Data Integrity

As with confidentiality, **integrity** can apply to a stream of messages, a single message, or selected fields within a message. Again, the most useful and straightforward approach is total stream protection.

A connection-oriented integrity service deals with a stream of messages and assures that messages are received as sent with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service deals with individual messages without regard to any larger context and generally provides protection against message modification only.

We can make a distinction between service with and without recovery. Because the integrity service relates to active attacks, we are concerned with detection rather than prevention. If a violation of integrity is detected, then the service may simply report this violation, and some other portion of software or human intervention is required to recover from the violation. Alternatively, there are mechanisms available to recover from the loss of integrity of data, as we will review subsequently. The incorporation of automated recovery mechanisms is typically the more attractive alternative.

Nonrepudiation

Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

Availability Service

Both X.800 and RFC 4949 define **availability** to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them). A variety of attacks can result in the loss of or reduction in availability. Some of these attacks are amenable to automated countermeasures, such as authentication and encryption, whereas others require some sort of physical action to prevent or recover from loss of availability of elements of a distributed system.

X.800 treats availability as a property to be associated with various security services. However, it makes sense to call out specifically an availability service. An availability service is one that protects a system to ensure its availability. This service addresses the security concerns raised by denial-of-service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

1.5 SECURITY MECHANISMS

Table 1.3 lists the security mechanisms defined in X.800. The mechanisms are divided into those that are implemented in a specific protocol layer, such as TCP or an application-layer protocol, and those that are not specific to any particular protocol layer or security service. These mechanisms will be covered in the appropriate places in the book, so we do not elaborate now except to comment on the

Table 1.3 Security Mechanisms (X.800)

SPECIFIC SECURITY MECHANISMS	PERVASIVE SECURITY MECHANISMS
May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.	Mechanisms that are not specific to any particular OSI security service or protocol layer.
Encipherment The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.	**Trusted Functionality** That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).
Digital Signature Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).	**Security Label** The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.
Access Control A variety of mechanisms that enforce access rights to resources.	**Event Detection** Detection of security-relevant events.
Data Integrity A variety of mechanisms used to assure the integrity of a data unit or stream of data units.	**Security Audit Trail** Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
Authentication Exchange A mechanism intended to ensure the identity of an entity by means of information exchange.	**Security Recovery** Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.
Traffic Padding The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.	
Routing Control Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.	
Notarization The use of a trusted third party to assure certain properties of a data exchange.	

Table 1.4 Relationship between Security Services and Mechanisms

Service	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
	Mechanism							
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

definition of encipherment. X.800 distinguishes between reversible encipherment mechanisms and irreversible encipherment mechanisms. A reversible encipherment mechanism is simply an encryption algorithm that allows data to be encrypted and subsequently decrypted. Irreversible encipherment mechanisms include hash algorithms and message authentication codes, which are used in digital signature and message authentication applications.

Table 1.4, based on one in X.800, indicates the relationship between security services and security mechanisms.

1.6 FUNDAMENTAL SECURITY DESIGN PRINCIPLES

Despite years of research and development, it has not been possible to develop security design and implementation techniques that systematically exclude security flaws and prevent all unauthorized actions. In the absence of such foolproof techniques, it is useful to have a set of widely agreed design principles that can guide the development of protection mechanisms. The National Centers of Academic Excellence in Information Assurance/Cyber Defense, which is jointly sponsored by the U.S. National Security Agency and the U.S. Department of Homeland Security, list the following as fundamental security design principles [NCAE13]:

- Economy of mechanism
- Fail-safe defaults
- Complete mediation

- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Isolation
- Encapsulation
- Modularity
- Layering
- Least astonishment

The first eight listed principles were initially proposed in [SALT75] and have withstood the test of time. In this section, we briefly discuss each principle.

Economy of mechanism means that the design of security measures embodied in both hardware and software should be as simple and small as possible. The motivation for this principle is that relatively simple, small design is easier to test and verify thoroughly. With a complex design, there are many more opportunities for an adversary to discover subtle weaknesses to exploit that may be difficult to spot ahead of time. The more complex the mechanism, the more likely it is to possess exploitable flaws. Simple mechanisms tend to have fewer exploitable flaws and require less maintenance. Further, because configuration management issues are simplified, updating or replacing a simple mechanism becomes a less intensive process. In practice, this is perhaps the most difficult principle to honor. There is a constant demand for new features in both hardware and software, complicating the security design task. The best that can be done is to keep this principle in mind during system design to try to eliminate unnecessary complexity.

Fail-safe default means that access decisions should be based on permission rather than exclusion. That is, the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. This approach exhibits a better failure mode than the alternative approach, where the default is to permit access. A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation that can be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure that may long go unnoticed in normal use. For example, most file access systems work on this principle and virtually all protected services on client/server systems work this way.

Complete mediation means that every access must be checked against the access control mechanism. Systems should not rely on access decisions retrieved from a cache. In a system designed to operate continuously, this principle requires that, if access decisions are remembered for future use, careful consideration should be given to how changes in authority are propagated into such local memories. File access systems appear to provide an example of a system that complies with this principle. However, typically, once a user has opened a file, no check is made to see if permissions change. To fully implement complete mediation, every time a user

reads a field or record in a file, or a data item in a database, the system must exercise access control. This resource-intensive approach is rarely used.

Open design means that the design of a security mechanism should be open rather than secret. For example, although encryption keys must be secret, encryption algorithms should be open to public scrutiny. The algorithms can then be reviewed by many experts, and users can therefore have high confidence in them. This is the philosophy behind the National Institute of Standards and Technology (NIST) program of standardizing encryption and hash algorithms and has led to the widespread adoption of NIST-approved algorithms.

Separation of privilege is defined in [SALT75] as a practice in which multiple privilege attributes are required to achieve access to a restricted resource. A good example of this is multifactor user authentication, which requires the use of multiple techniques, such as a password and a smart card, to authorize a user. The term is also now applied to any technique in which a program is divided into parts that are limited to the specific privileges they require in order to perform a specific task. This is used to mitigate the potential damage of a computer security attack. One example of this latter interpretation of the principle is removing high privilege operations to another process and running that process with the higher privileges required to perform its tasks. Day-to-day interfaces are executed in a lower privileged process.

Least privilege means that every process and every user of the system should operate using the least set of privileges necessary to perform the task. A good example of the use of this principle is role-based access control, described in Chapter 4. The system security policy can identify and define the various roles of users or processes. Each role is assigned only those permissions needed to perform its functions. Each permission specifies a permitted access to a particular resource (such as read and write access to a specified file or directory, connect access to a given host and port, etc.). Unless a permission is granted explicitly, the user or process should not be able to access the protected resource. More generally, any access control system should allow each user only the privileges that are authorized for that user. There is also a temporal aspect to the least privilege principle. For example, system programs or administrators who have special privileges should have those privileges only when necessary; when they are doing ordinary activities, the privileges should be withdrawn. Leaving them in place just opens the door to accidents.

Least common mechanism means that the design should minimize the functions shared by different users, providing mutual security. This principle helps reduce the number of unintended communication paths and reduces the amount of hardware and software on which all users depend, thus making it easier to verify if there are any undesirable security implications.

Psychological acceptability implies that the security mechanisms should not interfere unduly with the work of users, while at the same time meeting the needs of those who authorize access. If security mechanisms hinder the usability or accessibility of resources, then users may opt to turn off those mechanisms. Where possible, security mechanisms should be transparent to the users of the system or at most introduce minimal obstruction. In addition to not being intrusive or burdensome, security procedures must reflect the user's mental model of protection. If the protection procedures do not make sense to the user or if the user must translate his image of protection into a substantially different protocol, the user is likely to make errors.

Isolation is a principle that applies in three contexts. First, public access systems should be isolated from critical resources (data, processes, etc.) to prevent disclosure or tampering. In cases where the sensitivity or criticality of the information is high, organizations may want to limit the number of systems on which that data is stored and isolate them, either physically or logically. Physical isolation may include ensuring that no physical connection exists between an organization's public access information resources and an organization's critical information. When implementing logical isolation solutions, layers of security services and mechanisms should be established between public systems and secure systems responsible for protecting critical resources. Second, the processes and files of individual users should be isolated from one another except where it is explicitly desired. All modern operating systems provide facilities for such isolation, so that individual users have separate, isolated process space, memory space, and file space, with protections for preventing unauthorized access. Finally, security mechanisms should be isolated in the sense of preventing access to those mechanisms. For example, logical access control may provide a means of isolating cryptographic software from other parts of the host system, and for protecting cryptographic software from tampering and the keys from replacement or disclosure.

Encapsulation can be viewed as a specific form of isolation based on object-oriented functionality. Protection is provided by encapsulating a collection of procedures and data objects in a domain of its own so that the internal structure of a data object is accessible only to the procedures of the protected subsystem, and the procedures may be called only at designated domain entry points.

Modularity in the context of security refers both to the development of security functions as separate, protected modules and to the use of a modular architecture for mechanism design and implementation. With respect to the use of separate security modules, the design goal here is to provide common security functions and services, such as cryptographic functions, as common modules. For example, numerous protocols and applications make use of cryptographic functions. Rather than implementing such functions in each protocol or application, a more secure design is provided by developing a common cryptographic module that can be invoked by numerous protocols and applications. The design and implementation effort can then focus on the secure design and implementation of a single cryptographic module and including mechanisms to protect the module from tampering. With respect to the use of a modular architecture, each security mechanism should be able to support migration to new technology or upgrade of new features without requiring an entire system redesign. The security design should be modular so that individual parts of the security design can be upgraded without the requirement to modify the entire system.

Layering refers to the use of multiple, overlapping protection approaches addressing the people, technology, and operational aspects of information systems. By using multiple, overlapping protection approaches, the failure or circumvention of any individual protection approach will not leave the system unprotected. We will see throughout this text that a layering approach is often used to provide multiple barriers between an adversary and protected information or services. This technique is often referred to as *defense in depth*.

Least astonishment means that a program or user interface should always respond in the way that is least likely to astonish the user. For example, the mechanism for authorization should be transparent enough to a user that the user has a good intuitive understanding of how the security goals map to the provided security mechanism.

1.7 ATTACK SURFACES AND ATTACK TREES

In Section 1.3, we provided an overview of the spectrum of security threats and attacks facing computer and network systems. Section 11.1 goes into more detail about the nature of attacks and the types of adversaries that present security threats. This section elaborates on two concepts that are useful in evaluating and classifying threats: attack surfaces and attack trees.

Attack Surfaces

An attack surface consists of the reachable and exploitable vulnerabilities in a system [MANA11, HOWA03]. Examples of attack surfaces are the following:

- Open ports on outward facing Web and other servers, and code listening on those ports
- Services available on the inside of a firewall
- Code that processes incoming data, e-mail, XML, office documents, and industry-specific custom data exchange formats
- Interfaces, SQL, and Web forms
- An employee with access to sensitive information vulnerable to a social engineering attack

Attack surfaces can be categorized in the following way:

- **Network attack surface:** This category refers to vulnerabilities over an enterprise network, wide-area network, or the Internet. Included in this category are network protocol vulnerabilities, such as those used for a denial-of-service attack, disruption of communications links, and various forms of intruder attacks.
- **Software attack surface:** This refers to vulnerabilities in application, utility, or operating system code. A particular focus in this category is Web server software.
- **Human attack surface:** This category refers to vulnerabilities created by personnel or outsiders, such as social engineering, human error, and trusted insiders.

An attack surface analysis is useful for assessing the scale and severity of threats to a system. A systematic analysis of points of vulnerability makes developers and security analysts aware of where security mechanisms are required. Once an attack surface is defined, designers may be able to find ways to make the surface

smaller, thus making the task of the adversary more difficult. The attack surface also provides guidance on setting priorities for testing, strengthening security measures, or modifying the service or application.

As illustrated in Figure 1.3, the use of layering, or defense in depth, and attack surface reduction complement each other in mitigating security risk.

Attack Trees

An attack tree is a branching, hierarchical data structure that represents a set of potential techniques for exploiting security vulnerabilities [MAUW05, MOOR01, SCHN99]. The security incident that is the goal of the attack is represented as the root node of the tree, and the ways that an attacker could reach that goal are iteratively and incrementally represented as branches and subnodes of the tree. Each subnode defines a subgoal, and each subgoal may have its own set of further subgoals, etc. The final nodes on the paths outward from the root, that is, the leaf nodes, represent different ways to initiate an attack. Each node other than a leaf is either an AND-node or an OR-node. To achieve the goal represented by an AND-node, the subgoals represented by all of that node's subnodes must be achieved; and for an OR-node, at least one of the subgoals must be achieved. Branches can be labeled with values representing difficulty, cost, or other attack attributes, so that alternative attacks can be compared.

The motivation for the use of attack trees is to effectively exploit the information available on attack patterns. Organizations such as CERT publish security advisories that have enabled the development of a body of knowledge about both general attack strategies and specific attack patterns. Security analysts can use the attack tree to document security attacks in a structured form that reveals key vulnerabilities. The attack tree can guide both the design of systems and applications, and the choice and strength of countermeasures.

Figure 1.4, based on a figure in [DIMI07], is an example of an attack tree analysis for an Internet banking authentication application. The root of the tree is

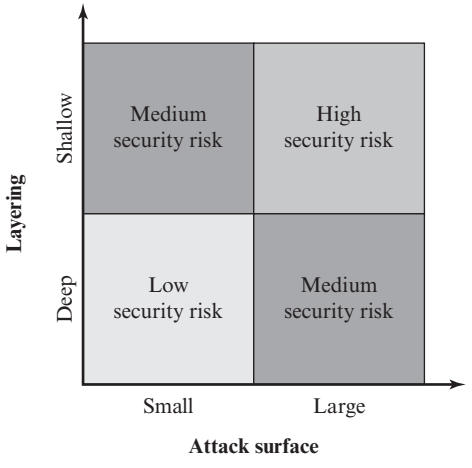

Figure 1.3 Defense in Depth and Attack Surface

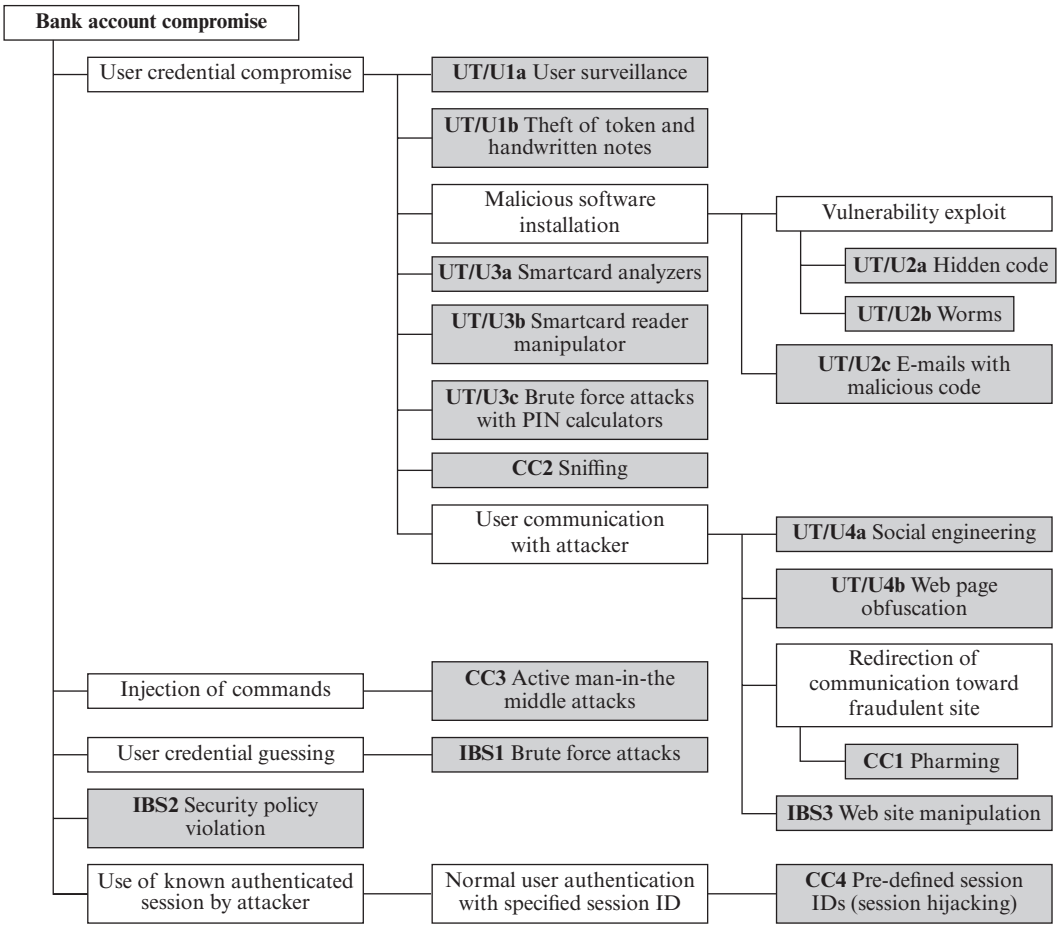

Figure 1.4 An Attack Tree for Internet Banking Authentication

the objective of the attacker, which is to compromise a user's account. The shaded boxes on the tree are the leaf nodes, which represent events the comprise the attacks. Note that in this tree in this example, all the nodes other than leaf nodes are OR-nodes. The analysis to generate this tree considered the three components involved in authentication:

- **User terminal and user (UT/U):** These attacks target the user equipment, including the tokens that may be involved, such as smartcards or other password generators, as well as the actions of the user.
- **Communications channel (CC):** This type of attack focuses on communication links.
- **Internet banking server (IBS):** These types of attacks are offline attacks against the servers that host the Internet banking application.

Five overall attack strategies can be identified, each of which exploits one or more of the three components. The five strategies are as follows:

- **User credential compromise:** This strategy can be used against many elements of the attack surface. There are procedural attacks, such as monitoring a user's action to observe a PIN or other credential, or theft of the user's token or handwritten notes. An adversary may also compromise token information using a variety of token attack tools, such as hacking the smartcard or using a brute force approach to guess the PIN. Another possible strategy is to embed malicious software to compromise the user's login and password. An adversary may also attempt to obtain credential information via the communication channel (sniffing). Finally, an adversary may use various means to engage in communication with the target user, as shown in Figure 1.4.
- **Injection of commands:** In this type of attack, the attacker is able to intercept communication between the UT and the IBS. Various schemes can be used to be able to impersonate the valid user and so gain access to the banking system.
- **User credential guessing:** It is reported in [HILT06] that brute force attacks against some banking authentication schemes are feasible by sending random usernames and passwords. The attack mechanism is based on distributed zombie personal computers, hosting automated programs for username- or password-based calculation.
- **Security policy violation:** For example, violating the bank's security policy in combination with weak access control and logging mechanisms, an employee may cause an internal security incident and expose a customer's account.
- **Use of known authenticated session:** This type of attack persuades or forces the user to connect to the IBS with a preset session ID. Once the user authenticates to the server, the attacker may utilize the known session ID to send packets to the IBS, spoofing the user's identity.

Figure 1.4 provides a thorough view of the different types of attacks on an Internet banking authentication application. Using this tree as a starting point, security analysts can assess the risk of each attack and, using the design principles outlined in the preceding section, design a comprehensive security facility. [DIMI07] provides a good account of the results of this design effort.

1.8 A MODEL FOR NETWORK SECURITY

A model for much of what we will be discussing is captured, in very general terms, in Figure 1.5. A message is to be transferred from one party to another across some sort of Internet service. The two parties, who are the *principals* in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the Internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.