Chapter 9

Security Models and Practice

Contents

9.1 Policy, Models, and Trust

Designing secure systems requires a clear idea of the security goals that are to be achieved, and an implementation framework in which to try to achieve those goals.

9.1.1 Security Policy

A key component of such a framework is for designers to define a ***security policy***, which is a well-defined set of rules that include the following components:

- ***Subjects***: the agents who interact with the system, which could be defined in terms of specific individuals or in terms of roles or ranks that groups of individuals might hold within an organization. Individuals could be identified by their names or by their job titles, like President, CEO, or CFO. Groups could be defined using terms such as users, administrators, generals, majors, faculty, deans, managers, and administrative assistants. This category also includes outsiders, such as attackers and guests.
- ***Objects***: the informational and computational resources that a security policy is designed to protect and manage. Informational examples include critical documents, files, and databases, and computational resources include servers, workstations, and software.
- ***Actions***: the things that subjects may or may not do with respect to the objects. Examples include the reading and writing of documents, updating software on a web server, and accessing the contents of a database.
- ***Permissions***: mappings between subjects, actions, and objects, which clearly state what kinds of actions are allowed or disallowed.
- ***Protections***: the specific security features or rules that are included in the policy to help achieve particular security goals, such as confidentiality, integrity, availability, or anonymity.

Thus, a security policy places constraints on what actions the subjects in a system can do with respect to the objects in that system, in order to achieve specific security goals. These policies are useful and often required. Several compliance regulations, such as the ***Health Insurance Portability and Accountability Act*** (***HIPAA***), the ***Gramm-Leach-Bliley Act*** (***GLBA***), and the ***Sarbanes-Oxley Act*** (***SOX***), require that an organization, such as a hospital, financial institution, or public corporation, have such policies.

9.1.2 Security Models

A ***security model*** is an abstraction that provides a conceptual language for administrators to specify security policies. Typically, security models define hierarchies of access or modification rights that members of an organization can have, so that subjects in an organization can easily be granted specific rights based on the position of these rights in the hierarchy. Examples include military classifications of access rights for documents based on concepts like "unclassified," "confidential," "secret," and "top secret."

These abstractions give policy writers a notational shorthand for defining access rights. Without such abstractions, security policies would be needlessly long. For instance, it is typical for a manager to have all of the same access rights as his subordinates, and more. A policy could spell this out in laborious detail or it could simply define a manager's rights in terms of a security model that automatically includes subordinate rights in a manager's list by using a hierarchy.

Discretionary and Mandatory Access Control

As mentioned in Section 3.3.3, two of the most widely used models of access control are ***discretionary*** and ***mandatory*** access control.

In general, ***discretionary access control***, or ***DAC***, refers to a scheme where users are given the ability to determine the permissions governing access to their own files. DAC typically features the concept of both users and groups, and allows users to set access-control measures in terms of these categories. In addition, DAC schemes allow users to grant privileges on resources to other users on the same system. Most computer systems employ some variant of a DAC scheme for access control to its resources. For example, both Linux and Windows allow users to specify file and folder permissions by means of access control lists (Section 3.3.3). These permissions in turn affect the rights other users have with respect to these files.

In contrast, ***mandatory access control*** is a more restrictive scheme that does not allow users to define permissions on files, regardless of ownership. Instead, security decisions are made by a central policy administrator. Each security rule consists of a ***subject***, which represents the party attempting to gain access, an ***object***, referring to the resource being accessed, and a series of permissions that define the extent to which that resource can be accessed. ***Security-Enhanced Linux*** (***SELinux***) incorporates mandary access control as a means of explicitly defining permissions to minimize abuse and misconfiguration issues. Mandatory access-control models attempt to prevent transfer of information that is not allowed by the rules.

9.1.3 Trust Management

The concept of ***trust*** is difficult to define. We know it involves a confidence in an entity's ability and intentions, but there are also subjective elements, including our own risk tolerance and culture. So, rather than try to formalize a rigorous definition of trust, let us consider a related concept instead.

A ***trust management*** system is a formal framework for specifying security policy in a precise language, which is usually a type of logic or programming language, together with a mechanism for ensuring that the specified policy is enforced. Thus, a trust management system consists of two main components, a ***policy language*** and a ***compliance checker***. Policy rules are specified in the policy language and are enforced by the compliance checker. These language and enforcement components typically involve rules describing four concepts. (See Figure 9.1.)

- ***Actions***: operations with security-related consequences on the system
- ***Principals***: users, processes, or other entities that can perform actions on the system
- ***Policies***: precisely written rules that govern which principals are authorized to perform which actions
- ***Credentials***: digitally signed documents that bind principal identities to allowable actions, including the authority to allow principals to delegate authority to other principals.

KeyNote

There are several languages that specify these terms and how they are applied. The ***KeyNote*** system, first presented in 1999, is one such language. In addition to implementing the terms defined above, KeyNote defines an ***application*** to be a program or system that uses KeyNote. A ***policy compliance value*** (***PCV***) is the answer issued by KeyNote in response to a ***request*** by a principal to perform some action—it indicates whether the requested action conforms to existing policies. To use KeyNote, an application queries the KeyNote system when a principal requests an action, including the appropriate policies and credentials in the query. The action is described using a set of attribute-value pairs known as an ***action attribute set*** that illustrates the security implications of that action. KeyNote then replies with a PCV indicating whether or not the action should be allowed, and the application behaves accordingly. Note that KeyNote merely interprets whether or not a given action should be permitted according to the provided policies, and it is up to the application to invoke KeyNote properly and correctly interpret its responses.

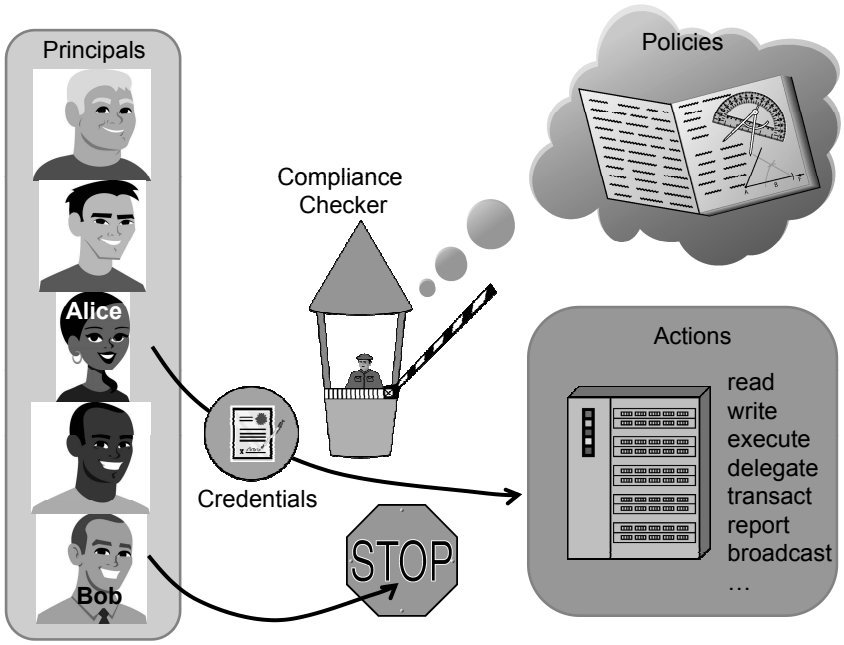

Figure 9.1: A trust management system. In this example, Alice has sufficient valid credentials for her requested action, with respect to the specified policies, and Bob does not.

XACML

A newer policy language, the ***Extensible Access Control Markup Language*** (***XACML***), was released in 2009. XACML leverages the Extensible Markup Language (XML) to define security policies and describe how these policies should be enforced. Using XML allows administrators to declare policies in a widely adopted and easy-to-use format that is already well supported in many environments. Like KeyNote, XACML includes all of the traditional trust management features: a principal is referred to as a ***subject*** that can request to perform actions on a ***resource***. XACML also introduces several new concepts. A ***policy administration point*** (***PAP***) manages security policies. A ***policy decision point*** (***PDP***) is responsible for issuing authorizations, analogously to how KeyNote issues PCVs. A ***policy enforcement point*** (***PEP***) requests access on behalf of a principal and behaves in accordance with the PDP's response. Finally, a ***policy information point*** (***PIP***) provides additional secutiry-related information. XACML also includes an approach to delegation that allows parties to grant rights to other parties without a central policy administrator. This is possible because the rights to delegate privileges are maintained independently from access rights.

9.2 Access-Control Models

In this section, we overview various models that were developed to formalize mechanisms to protect the confidentiality and integrity of documents stored in a computer system.

9.2.1 The Bell-La Padula Model

The ***Bell-La Padula*** (***BLP***) model is a classic example of a mandatory access-control model for protecting confidentiality. The BLP model is derived from the military ***multilevel security*** paradigm, which has been traditionally used in military organizations for document classification and personnel clearance. Such a security model has a strict, linear ordering on the security of levels of documents, so that each document has a specific security level in this ordering and each user is assigned a strict level of access that allows them to view all documents with the corresponding level of security or below. (See Figure 9.2.)

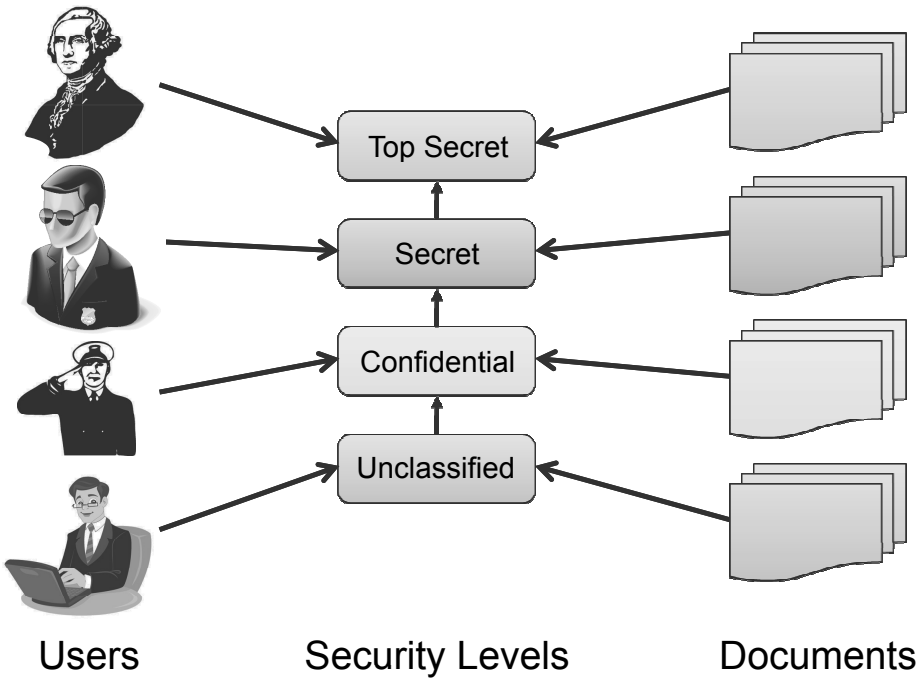

Figure 9.2: A multilevel security system, as in the military model, which defines a strict linear order on the security levels of documents.

Military Classification Hierarchies

For example, a typical multilevel security system based on the military security framework works as follows:

- There are several security levels. The bottom level is ***unclassified***. The other levels are in increasing order of security, with names like ***confidential***, ***secret***, and ***top secret***, respectively.
- Each document is classified at one of the security levels.
- Each user obtains "clearance" at one of the security levels.
- A document of a certain level can be accessed only by users with the same or higher clearance level.

Total Orders and Partial Orders

Such a strict, linear ordering for documents can be defined in terms of a comparison rule, $\leq$. We say that such a rule defines a ***total order*** on a universe set, U, if it satisfies the following properties:

- ***Reflexivity***: If x is in U, then $x \leq x$.
- ***Antisymmetry***: If $x \leq y$ and $y \leq x$, then $x = y$.
- ***Transitivity***: If $x \leq y$ and $y \leq z$, then $x \leq z$.
- ***Totality***: If x and y are in U, then $x \leq y$ or $y \leq x$.

All of the usual definitions of "less than or equal to" for numbers, such as integers and real numbers, are total orders.

We can still define a notion of "less than or equal to," however, if we drop the requirement of totality. In this case, we get a ***partial order***, denoted with $\preceq$. The classic example of a partial order is the set of courses taught at a college or university, where we say that, for two courses A and B, $A \preceq B$, if A is a prerequisite for B. Note, in particular, that there are no cycles of prerequisites, for otherwise no one would be able to satisfy the prerequisites for any course in such a cycle.

Given a partial order $\preceq$, it is always possible to find an associated total order $\leq$ compatible with $\preceq$, that is, if $x \preceq y$, then $x \leq y$. Such a total order is not unique in general.

How the BLP Model Works

Similar to a military security model, the BLP model also has security levels. Instead of forming a strict linear order, however, as in the military model, the security levels in BLP form a partial order, $\preceq$.

In addition, instead of documents, more general objects are considered. Each object, x, is assigned to a security level, $L(x)$. Similarly, each user, u, is assigned to a security level, $L(u)$. Access to objects by users is controlled by the following two rules:

- ***Simple security property***. A user u can read an object x only if

$$L(x) \preceq L(u).$$

- ****-property.*** A user u can write (create, edit, or append to) an object x only if

$$L(u) \preceq L(x).$$

The simple security property is also called the "no read up" rule, as it prevents users from viewing objects with security levels higher than their own. The *-property is also called the "no write down" rule. It is meant to prevent propagation of information to users with a lower security level. The BLP rules capture the principle that information can only flow up, going from lower security levels to higher security levels.

A consequence of these rules is that users with different security levels can have only one-way communication. Namely, if $L(u) \preceq L(v)$, then u can send a message to v (u writes up the message and v reads down the message). However, the opposite is not possible. To overcome this problem, one can change the meaning of $L(u)$ to represent the ***maximum security level*** that u can have and allow u to assume a ***current security level***, $C(u)$, such that $C(u) \preceq L(u)$. Now, if $L(u) \preceq L(v)$, user v can assume current security level $C(v) = L(u)$ to have a two-way communication with user u.

Using the BLP Model

In practical applications of the BLP model, it is common to define the partial order $\preceq$ of security levels starting from a set B of ***basic levels*** that have a linear order $\leq$ and a collection S of ***categories*** (also called ***compartments***). A security level $L(x)$ now consists of a pair $(b(x), S(x))$, where $b(x) \in B$ and $S(x) \subseteq S$. Also, the level of x precedes the level of y if the basic level of x is less than the basic level of y and the subset of categories of x is contained in the subset of categories of y.

Defining Security Levels Using Categories

In other words, using the approach of defining security levels as pairs of traditional security labels and categories, we can write the comparison rule as follows:

$$(b(x), S(x)) \preceq (b(y), S(y)) \iff b(x) \leq b(y) \text{ and } S(x) \subseteq S(y).$$

An example of partial order built from the military basic levels and a set of categories associated with geographic regions is shown in Figure 9.3.

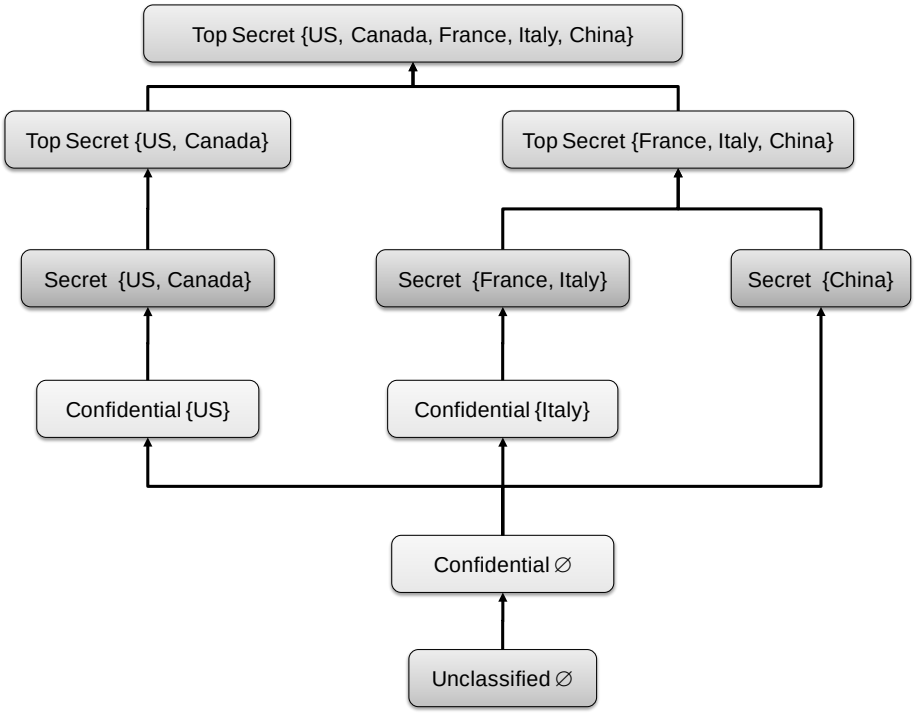

Figure 9.3: A partial order of security levels induced by basic levels and categories in the BLP model.

Incidentally, the BLP model can be further augmented with discretionary access-control rules, expressed, for example, by access control lists for every object. While such discretionary access-control rules can be created and updated by users, only system administrators can modify the mandatory access-control rules associated with security levels in such a scheme.

9.2.2 Other Access-Control Models

There are several other access-control models that differ from the Bell-La Padula model. Some alternative models address security goals other than confidentiality, and other models make changes in the two basic access-control rules of the BLP model.

The Biba Model

The ***Biba model*** has a similar structure to the BLP model, but it addresses integrity rather than confidentiality. Objects and users are assigned ***integrity levels*** that form a partial order, similar to the BLP model. The integrity levels in the Biba model indicate degrees of trustworthiness, or accuracy, for objects and users, rather than levels for determining confidentiality. For example, a file stored on a machine in a closely monitored data center would be assigned a higher integrity level than a file stored on a laptop. In general, a data-center computer is less likely to be compromised than a random laptop computer. Likewise, when it comes to users, a senior employee with years of experience would have a higher integrity level than an intern.

The access-control rules for Biba are the reverse of those for BLP. That is, Biba does not allow reading from lower levels and writing to upper levels. In particular, if we let $I(u)$ denote the integrity level of a user u and $I(x)$ denote the integrity level for an object, x, we have the following rules in the Biba model:

- A user u can read an object x only if

$$I(u) \preceq I(x).$$

- A user u can write (create, edit or append to) an object x only if

$$I(x) \preceq I(u).$$

Thus, the Biba rules express the principle that information can only flow down, going from higher integrity levels to lower integrity levels.

The Low-Watermark Model

The ***low-watermark model*** is an extension to the Biba model that relaxes the "no read down" restriction, but is otherwise similar to the Biba model. In other words, users with higher integrity levels can read objects with lower integrity levels. After such a reading, the user performing the reading is demoted such that his integrity level matches that of the read object. One

example of an implementation of the low-watermark model is LOMAC, a security extension that can be loaded as a kernel module on Linux.

The Clark-Wilson Model

Rather than dealing with document confidentiality and/or integrity, the ***Clark-Wilson*** (***CW***) model deals with systems that perform transactions. It describes mechanisms for assuring that the integrity of such a system is preserved across the execution of a transaction. Key components of the CW model include the following:

- Integrity constraints that express relationships among objects that must be satisfied for the system state to be valid. A classic example of an integrity constraint is the relationship stating that the final balance of a bank account after a withdrawal transaction must be equal to the initial balance minus the amount withdrawn.

- Certification methods that verify that transactions meet given integrity constraints. Once the program for a transaction is certified, the integrity constraints do not need to be verified at each execution of the transaction.

- Separation of duty rules that prevent a user that executes transaction from certifying it. In general, each transaction is assigned disjoint sets of users that can certify and execute it, respectively.

The Chinese Wall Model

The ***Brewer and Nash model***, commonly referred to as the ***Chinese wall model***, is designed for use in the commercial sector to eliminate the possibility of conflicts of interest. To achieve this, the model groups resources into "conflict of interest classes." The model enforces the restriction that each user can only access one resource from each conflict of interest class. In the financial world, such a model might be used, for instance, to prevent market analysts from receiving insider information from one company and using that information to provide advice to that company's competitor. Such a policy might be implemented on computer systems to regulate users' access to sensitive or proprietary data.

9.2.3 Role-Based Access Control

The ***role-based access control*** model can be viewed as an evolution of the notion of group-based permissions in file systems. An RBAC system is defined with respect to an organization, such as company, a set of resources, such as documents, print services, and network services, and a set of users, such as employees, suppliers, and customers.

Core RBAC

The main components of the RBAC model are users, roles, permissions, and sessions, defined as follows:

- A ***user*** is an entity that wishes to access resources of the organization to perform a task. Usually, users are actual human users, but, more generally, a user can also be a machine or an application, if such entities can be assigned identities.
- A ***role*** is defined as a collection of users with similar functions and responsibilities in the organization. Examples of roles in a university may include "student," "alum," "faculty," "dean," "staff," and "contractor." In general, a user may have multiple roles. For example, an administrative assistant at a university who enrolls for an accounting course may have the roles "staff" and "student." Note that some roles may be a subset of other roles. For instance, deans are usually a subset of the faculty of a university. Also, some roles may have a unique user, such as the president of a university or CEO of a company. Roles and their functions are often specified in the written documents of the organization, such as bylaws and statutes. The assignment of users to roles follows resolutions by the organization, such as employment actions (e.g., hiring, promotion, and resignation) and academic actions (e.g., admission, degree conferral, and suspension).
- A ***permission*** describes an allowed method of access to a resource. More specifically, a permission consists of an operation performed on an object, such as "read a file" or "open a network connection." Each role has an associated set of permissions.
- A ***session*** consists of the activation of a subset of the roles of a user for the purpose of performing a certain task. For example, a laptop user may create a session with the administrator role to install a new program. Later on, the same user may create another session with a nonprivileged role to use the application. Sessions support the principle of least privilege (Section 1.1.4).

The Power of Role-Based Access Control

The components described above characterize what is known as the ***core*** role-based, access-control (RBAC) model, which generalizes the widely used concept of user groups and introduces the notion of sessions.

The power of the RBAC model is given by two additional components, however:

- ***Role hierarchy***
- ***Role constraints***

We describe these next.

Hierarchical RBAC

In the role-based access control model, roles can be structured in a hierarchy similar to an organization chart. More formally, we define a partial order among roles by saying that a role R_1 ***inherits*** role R_2, which is denoted

$$R_1 \succeq R_2,$$

if R_1 includes all permissions of R_2 and R_2 includes all users of R_1. When $R_1 \succeq R_2$, we also say that role R_1 is ***senior*** to role R_2 and that role R_2 is ***junior*** to role R_1.

For example, in a company, the role "manager" inherits the role "employee" and the role "vice president" inherits the role "manager." Also, in a university, the roles "undergraduate student" and "graduate student" inherit the role "student." Informally, the notion of inheritance is captured by the phrase "is a," as in the phrase "an assistant professor ***is a*** faculty member" or the phrase "a provost ***is an*** administrator."

Inheritance simplifies the administrative management of permissions and users associated with roles. That is, when the system administrator adds a permission to a role, the system can propagate this permission addition to senior roles. For example, adding the permission of viewing student grades to the role professor, automatically adds this permission also to the role dean and other roles senior to professor.

Similarly, when the system administrator adds a user to a role, the system can propagate this user addition to all junior roles. For instance, adding user Mike to the role professor automatically adds Mike to the role employee and other roles junior to professor.

Visualizing Role Hierarchy

Role hierarchies can be graphically represented with a diagram where each role is connected to its immediate predecessors and successors in the hierarchy. That is, an edge is drawn from role a R_1 to a role R_2 if

$$R_1 \succeq R_2$$

and there is no other role R_3 distinct from R_1 and R_2 such that

$$R_1 \succeq R_3 \succeq R_2.$$

Also, the diagram is drawn so that each role is placed at a higher y-coordinate than its junior roles.

An example of such a diagram for a role hierarchy is shown in Figure 9.4.

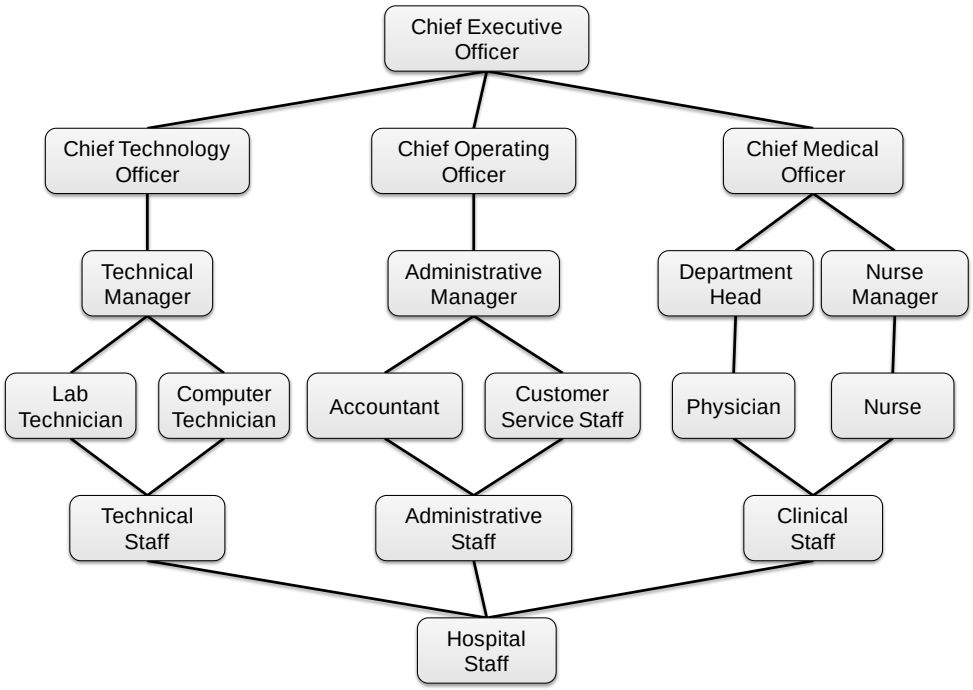

Figure 9.4: A simplified role hierarchy for a hospital. Note that, in going down from one node X to a connected node Y below it, we can always say "X is a Y."

Constrained RBAC

To provide support for the principle of separation of privilege, the RBAC model allows us to define ***constraints*** that prevent users from having incompatible roles that create conflicts of interest.

The simplest form of constraint is a pair of roles (R_1, R_2) indicating that no user can be assigned to both roles R_1 and R_2. For example, in a university, no user should have the roles of teaching assistant (who recommends grades) and instructor (who is responsible for a course and finalizes grades based on recommendations from the teaching assistant) at the same time. Similarly, in a company, no user should have both the roles of buyer, who proposes purchases of goods, and controller, who reviews and approves purchase orders. A more general constraint is defined by a pair (S, k), where S is a subset of roles and $k \geq 2$ is an integer. This constraint, called a ***separation of duty*** relation, stipulates that no user can have k or more roles from S.

Separation of duty relations can have two different meanings, static and dynamic.

- In a ***static*** separation of duty relation (S, k), the constraint holds for the assignment of users to roles. That is, no user can be ***assigned*** to k or more roles in S.

- In a ***dynamic*** separation of duty relation (S, k), the constraint holds for the activation of roles of users in sessions. That is, no user can have k or more roles in S ***activated*** in a session.

Dynamic separation of duty relations are more flexible, as they allow users to have different roles from an incompatible set in different sessions, so long as the sessions don't overlap in time. For example, suppose that Anna is the head of the research division of a company. In one session, Anna activates the role "supervisor" to approve a travel expense report from an employee of her division. In another session, she activates the role "traveler" to submit her own travel expense report. The following dynamic separation of duty relation assures that Anna cannot approve her own travel expense report.

$$(\{\mathsf{supervisor}, \mathsf{traveler}\}, 2)$$

When constrained RBAC and hierarchical RBAC coexist, separation of duties should be interpreted in the context of inheritance. Specifically, if a role R_1 inherits a role R_2 and R_2 is involved in a separation of duty relation (S, k), that is, $R_2 \in S$, then the assignment or activation constraint holds for users of role R_1 as well.

9.3 Security Standards and Evaluation

Many different organizations have developed standards that define how to enforce and assess security practices and policies in high-security contexts. In particular, various government organizations, including the United States Department of Defense and the National Security Agency, have developed stringent regulations regarding computer systems which may be used to store and transfer highly sensitive information. We review some of these standards in this section.

9.3.1 Orange Book and Common Criteria

The ***Trusted Computer System Evaluation Criteria*** (***TCSEC***), commonly referred to as the ***Orange Book*** (because of its orange cover), was developed in 1983, and updated in 1985, as a standard for evaluating the security of computers storing classified information. (See Figure 9.5.)

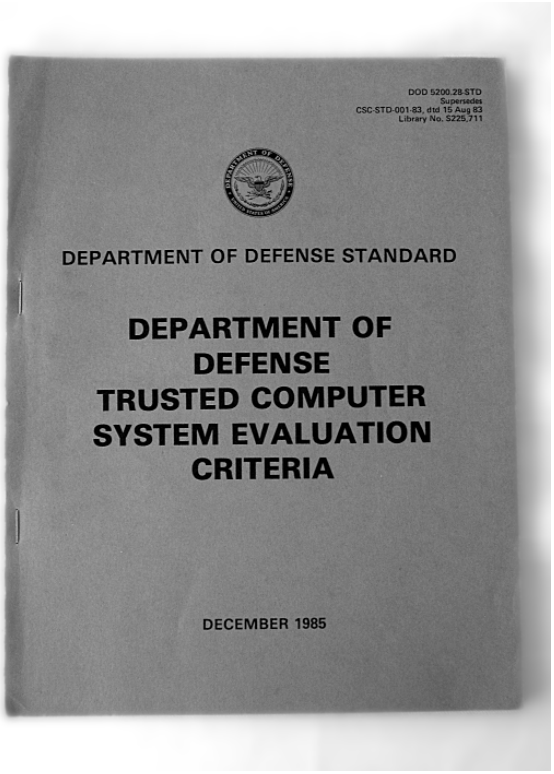

Figure 9.5: The Orange Book, as updated in 1985. (Public domain image.)

The Orange Book defines four "divisions" of security criteria:

- Division D represents a system with "minimal protection." This status is assigned to systems that have been evaluated by TCSEC but do not meet security requirements for a higher-level division.
- Division C guarantees "discretionary protection," indicating the system makes use of some type of discretionary access-control system (Section 3.3.3).
- Division B guarantees "mandatory protection," indicating the system implements mandatory access control (Section 3.3.3).
- Division A guarantees "verified protection," demonstrating that a system has a formal process for verification of security.

Common Criteria

The ***Common Criteria for Information Technology Security Evaluation***, commonly referred to as ***Common Criteria***, is a set of international standards describing a computer security certification. In the United States, it has replaced TCSEC as the standard measure of computer security in government organizations. Specifically, it defines key concepts related to security evaluation and details how to conduct evaluations in a standardized manner:

- The ***target of evaluation*** (***TOE***) is the system subject to evaluation.
- A ***protection profile*** (***PP***) describes a set of security requirements for a broad class of security devices, such as an operating system or firewall.
- A ***security target*** (***ST***) is a document that defines the vendor's security goals for the TOE, each of which is evaluated based on the implementation of the system.

Common Criteria is not a certification that vouches for the security of a product. Instead, it is a framework by which vendors can document the security goals of their products and evaluate their systems in the context of those goals. For example, newer versions of Microsoft Windows are certified as having been evaluated according to the Common Criteria, but security vulnerabilities in Windows are not uncommon. The certification indicates that Microsoft was able to carefully define security goals and assess Windows according to the Common Criteria framework, but it does not assert that Windows is secure in a more general sense. Some researchers have therefore criticized the Common Criteria as expensive, time-consuming, and not particularly effective at guaranteeing functional security.

9.3.2 Government Regulations and Standards

With the increase in importance of computer security, there are now several government regulations and standards regarding security and privacy requirements of systems that impact citizens.

FIPS 140

The ***Federal Information Processing Standardization*** (***FIPS***) ***140*** are a set of standards setting requirements for cryptographic modules used by government organizations in the United States. FIPS discusses requirements in eleven areas, including documentation, flow of information, physical security, key management, and attack mitigation. The newest release of these standards, FIPS 140-2, defines the specifications for four "levels" of security, each with different requirements.

Security Level 1 is the lowest level of security. It provides no mechanism for ensuring physical security, and allows the cryptographic module to be executed on a general-purpose computer system such as a personal computer. Level 2 increases stringency by requiring physical security measures, such as tamper-evident coatings and pick-resistant locks, introduces a requirement for a role-based authentication system, and mandates a trusted operating system adhering to additional standards. At Level 3, requirements are provided to prevent (rather than merely detect) physical tampering, and identity-based authentication replaces the role-based requirements of Level 2. The strictest level, Level 4, tightens physical security measures in that all sensitive cryptographic keys and messages are destroyed in the event of unauthorized attempts at physical access. In addition, further measures are implemented to protect against certain environmental conditions such as extremes in temperature and voltage.

Other Standards

Certain industries are legally bound to adhere to various standards that dictate requirements for storing information. For example, the storage of healthcare records in the United States is regulated according to the ***Health Insurance Portability and Accountability Act*** (***HIPAA***). HIPAA establishes standards requiring healthcare providers and employers to maintain the privacy of patient records. Title II of HIPAA defines five rules dealing with healthcare documents. In particular, the "Privacy Rule" defines the concept of ***Protected Health Information*** (***PHI***) and sets regulations on the use and disclosure of this information. The Privacy Rule, which applies to both paper and electronic documents, requires that if healthcare providers

need to share PHI without a patient's permission (to facilitate medical care, for example), only the minimum amount of information necessary for treatment is disclosed. The "Security Rule" defines administrative, physical, and technical security safeguards designed to prevent access by unauthorized parties to PHI stored in electronic form. Small health care organizations (e.g., a dental practice with one or two dentists) often find the HIPAA "Security Rule" to be onerous to implement. This has slowed adoption of electronic record keeping in the health care sector. Adherence to HIPAA is mandated by law, and healthcare providers may be subject to legal action if they are found to be in breach of HIPAA standards.

The ***Family Educational Rights and Privacy Act*** (***FERPA***) establishes similar requirements for protecting the privacy of educational records in the United States. As with HIPAA, both electronic and paper records are covered by the law. Under FERPA, all students must have access to their own student records. In addition, schools must request consent from a student before disclosing that student's educational records to another party. Students are also given the right to view recommendations included in applications to educational institutions, but students commonly waive this right at the request of the institution or recommender.

In contrast to the United States' compartmentalized approach to privacy standards, the European Union established the ***Data Protection Directive*** as a single standard to regulate the processing of any type of personal information, including bank statements, criminal records, and healthcare information. This directive defines three categories of conditions that must be met to warrant the disclosure of sensitive information: transparency, legitimate purpose, and proportionality. Transparency dictates that the data subject is informed of the disclosure of his or her information, and that either consent or allowed cause for disclosure is provided. Legitimate purpose requires that information can only be disclosed for specified reasons, and must not be used in any other manner. Finally, proportionality requires that only necessary personal data is processed.

In addition to standards designed to regulate personal information, public companies in the United States must adhere to strict guidelines detailing how financial records are processed and stored. The ***Sarbanes-Oxley Act***, also known as ***SOX***, was passed in 2002 and lays out stringent rules dictating how corporate accounting should be conducted and audited. In particular, executives of a company may be held personally liable for fraudulent record keeping by the company. SOX has been criticized as overly rigorous and costly, potentially putting United States corporations at a disadvantage in the international market. Nevertheless, many see SOX compliance as a necessity to guarantee corporate transparency and accountability for fraud.

9.4 Software Vulnerability Assessment

A modern computer system is a collection of many complex components working together. A single flaw in any of these components could result in a compromise of the security of the entire system. Such flaws may be extremely subtle, ranging from minor coding errors to device misconfiguration.

The process of identifying these types of flaws, whether they reside in an operating system, application software, or in the configuration of network devices, is known as ***vulnerability assessment***. We begin by discussing techniques used in the analysis of software, before examining how the security of a network is assessed. Similar terminology is used in both situations.

Black-Box Analysis

Black-box analysis refers to an assessment where the inner workings of the target are hidden from the auditor. Intuitively, the system is sitting inside a "black box," which can only be observed in terms of its input-output behavior. For example, a network auditor performing a black-box test may have the public address of a target web site, but no knowledge of the internals of that web site's surrounding network.

In software, black-box assessments are typically performed by independent vulnerability research groups auditing commercial software. In these situations, auditors may have working copies of the software, but no access to its source code. Black-box testing is designed to simulate the capabilities of a real-life attacker and attempt to address issues that are most likely to occur in real situations.

White-Box Analysis

In contrast, ***white-box analysis*** gives auditors access to any additional information required to conduct a full assessment, such as source code, documentation, and detailed network topology, besides the system's input-output behavior.

White-box analysis enables auditors to discover vulnerabilities that may be difficult to find without this additional transparency, but this extra knowledge may come with the cost of greater time and financial investment to complete an analysis.

Gray-Box Analysis

Gray-box analysis falls somewhere between these two extremes. It requires that a carefully selected subset of details be available to auditors, often chosen to encourage focus on high-risk areas, but not the full disclosure that would be required for a white-box analysis.

9.4.1 Static and Dynamic Analysis

Once the scope of an audit has been determined, auditors can employ a variety of techniques to attempt to discover vulnerabilities in targets, which broadly fall into two categories, ***static analysis*** and ***dynamic analysis***. (See Figure 9.6.)

- Static analysis involves the examination of a system just by looking at its code and data.
- Dynamic analysis involves the examination of a system while it is running.

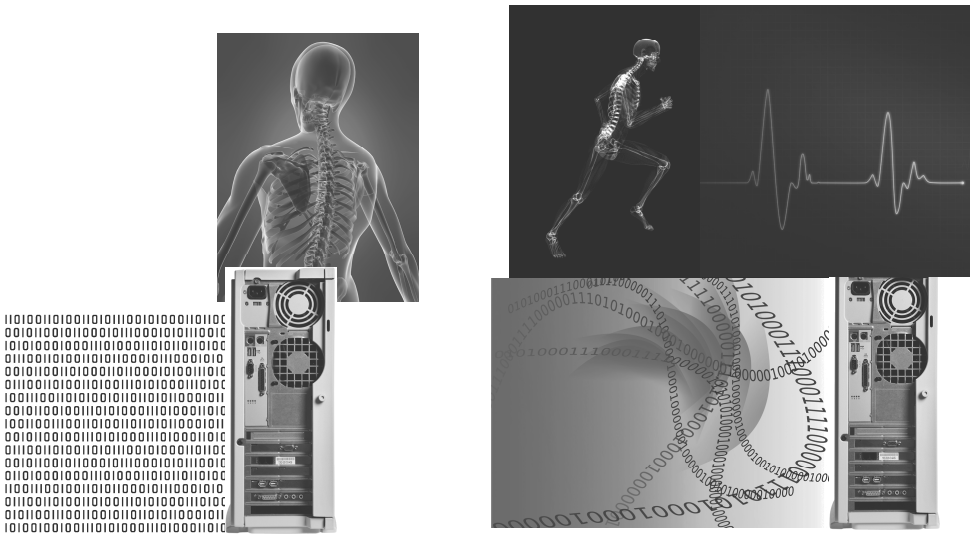

Figure 9.6: The difference between static and dynamic analysis: (a) Static analysis examines a system from its code and data, without running it. (b) Dynamic analysis examines an active, running system.

Static Analysis

As mentioned above, static analysis refers to the process of analyzing a system without actually executing the targeted software. Static analysis typically includes analysis of source code or binary code.

Source-Code Auditing

Source-code auditing is the process of carefully examining the source code of a target application in an attempt to uncover security vulnerabilities and other software bugs. Source-code auditing may take place at any point in the software development life cycle—sometimes code is audited early in the development of an application, while other times an audit is performed in response to security problems found in a released product.

Source-code auditing requires a highly specialized skill set that is not necessarily the same as that of a traditional software developer. There are a wide variety of different strategies used in structuring an audit, and these strategies should be carefully chosen based on the nature of the targeted software, the amount of code being audited, and time constraints. In general, auditors start by familiarizing themselves with the basic functions of the target application before examining its code. In some situations, an auditor may choose to identify security-critical areas to focus on. In other circumstances, an auditor may identify locations in a program that accept external input and trace through a program's code following that input. Once a potential vulnerability has been identified, source-code auditors will typically utilize dynamic analysis techniques to verify its exploitability.

Several automated analysis tools have been developed to assist auditors in identifying vulnerable source code. The simplest of these tools just searches code for potentially unsafe functions that are frequently used incorrectly, and identifies code patterns that may result in a security vulnerability. For example, copying data into a fixed-length buffer may be flagged as a potential buffer overflow (Section 3.4.3). More sophisticated scanners use complex heuristics to analyze data flow and ensure expected behavior.

Such tools may be very effective at identifying some types of vulnerabilities, such as memory corruption issues or other bugs that typically stem from improper function use or low-level syntactical issues. Other classes of bugs related to high-level design issues or unexpected behavior may go undetected, however. In fact, it has been proven that finding all possible errors in a given program is computationally undecidable. Static analysis tools attempt to provide useful approximate solutions to this problem, but will never be able to guarantee the security of a program.

Binary Auditing

In many situations, the source code of a targeted application is unavailable. For example, a vulnerability research lab may wish to audit the security of a closed-source commercial software without the cooperation of the vendor. In other cases, auditors are hired by software companies to perform black-box testing of their software, restricting access to source code. In these scenarios, auditors must use tools and techniques designed to analyze ***binary code***, usually with the help of a ***disassembler***—an application that can interpret compiled ***machine code*** into human-readable ***assembly language*** for analysis.

The process of investigating the inner workings of a compiled program is known as ***reverse engineering***. Many of the same techniques developed for source-code auditing can be applied to reverse engineering, with additional complexity introduced due to the potentially poor readability and complexity of assembly code.

Dynamic Analysis

Dynamic analysis is a method of vulnerability assessment that involves actually running live software to uncover flaws.

Most often, this type of analysis is done with the assistance of a ***debugger***—a piece of software that allows a developer or auditor to carefully control a program's execution at a low level, including the ability to manipulate a process's address space manually or step through a program's execution one instruction at a time.

By supplementing static analysis, such as code review and reverse engineering, with dynamic analysis techniques, auditors can identify potentially vulnerable situations, provide input triggering the desired situation, and trace execution of the program step by step.

More recently, virtual machine technology is being used by auditors performing dynamic analysis. Virtual machines provide the ability to create a snapshot capturing the exact state of an operating system and all its programs. During dynamic analysis, an auditor can create a snapshot before testing an attack scenario. After completing this analysis, the auditor can revert the virtual machine to the state contained in the snapshot, to guarantee fully reproducible results in subsequent tests.

Fuzzing

At any point that an application receives input from an external source, there exists the possibility of introducing malicious code designed to exploit a vulnerability in the application. Collectively, these points are referred to as a program's ***attack surface***, and represent all of the locations in which the application has contact with unknown or uncontrolled factors. By providing malformed or otherwise unexpected input to test each of these points, an auditor may be able to identify situations in which the targeted application does not function as expected. In many cases, provoking this unexpected behavior is the first step in discovering vulnerabilities that may be used to completely compromise the application.

Fuzzing is a means of automating the process of injecting unexpected input into an application with the goal of uncovering exploitable vulnerabilities. Fuzzers typically produce input for the program and repeatedly run the program with each generated input, recording events such as crashes and error messages for future analysis. The most primitive fuzzers simply generate random streams of input to be provided to the target program. While this may uncover more obvious bugs, more sophisticated techniques must be used to uncover more subtle vulnerabilities. Fuzzers are often developed for specific programs, network protocols, or file formats. For example, a fuzzer may start with valid input specified by the auditor and selectively mutate portions of this input in an attempt to produce error conditions in the target application.

9.4.2 Exploit Development and Vulnerability Disclosure

An ***exploit*** is a piece of code specifically designed to take advantage of a software vulnerability to achieve a result unintended by the vulnerable program ranging from denial of service to escalation of privileges. Exploits are often developed by vulnerability researchers as a proof of concept to establish that a software bug is exploitable in practice. With the advent of automated network scanners for use in penetration testing, exploitation of software has become its own specialized industry. Network scanning companies frequently employ their own exploit developers, who specialize in writing robust, portable exploit code. Other companies purchase exploit code from independent developers. Network scanners and other exploitation frameworks make exploiting software as simple as selecting a target. Because of this, they have generated some controversy—while they can be invaluable tools for network security specialists conducting legitimate audits with permission, they can also be used by malicious parties.

Vulnerability Disclosure

In many circumstances, software audits are conducted by consultants or employees, and the results are handled internally, by either simply correcting the code or issuing a security patch to fix existing installations. Still, independent security researchers are not subject to the same restrictions, and have a degree of choice in how they choose to disclose security vulnerabilities to the general public.

The Ethics of Disclosure

Some security professionals are committed to the concept of ***responsible disclosure***, which advocates reporting security issues to software and hardware vendors, giving vendors an opportunity to release a patch before the issue becomes public. After such a patch becomes available, the vendor or researcher typically publicizes a security advisory or bulletin, with varying levels of detail regarding the vulnerability and how to mitigate its effects. Some larger software developers actually provide financial incentives to vulnerability researchers who responsibly report bugs to the vendor before disclosing them to the public.

Other vulnerability researchers believe that responsible disclosure does not hold software companies accountable for the quality and security of their products. These researchers advocate ***full disclosure***, which involves publicizing all details of a vulnerability immediately. Many consider this disclosure policy to be irresponsible, since it may inform malicious parties of vulnerabilities before giving vendors an opportunity to provide end users with patches and attack mitigation advice. Even so, it often results in much faster response times from vendors, who must react to the issue promptly to prevent widespread exploitation of their product.

To limit public disclosures of unpatched vulnerabilities, some software vendors attempt to suppress vulnerability disclosures by taking legal action against researchers. For example, reverse engineering and publishing the details of a product may constitute disclosure of trade secrets, which may be illegal. Attempts to silence security disclosures have often generated negative publicity for vendors, and may do little to actually prevent publication of security flaws. Other vendors may attempt to reduce public disclosure by including in software licenses acceptable use clauses that restrict testing and reverse engineering.

9.5 Administration and Auditing

Much of the responsibility for establishing secure computers and networks rests on system and network administrators. While software and operating system vulnerabilities are commonplace and difficult to predict in advance, administrators can implement precautions to minimize the impact of these flaws. In fact, many dangerous scenarios may arise from the misconfiguration of settings at the software or hardware level, both of which may lie within the responsibility of an administrator.

9.5.1 System Administration

Learning how to properly administer systems is an expansive topic on which several books have been written. Even when an administrator understands the intricacies of the individual components of a network, complex issues may arise due to the unpredictable interactions between these components. Rather than attempt to cover the details of system administration, we will explain how previously described security principles and techniques can be applied in a system administration context instead.

User Policies

Least privilege should be employed by restricting the rights of each user and system component to the bare minimum necessary for smooth operation. For example, ordinary users should not have access to the administrator account of a machine, except when absolutely necessary. Users should only have access to files necessary for their work, and users should not be able to install unnecessary software without permission. If a machine is running any services that are accessible to the public, these services should be run with the lowest level of privileges possible, to mitigate the effects of a potential compromise.

A sound user ***access control*** system should be established to set rules on who receives accounts and with what access. Procedures to grant and revoke accounts with varying levels of privileges should be created, and each account should have the appropriate restrictions granting only necessary privileges.

Proper use of encryption and ***strong passwords*** are essential in preventing unauthorized access by intruders. Users should be educated on password strength, and encouraged or required to change passwords regularly, especially in the event of a suspected intrusion. Network adminis-

trators may chose to run password-cracking programs proactively in order to detect and fix weak passwords. All sensitive communications should take place over encrypted channels, using appropriate encryption protocols that have been shown to be secure by experts. Use of "home-grown" cryptographic solutions is discouraged.

System Policies

Prompt and frequent ***patching*** is important to prevent compromise due to vulnerabilities in software. Exploitation of unpatched vulnerabilities comprise a large portion of intrusion scenarios, and could be easily prevented by implementing an efficient program to monitor and apply software updates in response to security announcements. Clear policies should be set regarding how updates are installed on user machines, and end users should not be relied upon to update their own software. Many tools are available to manage the propagation of software updates throughout an organization.

Policies should be set to create acceptable levels of ***physical security***. Decisions should be made as to whether or not ordinary users should have access to physical resources, such as servers, storage media, and network devices. Rules should be set regarding the use of removable media, such as USB flash drives. To prevent live CD attacks (Section 2.4.4), machines should be configured to boot only from the hard disk. The BIOS password, known only to system administrators, would have to be entered in order to boot from a CD or other external media. In high-security contexts, access to networking cables should be restricted to authorized parties.

Organizations should create policies that define ***acceptable use*** of internal computer systems. To limit organizational liability, companies should consider requiring employees to sign an agreement to these policies, which should be clearly written and readily accessible.

Network Policies

Administrators should minimize the ***attack surface*** of their networks by deploying a firewall (Section 6.2) and properly configuring it to allow the bare minimum of necessary traffic. Larger networks should be segmented such that any machines providing services to external users are placed in a DMZ. Machines for internal use only should be in a separate segment behind a firewall that manages the flow of information between the internal network and public Internet. Machines that do not need access to the Internet should be isolated. Administrators should minimize the number of

externally accessible services running on each machine to keep the number of open ports at a minimum.

Segmentation of the network into regions, each residing behind its own router or switch, can minimize the impact of an intrusion by restricting the intruder to a limited set of resources. Trusted and untrusted machines should be located on separate segments to minimize exposure of internal resources to potentially malicious parties. (See Figure 9.7.) Administrators should keep track of all machines and devices connected to the network. By monitoring MAC addresses and logging activity, administrators can detect and defend against unauthorized access attempts.

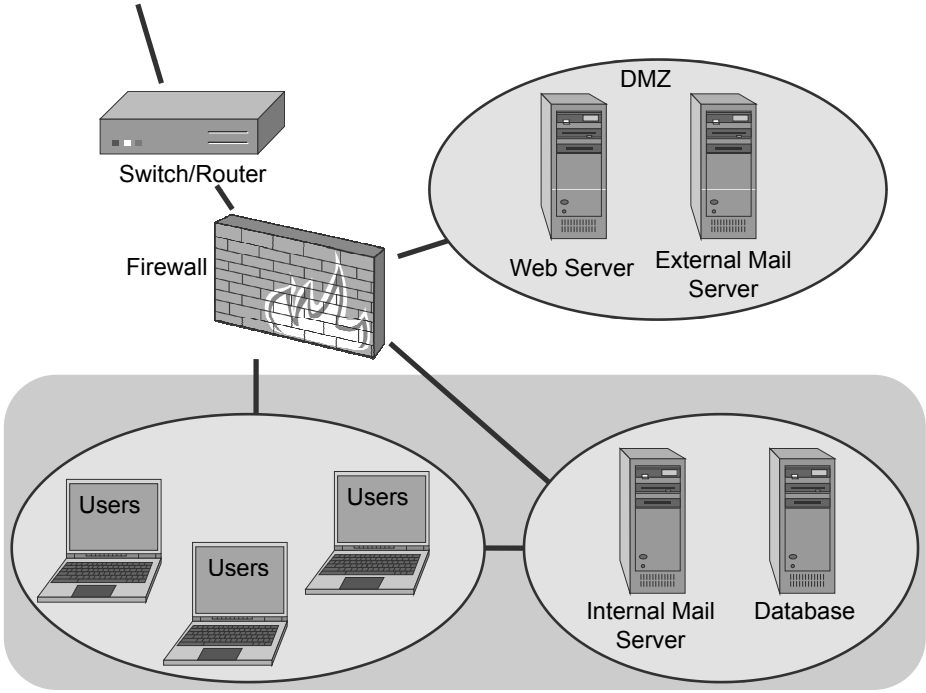

Figure 9.7: Example of segmentation of a network into regions.

A ***mail infrastructure*** should be created that protects internal users from spam, phishing attempts, and malware. (See Section 10.2.) Positioning the mail server within a DMZ and performing virus scanning and spam filtering before mail reaches the internal network is advisable.

Network administrators should regularly conduct audits to ensure compliance with regulations throughout an organization. Policies should define who conducts these audits, what criteria are to be observed, and how often they must be performed.

9.5.2 Network Auditing and Penetration Testing

There are many approaches to testing the security of a network. Network security audits may be performed to test a network for compliance with a set of standards, ranging from internal policies to federally mandated regulations. The scope of an audit can vary dramatically, depending on which regulations are to be tested.

One of the most common targets of an audit is an organization's password policy. Examples of questions that should be answered by such a policy include the following:

- In what situations are passwords used?
- What steps are being taken to ensure that users make use of strong passwords?
- Is there a single sign-on system in place or are users responsible for multiple passwords?
- Is there an account lockout policy in place in the event of repeated failed attempts to authenticate?
- In what circumstances can users reset or recover lost passwords and how is this performed?

A thorough audit will answer these questions and assess whether or not the organization is in compliance with appropriate standards.

Penetration Testing

A ***penetration test*** is a hands-on audit that aims to simulate an attack by an actual intruder. As with source-code auditing, penetration tests grant varying levels of visibility to auditors, ranging from black-box to white-box testing. Penetration tests can potentially be disruptive to an organization, because they may involve exploitation of vulnerabilities and accidental denial of service to users. Therefore, it is critical that auditors conducting a penetration test have explicit permission from appropriate parties within the organization, and clearly define what types of attacks and side effects are considered acceptable. Without explicit agreement from authorized parties, a penetration tester may be held liable for any damage incurred during a test. Some penetration tests are comprehensive and allow auditors to perform social engineering attacks (Section 1.4.3) and attempt physical intrusion (Chapter 2), while others are more limited in scope.

Penetration testers typically follow a strict methodology that defines exactly how the test is to be conducted. Such methodologies will vary depending on the scope of the audit and who is performing it, but the penetration testing process can typically be divided into three broad phases.

- First, an auditor must gain as much information as possible about the topology of the target network, a phase known as ***network discovery*** or ***host enumeration***. Auditors can determine which hosts are accessible via the Internet using techniques such as ***ping sweeping*** (issuing ping commands to ranges of IP addresses and recording responses) and by investigating domain-name registration information and DNS resolution. Additional topology information can be gathered by using tools such as traceroute, which returns information about each host along the path to a target. Ideally, the discovery phase should allow the auditor to determine which hosts may be promising targets in later stages of testing.

- After this information-gathering phase, most testers begin a second phase focusing on ***network vulnerability analysis***. During this stage, the tester may conduct port scans (Section 6.4.4) to determine which hosts have open ports. Fingerprinting techniques may be used to determine which operating systems are in use and which applications are accessible remotely. In addition, the auditor may begin researching existing vulnerabilities in these applications—if a vulnerable host is present, that may allow the auditor to gain access to that host and conduct additional attacks against the network.

- The final phase of a typical penetration test involves actually ***exploiting known vulnerabilities*** and attempting to gain access to internal resources. On a successful intrusion, testers may conduct additional information gathering to continue mapping out the network topology and identify additional targets. Often, it is necessary for an auditor to leverage a compromised system to gain additional access within a network.

Throughout the penetration testing process, auditors must keep detailed documentation describing what information was discovered, which techniques were used to attack the target network, and any vulnerabilities or misconfigurations that allowed the auditor to gain access to restricted resources. On completion of a penetration test, the auditor must provide this information to network administrators and suggest mitigation measures that would defend against future exploitation attempts.

9.6 Kerberos

Kerberos is an authentication protocol and a software suite implementing this protocol. Kerberos uses symmetric cryptography to authenticate clients to services and vice versa. For example, Windows servers use Kerberos as the primary authentication mechanism, working in conjunction with Active Directory to maintain centralized user information. Other possible uses of Kerberos include allowing users to log into other machines in a local-area network, authentication for web services, authenticating email client and servers, and authenticating the use of devices such as printers. Services using Kerberos authentication are commonly referred to as "Kerberized".

9.6.1 Kerberos Tickets and Servers

Kerberos uses the concept of a ***ticket*** as a token that proves the identity of a user. Tickets are digital documents that store session keys. They are typically issued during a login session and then can be used instead of passwords for any Kerberized services. During the course of authentication, a client receives two tickets:

- A ***ticket-granting ticket*** (***TGT***), which acts as a global identifier for a user and a session key
- A ***service ticket***, which authenticates a user to a particular service

These tickets include time stamps that indicate an expiration time after which they become invalid. This expiration time can be set by Kerberos administrators depending on the service.

To accomplish secure authentication, Kerberos uses a trusted third party known as a ***key distribution center*** (***KDC***), which is composed of two components, typically integrated into a single server:

- An ***authentication server*** (***AS***), which performs user authentication
- A ***ticket-granting server*** (***TGS***), which grants tickets to users

The authentication server keeps a database storing the secret keys of the users and services. The secret key of a user is typically generated by performing a one-way hash of the user-provided password. Kerberos is designed to be modular, so that it can be used with a number of encryption protocols, with AES (Section 8.1.6) being the default cryptosystem. Kerberos aims to centralize authentication for an entire network—rather than storing sensitive authentication information at each user's machine, this data is only maintained in one presumably secure location. Even in the event of a compromise of the KDC, the users' plaintext passwords will remain secret, since an attacker would only recover the passwords' hashes.

9.6.2 Kerberos Authentication

Kerberos is based on a protocol designed by Needham and Schroeder in 1978 for authentication using symmetric encryption. When a user wishes to access services, the following steps are performed. (See Figure 9.8.)

1. The user provides a username and password on the client machine, which is cryptographically hashed to form the secret key for the client.
2. The client contacts the AS, which replies with the following items:
   - The ***client-TGS session key***, K_{CT}, encrypted using the client's secret key, K_C (which the AS has stored in its database).
   - The ***ticket-granting ticket*** (***TGT***), encrypted with the secret key of the TGS, K_T (also stored in the AS database). The TGT includes key K_{CT} and a validity period.
3. The client decrypts the TGS session key K_{CT} using K_C. To request a service, the client sends the following two messages to the TGS:
   - The TGT (still encrypted using the TGS's secret key, K_T) and the name, S, of the service being requested.
   - An ***authentication token*** consisting of the client ID and time stamp, encrypted using the client-TGS session key K_{CT}.
4. The TGS decrypts the TGT using K_T, thus retrieving the client-TGS session key K_{CT} and the validity period of the TGT. If the current time is within the validity period, the TGS decrypts the authentication token with key K_{CT} and sends two messages to the client:
   - A new ***client-server session key***, K_{CS}, encrypted with K_{CT}.
   - A ***client-to-server ticket***, encrypted using the specific service's secret key, K_S, which is known to the TGS. This ticket contains the client ID, network address, validity period, and key K_{CS}.
5. After decrypting the client-server session keyK_{CS}, the client authenticates itself to service S by sending the following two messages:
   - The client-to-server ticket, sent by the TGS in the previous step.
   - The client ID and time stamp, encrypted with K_{CS}.
6. The service decrypts the client-to-server ticket using its secret key K_S and obtains the client-server session key K_{CS}. Using K_{CS}, it decrypts the client ID and time stamp. Finally, to prove its identity to the client, it increments the time stamp by 1 and sends it back to the client reencrypted with K_{CS}.
7. The client decrypts and verifies this response using K_{CS}. If the verification succeeds, the client-server session can begin .

Steps 3–7 of the protocol can be repeated by the client to access multiple services within the validity period of the ticket-granting ticket.

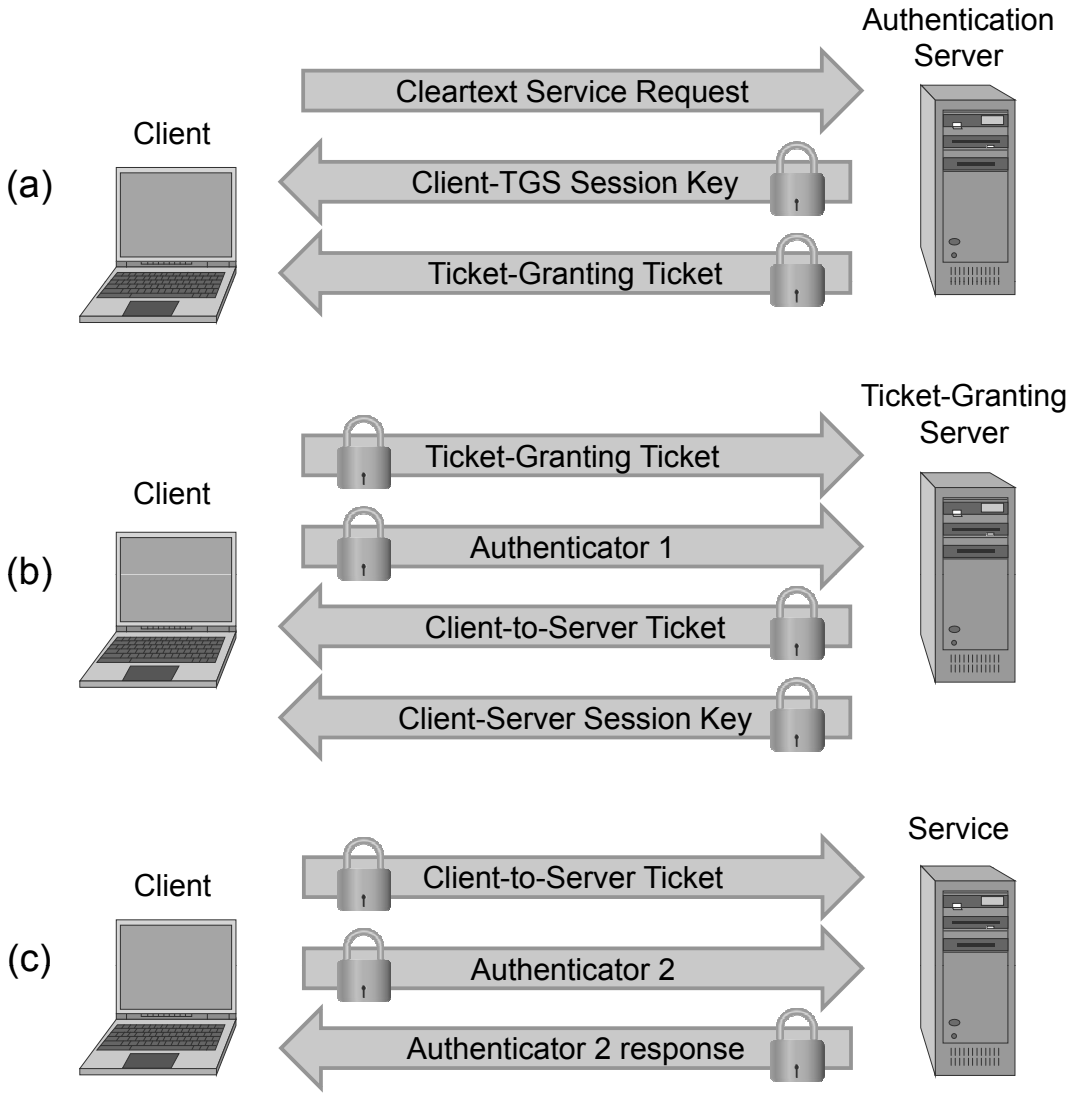

Figure 9.8: Kerberos authentication: (a) The client and authentication server authenticate themselves to each other. (b) The client and ticket-granting server authenticate themselves to each other. (c) The client and requested service authenticate themselves to each other, at which point the service will be provided to the client.

Kerberos Advantages

Because of its distributed architecture, the Kerberos protocol is designed to be secure even when performed over an insecure network. Since each transmission is encrypted using an appropriate secret key (except the initial plaintext request to the authentication server, which contains no secret information), an attacker cannot forge a valid ticket to gain unauthorized access to a service without compromising an encryption key or breaking the underlying encryption algorithm, which is assumed to be secure. The integrity of each message can also be further protected from tampering by including a cryptographic message authentication code, created with the appropriate session key with each transmission.

Kerberos is also desiged to protect against replay attacks, where an attacker eavesdrops legitimate Kerberos communications and retransmits messages from an authenticated party to perform unauthorized actions. The inclusion of time stamps in Kerberos messages restricts the window in which an attacker can retransmit messages. In addition, tickets may contain the IP addresses associated with the authenticated party to prevent replaying messages from a different IP address. Finally, Kerberized services make use of a "replay cache," which stores previous authentication tokens and detects their reuse. Collectively, these measures provide strong protection against known types of replay attacks.

Additional advantages of Kerberos include the use of symmetric encryption instead of public-key encryption, which makes Kerberos computationally efficient, and the availability of an open-source implementation, which has facilitated the adoption of Kerberos.

Kerberos Disadvantages

While Kerberos provides strong security, it has some drawbacks. Most notably, Kerberos has a single point of failure: if the Key Distribution Center becomes unavailable, the authentication scheme for an entire network may cease to function. Larger networks sometimes prevent such a scenario by having multiple KDCs, or having backup KDCs available in case of emergency. In addition, if an attacker compromises the KDC, the authentication information of every client and server on the network would be revealed. Finally, Kerberos requires that all participating parties have synchronized clocks, since time stamps are used. While these weaknesses should be considered before deploying Kerberos, they have not prevented the widespread adoption of Kerberos as a strong authentication protocol.

9.7 Secure Storage

As discussed in Chapter 2, it is difficult to defend a computer system against an attacker who has physical access to that system. Nevertheless, this scenario occurs more often than one might think. For example, an estimated 12,000 laptops are lost or stolen in U.S. airports every week. Besides the obvious cost of replacing equipment, lost laptops generate a significant expense for an organization due to the serious risk of data breach. Research suggests that the average cost of a lost laptop to a corporation is around $50,000, mostly due to costs associated with intellectual property loss, forensics, lost productivity, and legal expenses, not the hardware itself. To help mitigate this problem, a number of technologies have been developed to protect the confidentiality of data on computer systems, even in the event of physical compromise.

9.7.1 File Encryption

Password Protection of Files

One approach to protecting sensitive information is to perform encryption on individual files. Many popular software suites, including Microsoft Office and Adobe Acrobat, allow users to protect their documents by encrypting their contents. Early file encryption solutions, such as the password protection provided by early versions of Microsoft Office, were designed to withstand casual attempts at data compromise, as they used naive encryption solutions such as a simple XOR algorithm.

Modern file encryption, on the other hand, is designed to be resilient against determined attackers. For example, both Microsoft Office 2007 and Adobe Acrobat 9 make use of the AES block cipher for encryption. Office derives a secret key by iteratively hashing a user-provided password 50,000 times with SHA-1. Repeatedly hashing the password does not provide increased cryptographic security, but rather is designed to slow down brute-force attempts by requiring each password guess to perform a time-consuming computation. In comparison, Acrobat 9 uses the SHA-256 algorithm, which is considered stronger than SHA-1, but it only hashes the user-supplied password once to derive a secret key. The effects of this difference can be observed in practice. A password-recovery tool known as Elcomsoft advertises that it can achieve 5,000 password attempts per second with Office 2007, as opposed to 75 million per second for Acrobat 9.

Filesystem Encryption

The ***Encrypting File System*** (***EFS***) is an example of a filesystem-level encryption scheme that is available on recent versions of the Microsoft Windows operating system. EFS works by transparently providing automatic encryption and decryption of specified files and folders, such that if an attacker gained physical access to a machine, these files would be indecipherable. Files and folders must be specifically tagged for use with EFS; by default, all files are left unencrypted.

EFS uses both symmetric and asymmetric cryptography. For performance reasons, each file is encrypted with a separate symmetric ***file-encryption key*** (***FEK***), using AES. The FEK used to encrypt the data is then encrypted using the user's public key and stored in the file's metadata. To decrypt the file, the FEK is decrypted using the user's private key, and is then used to decrypt the data. To support sharing among users, multiple copies of the FEK can be included in the encrypted file, each encrypted with a different user's public key. (See Figure 9.9.) In addition, to ensure that data can be recovered in the event of a forgotten password or lost private key, ***data recovery agnets*** (***DRAs***) can be identified by administrators as parties authorized to decrypt all EFS encrypted files.

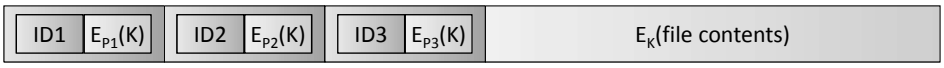

Figure 9.9: Format of a file encrypted with Window's EFS. The FEK, denoted with K, is encrypted with the public keys of the users sharing the file.

Security Challenges with EFS

A number of security issues have been identified for EFS. First, only the contents of files are encrypted, so information such as file names and other metadata is not protected. Secondly, encryption is only applied on EFS enabled filesystems, so transferring files to other filesystems may result in accidental decryption. Similarly, file contents may be exposed via unprotected temporary files.

By default, EFS private keys for the users are stored on disk after being encrypted using a salted hash of the user's Windows password. Therefore, if an attacker can recover a user's password, their private key can be decrypted, resulting in the compromise of any EFS encrypted files. In addition, if the accounts of any users designated as DRAs can be compromised, then an attacker will gain the ability to decrypt all files.

9.7.2 Disk Encryption

Rather than encrypting individual files or folders, it may be desirable to encrypt entire physical or logical disks. Two of the most popular disk encryption solutions are ***BitLocker***, available on Windows Vista and 7, and the open source ***TrueCrypt***.

TrueCrypt

TrueCrypt is a full-disk encryption technology that is designed to protect disk contents from compromise by an adversary who has obtained physical access. TrueCrypt can create a virtual encrypted disk within a file and mount it as if it were a physical drive. Using this setting in Windows, a TrueCrypt file becomes a volume in Windows Explorer with a drive letter, just as though an external drive was mounted. TrueCrypt can also encrypt an entire partition or storage device. TrueCrypt encrypts each sector in the volume and supports a number of strong symmetric encryption algorithms, including AES. Note that typical disk sector sizes are powers of two in the range 512B through 8,192B. Encryption and decryption are performed automatically by TrueCrypt and are transparent to the user. However, the TrueCrypt password is independent from the login password and must be entered by the user when a TrueCrypt file is mounted as a drive.

The ability to deny the presence of data hidden within a computer upon its examination by an adversary is known as ***plausible deniability***. In situations where an attacker has the means to force the owner of a computer system to reveal decryption keys for known encrypted volumes via means such as extortion, threats, or even torture, being able to deny the existence of informataion could protect valuable data.

TrueCrypt attempts to provide plausible deniability by allowing users to create hidden encrypted volumes that are designed to be undetectable to an adversary obtaining physical access. Hidden volumes are created by placing a TrueCrypt-encrypted volume within the free space of another TrueCrypt volume, without modifying any of the outer volume's metadata. With TrueCrypt, all free space is initialized with random data, so the existence of a hidden encrypted volume, which is indistinguishable from random data, is impossible to prove. When confronted with an adversary demanding the password to decrypt TrueCrypt volumes, the password for the outer volume can be revealed, knowing that the existence of the hidden volume will remain undetected. To decrypt this hidden volume, a user instructs TrueCrypt to attempt to detect a hidden volume encrypted using a given password. If the corret password is given and there is in fact a hidden volume, a TrueCrypt volume header will be properly decrypted

and verified, giving that user access to the data. If an incorrect password is given or there is no hidden volume, TrueCrypt will fail to decrypt a valid header and indicate that a hidden volume could not be found.

While the design of hidden volumes in TrueCrypt is sound, the operating system or applications that access files in a hidden volume may leave traces of the use of the hidden volume, thus compromising plausible deniability. Examples include shortcuts to recently open files in Windows, temporary backup files created by Microsoft Office applications for crash recovery, and indices and snapshot files created by Google Desktop.

BitLocker

Some versions of Windows provide a disk-encryption technology known as BitLocker. Like TrueCrypt, BitLocker encrypts disk sectors with symmetric encryption, specifically AES. To decrypt a volume, the user has several options. A password can be provided at boot time via keyboard, or a decryption key can be loaded from a USB device or a ***Trusted Platform Module*** (***TPM***), which we discuss below.

BitLocker makes use of two NTFS formatted volumes, one containing the operating system and data that is to be encrypted, and another to be used as an unencrypted boot volume. When the user authenticates at boot time, the ***volume master key*** is unlocked. Using this key, BitLocker decrypts the ***full-volume encryption key***, which is stored encrypted on the boot volume. This key is then kept in memory and used to decrypt the data on the encrypted volume.

9.7.3 Trusted Platform Module

The ***Trusted Platform Module*** (***TPM***) is a chip designed to be mounted on the motherboard for use as a secure cryptoprocessor that can securely generate and store cryptographic keys. Each TPM chip has a unique RSA private key burned into the hardware at the time of production. The TPM is designed to be tamper-resistant, so this key is hard to recover by attackers with physical access.

TPM chips feature several ***platform configuration registers*** (***PCRs***), which are used to store keys and ciphertexts for several cryptographic operations.

- The extend operation updates the value of a specified PCR with a cryptographic hash of the previous value of that PCR concatenated with data provided to the operation.

- The seal operation encrypts a supplied plaintext with the TPM private key and associates it with the current contents of a specified PCR. The operation returns the ciphertext, as well as a MAC computed from the current value of the specified PCR and the TPM private key.
- Given a ciphertext, a hash value, and the name of a PCR, the unseal operation decrypts the ciphertext only if computing the MAC of the current value of the PCR yields the given hash value.

Collectively, these operations allow hardware and software components, including the BIOS, bootloader, operating system, and applications, to bind secret data to the TPM that can only be extracted if the state of the machine is identical to when the data was stored.

For example, BitLocker can use the TPM as a means of guaranteeing the integrity of trusted operating system components before decrypting the contents of the hard drive. First, the TPM is initialized by performing the extend operation to initialize specific PCRs to capture the desired state of the BIOS, bootloader, kernel, and other trusted components. Next, the volume master key is sealed to the values of these PCRs and stored. On booting, the operating system repeats these extension operations, and attempts to unseal the key to decrypt the BitLocker drives. If any of the trusted system components have been altered, the state of the PCRs will differ from when the seal operation was performed, and the TPM will not unseal the volume master key. The TPM can also be used for a number of other cryptographic applications, including digital-rights management (Sections 10.4.1–10.4.2) and software licensing (Section 10.4.3).

Using at TPM to store the volume master key increases the usability of BitLocker since the user does not have to enter a password or insert a USB token. However, this mode of operation for BitLocker is vulnerable to the cold boot attacks described in Section 2.4.5.

Also, while the TPM is designed to be impervious to physical tampering, an attack was presented in 2010 by security researcher Christopher Tarnovsky. In his attack, Tarnovsky applied acid and rust remover to remove the outer shell and several layers of mesh wiring, exposing the chip's core. Then, by carefully using a microprobe to tap communication channels in the chip's core, he was able to extract CPU instructions and recover protected information from the TPM. While this attack may suggest that the TPM may not withstand determined attackers with physical access, Tarnovsky's approach was highly technical and required many months of patient work. As such, it may be infeasible for even sophisticated attackers to compromise the chip, but Tarnovsky's attack may provide the groundwork needed to make defeating the TPM more feasible in the future.

9.8 Exercises

For help with exercises, please visit **securitybook.net**.

Reinforcement

R-9.1 What are the five components of a security policy?

R-9.2 Describe the differences between discretionary and mandatory access-control policies.

R-9.3 What are the four components of a trust management system?

R-9.4 Consider a variation of the Bell La Padula model that does not have the *-property. Which security vulnerabilities arise?

R-9.5 What are the components of a total order and which one is missing in the definition of a partial order?

R-9.6 Compare and contrast the Biba model and the BLP model.

R-9.7 What is the difference between the Chinese wall model and the Brewer and Nash model.

R-9.8 Explain the difference between white-box and black-box assessments.

R-9.9 What types of records are protected under HIPAA? What about FERPA?

R-9.10 Describe how network segmentation might be used by system administrators to provide additional security.

R-9.11 What are some of the advantages of dynamic-analysis techniques over static-analysis techniques?

R-9.12 What are the types of tickets and servers used in Kerberos?

Creativity

C-9.1 Draw a diagram for a partial order and show that there are at least two total orders that include the same relationships.

C-9.2 UFO enthusiasts believe there might be as many as 38 classification levels above "top secret," which are so secret that people without those clearances can't even know their names. Give a reason why such classifications might be necessary for handling UFO related information and give some plausible names for such classification levels.

C-9.3 Describe an application suitable for the BLP model but not for the RBAC model.

C-9.4 Describe a situation where security levels for conflicts of interest would be important.

C-9.5 Compare the BLP model with the RBAC model.

C-9.6 Design a data structure for representing a hierarchical RBAC system and describe the algorithm for checking whether a user can access a resource. Analyze the space used by the data structure and the running time of the algorithm.

C-9.7 Briefly describe your own security standard for computer systems. What properties are most important? How can these security properties be regulated and monitored?

C-9.8 If an administrator discovers a vulnerability in his or her system, who should he tell? Should he make the vulnerability public? Why or why not?

C-9.9 ***White-hat*** testing is a set of vulnerability tests that are designed to be used by system administrators to uncover system vulnerabilities so that they can be fixed. Describe some white-hat system and network tests and describe some specific vulnerabilities that they are designed to discover.

C-9.10 Why does Kerberos need two types of tickets and two types of servers?

Projects

P-9.1 Implement a system for controlling access to a collection of web pages based on the BLP model.

P-9.2 Implement an RBAC system for controlling access to the pages of a web site.

P-9.3 Write a simple static-analysis tool for detecting potential vulnerabilities in source code.

P-9.4 With permission, conduct a simulated penetration test on a virtual machine network. Develop a full methodology, perform the audit, and present formal results.

P-9.5 Choose a piece of open source software with published vulnerabilities. After downloading the source code, identify the vulnerable code and develop a security advisory describing the bug, its severity, and other relevant information.

Chapter Notes

Most of the standards, specifications, and formal documents described in this chapter are available online:

- TCSEC: csrc.nist.gov/publications/history/dod85.pdf
- Common Criteria: www.commoncriteriaportal.org/thecc.html
- FIPS 140 csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
- HIPAA: www.hhs.gov/ocr/privacy
- FERPA: www.ed.gov/policy/gen/guid/fpco/ferpa
- Data Protection Directive: ec.europa.eu/justice_home/fsj/privacy
- SOX: pcaobus.org
- TPM: www.trustedcomputinggroup.org/resources/tpm_main_specification

They KeyNote trust management system, developed by Blaze, Feigenbaum, Ioannidis, and Keromytis, is described in RFC 2704. The Bell-La Padula model is described by its designers in a 1973 MITRE technical report [3]. Likewise, the Biba model is also described by its designer in a MITRE report [5]. The Brewer and Nash Chinese-wall model is presented in a 1989 paper [14]. Our description of the RBAC model follows the paper by Ferraiolo *et al.* [32] (see also the book by by Ferraiolo *et al.* [31]). The Kerberos protocol is based on a classic paper by Needham and Schroeder on establishing secure communication between two parties who share secret keys with a trusted third party [64]. For additional information on the concepts and implementation details behind Kerberos, see the book by Garman [35]. The Ponemon Institute (ponemon.org) has studied the cost of data breaches, including those caused by lost laptops. Information leakage by the operating system or applications that may compromise TrueCrypt hidden volumes has been investigated by Czekis *et al.* [21].