CHAPTER 13

TRUSTED COMPUTING AND MULTILEVEL SECURITY

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Explain the Bell-LaPadula model and its relevance to trusted computing.
- Summarize other formal models for computer security.
- Understand the concept of trusted systems.
- List and explain the properties of a reference monitor and explain the relationship between a reference monitor and a security kernel database.
- Present an overview of the application of multilevel security to role-based access control and to database security.
- Discuss hardware approaches to trusted computing.
- Explain and summarize the common criteria for information technology security evaluation.

This chapter deals with a number of interrelated topics having to do with the degree of confidence users and implementers can have in security functions and services:

- Formal models for computer security
- Multilevel security
- Trusted systems
- Mandatory access control
- Security evaluation

13.1 THE BELL-LAPADULA MODEL FOR COMPUTER SECURITY

Computer Security Models

Two historical facts highlight a fundamental problem that needs to be addressed in the area of computer security. First, all complex software systems have eventually revealed flaws or bugs that subsequently needed to be fixed. A good discussion of this can be found in the classic *The Mythical Man-Month* [BROO95]. Second, it is extraordinarily difficult, if not impossible, to build a computer hardware/software system that is not vulnerable to a variety of security attacks. An illustration of this difficulty is the Windows NT operating system, introduced by Microsoft in the early 1990s. Windows NT was promised to have a high degree of security and to be far superior to previous OSs, including Microsoft's Windows 3.0 and many other personal computer, workstation, and server OSs. Sadly, Windows NT did not deliver on this promise. This OS and its successor Windows versions have been chronically plagued with a wide range of security vulnerabilities.

Problems to do with providing strong computer security involved both design and implementation. It is difficult, in designing any hardware or software module, to be assured that the design does in fact provide the level of security that was intended. This difficulty results in many unanticipated security vulnerabilities. Even

if the design is in some sense correct, it is difficult, if not impossible, to implement the design without errors or bugs, providing yet another host of vulnerabilities.

These problems have led to a desire to develop a method to prove, logically or mathematically, that a particular design does satisfy a stated set of security requirements and that the implementation of that design faithfully conforms to the design specification. To this end, security researchers have attempted to develop formal models of computer security that can be used to verify security designs and implementations.

Initially, research in this area was funded by the U.S. Department of Defense and considerable progress was made in developing models and in applying them to prototype systems. That funding has greatly diminished as have attempts to build formal models of complex systems. Nevertheless, such models have value in providing a discipline and a uniformity in defining a design approach to security requirements [BELL05]. In this section, we look at perhaps the most influential computer security model, the Bell-LaPadula (BLP) model [BELL73, BELL75]. Several other models are examined in Section 13.2.

General Description

The BLP model was developed in the 1970s as a formal model for access control. The model relied on the access control concept described in Chapter 4 (e.g., Figure 4.4). In the model, each subject and each object is assigned a **security class**. In the simplest formulation, security classes form a strict hierarchy and are referred to as **security levels**. One example is the U.S. military classification scheme:

$$\text{top secret} > \text{secret} > \text{confidential} > \text{restricted} > \text{unclassified}$$

It is possible to also add a set of categories or compartments to each security level, so that a subject must be assigned both the appropriate level and category to access an object. We ignore this refinement in the following discussion.

This concept is equally applicable in other areas, where information can be organized into gross levels and categories and users can be granted clearances to access certain categories of data. For example, the highest level of security might be for strategic corporate planning documents and data, accessible by only corporate officers and their staff; next might come sensitive financial and personnel data, accessible only by administration personnel, corporate officers, and so on. This suggests a classification scheme such as

$$\text{strategic} > \text{sensitive} > \text{confidential} > \text{public}$$

A subject is said to have a **security clearance** of a given level; an object is said to have a **security classification** of a given level. The security classes control the manner by which a subject may access an object. The model defined four access modes, although the authors pointed out that in specific implementation environments, a different set of modes might be used. The modes are as follows:

- **read:** The subject is allowed only read access to the object.
- **append:** The subject is allowed only write access to the object.

- **write:** The subject is allowed both read and write access to the object.
- **execute:** The subject is allowed neither read nor write access to the object but may invoke the object for execution.

When multiple categories or levels of data are defined, the requirement is referred to as **multilevel security (MLS)**. The general statement of the requirement for confidentiality-centered multilevel security is that a subject at a high level may not convey information to a subject at a lower level unless that flow accurately reflects the will of an authorized user as revealed by an authorized declassification. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system for confidentiality must enforce the following:

- **No read up:** A subject can only read an object of less or equal security level. This is referred to in the literature as the **simple security property (ss-property)**.
- **No write down:** A subject can only write into an object of greater or equal security level. This is referred to in the literature as the ***-property**[1] (pronounced *star property*).

Figure 13.1 illustrates the need for the *-property. Here, a malicious subject passes classified information along by putting it into an information container labeled at a lower security classification than the information itself. This will allow a subsequent read access to this information by a subject at the lower clearance level.

These two properties provide the confidentiality form of what is known as **mandatory access control** (MAC). Under this MAC, no access is allowed that does not satisfy these two properties. In addition, the BLP model makes a provision for discretionary access control (DAC).

- **ds-property:** An individual (or role) may grant to another individual (or role) access to a document based on the owner's discretion, constrained by the MAC rules. Thus, a subject can exercise only accesses for which it has the necessary authorization and which satisfy the MAC rules.

The basic idea is that site policy overrides any discretionary access controls. That is, a user cannot give away data to unauthorized persons.

Formal Description of Model

We use the notation presented in [BELL75]. The model is based on the concept of a current state of the system. The state is described by the 4-tuple (b, M, f, H), defined as follows:

- **Current access set *b*:** This is a set of triples of the form (subject, object, access-mode). A triple (s, o, a) means that subject s has current access to o in access mode a. Note that this does not simply mean that s has the access right a to o. The triple means that s is currently exercising that access right; that is s is currently accessing o by mode a.

[1]The "*" does not stand for anything. No one could think of an appropriate name for the property during the writing of the first report on the model. The asterisk was a dummy character entered in the draft so that a text editor could rapidly find and replace all instances of its use once the property was named. No name was ever devised, and so the report was published with the "*" intact.

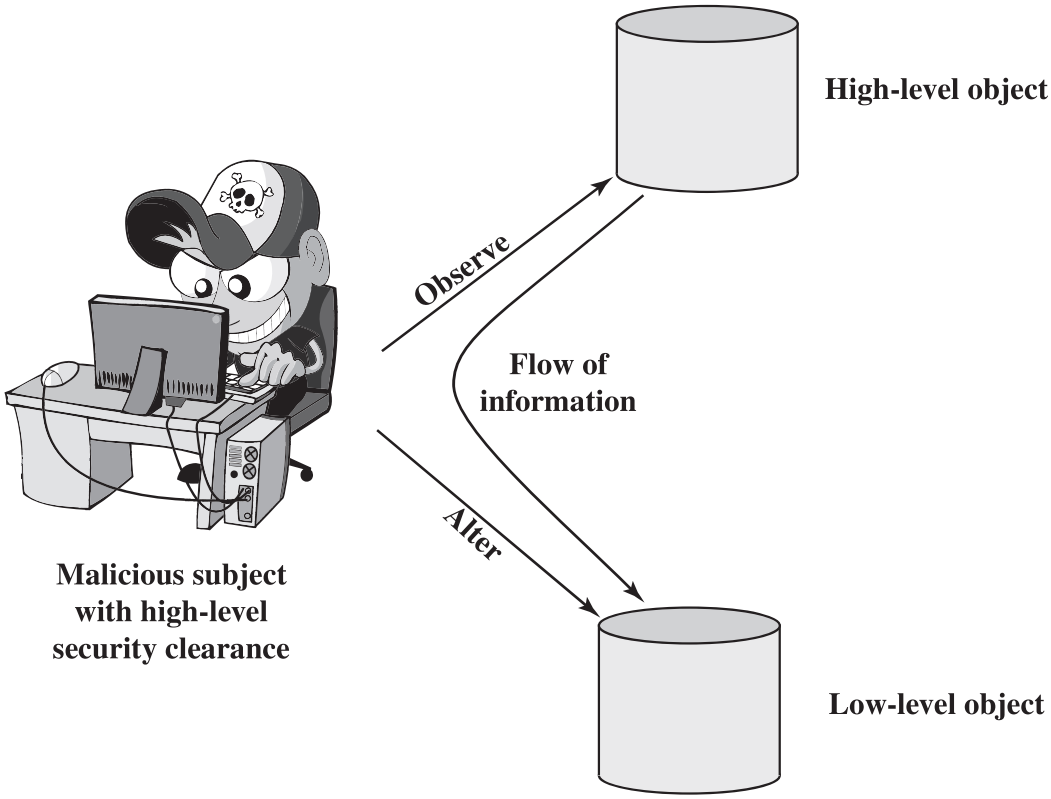


Figure 13.1 **Information Flow Showing the Need for the *-Property**

- **Access matrix *M*:** The access matrix has the structure indicated in Chapter 4. The matrix element M_{ij} records the access modes in which subject S_i is permitted to access object O_j.
- **Level function *f*:** This function assigns a security level to each subject and object. It consists of three mappings: $f_o(O_j)$ is the classification level of object O_j; $f_s(S_i)$ is the security clearance of subject S_i; $f_c(S_i)$ is the current security level of subject S_i. The security clearance of a subject is the maximum security level of the subject. The subject may operate at this level or at a lower level. Thus, a user may log onto the system at a level lower than the user's security clearance. This is particularly useful in a role-based access control system.
- **Hierarchy *H*:** This is a directed rooted tree whose nodes correspond to objects in the system. The model requires that the security level of an object must dominate the security level of its parent. For our discussion, we may equate this with the condition that the security level of an object must be greater than or equal to its parent.[2]

[2]The concept of dominance allows for a more complex security classification structure involving both security levels and compartments. This refinement, developed in the military, is not essential for our discussion.

We can now define the three BLP properties more formally. For every subject S_i and every object O_j, the requirements can be stated as follows:

- **ss-property:** Every triple of the form (S_i, O_j, read) in the current access set b has the property $f_c(S_i) \geq f_o(O_j)$.
- ***-property:** Every triple of the form (S_i, O_j, append) in the current access set b has the property $f_c(S_i) \leq f_o(O_j)$. Every triple of the form (S_i, O_j, write) in the current access set b has the property $f_c(S_i) = f_o(O_j)$.
- **ds-property:** If (S_i, O_j, A_x) is a current access (is in b), then access mode A_x is recorded in the (S_i, O_j) element of M. That is, (S_i, O_j, A_x) implies that $A_x \in M[S_i,O_j]$.

These three properties can be used to define a confidentiality secure system. In essence, a secure system is characterized by the following:

1. The current security state of the system (b, M, f, H) is secure if and only if every element of b satisfies the three properties.
2. The security state of the system is changed by any operation that causes a change any of the four components of the system, (b, M, f, H).
3. A secure system remains secure so long as any state change does not violate the three properties.

[BELL75] shows how these three points can be expressed as theorems using the formal model. Further, given an actual design or implementation, it is theoretically possible to prove the system secure by proving that any action that affects the state of the system satisfies the three properties. In practice, for a complex system, such a proof has never been fully developed. However, as mentioned earlier, the formal statement of requirements can lead to a more secure design and implementation.

Abstract Operations

The BLP model includes a set of rules based on abstract operations that change the state of the system. The rules are as follows:

1. **Get access:** Add a triple (*subject*, *object*, *access-mode*) to the current access set b. Used by a subject to initiate access to an object in the requested mode.
2. **Release access:** Remove a triple (*subject*, *object*, *access-mode*) from the current access set b. Used to release previously initiated access.
3. **Change object level:** Change the value of $f_o(O_j)$ for some object O_j. Used by a subject to alter the security level of an object.
4. **Change current level:** Change the value of $f_c(S_i)$ for some subject S_i. Used by a subject to alter the security level of a subject.
5. **Give access permission:** Add an access mode to some entry of the access permission matrix M. Used by a subject to grant an access mode on a specified object to another subject.
6. **Rescind access permission:** Delete an access mode from some entry of M. Used by a subject to revoke an access previously granted.

7. **Create an object:** Attach an object to the current tree structure H as a leaf. Used to create a new object or activate an object that has previously been defined but is inactive because it has not been inserted into H.
8. **Delete a group of objects:** Detach from H an object and all other objects beneath it in the hierarchy. This renders the group of objects inactive. This operation may also modify the current access set b because all accesses to the object are released.

Rules 1 and 2 alter the current access; rules 3 and 4 alter the level functions; rules 5 and 6 alter access permission; and rules 7 and 8 alter the hierarchy. Each rule is governed by the application of the three properties. For example, for get access for a read, we must have $f_c(S_i) \geq f_o(O_j)$ and $A_x \in M[S_i,O_j]$.

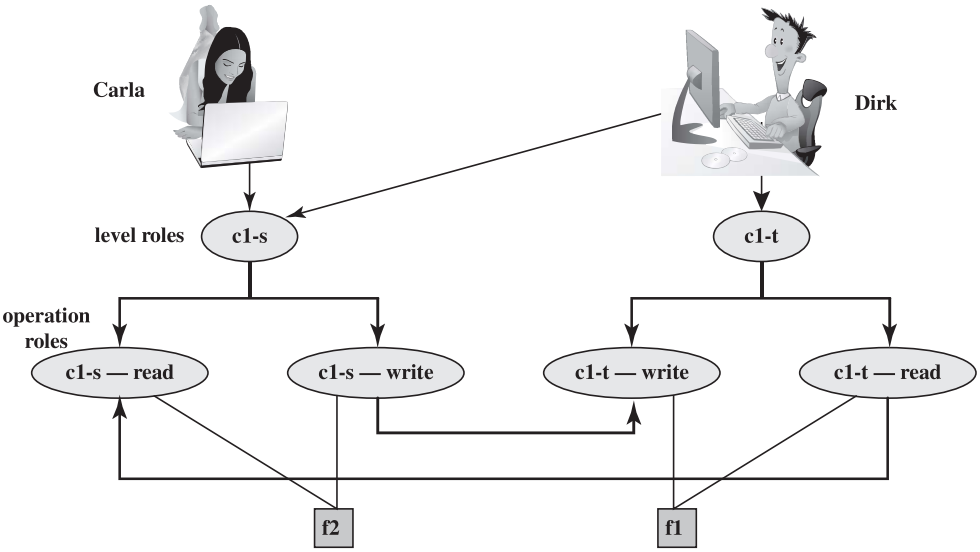
Example of BLP Use

This example illustrates the operation of the BLP model and also highlights a practical issue that must be addressed. We assume a role-based access control system. Carla and Dirk are users of the system. Carla is a student (s) in course c1. Dirk is a teacher (t) in course c1 but may also access the system as a student; thus two roles are assigned to Dirk:

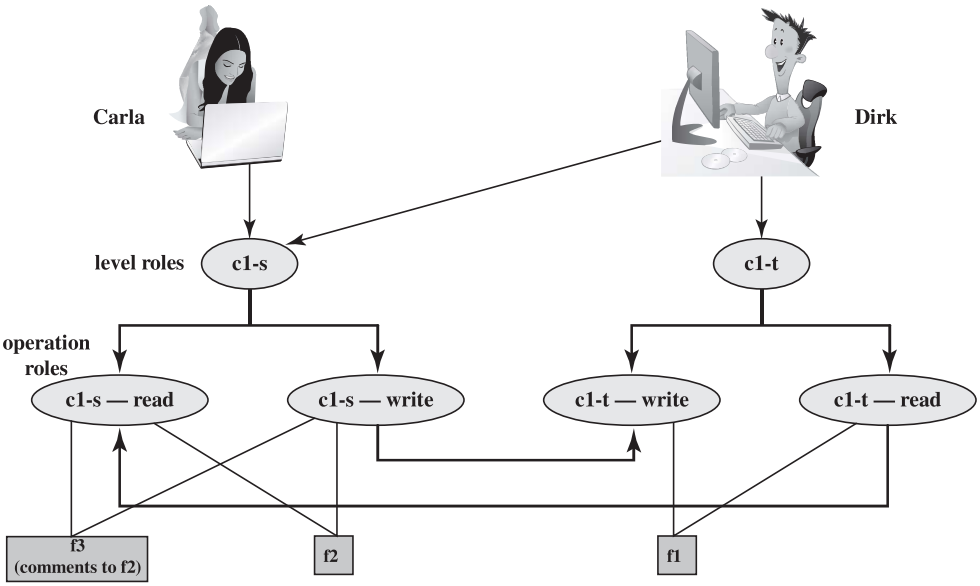
Carla: (c1-s)
Dirk: (c1-t), (c1-s)

The student role is assigned a lower security clearance and the teacher role a higher security clearance. Let us look at some possible actions:

1. Dirk creates a new file f1 as c1-t; Carla creates file f2 as c1-s (Figure 13.2a). Carla can read and write to f2, but cannot read f1, because it is at a higher classification level (teacher level). In the c1-t role, Dirk can read and write f1 and can read f2 if Carla grants access to f2. However, in this role, Dirk cannot write f2 because of the *-property; neither Dirk nor a Trojan horse on his behalf can downgrade data from the teacher level to the student level. Only if Dirk logs in as a student can he create a c1-s file or write to an existing c1-s file, such as f2. In the student role, Dirk can also read f2.
2. Dirk reads f2 and wants to create a new file with comments to Carla as feedback. Dirk must sign in student role c1-s to create f3 so that it can be accessed by Carla (Figure 13.2b). In a teacher role, Dirk cannot create a file at a student classification level.
3. Dirk creates an exam based on an existing template file store at level c1-t. Dirk must log in as c1-t to read the template and the file he creates (f4) must also be at the teacher level (Figure 13.2c).
4. Dirk wants Carla to take the exam and so must provide her with read access. However, such access would violate the ss-property. Dirk must downgrade the classification of f4 from c1-t to c1-s. Dirk cannot do this in the c1-t role because this would violate the *-property. Therefore, a security administrator (possibly Dirk in this role) must have downgrade authority and must be

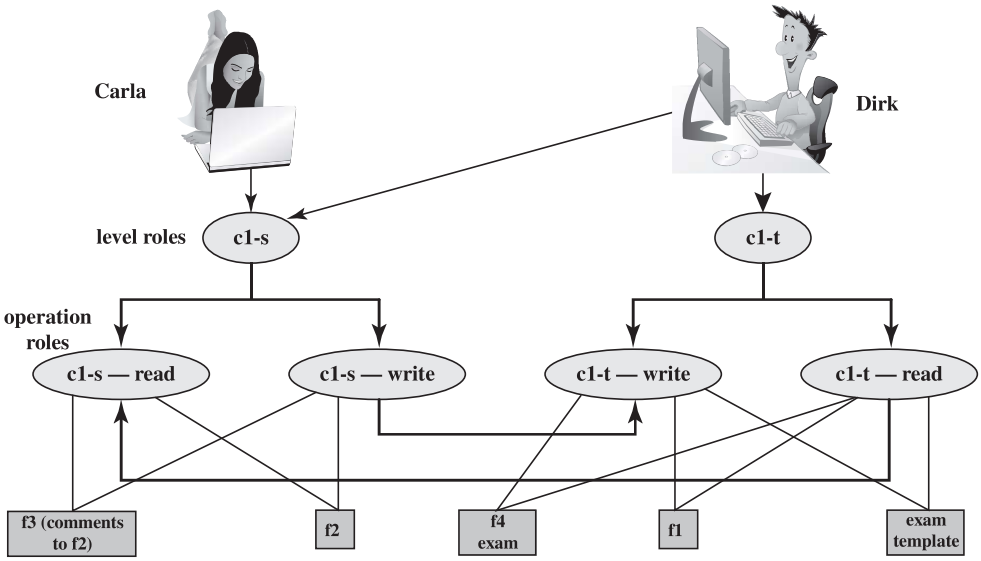


(a) Two new files are created: f1: c1-t; f2: c1-s

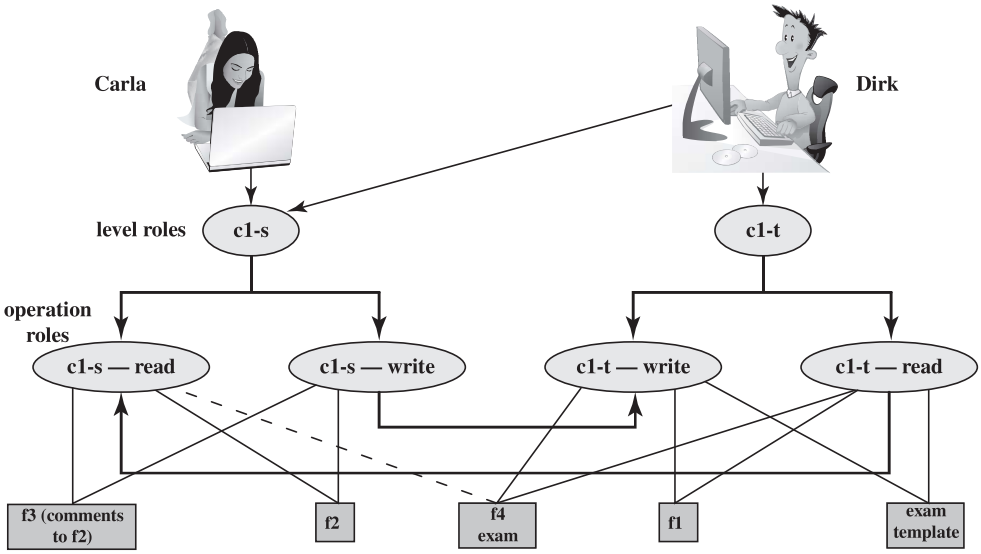


(b) A third file is added: f3: c1-s

Figure 13.2 **Example of Use of BLP Concepts**

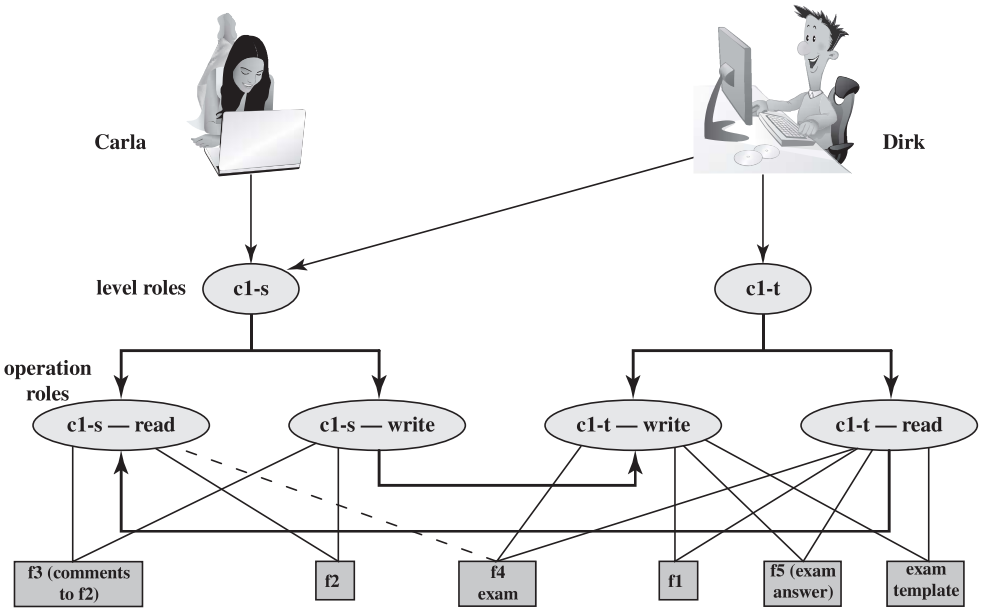


(c) An exam is created based on an existing template: f4: c1-t



(d) Carla, as student, is permitted acess to the exam: f4: c1-s

Figure 13.2 *(Continued)*



(e) The answers given by Carla are only accessible for the teacher: f5: c1-t

Figure 13.2 *(Continued)*

able to perform the downgrade outside the BLP model. The dotted line in Figure 13.2d connecting f4 with c1-s-read indicates that this connection has not been generated by the default BLP rules but by a system operation.

5. Carla writes the answers to the exam into a file f5. She creates the file at level c1-t so that only Dirk can read the file. This is an example of writing up, which is not forbidden by the BLP rules. Carla can still see her answers at her workstation but cannot access f5 for reading.

This discussion illustrates some critical practical limitations of the BLP model. First, as noted in step 4, the BLP model has no provision to manage the "downgrade" of objects, even though the requirements for multilevel security recognize that such a flow of information from a higher to a lower level may be required, provided it reflects the will of an authorized user. Hence, any practical implementation of a multilevel system has to support such a process in a controlled and monitored manner. Related to this is another concern. A subject constrained by the BLP model can only be "editing" (reading and writing) a file at one security level while also viewing files at the same or lower levels. If the new document consolidates information from a range of sources and levels, some of that information is now classified at a higher level than it was originally. This is known as *classification creep* and is a well-known concern when managing multilevel information. Again, some process of managed downgrading of information is needed to restore reasonable classification levels.

Implementation Example—Multics

[BELL75] outlines an implementation of MLS on the Multics operating system. We begin with a brief description of the relevant aspects of Multics.

Multics is a time-sharing operating system that was developed by a group at MIT known as Project MAC (multiple-access computers) in the 1960s. Multics was not just years but decades ahead of its time. Even by the mid-1980s, almost 20 years after it became operational, Multics had superior security features and greater sophistication in the user interface and other areas than other contemporary mainframe operating systems.

Both memory management and the file system in Multics are based on the concept of segments. Virtual memory is segmented. For most hardware platforms, paging is also used. In any case, the working space of a process is assigned to a segment and a process may create one or more data segments for use during execution. Each file in the file system is defined as a segment. Thus, the OS uses the same mechanism to load a data segment from virtual memory into main memory and to load a file from virtual memory into main memory. Segments are arranged hierarchically, from a root directory down to individual segments.

Multics manages the virtual address space by means of a descriptor segment, which is associated with a process and which has one entry for each segment in virtual memory accessible by this process. The descriptor segment base register points to the start of the descriptor segment for the process that is currently executing. The descriptor entry includes a pointer to the start of the segment in virtual memory plus protection information, in the form of read, write, and execute bits, which may be individually set to ON or OFF. The protection information found in a segment's descriptor is derived from the access control list for the segment.

For MLS, two additional features are required. A process-level table includes an entry of each active process, and the entry indicates the security clearance of the process. Associated with each segment is a security level, which is stored in the parent directory segment of the segment in question.

Corresponding to the security state of the BLP model (b, M, f, H) is a set of Multics data structures (Figure 13.3). The correspondence is as follows:

- *b:* Segment descriptor word. The descriptor segment identifies the subject (process). The segment pointer in segment descriptor word identifies the object (data segment). The three access control bits in the segment descriptor word identify the access mode.
- *M:* Access control list.
- *f:* Information in the directory segment and in the process-level table.
- *H:* Hierarchical segment structure.

With these data structures, Multics can enforce discretionary and mandatory access control. When a process attempts an access to a segment, it must have the desired access permission as specified by the access control list. Also, its security clearance is compared to the security classification of the segment to be accessed to determine if the simple security rule and *-property security rule are satisfied.

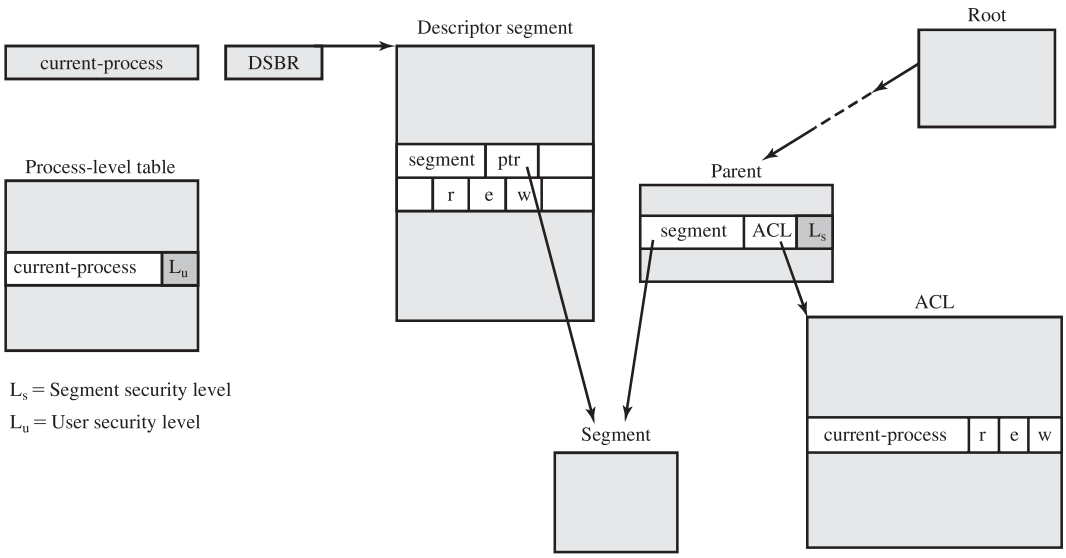


Figure 13.3 **Multics Data Structures for MLS**

Limitations to the BLP model

While the BLP model could in theory lay the foundations for secure computing within a single administration realm environment, there are some important limitations to its usability and difficulties to its implementation.

First, there is the incompatibility of confidentiality and integrity within a single MLS system. In general terms, MLS can work either for *powers* or for *secrets*, but not readily for both. This mutual exclusion excludes some interesting power and integrity centered technologies from being used effectively in BLP style MLS environments.

A second important limitations to usability is the so called *cooperating conspirator* problem in the presence of covert channels. In the presence of shared resources the *-property may become unenforceable. This is especially a problem in the presence of active content that is prevalent in current word processing and other document formats. A malicious document could carry in it a subject that would when executed broadcast classified documents using shared-resource covert channels. In essence, the BLP model effectively breaks down when (untrusted) low classified executable data are allowed to be executed by a high clearance (trusted) subject.

13.2 OTHER FORMAL MODELS FOR COMPUTER SECURITY

It is important to note that the models described in this chapter either focus on confidentiality or on integrity, with the exception of the Chinese Wall Model. The incompatibility of confidentiality and integrity concerns is recognized to be a major limitation to the usability of MLS in general, and to confidentiality focused MLS in specific.

This section explores some other important computer security models.

Biba Integrity Model

The BLP model deals with confidentiality and is concerned with unauthorized disclosure of information. The Biba [BIBA77] models deals with integrity and is concerned with the unauthorized modification of data. The Biba model is intended to deal with the case in which there is data that must be visible to users at multiple or all security levels but should only be modified in controlled ways by authorized agents.

The basic elements of the Biba model have the same structure as the BLP model. As with BLP, the Biba model deals with subjects and objects. Each subject and object is assigned an integrity level, denoted as $I(S)$ and $I(O)$ for subject S and object O, respectively. A simple hierarchical classification can be used, in which there is a strict ordering of levels from lowest to highest. As in the BLP model, it is also possible to add a set of categories to the classification scheme; this we ignore here.

The model considers the following access modes:

- **Modify:** To write or update information in an object.
- **Observe:** To read information in an object.
- **Execute:** To execute an object.
- **Invoke:** Communication from one subject to another.

The first three modes are analogous to BLP access modes. The invoke mode is new. Biba then proposes a number of alternative policies that can be imposed on this model. The most relevant is the strict integrity policy, based on the following rules:

- **Simple integrity:** A subject can modify an object only if the integrity level of the subject dominates the integrity level of the object: $I(S) \geq I(O)$.
- **Integrity confinement:** A subject can read an object only if the integrity level of the subject is dominated by the integrity level of the object: $I(S) \leq I(O)$.
- **Invocation property:** A subject can invoke another subject only if the integrity level of the first subject dominates the integrity level of the second subject: $I(S_1) \geq I(S_2)$.

The first two rules are analogous to those of the BLP model but are concerned with integrity and reverse the significance of read and write. The simple integrity rule is the logical write-up restriction that prevents contamination of high-integrity data. Figure 13.4 illustrates the need for the integrity confinement rule. A low-integrity

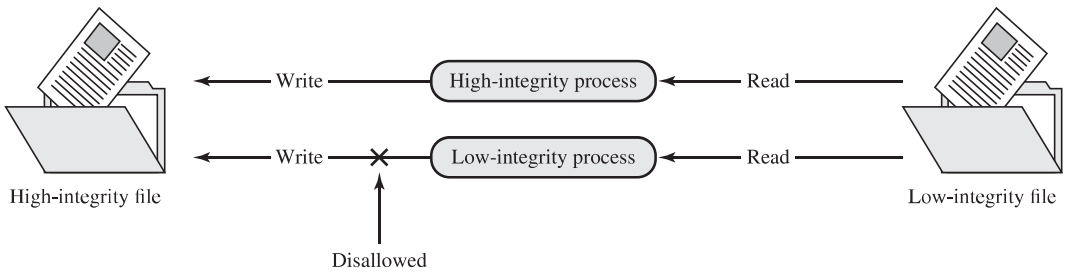


Figure 13.4 Contamination with Simple Integrity Controls
Source: [GASS88].

process may read low-integrity data but is prevented from contaminating a high-integrity file with that data by the simple integrity rule. If only this rule is in force, a high-integrity process could conceivably copy low-integrity data into a high-integrity file. Normally, one would trust a high-integrity process to not contaminate a high-integrity file, but either an error in the process code or a Trojan horse could result in such contamination; hence the need for the integrity confinement rule.

Clark-Wilson Integrity Model

A more elaborate and perhaps more practical integrity model was proposed by Clark and Wilson [CLAR87]. The Clark-Wilson integrity model (CWM) is aimed at commercial rather than military applications and closely models real commercial operations. The model is based on two concepts that are traditionally used to enforce commercial security policies:

- **Well-formed transactions:** A user should not manipulate data arbitrarily, but only in constrained ways that preserve or ensure the integrity of the data.
- **Separation of duty among users:** Any person permitted to create or certify a well-formed transaction may not be permitted to execute it (at least against production data).

The model imposes integrity controls on data and the transactions that manipulate the data. The principal components of the model are as follows:

- **Constrained data items (CDIs):** Subject to strict integrity controls.
- **Unconstrained data items (UDIs):** Unchecked data items. An example is a simple text file.
- **Integrity verification procedures (IVPs):** Intended to assure that all CDIs conform to some application-specific model of integrity and consistency.
- **Transformation procedures (TPs):** System transactions that change the set of CDIs from one consistent state to another.

The CWM enforces integrity by means of certification and enforcement rules on TPs. **Certification rules** are security policy restrictions on the behavior of IVPs and TPs. **Enforcement rules** are built-in system security mechanisms that achieve the objectives of the certification rules. The rules are as follows:

C1: All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

C2: All TPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate, the security officer must specify a relation, which defines that execution. A relation is thus of the form (TPi, (CDIa, CDIb, CDIc . . .)), where the list of CDIs defines a particular set of arguments for which the TP has been certified.

E1: The system must maintain the list of relations specified in rule C2 and must ensure that the only manipulation of any CDI is by a TP, where the TP is operating on the CDI as specified in some relation.

E2: The system must maintain a list of relations of the form (UserID, TPi, (CDIa, CDIb, CDIc, . . .)), which relates a user, a TP, and the data objects that TP may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

C3: The list of relations in E2 must be certified to meet the separation of duty requirement.

E3: The system must authenticate the identity of each user attempting to execute a TP.

C4: All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

C5: Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected. Typically, this is an edit program.

E4: Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, the list of TPs associated with a CDI and the list of users associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

Figure 13.5 illustrates the rules. The rules combine to form a two-part integrity assurance facility, in which certification is done by a security officer with respect to an integrity policy, and enforcement is done by the system.

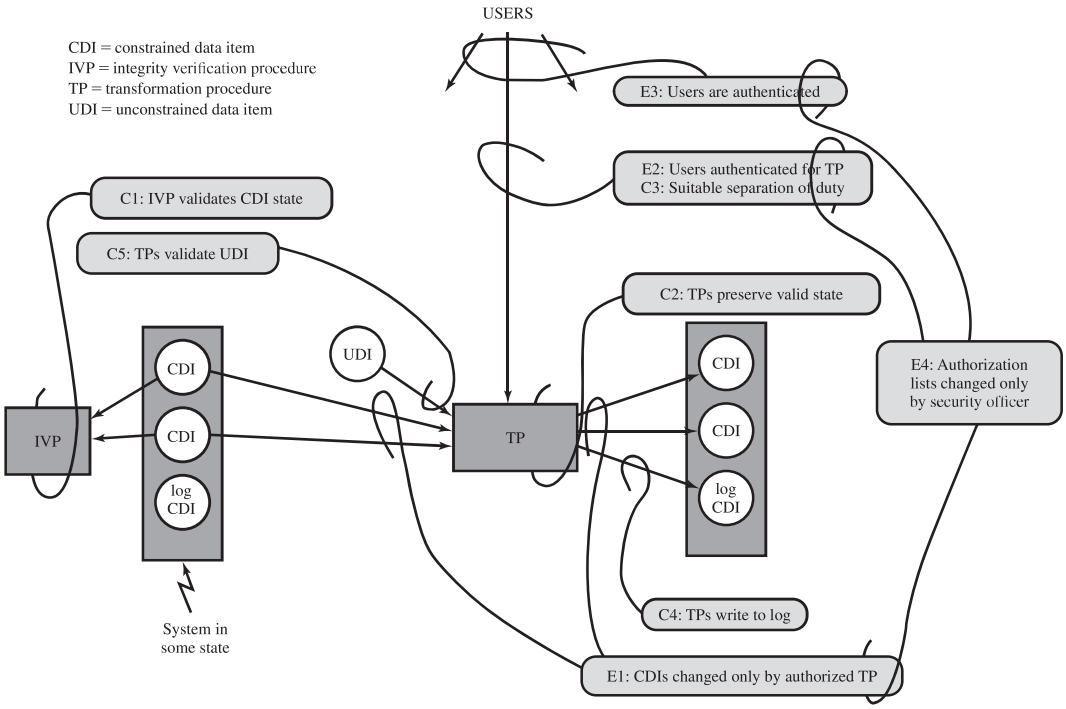


Figure 13.5 Summary of Clark-Wilson System Integrity Rules
Source: [CLAR87].

Chinese Wall Model

The Chinese Wall Model (CWM) takes a quite different approach to specifying integrity and confidentiality than any of the approaches we have examined so far. The model was developed for commercial applications in which conflicts of interest can arise. The model makes use of both discretionary and mandatory access concepts.

The principal idea behind the CWM is a concept that is common in the financial and legal professions, which is to use a what is referred to as a Chinese wall to prevent a conflict of interest. An example from the financial world is that of a market analyst working for a financial institution providing corporate business services. An analyst cannot be allowed to provide advice to one company when the analyst has confidential information (insider knowledge) about the plans or status of a competitor. However, the analyst is free to advise multiple corporations that are not in competition with each other and to draw on market information that is open to the public.

The elements of the model are the following:

- **Subjects:** Active entities that may wish to access protected objects; includes users and processes.
- **Information:** Corporate information organized into a hierarchy with three levels:
  - **Objects**: Individual items of information, each concerning a single corporation.
  - **Dataset (DS):** All objects that concern the same corporation.
  - **Conflict of interest (CI) class:** All datasets whose corporations are in competition.
- **Access rules:** Rules for read and write access.

Figure 13.6a gives an example. There are datasets representing banks, oil companies, and gas companies. All bank datasets are in one CI, all oil company datasets in another CI, and so forth.

In contrast to the models we have studies so far, the CWM does not assign security levels to subjects and objects and is thus not a true multilevel secure model. Instead, the history of a subject's previous access determines access control. The basis of the Chinese wall policy is that subjects are only allowed access to information that is not held to conflict with any other information that they already possess. Once a subject accesses information from one dataset, a wall is set up to protect information in other datasets in the same CI. The subject can access information on one side of the wall but not the other side. Further, information in other CIs is initially not considered to be on one side or the other of the wall but out in the open. When additional accesses are made in other CIs by the same subject, the shape of the wall changes to maintain the desired protection. Further, each subject is controlled by his or her own wall—the walls for different subjects are different.

To enforce the Chinese wall policy, two rules are needed. To indicate the similarity with the two BLP rules, the authors gave them the same names. The first rule is the simple security rule:

Simple security rule: A subject S can read on object O only if

- O is in the same DS as an object already accessed by S, **OR**
- O belongs to a CI from which S has not yet accessed any information.

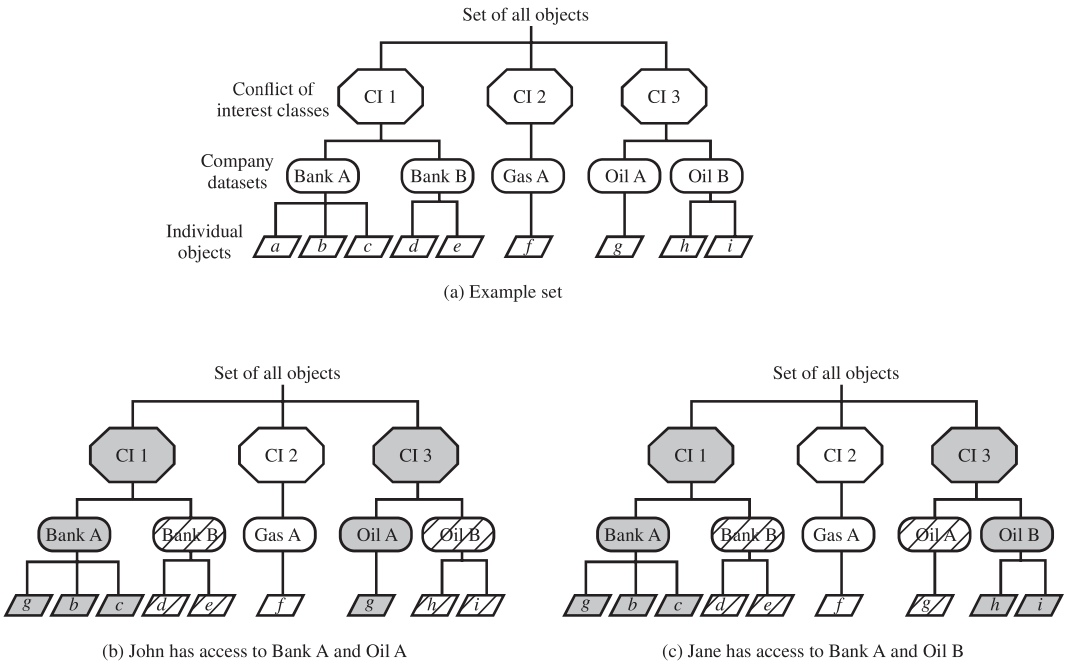


Figure 13.6 Potential Flow of Information between Two CIs

Figures 13.6b and c illustrate the operation of this rule. Assume that at some point, John has made his first read request to any object in this set for an object in the Bank A DS. Because John has not previously accessed an object in any other DS in CI 1, the access is granted. Further, the system must remember that access has been granted so that any subsequent request for access to an object in the Bank B DS will be denied. Any request for access to other objects in the Bank A DS is granted. At a later time, John requests access to an object in the Oil A DS. Because there is no conflict, this access is granted, but a wall is set up prohibiting subsequent access to the Oil B DS. Similarly, Figure 13.6c reflects the access history of Jane.

The simple security rule does not prevent an indirect flow of information that would cause a conflict of interest. In our example, John has access to Oil A DS and Bank A DS; Jane has access to Oil B DS and Bank A DS. If John is allowed to read from the Oil A DS and write into the Bank A DS, John may transfer information about Oil A into the Bank A DS; this is indicated by changing the value of the first object under the Bank A DS to *g*. The data can then subsequently be read by Jane. Thus, Jane would have access to information about both Oil A and Oil B, creating a conflict of interest. To prevent this, the CWM has a second rule:

***-property rule:** A subject S can write an object O only if

- S can read O according to the simple security rule, **AND**
- All objects that S can read are in the same DS as O.

Put another way, either subject cannot write at all, or a subject's access (both read and write) is limited to a single dataset. Thus, in Figure 13.6, neither John nor Jane has write access to any objects in the overall universe of data.

The *-property rule is quite restrictive. However, in many cases, a user only needs read access because the user is performing some analysis role.

To somewhat ease the write restriction, the model includes the concept of **sanitized data**. In essence, sanitized data are data that may be derived from corporate data but that cannot be used to discover the corporation's identity. Any DS consisting solely of sanitized data need not be protected by a wall; thus the two CWM rules do not apply to such DSs.

13.3 THE CONCEPT OF TRUSTED SYSTEMS

The models described in the preceding two sections are all aimed at enhancing the trust that users and administrators have in the security of a computer system. The concept of trust in the context of computer security goes back to the early 1970s, spurred on by the U.S. Department of Defense initiative and funding in this area. Early efforts were aimed to developing security models and then designing and implementing hardware/software platforms to achieve trust. Because of cost and performance issues, trusted systems did not gain a serious foothold in the commercial market. More recently, the interest in trust has reemerged, with the work on trusted computer platforms, a topic we explore in Section 13.5. In this section, we examine some basic concepts and implications of trusted systems.

Some useful terminology related to trusted systems is listed in Table 13.1.

Table 13.1 Terminology Related to Trust

Trust The extent to which someone who relies on a system can have confidence that the system meets its specifications (i.e., that the system does what it claims to do and does not perform unwanted functions).
Trusted system A system believed to enforce a given set of attributes to a stated degree of assurance.
Trustworthiness Assurance that a system deserves to be trusted, such that the trust can be guaranteed in some convincing way, such as through formal analysis or code review.
Trusted computer system A system that employs sufficient hardware and software assurance measures to allow its use for simultaneous processing of a range of sensitive or classified information.
Trusted computing base (TCB) A portion of a system that enforces a particular policy. The TCB must be resistant to tampering and circumvention. The TCB should be small enough to be analyzed systematically.
Assurance A process that ensures a system is developed and operated as intended by the system's security policy.
Evaluation Assessing whether the product has the security properties claimed for it.
Functionality The security features provided by a product.

Reference Monitors

Initial work on trusted computers and trusted operating systems was based on the **reference monitor** concept, depicted in Figure 13.7. The reference monitor is a controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object. The reference monitor has access to a file, known as the **security kernel database**, that lists the access privileges (security clearance) of each subject and the protection attributes (classification level) of each object. The reference monitor enforces the security rules (no read up, no write down) and has the following properties:

- **Complete mediation:** The security rules are enforced on every access, not just, for example, when a file is opened.
- **Isolation:** The reference monitor and database are protected from unauthorized modification.
- **Verifiability:** The reference monitor's correctness must be provable. That is, it must be possible to demonstrate mathematically that the reference monitor enforces the security rules and provides complete mediation and isolation.

These are stiff requirements. The requirement for complete mediation means that every access to data within main memory and on disk and tape must be mediated. Pure software implementations impose too high a performance penalty to be practical;

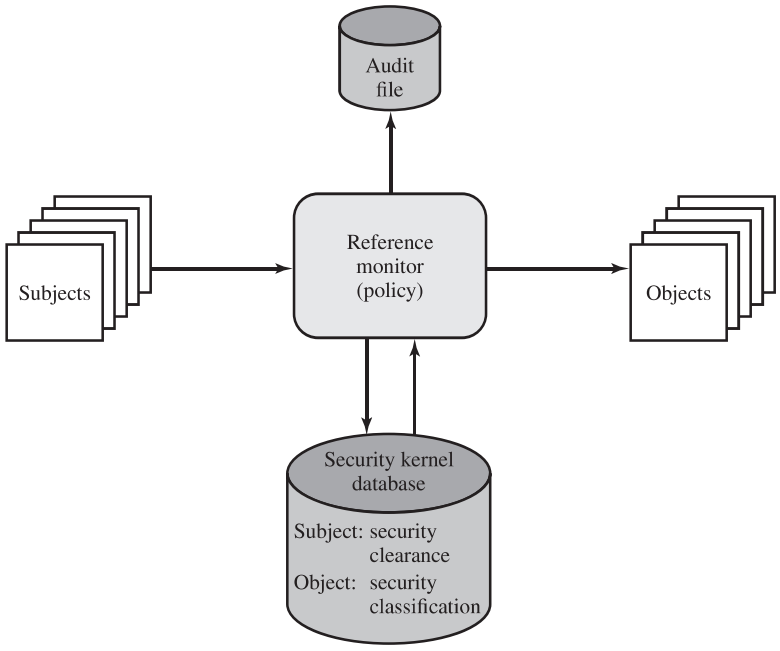


Figure 13.7 **Reference Monitor Concept**

the solution must be at least partly in hardware. The requirement for isolation means that it must not be possible for an attacker, no matter how clever, to change the logic of the reference monitor or the contents of the security kernel database. Finally, the requirement for mathematical proof is formidable for something as complex as a general-purpose computer. A system that can provide such verification is referred to as a **trustworthy system**.

A final element illustrated in Figure 13.7 is an audit file. Important security events, such as detected security violations and authorized changes to the security kernel database, are stored in the audit file.

In an effort to meet its own needs and as a service to the public, the U.S. Department of Defense in 1981 established the Computer Security Center within the National Security Agency (NSA) with the goal of encouraging the widespread availability of trusted computer systems. This goal is realized through the center's Commercial Product Evaluation Program. In essence, the center attempts to evaluate commercially available products as meeting the security requirements just outlined. The center classifies evaluated products according to the range of security features that they provide. These evaluations are needed for Department of Defense procurements but are published and freely available. Hence, they can serve as guidance to commercial customers for the purchase of commercially available, off-the-shelf equipment.

Trojan Horse Defense

One way to secure against Trojan horse attacks is the use of a secure, trusted operating system. Figure 13.8 illustrates an example. In this case, a Trojan horse is used to get around the standard security mechanism used by most file management and operating systems: the access control list. In this example, a user named Bob interacts through a program with a data file containing the critically sensitive character string "CPE170KS." User Bob has created the file with read/write permission provided only to programs executing on his own behalf: that is, only processes that are owned by Bob may access the file.

The Trojan horse attack begins when a hostile user, named Alice, gains legitimate access to the system and installs both a Trojan horse program and a private file to be used in the attack as a "back pocket." Alice gives read/write permission to herself for this file and gives Bob write-only permission (Figure 13.8a). Alice now induces Bob to invoke the Trojan horse program, perhaps by advertising it as a useful utility. When the program detects that it is being executed by Bob, it reads the sensitive character string from Bob's file and copies it into Alice's back-pocket file (Figure 13.8b). Both the read and write operations satisfy the constraints imposed by access control lists. Alice then has only to access Bob's file at a later time to learn the value of the string.

Now consider the use of a secure operating system in this scenario (Figure 13.8c). Security levels are assigned to subjects at logon on the basis of criteria such as the terminal from which the computer is being accessed and the user involved, as identified by password/ID. In this example, there are two security levels, sensitive and public, ordered so that sensitive is higher than public. Processes owned by Bob and Bob's data file are assigned the security level sensitive. Alice's file and processes are restricted to public. If Bob invokes the Trojan horse program (Figure 13.8d), that program acquires

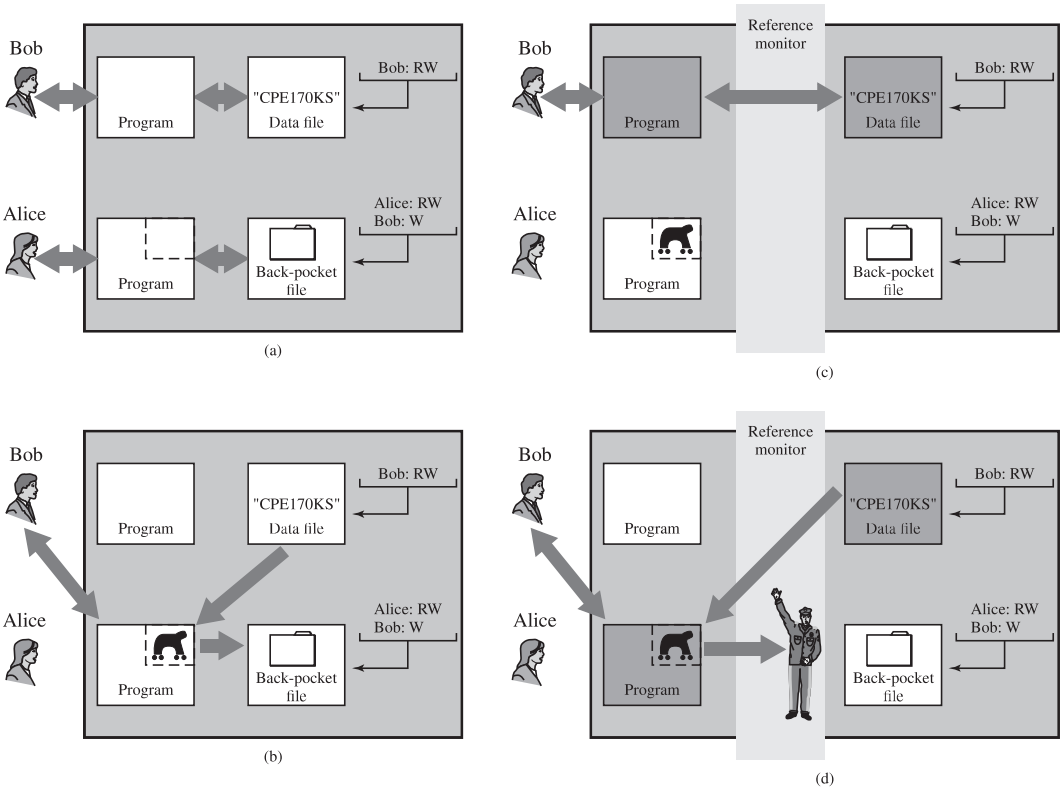


Figure 13.8 **Trojan Horse and Secure Operating System**

Bob's security level. It is therefore able, under the simple security property, to observe the sensitive character string. When the program attempts to store the string in a public file (the back-pocket file), however, the *-property is violated and the attempt is disallowed by the reference monitor. Thus, the attempt to write into the back-pocket file is denied even though the access control list permits it: The security policy takes precedence over the access control list mechanism.

13.4 APPLICATION OF MULTILEVEL SECURITY

RFC 4949 defines multilevel security as follows:

> **Multilevel Security (MLS)**: A mode of system operation wherein (a) two or more security levels of information are allowed to be to be handled concurrently within the same system when some users having access to the system have neither a security clearance nor need-to-know for some of the data handled by the system and (b) separation of the users and the classified material on the basis, respectively, of clearance and classification level are dependent on operating system control.

Multilevel security is of interest when there is a requirement to maintain a resource, such as a file system or database in which multiple levels of data sensitivity are defined. The hierarchy could be as simple as two levels (e.g., public and proprietary) or could have many levels (e.g., the military unclassified, restricted, confidential, secret, top secret). The preceding three sections have introduced us to the essential elements of multilevel security. In this section, we look at two applications areas where MLS concepts have been applied: role-based access control system and database security.

Multilevel Security for Role-Based Access Control[3]

[OSBO00] shows how a rule-based access control (RBAC) system can be used to implement the BLP multilevel security rules. Recall that the ANSI standard RBAC specification included the concept of administrative functions, which provide the capability to create, delete, and maintain RBAC elements and relations. It is useful here to assign special administrative roles to these functions. With this in mind, Table 13.2 summarizes the components of an RBAC.

The following formal specification indicates how a RBAC system can be used to implement MLS access:

- **Constraint on users:** For each user u in the set of users U, a security clearance $L(u)$ is assigned. Formally, $\forall u \in U [L(u) \text{ is given}]$.
- **Constraints on permissions:** Each permission assigns a read or write permission to an object o, and each object has one read and one write permission. All

Table 13.2 RBAC Elements

U, a set of users R and AR, disjoint sets of (regular) roles and administrative roles P and AP, disjoint sets of (regular) permissions and administrative permissions S, a set of sessions
$PA \subseteq P \times R$, a many-to-many permission to role assignment relation $APA \subseteq AP \times AR$, a many-to-many permission to administrative role assignment relation
$UA \subseteq U \times R$, a many-to-many user to role assignment relation $AUA \subseteq U \times AR$, a many-to-many user to administrative role assignment relation
$RH \subseteq R \times R$, a partially ordered role hierarchy $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy (both hierarchies are written as $\geq$ in infix notation)
User: $S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session's lifetime) *Roles:* $S \rightarrow 2^{RUAR}$ maps each session s_i to a set of roles and administrative roles *Roles:* $(S_i \subseteq \{ r \mid \exists r' \geq r) [(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time) sessions s_i has the permissions $\cup_{r \in roles(si)} \{p \mid (\exists r'' \leq r) \in PA \cup APA]\}$
There is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.

[3]The reader may wish to review Section 4.5 before proceeding.

objects have a security classification. Formally, $P = \{(o,r),(o,w)|\ o$ is an object in the system$\}$; $\forall o \in P$[L(o) is given].

- **Definitions:** The read-level of a role r, denoted r-level(r), is the least upper bound of the security levels of the objects for which (o, r) is in the permissions of r. The w-level of a role r (denoted w-level(r)) is the greatest lower bound (glb) of the security levels of the objects o for which (o, w) is in the permissions of r, if such a glb exists. If the glb does not exist, the w-level is undefined.
- **Constraints on UA:** Each role r has a defined write-level, denoted w-level(r). For each user assignment, the clearance of the user must dominate the r-level of the role and be dominated by the w-level of the role. Formally, $\forall r \in UA$ [w-level(r) is defined]; $\forall(u,r) \in UA$ [L(u) $\geq$ r-level(r)]; $\forall(u,r) \in UA$ [L(u) $\leq$ w-level(r)].

The preceding definitions and constraints enforce the BLP model. A role can include access permissions for multiple objects. The r-level of the role indicates the highest security classification for the objects assigned to the role. Thus, the simple security property (no read up) demands that a user can be assigned to a role only if the user's clearance is at least as high as the r-level of the role. Similarly, the w-level of the role indicates the lowest security classification of its objects. The *-security property (no write down) demands that a user be assigned to a role only if the user's clearance is no higher than the w-level of the role.

Database Security and Multilevel Security

The addition of multilevel security to a database system increases the complexity of the access control function and of the design of the database itself. One key issue is the granularity of classification. The following are possible methods of imposing multilevel security on a relational database, in terms of the granularity of classification (Figure 13.9):

- **Entire database:** This simple approach is easily accomplished on an MLS platform. An entire database, such as a financial or personnel database, could be classified as confidential or restricted and maintained on a server with other files.
- **Individual tables (relations):** For some applications, it is appropriate to assign classification at the table level. In the example of Figure 13.9a, two levels of classification are defined: unrestricted (U) and restricted (R). The Employee table contains sensitive salary information and is classified restricted, while the Department table is unrestricted. This level of granularity is relatively easy to implement and enforce.
- **Individual columns (attributes):** A security administrator may choose to determine classification on the basis of attributes, so that selected columns are classified. In the example of Figure 13.9b, the administrator determines that salary information and the identity of department managers is restricted information.
- **Individual rows (tuples):** In other circumstances, it may make sense to assign classification levels on the basis of individual rows that match certain properties.

Department Table - U		
Did	Name	Mgr
4	accts	Cathy
8	PR	James

Employee Table - R			
Name	Did	Salary	Eid
Andy	4	43K	2345
Calvin	4	35K	5088
Cathy	4	48K	7712
James	8	55K	9664
Ziggy	8	67K	3054

(a) Classified by table

Department Table		
Did - U	Name - U	Mgr - R
4	accts	Cathy
8	PR	James

Employee Table			
Name - U	Did - U	Salary - R	Eid - U
Andy	4	43K	2345
Calvin	4	35K	5088
Cathy	4	48K	7712
James	8	55K	9664
Ziggy	8	67K	3054

(b) Classified by column (attribute)

Department Table			
Did	Name	Mgr	
4	accts	Cathy	R
8	PR	James	U

Employee Table				
Name	Did	Salary	Eid	
Andy	4	43K	2345	U
Calvin	4	35K	5088	U
Cathy	4	48K	7712	U
James	8	55K	9664	R
Ziggy	8	67K	3054	R

(c) Classified by row (tuple)

Department Table		
Did	Name	Mgr
4 - U	accts - U	Cathy - R
8 - U	PR - U	James - R

Employee Table			
Name	Did	Salary	Eid
Andy - U	4 - U	43K - U	2345 - U
Calvin - U	4 - U	35K - U	5088 - U
Cathy - U	4 - U	48K - U	7712 - U
James - U	8 - U	55K - R	9664 - U
Ziggy - U	8 - U	67K - R	3054 - U

(d) Classified by element

Figure 13.9 Approaches to Database Classification

In the example of Figure 13.9c, all rows in the Department table that contain information relating to the Accounts Department (Dept. ID = 4), and all rows in the Employee table for which the Salary is greater than 50K are restricted.

- **Individual elements:** The most difficult scheme to implement and manage is one in which individual elements may be selectively classified. In the example of Figure 13.9d, salary information and the identity of the manager of the Accounts Department are restricted.

The granularity of the classification scheme affects the way in which access control is enforced. In particular, efforts to prevent inference depend on the granularity of the classification.

READ ACCESS For read access, a database system needs to enforce the simple security rule (no read up). This is straightforward if the classification granularity is the entire database or at the table level. Consider now a database classified by column (attribute). For example, in Figure 13.9b, suppose that a user with only unrestricted clearance issues the following SQL query:

```
SELECT Ename
     FROM Employee
     WHERE Salary > 50K
```

This query returns only unrestricted data but reveals restricted information, namely whether any employees have a salary greater than 50K and, if so, which employees. This type of security violation can be addressed by considering not only the data returned to the user but also any data that must be accessed to satisfy the query. In this case, the query requires access to the Salary attribute, which is unauthorized for this user; therefore, the query is rejected.

If classification is by row (tuple) rather than column, then the preceding query does not pose an inference problem. Figure 13.9c shows that in the Employee table, all rows corresponding to salaries greater than 50K are restricted. Because all such records will be removed from the response to the preceding query, the inference just discussed cannot occur. However, some information may be inferred, because a null response indicates either that salaries above 50 are restricted, or no employee has a salary greater than 50K.

The use of classification by rows instead of columns creates other inference problems. For example, suppose we add a new Projects table to the database of Figure13.9c consisting of attributes Eid, ProjectID, and ProjectName, where the Eid field in the Employee and Projects tables can be joined. Suppose that all records in the Projects table are unrestricted except for projects with ProjectID 500 through 599. Consider the following request:

```
SELECT Ename
     WHERE Employee.Eid = Projects.Eid
     AND Projects.ProjectID = 500
```

This request, if granted, returns information from the Employee table, which is unrestricted, although it reveals restricted information, namely that the selected employees are assigned to project 500. As before, the database system must consider not just the data returned to the user but any data that must be accessed to satisfy the query.

Classification by element does not introduce any new considerations. The system must prevent not only a read up but also a query that must access higher-level elements in order to satisfy the query.

As a general comment, we can say that dealing with read access is far simpler if the classification granularity is database or table. If the entire database has a single classification, then no new inference issues are raised. The same is true of classification by table. If some finer-grained classification seems desirable, it might be possible to achieve the same effect by splitting tables.

Write Access For write access, a database system needs to enforce the *-security rule (no write down). But this is not as simple as it may seem. Consider the following situation. Suppose the classification granularity is finer than the table level (i.e., by column, by row, or by element) and that a user with a low clearance (unrestricted) requests the insertion of a row with the same primary key as an existing row where the row or one of its elements is at a higher level. The DBMS has essentially three choices:

1. Notify the user that a row with the same primary key already exists and reject the insertions. This is undesirable because it informs the user of the existence of a higher-level row with the specified primary key value.
2. Replace the existing row with the new row classified at the lower level. This is undesirable because it would allow the user to overwrite data not visible to the user, thus compromising data integrity.
3. Insert the new row at the lower level without modifying the existing row at the higher level. This is known as **polyinstantiation**. This avoids the inference and data integrity problems but creates a database with conflicting entries.

The same alternatives apply when a user attempts to update a row rather than insert a row. To illustrate the effect of polyinstantiation, consider the following query applied to Figure 13.9c by a user with a low clearance (U).

```
INSERT INTO Employee
     VALUES (James,8,35K,9664,U)
```

The table already contains a row for James with a higher salary level, which necessitates classifying the row as restricted. This new tuple would have an unrestricted classification. The same effect would be produced by an update:

```
UPDATE Employee
     SET Salary=35K
     WHERE Eid=9664
```

The result is unsettling (Figure 13.10). Clearly, James can only have one salary and therefore one of the two rows is false. The motivation for this is to prevent

Employee				
Name	Did	Salary	Eid	
Andy	4	43K	2345	U
Calvin	4	35K	5088	U
Cathy	4	48K	7712	U
James	8	55K	9664	R
James	8	35K	9664	U
Ziggy	8	67K	3054	R

Figure 13.10 Example of Polyinstantiation

inference. If a unrestricted user queries the salary of James in the original database, the user's request is rejected and the user may infer that salary is greater than 50K. The inclusion of the "false" row provides a form of cover for the true salary of James. Although the approach may appear unsatisfactory, there have been a number of designs and implementations of polyinstantiation [BERT95].

The problem can be avoided by using a classification granularity of database or table, and in many applications, such granularity is all that is needed.

13.5 TRUSTED COMPUTING AND THE TRUSTED PLATFORM MODULE

The trusted platform module (TPM) is a concept being standardized by an industry consortium, the Trusted Computing Group. The TPM is a hardware module that is at the heart of a hardware/software approach to trusted computing. Indeed, the term **trusted computing** (TC) is now used in the industry to refer to this type of hardware/software approach.

The TC approach employs a TPM chip in personal computer motherboard or a smart card or integrated into the main processor, together with hardware and software that in some sense has been approved or certified to work with the TPM. We can briefly describe the TC approach as follows. The TPM generates keys that it shares with vulnerable components that pass data around the system, such as storage devices, memory components, and audio/visual hardware. The keys can be used to encrypt the data that flow throughout the machine. The TPM also works with TC-enabled software, including the OS and applications. The software can be assured that the data it receives are trustworthy, and the system can be assured that the software itself is trustworthy.

To achieve these features, TC provides three basic services: authenticated boot, certification, and encryption.

Authenticated Boot Service

The authenticated boot service is responsible for booting the entire operating system in stages and assuring that each portion of the OS, as it is loaded, is a version that is approved for use. Typically, an OS boot begins with a small piece

of code in the Boot ROM. This piece brings in more code from the Boot Block on the hard drive and transfers execution to that code. This process continues with more and larger blocks of the OS code being brought in until the entire OS boot procedure is complete and the resident OS is booted. At each stage, the TC hardware checks that valid software has been brought in. This may be done by verifying a digital signature associated with the software. The TPM keeps a tamper-evident log of the loading process, using a cryptographic hash function to detect any tampering with the log.

When the process is completed, the tamper-resistant log contains a record that establishes exactly which version of the OS and its various modules are running. It is now possible to expand the trust boundary to include additional hardware and application and utility software. The TC-enabled system maintains an approved list of hardware and software components. To configure a piece of hardware or load a piece of software, the system checks whether the component is on the approved list, whether it is digitally signed (where applicable), and that its serial number has not been revoked. The result is a configuration of hardware, system software, and applications that is in a well-defined state with approved components.

Certification Service

Once a configuration is achieved and logged by the TPM, the TPM can certify the configuration to other parties. The TPM can produce a digital certificate by signing a formatted description of the configuration information using the TPM's private key. Thus, another user, either a local user or a remote system, can have confidence that an unaltered configuration is in use because

1. The TPM is considered trustworthy. We do not need a further certification of the TPM itself.
2. Only the TPM possesses this TPM's private key. A recipient of the configuration can use the TPM's public key to verify the signature (Figure 2.7b).

To assure that the configuration is timely, a requester issues a "challenge" in the form of a random number when requesting a signed certificate from the TPM. The TPM signs a block of data consisting of the configuration information with the random number appended to it. The requester therefore can verify that the certificate is both valid and up to date.

The TC scheme provides for a hierarchical approach to certification. The TPM certifies the hardware/OS configuration. Then the OS can certify the presence and configuration of application programs. If a user trusts the TPM and trusts the certified version of the OS, then the user can have confidence in the application's configuration.

Encryption Service

The encryption service enables the encryption of data in such a way that the data can be decrypted only by a certain machine and only if that machine is in a certain configuration. There are several aspects of this service.

First, the TPM maintains a master secret key unique to this machine. From this key, the TPM generates a secret encryption key for every possible configuration

of that machine. If data are encrypted while the machine is in one configuration, the data can only be decrypted using that same configuration. If a different configuration is created on the machine, the new configuration will not be able to decrypt the data encrypted by a different configuration.

This scheme can be extended upward, as is done with certification. Thus, it is possible to provide an encryption key to an application so that the application can encrypt data, and decryption can only be done by the desired version of the desired application running on the desired version of the desired OS. These encrypted data can be stored locally, only retrievable by the application that stored them, or transmitted to a peer application on a remote machine. The peer application would have to be in the identical configuration to decrypt the data.

TPM Functions

Figure 13.11, based on the most recent TPM specification, is a block diagram of the functional components of the TPM. These are as follows:

- **I/O:** All commands enter and exit through the I/O component, which provides communication with the other TPM components.
- **Cryptographic co-processor:** Includes a processor that is specialized for encryption and related processing. The specific cryptographic algorithms implemented by this component include RSA encryption/decryption, RSA-based digital signatures, and symmetric encryption.

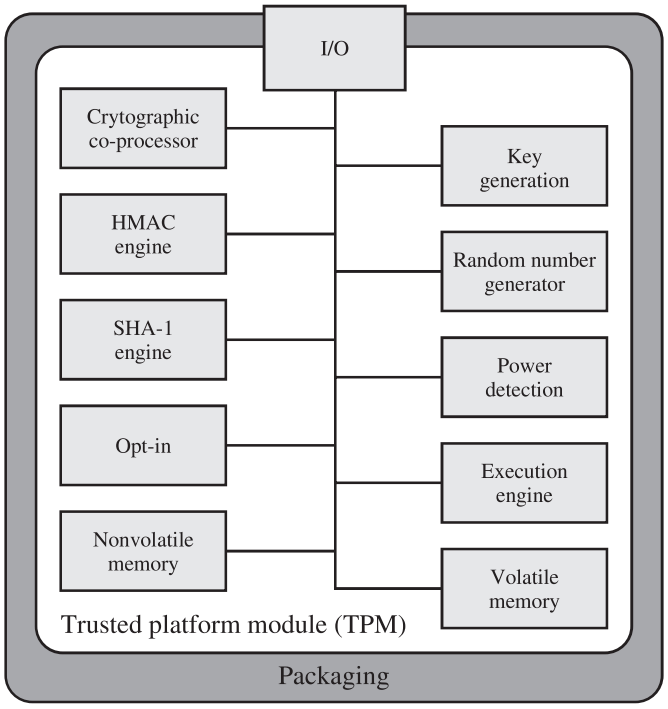


Figure 13.11 TPM Component Architecture

- **Key generation:** Creates RSA public/private key pairs and symmetric keys.
- **HMAC engine:** This algorithm is used in various authentication protocols.
- **Random number generator (RNG):** This component produces random numbers used in a variety of cryptographic algorithms, including key generation, random values in digital signatures, and nonces. A nonce is a random number used once, as in a challenge protocol. The RNG uses a hardware source of randomness (manufacturer specific) and does not rely on a software algorithm that produces pseudo random numbers.
- **SHA-1 engine:** This component implements the SHA algorithm, which is used in digital signatures and the HMAC algorithm.
- **Power detection:** Manages the TPM power states in conjunction with the platform power states.
- **Opt-in:** Provides secure mechanisms to allow the TPM to be enabled or disabled at the customer/user's discretion.
- **Execution engine:** Runs program code to execute the TPM commands received from the I/O port.
- **Nonvolatile memory:** Used to store persistent identity and state parameters for this TPM.
- **Volatile memory:** Temporary storage for execution functions, plus storage of volatile parameters, such as current TPM state, cryptographic keys, and session information.

Protected Storage

To give some feeling for the operation of a TC/TPM system, we look at the protected storage function. The TPM generates and stores a number of encryption keys in a trust hierarchy. At the root of the hierarchy is a storage root key generated by the TPM and accessible only for the TPM's use. From this key other keys can be generated and protected by encryption with keys closer to the root of the hierarchy.

An important feature of Trusted Platforms is that a TPM protected object can be "sealed" to a particular software state in a platform. When the TPM protected object is created, the creator indicates the software state that must exist if the secret is to be revealed. When a TPM unwraps the TPM protected object (within the TPM and hidden from view), the TPM checks that the current software state matches the indicated software state. If they match, the TPM permits access to the secret. If they do not match, the TPM denies access to the secret.

Figure 13.12 provides an example of this protection. In this case, there is an encrypted file on local storage that a user application wishes to access. The following steps occur:

1. The symmetric key that was used to encrypt the file is stored with the file. The key itself is encrypted with another key to which the TPM has access. The protected key is submitted to the TPM with a request to reveal the key to the application.

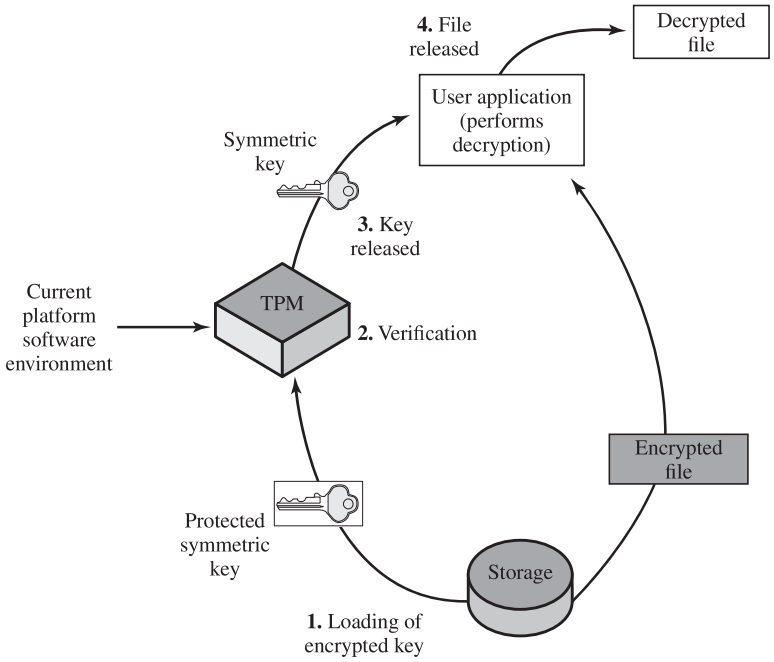


Figure 13.12 **Decrypting a File Using a Protected Key**

2. Associated with the protected key is a specification of the hardware/software configuration that may have access to the key. The TPM verifies that the current configuration matches the configuration required for revealing the key. In addition, the requesting application must be specifically authorized to access the key. The TPM uses an authorization protocol to verify authorization.
3. If the current configuration is permitted access to the protected key, then the TPM decrypts the key and passes it on to the application.
4. The application uses the key to decrypt the file. The application is trusted to then securely discard the key.

The encryption of a file proceeds in an analogous matter. In this latter case, a process requests a symmetric key to encrypt the file. The TPM then provides an encrypted version of the key to be stored with the file.

13.6 COMMON CRITERIA FOR INFORMATION TECHNOLOGY SECURITY EVALUATION

The work done by the National Security Agency and other U.S. government agencies to develop requirements and evaluation criteria for trusted systems resulted in the publication of the *Trusted Computer System Evaluation Criteria* (TCSEC),

informally known as the *Orange Book*, in the early 1980s. This focused primarily on protecting information confidentiality. Subsequently, other countries started work to develop criteria based on the TCSEC but that were more flexible and adaptable to the evolving nature of IT. The process of merging, extending, and consolidating these various efforts eventually resulted in the development of the Common Criteria in the late 1990s. The *Common Criteria (CC) for Information Technology and Security Evaluation* are ISO standards for specifying security requirements and defining evaluation criteria. The aim of these standards is to provide greater confidence in the security of IT products as a result of formal actions taken during the process of developing, evaluating, and operating these products. In the development stage, the CC defines sets of IT requirements of known validity that can be used to establish the security requirements of prospective products and systems. Then the CC details how a specific product can be evaluated against these known requirements, to provide confirmation that it does indeed meet them, with an appropriate level of confidence. Lastly, when in operation the evolving IT environment may reveal new vulnerabilities or concerns. The CC details a process for responding to such changes, and possibly reevaluating the product. Following successful evaluation, a particular product may be listed as CC certified or validated by the appropriate national agency, such as NIST/NSA in the United States. That agency publishes lists of evaluated products, which are used by government and industry purchasers who need to use such products.

Requirements

The CC defines a common set of potential security requirements for use in evaluation. The term **target of evaluation** (TOE) refers to that part of the product or system that is subject to evaluation. The requirements fall into two categories:

- **Functional requirements:** Define desired security behavior. CC documents establish a set of security functional components that provide a standard way of expressing the security functional requirements for a TOE.
- **Assurance requirements:** The basis for gaining confidence that the claimed security measures are effective and implemented correctly. CC documents establish a set of assurance components that provide a standard way of expressing the assurance requirements for a TOE.

Both functional requirements and assurance requirements are organized into classes: A **class** is a collection of requirements that share a common focus or intent. Tables 13.3 and 13.4 briefly define the classes for functional and assurance requirements. Each of these classes contains a number of families. The requirements within each **family** share security objectives but differ in emphasis or rigor. For example, the audit class contains six families dealing with various aspects of auditing (e.g., audit data generation, audit analysis and audit event storage). Each family, in turn, contains one or more components. A **component** describes a specific set of security requirements and is the smallest selectable set of security requirements for inclusion in the structures defined in the CC.

Table 13.3 CC Security Functional Requirements

Class	Description
Audit	Involves recognizing, recording, storing, and analyzing information related to security activities. Audit records are produced by these activities and can be examined to determine their security relevance.
Cryptographic support	Used when the TOE implements cryptographic functions. These may be used, for example, to support communications, identification and authentication, or data separation.
Communications	Provides two families concerned with nonrepudiation by the originator and by the recipient of data.
User data protection	Specifies requirements relating to the protection of user data within the TOE during import, export, and storage, in addition to security attributes related to user data.
Identification and authentication	Ensure the unambiguous identification of authorized users and the correct association of security attributes with users and subjects.
Security management	Specifies the management of security attributes, data and functions.
Privacy	Provides a user with protection against discovery and misuse of his or her identity by other users.
Protection of the TOE security functions	Focused on protection of TSF (TOE security functions) data rather than of user data. The class relates to the integrity and management of the TSF mechanisms and data.
Resource utilization	Supports the availability of required resources, such as processing capability and storage capacity. Includes requirements for fault tolerance, priority of service, and resource allocation.
TOE access	Specifies functional requirements, in addition to those specified for identification and authentication, for controlling the establishment of a user's session. The requirements for TOE access govern such things as limiting the number and scope of user sessions, displaying the access history, and modifying access parameters.
Trusted path/ channels	Concerned with trusted communications paths between the users and the TSF and between TSFs.

For example, the cryptographic support class of functional requirements includes two families: cryptographic key management and cryptographic operation. There are four components under the cryptographic key management family, which are used to specify key generation algorithm and key size; key distribution method; key access method; and key destruction method. For each component, a standard may be referenced to define the requirement. Under the cryptographic operation family, there is a single component, which specifies an algorithm and key size based on an assigned standard.

Sets of functional and assurance components may be grouped together into reusable packages, which are known to be useful in meeting identified objectives. An example of such a package would be functional components required for Discretionary Access Controls.

Table 13.4 CC Security Assurance Requirements

Class	Description
Configuration management	Requires that the integrity of the TOE is adequately preserved. Specifically, configuration management provides confidence that the TOE and documentation used for evaluation are the ones prepared for distribution.
Delivery and operation	Concerned with the measures, procedures, and standards for secure delivery, installation, and operational use of the TOE, to ensure that the security protection offered by the TOE is not compromised during these events.
Development	Concerned with the refinement of the TSF from the specification defined in the ST to the implementation, and a mapping from the security requirements to the lowest level representation.
Guidance documents	Concerned with the secure operational use of the TOE, by the users and administrators.
Life cycle support	Concerned with the life cycle of the TOE include life cycle definition, tools and techniques, security of the development environment, and remediation of flaws found by TOE consumers.
Tests	Concerned with demonstrating that the TOE meets its functional requirements. The families address coverage and depth of developer testing, and requirements for independent testing.
Vulnerability assessment	Defines requirements directed at the identification of exploitable vulnerabilities, which could be introduced by construction, operation, misuse, or incorrect configuration of the TOE. The families identified here are concerned with identifying vulnerabilities through covert channel analysis, analyzing the configuration of the TOE, examining the strength of mechanisms of the security functions, and identifying flaws introduced during development of the TOE. The second family covers the security categorization of TOE components. The third and fourth cover the analysis of changes for security impact and the provision of evidence that procedures are being followed. This class provides building blocks for the establishment of assurance maintenance schemes.
Assurance maintenance	Provides requirements that are intended to be applied after a TOE has been certified against the CC. These requirements are aimed at assuring that the TOE will continue to meet its security target as changes are made to the TOE or its environment.

Profiles and Targets

The CC also defines two kinds of documents that can be generated using the CC-defined requirements.

- **Protection profiles (PPs):** Define an implementation-independent set of security requirements and objectives for a category of products or systems that meet similar consumer needs for IT security. A PP is intended to be reusable and to define requirements that are known to be useful and effective in meeting the identified objectives. The PP concept has been developed to support the definition of functional standards and as an aid to formulating procurement specifications. The PP reflects user security requirements.

- **Security targets (STs):** Contain the IT security objectives and requirements of a specific identified TOE and defines the functional and assurance measures offered by that TOE to meet stated requirements. The ST may claim conformance to one or more PPs and forms the basis for an evaluation. The ST is supplied by a vendor or developer.

Figure 13.13 illustrates the relationship between requirements on the one hand and profiles and targets on the other. For a PP, a user can select a number of components to define the requirements for the desired product. The user may also refer to predefined packages that assemble a number of requirements commonly grouped together within a product requirements document. Similarly, a vendor or designer can select a number of components and packages to define an ST.

Figure 13.14 shows what is referred to in the CC documents as the security functional requirements paradigm. In essence, this illustration is based on the reference monitor concept but makes use of the terminology and design philosophy of the CC.

Example of a Protection Profile

The protection profile for a smart card, developed by the Smart Card Security User Group, provides a simple example of a PP. This PP describes the IT security requirements for a smart card to be used in connection with sensitive applications, such as banking industry financial payment systems. The assurance level for this PP is EAL 4, which is described in the following subsection. The PP lists **threats** that must be addressed by a product that claims to comply with this PP. The threats include the following:

- **Physical probing:** May entail reading data from the TOE through techniques commonly employed in IC failure analysis and IC reverse engineering efforts.
- **Invalid input:** Invalid input may take the form of operations that are not formatted correctly, requests for information beyond register limits, or attempts to find and execute undocumented commands. The result of such an attack

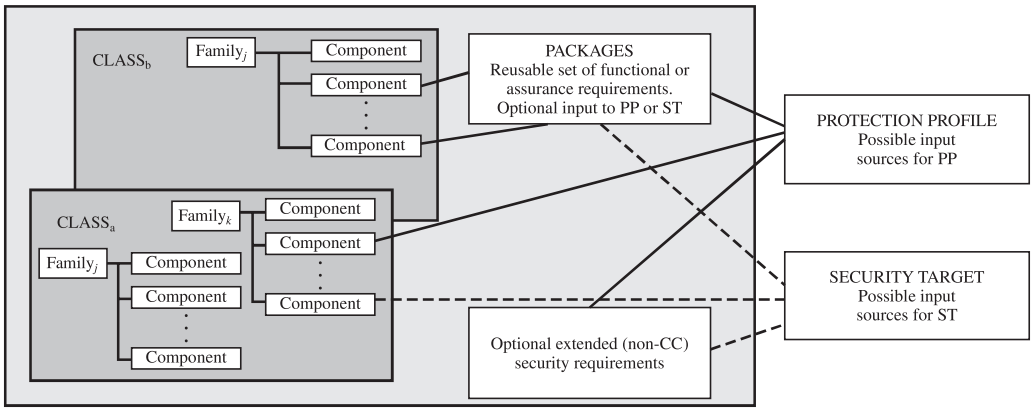


Figure 13.13 Organization and Construction of Common Criteria Requirements

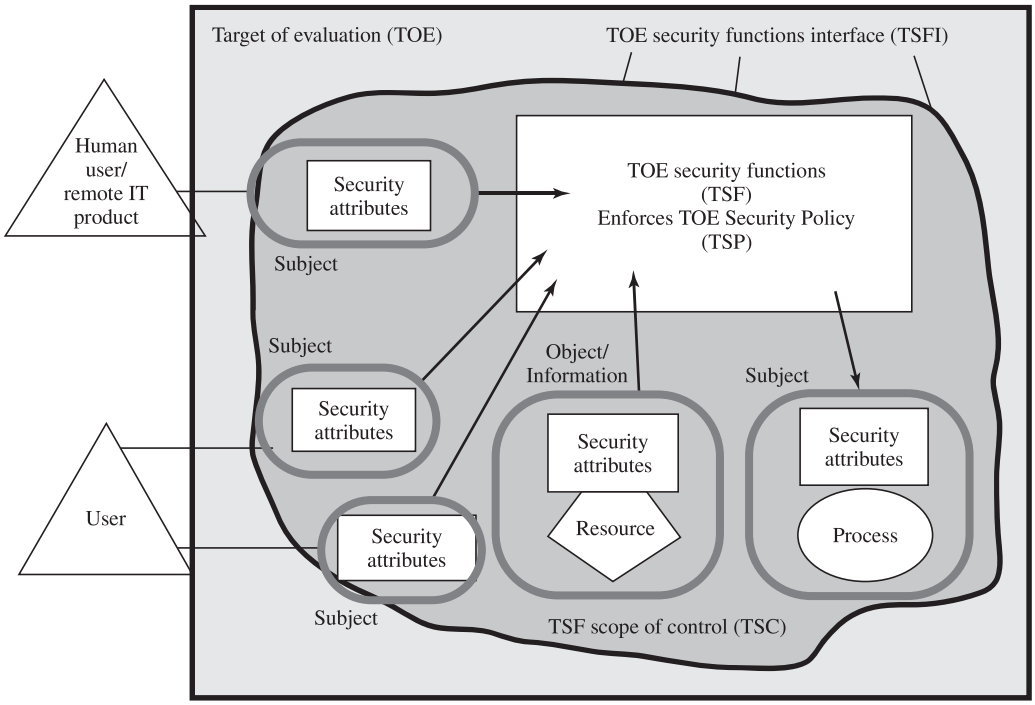


Figure 13.14 **Security Functional Requirements Paradigm**

may be a compromise in the security functions, generation of exploitable errors in operation, or release of protected data.

- **Linkage of multiple operations:** An attacker may observe multiple uses of resources or services and, by linking these observations, deduce information that that may reveal security function data.

Following a list of threats, the PP turns to a description of **security objectives**. These reflect the stated intent to counter identified threats and/or comply with any organizational security policies identified. Nineteen objectives are listed, including the following:

- **Audit:** The system must provide the means of recording selected security-relevant events, so as to assist an administrator in the detection of potential attacks or misconfiguration of the system security features that would leave it susceptible to attack.
- **Fault insertion:** The system must be resistant to repeated probing through insertion of erroneous data.
- **Information leakage:** The system must provide the means of controlling and limiting the leakage of information in the system so that no useful information is revealed over the power, ground, clock, reset, or I/O lines.

Security requirements are provided to thwart specific threats and to support specific policies under specific assumptions. The PP lists specific requirements in three general areas: TOE security functional requirements, TOE security assurance requirements, and security requirements for the IT environment. In the area of **security functional requirements**, the PP defines 42 requirements from the available classes of security functional requirements (Table 13.3). For example, for security auditing, the PP stipulates what the system must audit; what information must be logged; what the rules are for monitoring, operating and protecting the logs; and so on. Functional requirements are also listed from the other functional requirements classes, with specific details for the smart card operation.

The PP defines 24 **security assurance requirements** from the available classes of security assurance requirements (Table 13.4). These requirements were chosen to demonstrate:

- The quality of the product design and configuration
- That adequate protection is provided during the design and implementation of the product
- That vendor testing of the product meets specific parameters
- That security functionality is not compromised during product delivery
- That user guidance, including product manuals pertaining to installation, maintenance and use, are of a specified quality and appropriateness

The PP also lists **security requirements of the IT environment**. These cover the following topics:

- Cryptographic key distribution
- Cryptographic key destruction
- Security roles

The final section of the PP (excluding appendices) is a lengthy rationale for all of the selections and definitions in the PP. The PP is an industry-wide effort designed to be realistic in its ability to be met by a variety of products with a variety of internal mechanisms and implementation approaches.

13.7 ASSURANCE AND EVALUATION

The NIST *Computer Security Handbook* [NIST95] characterizes assurance in the following way: "Security assurance is the degree of confidence one has that the security controls operate correctly and protect the system as intended. Assurance is not, however, an absolute guarantee that the measures work as intended." As with any other aspect of computer security, resources devoted to assurance must be subjected to some sort of cost-benefit analysis to determine what amount of effort is reasonable for the level of assurance desired.

Target Audience

The design of assurance measures depends in part on the target audience for these measures. That is, in developing a degree of confidence in security measures, we need to specify what individuals or groups possess that degree of confidence. The CC document on assurance [CCPS12c] lists the following target audiences:

- **Consumers**: Select security features and functions for a system and determine the required levels of security assurance.
- **Developers**: Respond to actual or perceived consumer security requirements; interpret statements of assurance requirements; and determine assurance approaches and level of effort.
- **Evaluators**: Use the assurance requirements as a mandatory statement of evaluation criteria when evaluating security features and controls.

Evaluators may be in the same organization as consumers or a third-party evaluation team.

Scope of Assurance

Assurance deals with security features of IT products, such as computers, database management systems, operating systems, and complete systems. Assurance applies to the following aspects of a system:

- **Requirements**: This category refers to the security requirements for a product
- **Security policy**: Based on the requirements, a security policy can be defined
- **Product design**: Based on requirements and security policy
- **Product implementation**: Based on design
- **System operation**: Includes ordinary use plus maintenance

In each area, various approaches can be taken to provide assurance. [CCPS12c] lists the following possible approaches:

- Analysis and checking of process(es) and procedure(s)
- Checking that process(es) and procedure(s) are being applied
- Analysis of the correspondence between TOE design representations
- Analysis of the TOE design representation against the requirements
- Verification of proofs
- Analysis of guidance documents
- Analysis of functional tests developed and the results provided
- Independent functional testing
- Analysis for vulnerabilities (including flaw hypothesis)
- Penetration testing

A somewhat different take on the elements of assurance is provided in [CHOK92]. This report is based on experience with *Orange Book* evaluations but

is relevant to current trusted product development efforts. The author views assurance as encompassing the following requirements:

- **System architecture:** Addresses both the system development phase and the system operations phase. Examples of techniques for increasing the level of assurance during the development phase include modular software design, layering, and data abstraction/information hiding. An example of the operations phase is isolation of the trusted portion of the system from user processes.
- **System integrity:** Addresses the correct operation of the system hardware and firmware and is typically satisfied by periodic use of diagnostic software.
- **System testing:** Ensures that the security features have been tested thoroughly. This includes testing of functional operations, testing of security requirements, and testing of possible penetrations.
- **Design specification and verification:** Addresses the correctness of the system design and implementation with respect to the system security policy. Ideally, formal methods of verification can be used.
- **Covert channel analysis:** This type of analysis attempts to identify any potential means for bypassing security policy and ways to reduce or eliminate such possibilities.
- **Trusted facility management:** Deals with system administration. One approach is to separate the roles of system operator and security administrator. Another approach is detailed specification of policies and procedures with mechanisms for review.
- **Trusted recovery:** Provides for correct operation of security features after a system recovers from failures, crashes, or security incidents.
- **Trusted distribution:** Ensures that protected hardware, firmware, and software do not go through unauthorized modification during transit from the vendor to the customer.
- **Configuration management:** Requirements are included for configuration control, audit, management, and accounting.

Thus we see that assurance deals with the design, implementation, and operation of protected resources and their security functions and procedures. It is important to note that assurance is a process, not an attainment. That is, assurance must be an ongoing activity, including testing, auditing, and review.

Common Criteria Evaluation Assurance Levels

The concept of evaluation assurance is a difficult one to pin down. Further, the degree of assurance required varies from one context and one functionality to another. To structure the need for assurance, the CC defines a scale for rating assurance consisting of seven evaluation assurance levels (EALs) ranging from the least rigor and scope for assurance evidence (EAL 1) to the most (EAL 7). The levels are as follows:

- **EAL 1**: **functionally tested:** For environments where security threats are not considered serious. It involves independent product testing with no input

from the product developers. The intent is to provide a level of confidence in correct operation.

- **EAL 2**: **structurally tested:** Includes a review of a high-level design provided by the product developer. Also, the developer must conduct a vulnerability analysis for well-known flaws. The intent is to provide a low to moderate level of independently assured security.
- **EAL 3**: **methodically tested and checked:** Requires a focus on the security features. This includes requirements that the design separate security-related components from those that are not; that the design specifies how security is enforced; and that testing be based both on the interface and the high-level design, rather than a black-box testing based only on the interface. It is applicable where the requirement is for a moderate level of independently assured security, with a thorough investigation of the TOE and its development without incurring substantial reengineering costs.
- **EAL 4**: **methodically designed, tested, and reviewed:** Requires a low-level as well as a high-level design specification. Requires that the interface specification be complete. Requires an abstract model that explicitly defines security for the product. Requires an independent vulnerability analysis. It is applicable in those circumstances where developers or users require a moderate to high level of independently assured security in conventional commodity TOEs, and there is willingness to incur some additional security-specific engineering costs.
- **EAL 5**: **semiformally designed and tested:** Provides an analysis that includes all of the implementation. Assurance is supplemented by a formal model and a semiformal presentation of the functional specification and high-level design and a semiformal demonstration of correspondence. The search for vulnerabilities must ensure resistance to penetration attackers with a moderate attack potential. Covert channel analysis and modular design are also required.
- **EAL 6**: **semiformally verified design and tested:** Permits a developer to gain high assurance from application of specialized security engineering techniques in a rigorous development environment, and to produce a premium TOE for protecting high value assets against significant risks. The independent search for vulnerabilities must ensure resistance to penetration attackers with a high attack potential.
- **EAL 7**: **formally verified design and tested:** The formal model is supplemented by a formal presentation of the functional specification and high level design, showing correspondence. Evidence of developer "white box" testing of internals and complete independent confirmation of developer test results are required. Complexity of the design must be minimized.

The first four levels reflect various levels of commercial design practice. Only at the highest of these levels (EAL 4) is there a requirement for any source code analysis, and this only for a portion of the code. The top three levels provide specific guidance for products developed using security specialists and security-specific design and engineering approaches.

Evaluation Process

The aim of evaluating an IT product, a TOE, against a trusted computing standard is to ensure that the security features in the TOE work correctly and effectively, and that show no exploitable vulnerabilities. The evaluation process is performed either in parallel with, or after, the development of the TOE, depending on the level of assurance required. The higher the level, the greater the rigor needed by the process and the more time and expense that it will incur. The principle inputs to the evaluation are the security target, a set of evidence about the TOE, and the actual TOE. The desired result of the evaluation process is to confirm that the security target is satisfied for the TOE, confirmed by documented evidence in the technical evaluation report.

The evaluation process will relate the security target to one or more of the high-level design, low-level design, functional specification, source code implementation, and object code and hardware realization of the TOE. The degree of rigor used, and the depth of analysis are determined by the assurance level desired for the evaluation. At the higher levels, semiformal or formal models are used to confirm that the TOE does indeed implement the desired security target. The evaluation process also involves careful testing of the TOE to confirm it's security features.

The evaluation involves a number of parties:

- **Sponsor:** Usually either the customer or the vendor of a product for which evaluation is required. Sponsors determine the security target that the product has to satisfy.
- **Developer:** Has to provide suitable evidence on the processes used to design, implement, and test the product to enable its evaluation.
- **Evaluator:** Performs the technical evaluation work, using the evidence supplied by the developers, and additional testing of the product, to confirm that it satisfies the functional and assurance requirements specified in the security target. In many countries, the task of evaluating products against a trusted computing standard is delegated to one or more endorsed commercial suppliers.
- **Certifier:** The government agency that monitors the evaluation process and subsequently certifies that a product as been successfully evaluated. Certifiers generally manage a register of evaluated products, which can be consulted by customers.

The evaluation process has three broad phases:

1. **Preparation:** Involves the initial contact between the sponsor and developers of a product, and the evaluators who will assess it. It will confirm that the sponsor and developers are adequately prepared to conduct the evaluation and will include a review of the security target and possibly other evaluation deliverables. It concludes with a list of evaluation deliverables and acceptance of the overall project costing and schedule.
2. **Conduct of evaluation:** A structured and formal process in which the evaluators conduct a series of activities specified by the CC. These include reviewing the deliverables provided by the sponsor and developers, and other tests

of the product, to confirm it satisfies the security target. During this process, problems may be identified in the product, which are reported back to the developers for correction.

3. **Conclusion:** The evaluators provide the final evaluation technical report to the certifiers for acceptance. The certifiers use this report, which may contain confidential information, to validate the evaluation process and to prepare a public certification report. The certification report is then listed on the relevant register of evaluated products.

The evaluation process is normally monitored and regulated by a government agency in each country. In the United States the NIST and the NSA jointly operate the Common Criteria Evaluation and Validation Scheme (CCEVS). Many countries support a peering arrangement, which allows evaluations performed in one country to be recognized and accepted in other countries. Given the time and expense that an evaluation incurs, this is an important benefit to vendors and consumers. The Common Criteria Portal provides further information on the relevant agencies and processes used by participating countries.

13.8 RECOMMENDED READING

[LAND81] is a comprehensive survey of computer security models but does not present any of the mathematical or formal details. [BELL05] summarizes the Bell-LaPadula model and examines its relevance to contemporary system design and implementation.

[GALL09] is a worthwhile survey of the topics covered in this chapter. [GASS88] provides a comprehensive study of trusted computer systems. [SAYD04] is a historical summary of the evolution of multilevel security in military and commercial contexts.

[BERT95] and [LUNT90] examine the issues related to the use of multilevel security for a database system. [DENN85] and [MORG87] focus on the problem of inference in multilevel secure databases.

[OPPL05] and [FELT03] provide overviews of trusted computing and the TPM. [ENGL03] describes Microsoft's approach to implementing trusted computing on Windows.

BELL05 Bell, D. "Looking Back at the Bell-LaPadula Model." *Proceedings, 21st Annual IEEE Computer Security Applications Conference*, 2005.

BERT95 Bertino, E.; Japonica, S.; and Samurai, P. "Database Security: Research and Practice." *Information Systems*, Vol. 20, No. 7, 1995.

DENN85 Denning, D. "Commutative Filters for Reducing Interference Threats in Multilevel Database Systems." *Proceedings of 1985 IEEE Symposium on Security and Privacy*, 1985.

ENGL03 England, P., et al. "A Trusted Open Platform." *Computer*, July 2003.

FELT03 Felten, E. "Understanding Trusted Computing: Will Its Benefits Outweigh its Drawbacks?" *IEEE Security and Privacy*, May/June 2003.

GALL09 Galley, E., and Mitchell, C. "Trusted Computing: Security and Applications." *Cryptologia*, Volume 33, Number 1, 2009.

GASS88 Gasser, M. *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.

LAND81 Landwehr, C. "Formal Models for Computer Security." *Computing Surveys*, September 1981.

LUNT90 Lunt, T., and Fernandez, E. "Database Security." *ACM SIGMOD Record*, December 1990.

MORG87 Morgenstern, M. "Security and Inference in Multilevel Database and Knowledge-Base Systems." *ACM SIGMOD Record*, December 1987.

OPPL05 Oppliger, R., and Rytz, R. "Does Trusted Computing Remedy Computer Security Problems?" *IEEE Security and Privacy*, March/April 2005.

SAYD04 Saydjari, O. "Multilevel Security: Reprise." *IEEE Security and Privacy*, September/October 2004.

13.9 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

Bell-LaPadula (BLP) model	multilevel security (MLS)	security requirements
Biba integrity model	polyinstantiation	simple security property (ss-property)
certification rules	reference monitor	target of evaluation
Chinese Wall Model	sanitized data	threat
Clark-Wilson integrity model	security assurance requirements	Trojan horse
class	security class	trust
Common Criteria (CC)	security classification	trusted computer system
component	security clearance	trusted computing
ds-property	security functional requirements	trusted computing base
enforcement rules	security kernel database	trusted platform module (TPM)
family	security level	trusted system
mandatory access control (MAC)	security objective	trustworthy system
		*-property

Review Questions

13.1 Explain the differences among the terms *security class*, *security level*, *security clearance*, and *security classification*.

13.2 What are the three rules specified by the BLP model?

13.3 How is discretionary access control incorporated into the BLP models

13.4 What is the principal difference between the BLP model and the Biba model?

13.5 What are the three rules specified by the Biba model?

13.6 Explain the difference between certification rules and enforcement rules in the Clark-Wilson model.

13.7 What is the meaning of the term *Chinese wall* in the Chinese Wall Model?

13.8 What are the two rules that a reference monitor enforces?

13.9 What properties are required of a reference monitor?

13.10 In general terms, how can MLS be implemented in an RBAC system?

13.11 Describe each of the possible degrees of granularity possible with an MLS database system.

13.12 What is polyinstantiation?

13.13 Briefly describe the three basic services provided by a TPMs.

13.14 What is the aim of evaluating an IT product against a trusted computing evaluation standard?

13.15 What is the difference between *security assurance* and *security functionality* as used in trusted computing evaluation standards?

13.16 Who are the parties typically involved in a security evaluation process?

13.17 What are the three main stages in an evaluation of an IT product against a trusted computing standard, such as the Common Criteria?

Problems

13.1 The necessity of the "no read up" rule for a multilevel secure system is fairly obvious. What is the importance of the "no write down" rule?

13.2 The *-property requirement for append access $f_c(S_i) \leq f_o(O_j)$ is looser than for write access $f_c(S_i) = f_o(O_j)$. Explain the reason for this.

13.3 The BLP model imposes the ss-property and the *-property on every element of b but does not explicitly state that every entry in M must satisfy the ss-property and the *-property.

a. Explain why it is not strictly necessary to impose the two properties on M.

b. In practice, would you expect a secure design or implementation to impose the two properties on M? Explain.

13.4 In the example illustrated in Figure 13.2, state which of the eight BLP rules are invoked for each action in the scenario.

13.5 In Figure 13,2, the solid arrowed lines going from the level roles down to the operation roles indicate a role hierarchy with the operation roles having the indicated access rights (read, write) as a subset of the level roles. What do the solid arrowed lines going from one operation role to another indicate?

13.6 Consider the following system specification using a generic specification language:

constants

subjects = set of processes

sec_labels = {1, 2, 3, … MAX} such that $1 < 2 < \ldots <$ MAX

files = set of information sequences

label: subjects —> sec_labels

class(repository) = MAX

variables

respository: = set of all sets of files

initial state

repository = null set

actions