows having creating spurious account	Account
T13	CO	AIn	Collects customer account info to disseminate illegally	Account
T14	CO	Out/UIn	Eavesdropping	Account
T15	IN	AIn	Creates spurious account	Account
	-	-	-	-
T16	AV	AIn	Does not issue card	Card
T17	IN	AIn	Creates a spurious authorization / card	Card
T18	AC	Ain	Disavows having authorized money transfer	Account
T19	AV	Out	Overwhelms application	N/A
T20	CO	Out/UIn	Eavesdropping	Customer
T21	IN	Out	Transfers money between accounts illegally	Account

Legenda: CO - Confidentiality; IN - Integrity; AV - Availability; AC - Accountability; Out - Outsider; AIn - Authorized Insider; UIn - Unauthorized Insider

Figure 2.5 Policies for the financial institution

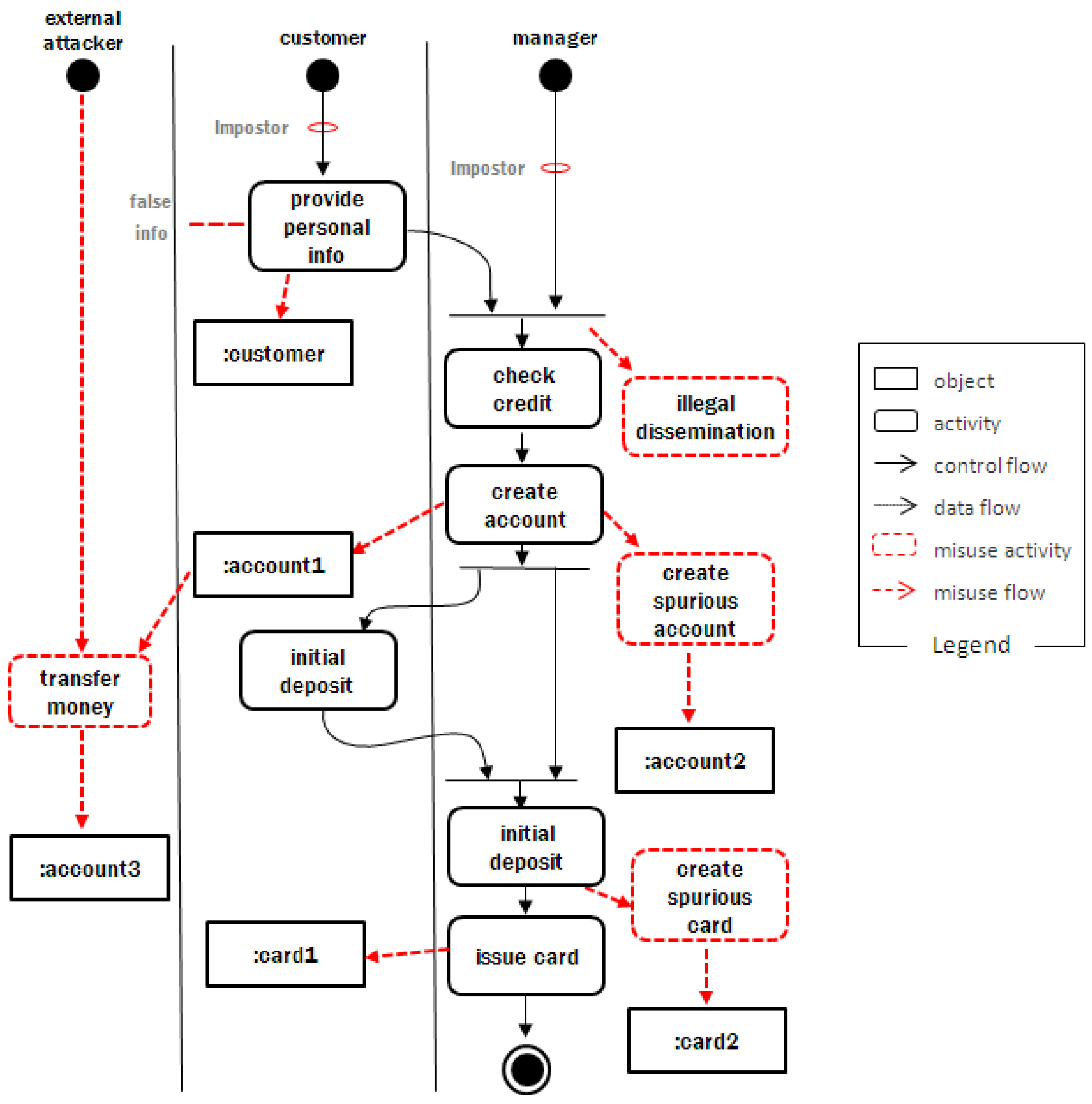


Figure 2.6 Activity Diagram for Use Case "Open Account" showing some threats

Figure 2.5 shows a table which results of applying this approach to the activity diagram in Figure 2.2. This table summarizes the results after applying the misuse activities approach to uncover threats. In order to make it easy to notice the differences, the threats already uncovered, shown in Figure 2.6, are in black (their numbers have been changed).

As presented in Figure 2.5, several more misuse activities have been uncovered, compared to the direct application of misuse actions, such as “T11 - Changes the customer credit info. to get more clients”. In this case, the analysis of integrity violation performed by some authorized internal user leads to consider this threat scenario. Such outcome is tied with the main goal of the misuse activities approach, which is to support the security requirements analysis in such a way that we would not forget any obvious threat. Additionally, when we give a first class treatment to the (misuse) activities performed by internal users, we can deal with the biggest source of threats in a proper way. It is still feasible that some threats might not be found by this approach, but in despite of this, the new approach appears to be more effective and useful than the earlier approach.

The context aspect can lead to some different conclusions in terms of the relevancy and suitability of some threats. In the example in Figure 2.2, if we consider the case when it is not possible to open accounts remotely, it means that any customer who wants to open an account must go to a bank agency to do it. In this case, some threats might be disabled, e.g. “T3 - Eavesdropping” and “T4 - Uncovers customer relationship with the institution by trying to create new account with his info”.

Security models

Security models are a more precise and detailed expression of relationships between some types of policies, usually authorization or authentication; they are used as guidelines to build and evaluate systems. They are described in formal or semi-formal way. 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. Mandatory and discretionary models can be combined with the access matrix and the multilevel model.

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 protections objects are classes

[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.

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 for the Access Matrix is given in Figure 2.7 (from [Fer13, Sch06]).

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 above, 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{t/t*} to M(s,o)---transfer can be destructive or not, depending on the policy.
grant{t/t*} to M(s,o)—in a grant both grantor and grantee have the right after the grant.
delete t from M(s,o)—a right is removed from a subject.
read M(s,o)—inspect the right of a subject for an object.
create object o
delete object o
create subject s
delete subject s

Harrison, Ruzzo, and Uhlman [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($x_1$, $x_2$, ….$x_k$) //the x's stand for s or o
    if $t_1$ in M($s_1$,$o_1$)  and
    if $t_2$ in M($s_2$,$o_2$)  and
      …..
    if $t_m$ in M($s_m$,$o_m$)
    then  $op_1$,$op_2$,…., $op_n$
    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 a version of RBAC, where general users cannot transfer their rights.

An implementation of the access matrix must have a way to properly store the authorization rules. However, we also need to enforce these rules, we need to intercept user or program requests, and to compare the request to the access matrix to decide access. This interceptor is the *reference monitor*. A pattern for the Reference Monitor is given at the end of the chapter and illustrates the template used to describe patterns in the standard literature. Typically, 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. Special policies are necessary when a subject or an object may imply others; this is discussed in the section on Implied Authorization. Another consideration is how to decide when there are predicates involved. This is discussed in the chapter on database security (Chapter 6).

Implied authorization

For convenience we may need to group or structure subjects, protection objects, and some access types in such a way that they 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 *implied authorization* was introduced in [Fer75b] and subsequently has been applied in some research projects and used in the later versions of the Windows operating system (starting with NT Version 4.0) and the .NET component authorization. It has also been applied to object-oriented databases where access rights can be inherited along generalization or aggregation hierarchies as we discuss in the chapter on database systems.

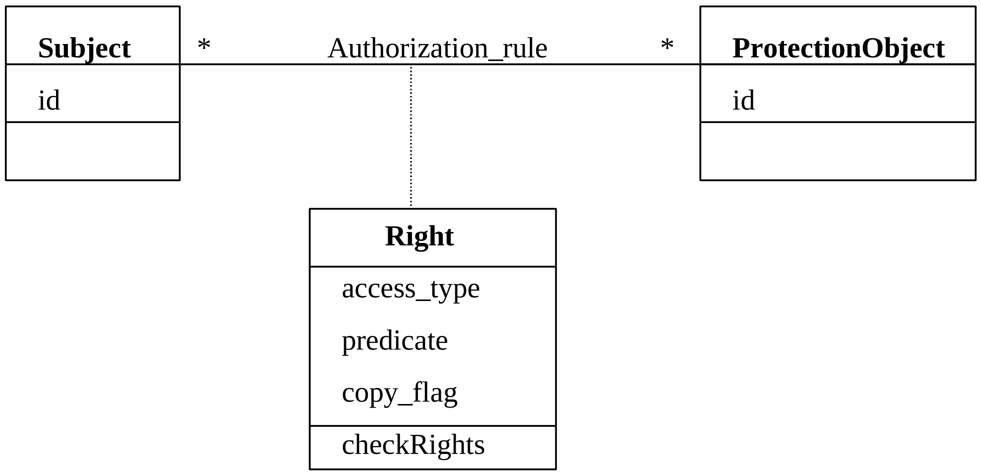


Figure 2.7. The Authorization (Access Matrix) pattern

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 2.8 [Fer01, Fer13, Sch06]. This model uses also the concept of session to further limit the rights used at a given moment and to enforce separation of duty.

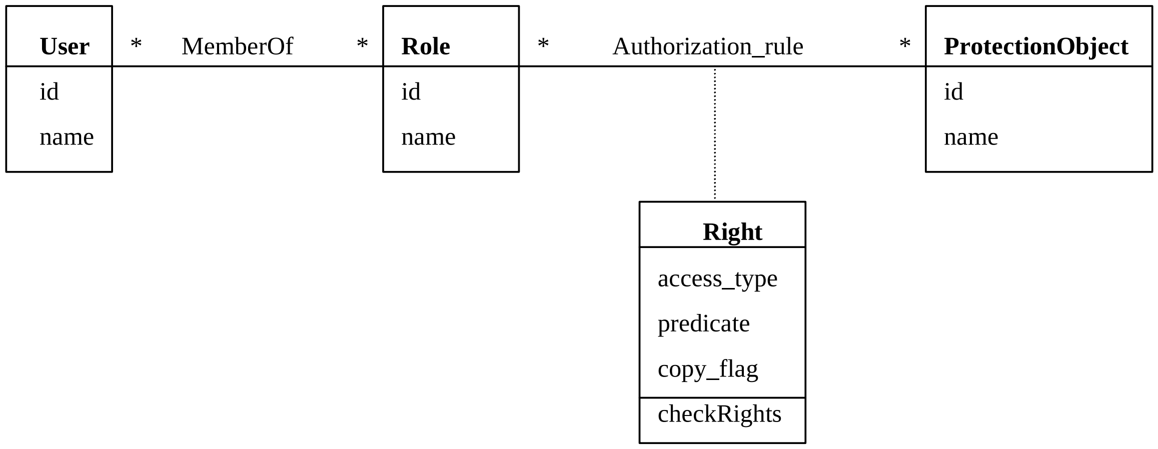


Figure 2.8 The basic RBAC model

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 define its needed rights. Roles can be structured (as shown in the pattern of Figure 2.9), 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].

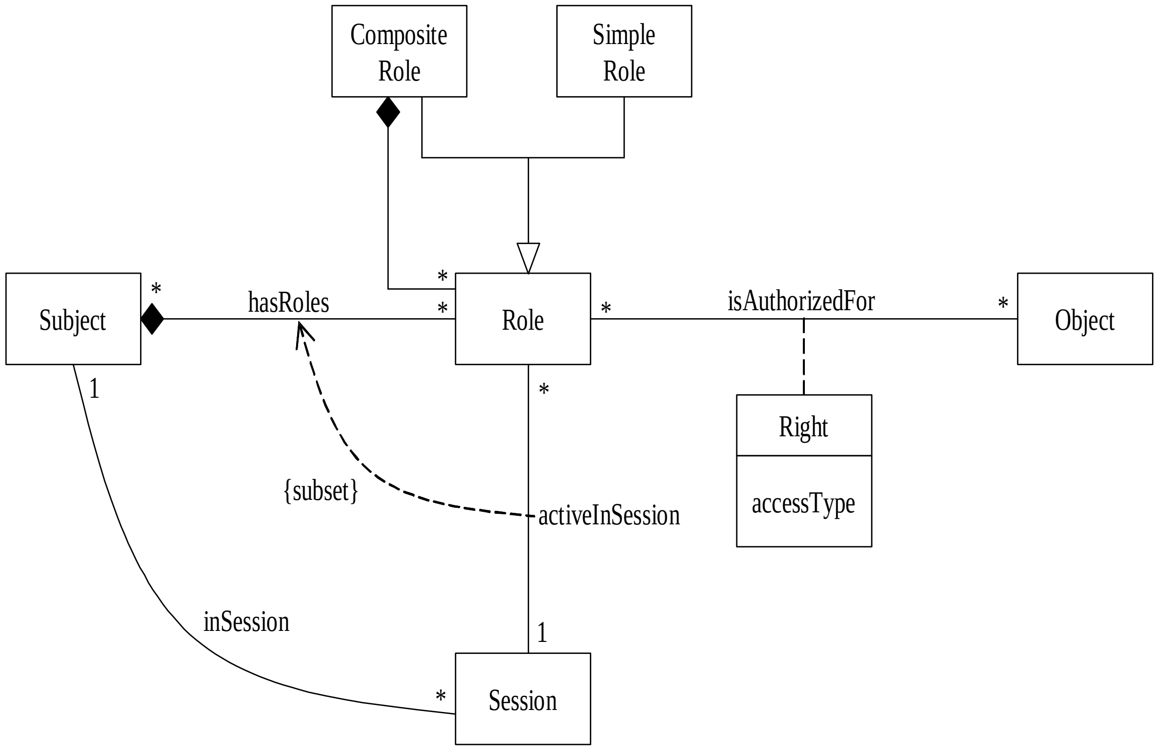


Figure 2.9 The RBAC pattern

Multilevel models

This type of model corresponds to the multilevel policies where data is classified into sensitivity levels and users have access according to their clearances. Vertical partitions are defined by *categories* or *compartments* (Figure 2.10). Because of the way they control security they have also been called *data flow* models, they control the allowed flow of data between levels. These models have been formalized in three different ways:

- The Bell-La Padula model, intended to control leakage of information between levels.
- The Biba model, which controls data integrity.
- The Lattice model, which generalizes the partially ordered levels of the previous models using the concept of mathematical lattices.

The Bell-La Padula confidentiality model

This model classifies subjects and data into *sensitivity levels*. Orthogonal to these levels, *compartments* or *categories* are defined, which correspond to divisions or groupings within each level. The *classification*, C, of a data object defines its sensitivity. Similarly, users or subjects in general are given *clearance* levels. In each level an access matrix may further refine access control.

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. For example, in Figure 2.11 (from [Woo80]), level L_1 dominates level L_2. Security levels L_1 and L_3 or L_2 and L_3 are incomparable. 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) => 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) => 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 2.11 shows a pattern to describe this model.

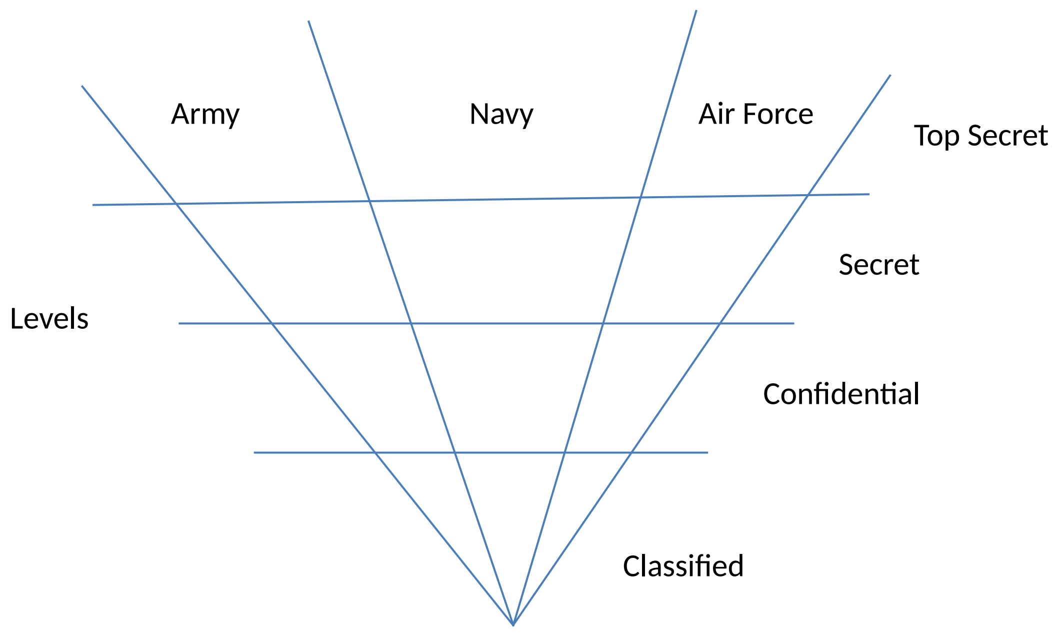


Figure 2.10 The multilevel model

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 the properties:

Single integrity property. Subject s can modify object o only if I(s) >= I(o).
Integrity *-property. If subject s has read access to object o with integrity level I(o), s can write object p only if I(o) >= I(p).

The first property establishes that an untrustful subject cannot write in objects of a higher level of integrity or she would degrade that object.

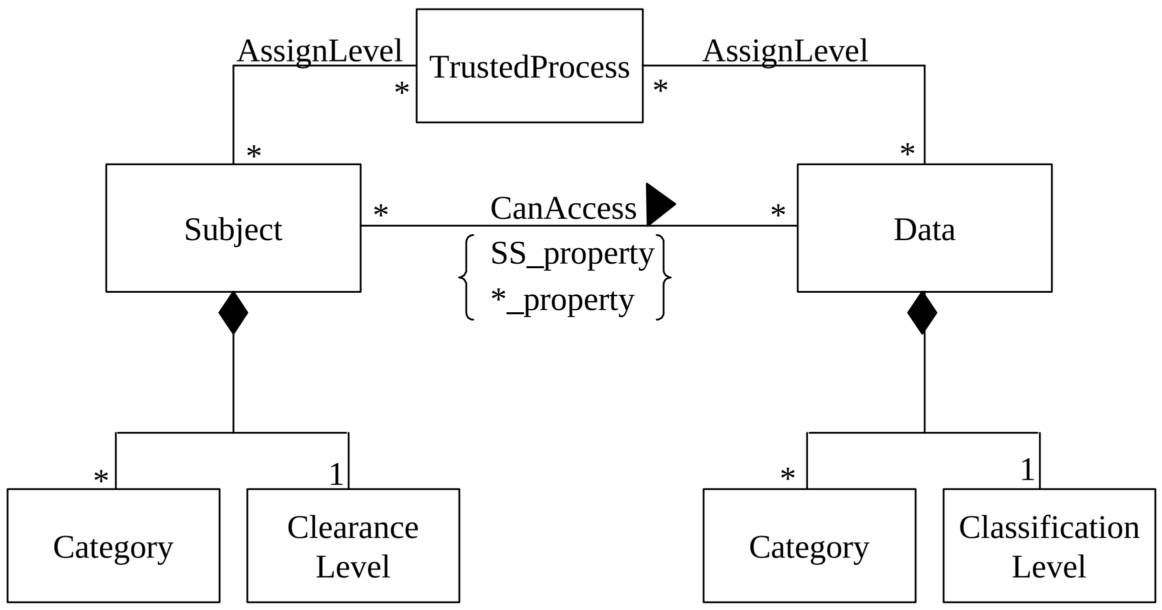
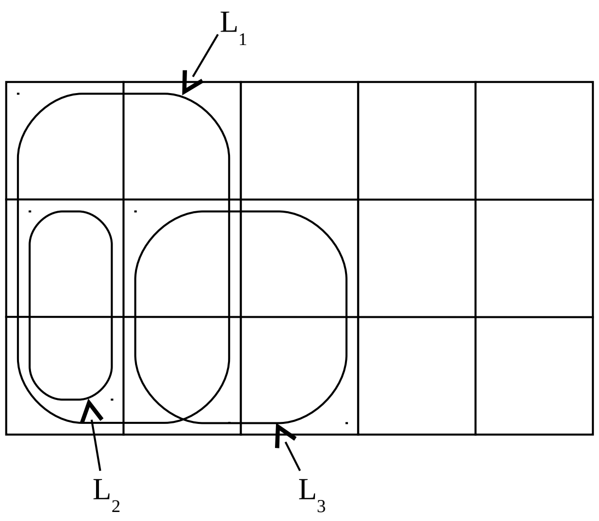


Figure 2.11. Multilevel security pattern



L_1 dominates L_2
L_1 and L_3 are not comparable

Figure 2.12 Levels and compartments

The Lattice model

A lattice is a mathematical structure consisting of partially ordered elements, 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 than simple hierarchies. They are also hard to use; for example, it is very difficult to make a lattice out of a set of data items with different access constraints. Figure 2.12 shows a lattice (from [Den82]) that combines data from three units (m.f,c) in a company (using a class-combining operator); the second level allows some users to have access to data from pairs of units, and the third level gives access to the whole data. It is clear that most organizations are not hierarchic and it is not easy to define who should access which level. Because of this they are rarely used in practice. A detailed description can be found in D. Denning's book [Den82].

Application of multilevel models

Multilevel models are theoretically the most secure of the three basic security models. As we will see later, it is possible to build DBMSs and operating systems 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 layer or compartmentalize their functions or their administration for security; e.g., operating systems and firewalls.

The Clark-Wilson model
When the military were controlling the study of security in the US, this model drew attention to the fact that for business, application integrity was a much more important aspect than confidentiality [Clar87]. It emphasizes well-formed transactions and separation of duty.

Enforcing authorization constraints
The three models described above describe the structure of authorization constraints, more precisely who can access what and how. To have a secure system these authorization constraints must be enforced when some active entity (user, process) tries to access a resource. We can do that by applying the principle of complete mediation, we intercept every request for resources and we validate it according to the authorization rules. The interceptor is known as the Reference Monitor. Figure 2.7 describes the general idea: process pi running on behalf of user U1 attempts to read file F1, the Reference Monitor RM intercepts the request and validates it according to the authorization rules; if one of them authorizes the request the action proceeds.

$$\begin{tikzcd}
 & \{m, f, c\} & \\
\{m, f\} \arrow[ur] & \{m, c\} \arrow[u] & \{f, c\} \arrow[ul] \\
\{m\} \arrow[u] \arrow[ur] & \{f\} \arrow[ul] \arrow[ur] & \{c\} \arrow[ul] \arrow[u] \\
 & \emptyset \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2.12 A lattice

The Reference Monitor pattern [Fer13]

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 and provide the subject with a decision which can have conditions.

Context

A multiprocessing environment where subjects request protection objects to perform their functions and access resources maybe bsed on conditions.

Problem

Authorization can be defined according to different models, e.g. access matrix or Role-Based Access Control. 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?

Decisions can be sometimes more complex than a Boolean response, e.g. when a user wants to access some database items, she may be authorized to access a subset of the data she requests. In this case, the decision is not Boolean (yes/no) but only some items are released. Also, many times there is a need to store decisions for possible reuse.

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, checks them for compliance with authorizations, makes decisions based on these authorization rules and stores those decisions including their attributes (Figure 2.13).

Figure 2.14 shows a class diagram showing a Reference Monitor with a reified decision. In this figure **Set_of_Authorization_Rules** denotes a collection of **Authorization** rules organized in a convenient way and corresponding to some security model. The Actual Subject (Client) can be a process which makes a **Request** for a Protection Object with the intention of applying to it some access type. The **Reference Monitor** makes a **Decision** according to the authorization rules. Examples of **Concrete Reference Monitors** are file permission systems, **memory** managers, or other resource controllers. Figure 2.15 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.

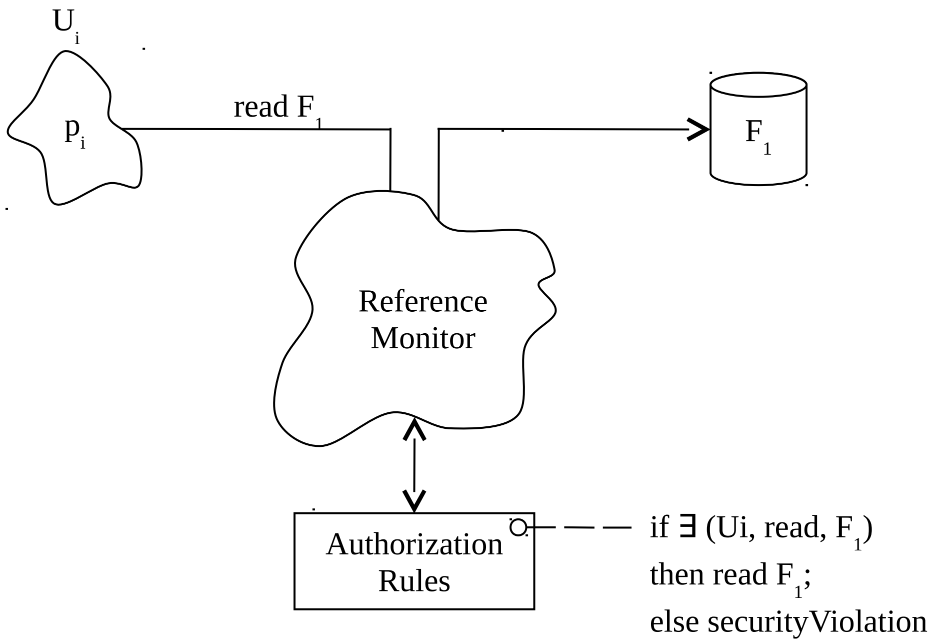


Figure 2.13 The idea of the Reference Monitor

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.

Models and secure system design

The access matrix and RBAC control access to resources while the multilevel models control information flow. Both the access Matrix and RBAC have been extended in many ways to consider all kinds of specialized restrictions. Both can represent content-dependent restrictions for example. In general, multilevel and lattice models are too rigid for non-military applications. All these models define the structure of the rules that condition access to resources, we still need a way to enforce these rules, this is the concept of Reference Monitor, described above as a pattern.

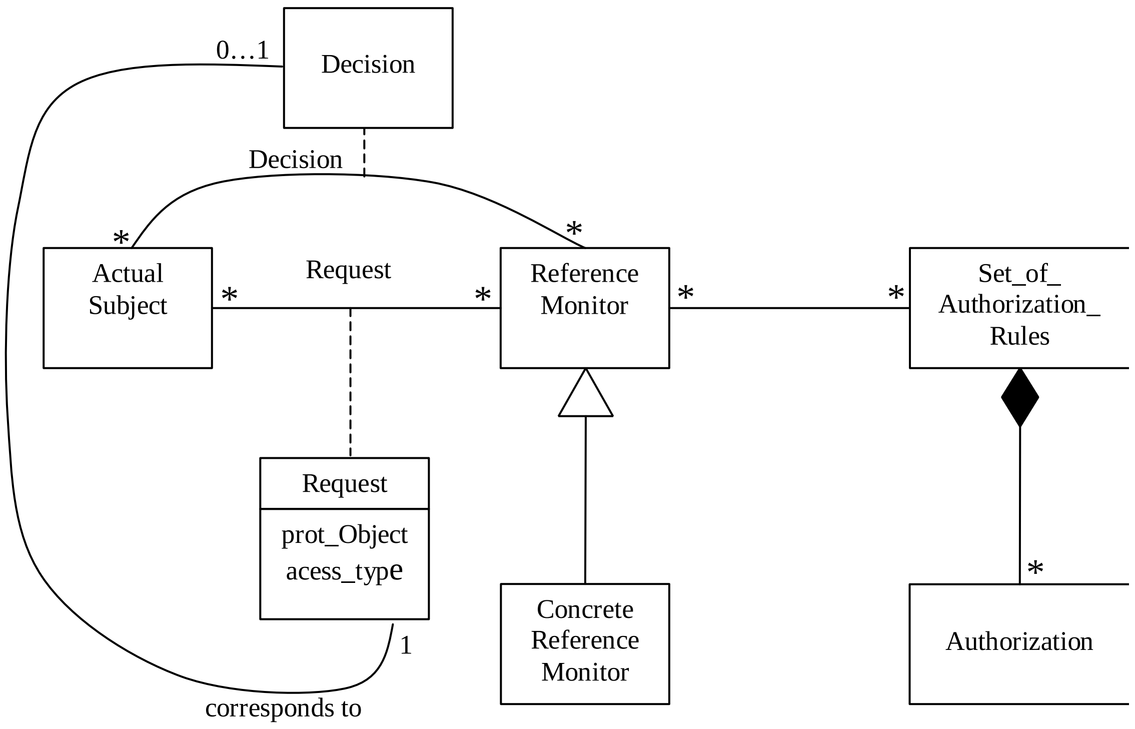
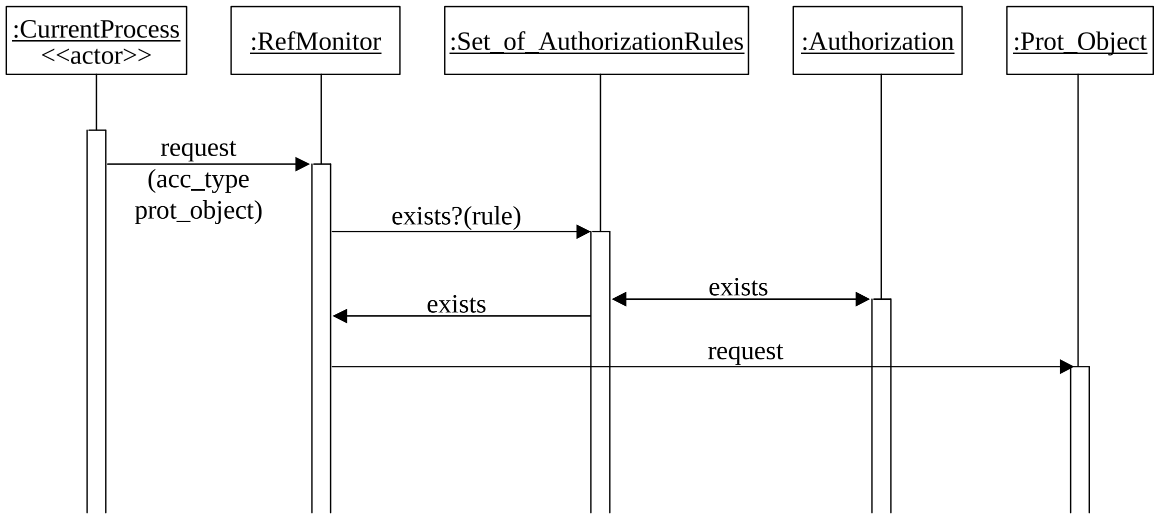


Figure 2.14 Class diagram for the Reference Monitor



:CurrentProcess <<actor>>
:RefMonitor
:Set_of_AuthorizationRules
:Authorization
:Prot_Object
request (acc_type prot_object)
exists?(rule)
exists
exists
request

Figure 2.15 Sequence diagram for enforcing authorization rules

The models we have discussed 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. What this means is that the models just provide mechanisms and it is still necessary to define the rules considering the semantics of the application and institution policies.

There are examples of three of the four 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 application servers. RBAC models can be considered mandatory in that users don't have the freedom to change roles. 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.

Once we have decided about the policies we want for a given system, 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 2.6 is then used as a starting point to further define the system: Do we need the concept of session? Do we need structures of roles? Figure 2.16 is a pattern diagram showing several access control patterns that a designer could use to make this decision. 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 4 we consider each level in detail.

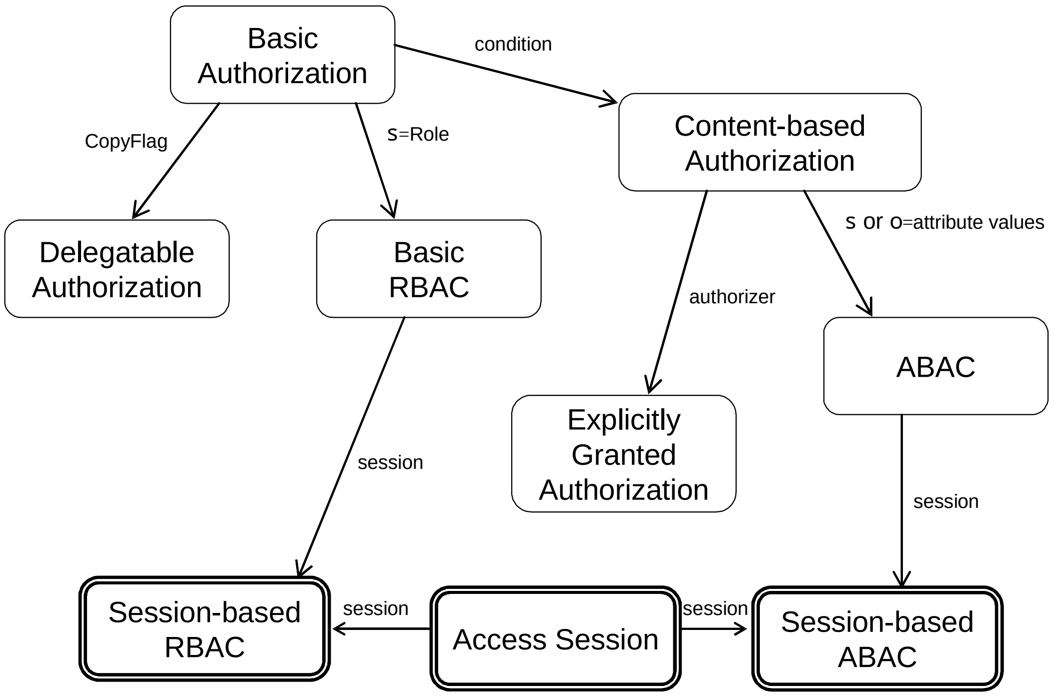


Figure 2.16. Relationships between access control patterns

Governance

Exercises

2.1 Consider some of the examples of attacks in Chapter 1, e.g. the Bagle worm, and study if they could have been avoided by having mechanisms that implemented appropriate policies. Indicate these policies and discuss how they could prevent the attack.

2.2 The flowchart of Figure 2.17 describes how a type of screensaver works. In addition to its normal work this software contains a Trojan Horse that creates a trapdoor in the form of a new account to which the hacker has access. Indicate which of the policies discussed in this chapter could prevent this Trojan Horse from creating this account.

2.3 Enumerate the threats for a use case to enroll students into courses in a university using the Internet. This use case requires the following activities: check that the course is not full, check that the student has the prerequisites for the course, add the student to the course roster, add the course to the student’s transcript, calculate fees.

2.4 Write a set of policies for dealing with medical records related to general patient treatment. Use the HIPAA or the BMA regulations and try to express them in terms of the policies discussed here.

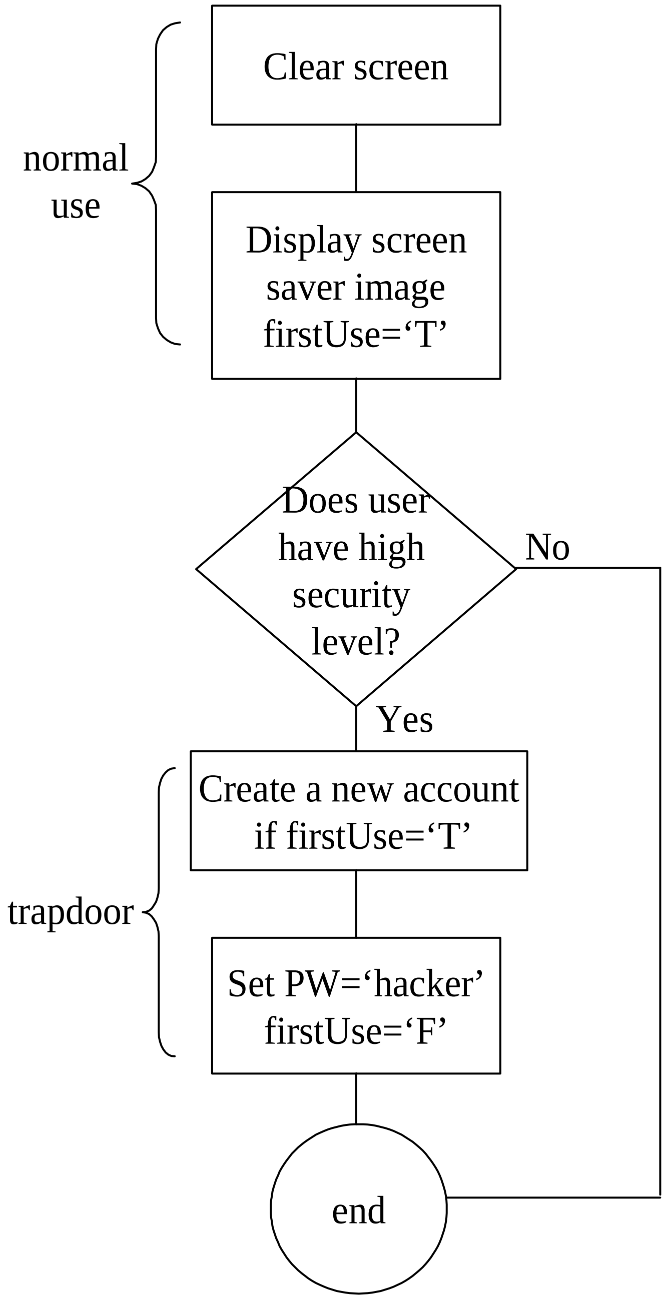

Clear screen
normal use
Display screen saver image firstUse='T'
Does user have high security level?
No
Yes
Create a new account if firstUse='T'
trapdoor
Set PW='hacker' firstUse='F'
end

Figure 2.17 Flowchart for the Trojan Horse of Exercise 2.2

2.5 Express the university policies of Section 2.5 as RBAC authorization rules

2.6 Discuss how appropriate is a multilevel security model for medical applications.

2.7 Study the security regulations of a credit card company or a consortium of credit cards, and evaluate them. If you find flaws, propose improvements.

2.8 Consider a software development environment where groups of software engineers work on projects. What type of security model would you use for this environment?

2.9 Assume that we intend to use an RBAC model for the voting application of the Appendix. Select appropriate roles and assign access rights for each role.

2.10 Add to the model of Figure 2.6 (RBAC class model) the possibility of having named groups of subjects sharing a role. What are the advantages/disadvantages of doing this as compared to assigning individual users to roles. In which situations could this approach be useful?

2. 11 Draw a UML model for the policies:

- Customers in a financial institution can read their own financial records.
- Brokers can read and write their customers' records.
- When a financial record is modified, the customer must be notified.

2.12 Enumerate the threats for a use case to enroll students into courses in a university using the Internet. This use case requires the following activities: check that the course is not full, check that the student has the prerequisites for the course, add the student to the course roster, add the course to the student's transcript, calculate fees.

2.13 One way to anonymize data for research purposes in Health Industry is the approach used by "IBM Anonymous Resolution Information Center ".
In this system there are three levels of security:
Level 1. Sensitive data is visible in normal and encrypted form . For this view people need be owners of data.
Level 2. Sensitive data is visible only in encrypted form. For this view people need to receive a key that will not show them the data
Level 3. Sensitive data is invisible in both visible and encrypted form. This level is for people that had not received the key

The way this program works is the following.
1.The original document that is to be distributed is kept unchanged and a copy is made. The secure access to the original document would be allowed to the document owner.
2.Data that can be dangerous for identity is encrypted using a key
3.Encrypted data and the key is exported for view to other entities.

4.In order to process the data the key is needed. Even with key the secure data is not displayed. It is only processed in encrypted form to make sure there are no duplicates in data. If conflict that needs resolution arrives, problem of data is sent to source for resolution.

The advantage of this system is that even if data is visible by anyone, sensitive data is encrypted being actually useless. The system can see sensitive data internally making sure there is no entity viewed as one.

These rules are confusing. Simplify and express these policies using the policies we discussed in Section 2.2.

References

[And96] R.J.Anderson, "Security in clinical information systems", January 1996.

[And02] M. Andress, "An overview of security policies", http://searchsecurity.techtarget.com, December, 2002.

[Bla00] D. Blacharski, "Emerging Technology: Create order with a strong security policy", *Network Magazine*, July 2000, http://www.networkmagazine.com/article/NMG20000710S0015

[Bra08] F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" *Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07)*. Turin, Italy, September 1-5, 2008. 328-333.

[cc] Common Criteria Portal, http://www.commoncriteriaportal.org/

[Cla96] D.D.Clark and D.R.Wilson, "A comparison of commercial and military security policies", *Procs. of the 1987 IEEE Comp. Sec. Symp. On Research on Security and Privacy*, 184-194.

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

[Fer75a] E. B. Fernandez, R. C. Summers, and C. B. Coleman, ``An authorization model for a shared data base," *Proceedings of the 1974 SIGMOD International Conference*, ACM, New York, pp. 23-31, 1975.

[Fer75b] 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.

[Fer81] E.B.Fernandez, R.C.Summers, and C. Wood, *Database security and integrity*, Addison-Wesley 1981.

[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.

[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

[Fer13] E.B.Fernandez, "*Security patterns in practice: Building secure architectures using software patterns*", Wiley Series on Software Design Patterns, 2013.

[Gra72] G.S. Graham and P. Denning, "Protection: Principles and practice", *AFIPS Conf. Procs*., 40, 1972 SJCC, 417-429.

[Gre06] S.S. Greene, *Security policies and procedures, Principles and practice*, Pearson Prentice-Hall, 2006.

[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.

[iet04] IETF Policy Framework, 2004, http://www.ietf.org/wg/concluded/policy.html

[Jaj01] S. Jajodia, P. Samarati, M.L. Sapino, and V. S. Subrahmanian. "Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26(2):214--260, 2001.

[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.

[Kue95a] W.E. Kuenhauser and M. von Kopp Ostrowski, "A framework to support multiple security policies", *Procs. of the 7th Annual Canadian Comp. Security Symp.*, 1995.

[Kue95b] W.E. Kuehnhauser, "A paradigm for user-defined security policies", *Procs. of the 14th IEEE Symp. on Reliable Distributed Systems*. 1995.

[Lam71] B.W. Lampson, "Protection", *Procs. 5th Annual Conf. on Info. Sciences and Systs.*,1971, 437-443. Reprinted in *ACM Operating Systs. Review*, 8, 1 (January 1974), 18-24.

[list] George Mason University Laboratory for Information Security Technology, http://www.list.gmu.edu

[Lup97a] E. Lupu and M.Sloman, "Conflict analysis for management policies", May 1997.

[Lup97b] E. Lupu and M. Sloman, "A policy based role object model", *Procs. of EDOC'97*, IEEE 1997.

[McG00] G. McGraw and G. Morrisett, "Attacking malicious code: A Report to the Infosec Research Council", IEEE Software, Sept./October 2000, 33-41.

[Mea00] N. Mead (coordinator), Malicious IT Roundtable: Roundtable on Information Security Policy, *IEEE Software*, July 2000.

[Mof88] J.D.Moffett and M. Sloman, "The source of authority for commercial access control", *Computer*, IEEE, February 1988, 59-69.

[nis] NIST , http://csrc.nist.gov/rbac

[nis09] NIST Special Publication 800-53 Revision 3 Recommended Security Controls for Federal Information Systems and Organizations http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final-errata.pdf

[Per92] G. Pernul and A.M.Tjoa, "Security policies for databases", *Procs. of Int. Conf. on Safety and Reliability of Computers (SAFECOMP-92),* Zurich, Switzerland, Oct. 1992.

[Pol] International Workshop on Policies for Distributed Systems and Networks, http://www.policy-workshop.org/

[Sal75] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems", Procs. of the IEEE, vol. 63, No 9, Sept.1975, 1278-1308.

[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.

[Sch06] M. Schumacher, E. B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering"*, Wiley 2006. Wiley Series on Software Design Patterns.

[Slo02] M. Sloman and E. Lupu, "Security and management policy specification", *IEEE*

Network, March/April 2002, 10-19.

[Sny81] L. Snyder, "Formal models of capability-based protection systems", *IEEE Trans.*
on Computers, Vol. C-30, No 3, March 1981, 172-181.

[Str01] J. Strassner et al., "Policy Core Information Model"—Version 1 Specification", RFC 3060, February 2001, http://www.ietf.org/rfc/rfc3060.txt

[Sum97] R.C.Summers, *Secure computing: Threats and safeguards*, McGraw-Hill, 1997.

[Ver06] D. Verdon, "Security policies and the software developer", *IEEE Security & Privacy*, July/August 2006, 42-49.

[Vij02] J. Vijayan, "Employee data exposed on web", Computerworld, Feb. 11, 2002, http://www.computerworld.com

[Woo80] C.Wood, E.B.Fernandez, and R.C. Summers, "Data base security: requirements, policies, and models", *IBM Systems Journal*, vol. 19, No 2, 1980, 229-252.

[Woo00] C.C. Wood, *Information security policies made easy*, Version 7, 2000. http://www.netiq.com/products/securitymgmt/default.asp

[Yod97] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security". *Procs. PLOP'97*. Also Chapter 15 in *Pattern Languages of Program Design*, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Addison-Wesley, 2000.