A Structured Approach to Computer Security*

Tomas Olovsson
Department of Computer Engineering
Chalmers University of Technology
S-412 96 Gothenburg
SWEDEN

Technical Report No 122, 1992

ABSTRACT

Security and dependability are two closely connected areas. Recently, some attempts have been made to integrate the two concepts by integrating security into the more general topic of dependability. This paper describes security concepts and gives a survey of security terminology. It also establishes a taxonomy reflecting concepts within the security area which enables the integration of security and dependability. Special concern is given to the problems which inevitably arise from the integration, for example, a somewhat different definition of security is introduced. This paper does not pretend to cover every single mechanism found in security, but is rather an attempt to create a taxonomy for security analysis, estimation and design; a taxonomy that should be useful for further research within this area.

* This work was partially supported by the Swedish National Board for Industrial and Technical Development (NUTEK) under contract #90-02692P.

1. INTRODUCTION

The use of computers has increased considerably in the last few years and this development has given users a tremendous amount of computing power and easy methods for information exchange. During this period, large computer networks have been built to connect these computer systems. Computers connected to networks are often operated and owned by different companies, each with their own views of security. This, and different demands on computer security, makes it very hard to agree about what a secure system really is.

Today, computers are also used in applications requiring a very high level of dependability. Computer security has traditionally been treated as an independent subject having nothing in common with dependability. At the same time dependability has been discussed without attention being paid to security. A recent suggestion in this area is to integrate security with dependability [25]. Since security as well as dependability are rather well established, it will take time before such an integration is fully accepted, especially since changes in security terminology and taxonomy are needed. However, both areas will benefit from this integration, even if there are some initial problems such as the use of different terms and the presence of overlapping areas, for example system availability which is present in both dependability and in security.

Recently there has been a lot of work in the area of dependability to find a proper taxonomy and to define a common terminology. Such an approach is needed within computer security as well, especially since unification of security and dependability concepts requires the taxonomy for security to match that which is used in dependability. There has been one approach within the PDCS project [7], but its primary goal was not to present a complete taxonomy. This paper makes a much deeper analysis of how to define a proper taxonomy, and tries to cover most aspects and forms of security.

Both a terminology and a taxonomy reflecting the concepts needed for security analysis, estimation and design are presented. The taxonomy makes it possible to fully integrate security into dependability, thus changes have been made according to traditional views in order to make this integration possible, such as giving a somewhat different definition of security. Special concern has been spent to maintain existing terminology as much as possible, and, where possible, to present a taxonomy similar to the one found in the area of dependability.

2. DEPENDABILITY AND SECURITY

Security is closely related to the more general topic of how to obtain a dependable computing system. **Dependability** is the trustworthiness of a system and can be seen as the quality of the service a system offers. Integrating security and dependability can be done in various ways. One approach is to treat security as one characteristic of dependability on the same level as availability, reliability and safety, as shown in figure 1 [25].

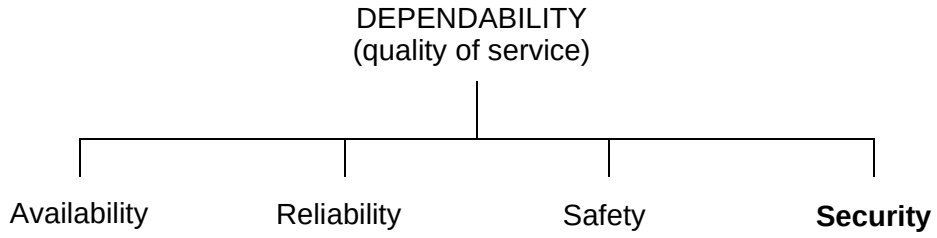

Figure 1: The relation between dependability and security

The **availability** *A(t)* of a system is defined as the probability that a system be operational at a time *t* or, in other words, the percentage of operational lifetime a system performs its functions. Loss of availability is often referred to as *denial of service*. High availability is mostly achieved by using redundant hardware in a system.

System **reliability** *R(t)* is closely related to availability, but reliability is the probability that a system will perform its functions during a time period $[t_0, t]$, given that it was working at the time t_0. Reliability is quite different from availability since reliability is a measure of the continuity of a service.

Safety *S(t)* is the probability that a system either performs its intended functions correctly or that the system has failed in such a manner that no catastrophic consequences occur (fail-safe operation). Safety is especially important in systems interacting with other systems which in turn may fail, and in applications where an uncontrolled failure of a system may cause major damage or personal injuries.

In this paper, **security** is defined as *the possibility for a system to protect objects with respect to confidentiality and integrity*. An **object** is a "passive" component within a system and consist of information and system resources, such as CPUs, disks and programs. An **entity** (which is sometimes referred to as a subject) is an active component in a system that causes information to flow among objects or causes a change in the system state [22], for example a user, a process or a device. What is considered as an entity in one operation may be an object in another operation, e.g. an entity reading information from a database is itself an object when another entity asks the process for the information.

This definition of security is similar to existing definitions [25], but instead of defining security as protection of information only, this definition also includes protection of system resources such as protection against illegal use of processor time, disks and networks.

Security is of special interest for those systems which need to preserve objects from threats in their environment. Several attempts have been made in the past to find a proper definition of security. The problem is to find a definition general enough to be accepted regardless of what kind of system is being described, yet detailed enough to describe what security really

is. For example, it is possible to express security in general terms: a secure system is a system on which enough trust can be put to use it together with sensitive information. This statement is valid and could also be chosen as a definition of security. However, it does not say anything at all about what security really is.

The integration of security with dependability allows dependability analysis more accurately to describe events causing a system to fail, since dependability analysis now includes not only traditional issues but also failures caused by security problems. Since it is quite clear that security problems affect the dependability of a system, it would then also seem reasonable to integrate security and dependability.

Not only dependability but also security will benefit from the proposed integration. Some mechanisms used to achieve a high reliability of a system, for example the use of design diversity, can increase the quality of programs maintaining security, and by making these programs less vulnerable the overall security for a system is increased. However, an increase in reliability may not necessarily imply improved security, and there are examples where increased reliability might degrade system security [22].

However, some complications arise from this integration. In traditional security analysis, availability is seen as one aspect of security, which is a problem since availability is also one aspect of dependability. This problem highly affects the discussion in section 5 which describes different aspects of security.

In the future, it would be desirable to extend the definition of security to be a function of time, *Sec(t)*, denoting *the probability for a system to protect objects with respect to confidentiality and integrity during a specified time period* $[t_0, t]$. With this definition, security and reliability become two very similar aspects of dependability. It opens new possibilities of expressing the security level for a system in absolute numbers and also to have a measure similar to MTTF (mean time to failure, derived from reliability *R(t)*) in the field of security. However, it is not an easy task to accomplish this, since it implies that there are methods available that describe security in absolute numbers. Extensive research is needed before such a definition can be accepted.

3. THE SECURITY CONCEPT

3.1 Security Cost Function

Security requirements for a computer system differ depending on applications: electronic funds transfers, reservation systems and control systems all have different demands. Also the amount of money the owners of the systems are willing to spend on maintaining security varies. There exist no absolutely secure systems and there are no absolutely reliable systems. Instead security can be measured on a continuous scale from 0 to 1 or from completely insecure to totally secure. Intuitively, a “secure system” is a system where an intruder has to spend an unacceptable amount of time or money in order to make an intrusion. Moreover the risk an intruder has to take may be considered to be too high.

Increased security most often results in increased cost for the system. The cost for security is a combination of many factors, for example cost for decreased system performance; cost for increased system complexity; cost for decreased usability of the system and increased operation and maintenance costs. Note that many of these costs are related in a complicated manner, for example it is possible to achieve a higher level of security by removing most or perhaps all functionality from a system, but this would result in increased an cost as a result of decreased usability.

Note that, for most systems, the cost for security exponentially increases when the security level approaches 100%, thus it is necessary to optimize the extent to which system resources should be protected. There must be a trade-off between the cost for increasing system security and the potential cost incurred as a result of successful security violations (figure 2).

The total cost for security violations must be calculated as the cost for a single security violation times the frequency of violations, a cost which is very hard to estimate.

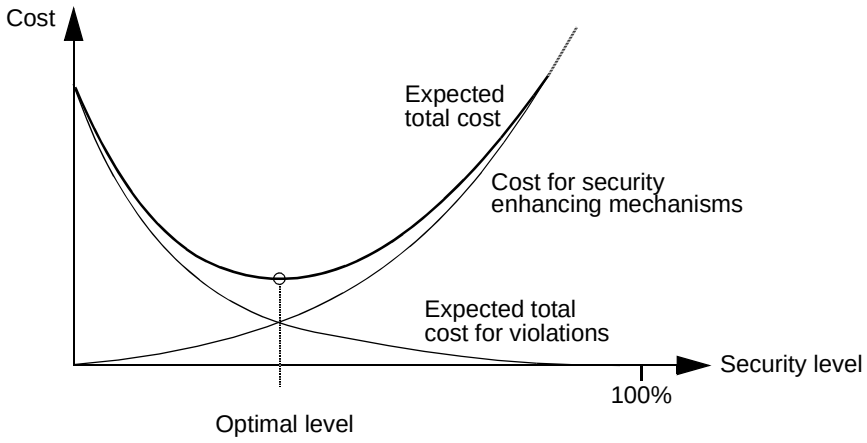

Figure 2: The security cost “function”

3.2 Security Policy

A **security policy** is a set of rules stating what is permitted and what is not permitted in a system during normal operation [19]. It is written in general terms and describes the security requirements for a system. The task to define a proper security policy is often a political decision to be taken by corporate management.

The security policy regulates how entities can gain access to objects in a system. The security policy should describe the well-balanced cost-effective protection of the system, as seen in figure 2, and should include all objects as well as all entities in the system. There is a similar action for specifying a security policy in the field of dependability, called to define a **failure semantics** [9], which is to define in what ways a system can fail to deliver its intended service.

Threat analysis is an important aid when defining the security policy. A threat analysis is a process where all possible threats to a system are identified. A list containing these threats and the severity of each threat is created. This list is then used as a basis for defining the security policy.

After the security policy has been defined, it can be used to decide what security mechanisms to select. **Security mechanisms** are the basic mechanisms used to implement security in a system, for example an access control mechanism which decides what entities are allowed to access an object.

Figure 3: The role of the security policy

When the security policy is defined, it is important to realize that the rules highly affect what security mechanisms are to be selected. This makes it important to define a security policy which enables the design of both a correct but also a practical, usable system. A correctly designed system is like a door to a building: if the door and the locks are good enough, most intruders will leave the building alone. However, if an intruder really wants to get in, he/she will be able to do that no matter what locks there are on the door. Also, if the locks are not easy enough to use, there is a tendency not to use them for their specific function at least for shorter periods of time, and by that create a situation an intruder may use. The same rule applies for computer security: the tools used to enforce security must be good enough and at the same time easy enough to use, to be accepted and to be used by the users of the system.

Any action, intentional or unintentional, that violates the rules stated in the security policy is a **security violation**. This definition allows for different interpretations of what is considered to be a security violation since two security policies can have different notions of security issues; they can attack different problems, and they can demand different solutions.

A **formal security model** is a mathematical formalization of the rules stated in the security policy and can be used to mathematically prove various properties about a system. To be precise enough, it needs to be written in a specification language or in a formal language [1], as opposed to the security policy.

4. THREATS

Threats can be seen as **potential violations of security** [19] and exist because of vulnerabilities, i.e. weaknesses, in a system. There are two basic types of threats: **accidental threats** that results in either an exposure of confidential information or causes an illegal system state to occur; and attacks that are **intentional threats**.

4.1 Accidental Threats

An **accidental threat** can be realized (or manifested) and result in either an exposure or as a modification of an object. **Exposures** can emerge from both hardware and software failures as well as from user and operational mistakes, and result in a violation of object confidentiality. For example, an exposure occurs when a user sends confidential mail to the wrong person.

An accidental threat may also realize itself as a **modification** of the object, which is a violation of object integrity. An object can be both information and resources, and a modification of a resource occurs when the resource enters an illegal state as a result of an accidental event.

4.2 Attacks

An attack is an intentional threat and is an action performed by an entity with the intention to violate security. Examples of attacks are destruction, modification, fabrication, interruption or interception of data [33]. An attack results in **disclosure** of information, a violation of object confidentiality, or in **modification** of objects, a violation of object integrity.

Figure 4: Two attacks against an object resulting in
a) disclosure b) modification of information

The definition of security as protection of objects and the definition of a security violation as an action violating the rules stated in the security policy (which describes how objects are allowed to be accessed), implies that a security violation is always an illegal access to an object. An attacker can gain access to a specific object by doing his attack in several steps, where each step involves an illegal access to an object. For example, system software could be the first target for an attacker, which in turn may help him to gain access to another object.

Attacks can be both direct and indirect. A **direct attack** aims directly at an object. Several components in a system may be attacked before the intended (final) object can be accessed. In this case, all these intermediate objects are targets for direct attacks. In an **indirect attack**, information is received from or about an object without attacking the object itself. For example, it may be possible to derive confidential information without accessing an object at all, by gathering statistics and thereby derive the desired information. Indirect attacks are especially troublesome in database systems where it is possible to ask indirect questions to a database about an object, and from the answers derive confidential information. Such an indirect attack is often called **inference**.

There are two different kinds of attacks: passive and active attacks. **Passive attacks** are done by monitoring a system performing its tasks and collecting information. In general, it is very hard to detect passive attacks since they do not interact or disturb normal system functions. Examples of passive attacks are monitoring network traffic, CPU and disk usage. Encryption of network traffic can only partly solve the problem since even the presence of traffic on a network may reveal some information. Traffic analysis such as measuring the length, time and frequency of transmissions can be very valuable to detect unusual activities. (Rumors say that prior to the US Panama invasion, Domino's pizza deliveries to the Pentagon jumped 25%, a situation in which an external observer could detect that something unusual was going on.)

An **active attack** changes the system behavior in some way. Examples can be to insert new messages on a network, to modify, delay, reorder, duplicate or delete existing messages, to deliberately abuse system software causing it to fail and to steal magnetic tapes. A simple operation such as the modification of a negative acknowledgment (NACK) from a database server into a positive acknowledgment (ACK) could result in great confusion and/or damage. Active attacks are, in contrast to passive attacks, more easy to detect if proper precautions have been taken.

The following paragraphs will describe some important types of attacks: Trojan horses, viruses, worms and covert channels.

4.2.1 Example: Trojan Horses

A Trojan Horse is a program performing one action but secretly also performing another. An example of a Trojan horse is a text editor searching documents for special keywords and, if a keyword is found, making a copy of the document available to someone else. The protection mechanisms in most systems have problems protecting information against such an attack. A document may be protected, but when entities have the possibility to select protection of objects at their own will, it is very hard for a system to stop a Trojan horse from requesting a change of protection for an object. In fact, most actions an entity may perform can be performed secretly by a Trojan horse, since the Trojan horse normally executes with the same privileges as the entity using it.

Another task a Trojan horse can perform is to open a **back-door** into a system. If a system administrator or any highly privileged user executes a program containing a Trojan horse, almost any action can be performed. For example, the Trojan horse may create new user accounts, modify the system into accepting users secretly or to modify encryption algorithms.

A special type of Trojan horse is a **logic bomb**. It is a program with a "feature" incorporated into it and this feature often consists of the destruction of objects. The bomb is programmed to go off at a specific time or when a specific event occurs. The idea behind a logic bomb is often to cause as much damage to a system as possible.

A Trojan horse can enter a system in many ways: it can be planted there by another user, it may have entered a system from the network (like viruses and worms) or it may have come with any piece of software installed in the system. It is normally very hard to identify what programs may contain a Trojan horse as well as it can be very hard to get rid of a Trojan horse [34]. Since the most vulnerable target for a Trojan horse is an entity with high privileges, this is a motivation for giving entities the least possible amount of privileges within a system, as long as they can fulfill their working tasks.

4.2.2 Example: Viruses and worms

Viruses and worms are relatives of Trojan horses. They are programs or code sequences designed to spread copies of themselves into other programs and to other computers. A **virus** is a small code sequence that modifies other programs into containing a copy of the same virus. A virus cannot survive by itself but it needs another program to modify and to insert itself into. By “infecting” other programs in this way, it will spread itself within a system. A **worm** on the other hand, is a program that spreads throughout a system without affecting other programs.

The function of a virus or a worm can be to disrupt the service of a system or to plant a Trojan horse or a logic bomb into a system. Worms and viruses are common in smaller systems lacking protection, but systems with a high degree of protection can also be the subject of a virus or worm attack [39]. Improper handling and incomplete testing of software are important channels for the spreading of viruses and worms.

4.2.3 Example: Covert channels

A covert channel is an unprotected channel that can be used by an entity to send confidential information to unauthorized entities and thereby violate security. In general, it is very hard to identify covert channels in a system since they can be of many different types: message length variations during transmissions, time and length of transmissions, presence and size of files, creation time for objects, modulation of disk usage, CPU time usage, etc. It is impossible to give a complete list, resulting in that there are no simple workable solutions solving all problems with covert channels.

Mandatory encryption of communication is no guarantee that entities will not (deliberately or not) send information to another entity over a covert channel. For example, it is still possible, while sending legal messages to another entity, to modulate the length of messages. More importantly this can be done without any users of the system being aware of it, since information can be created and sent without their knowledge by a Trojan horse.

Sometimes covert channels are divided into two groups: **timing channels** are covert channels modulating a resource in time, and **storage channels** are channels where actions like creation of objects reveal information to other entities, for example to choose specific file names, file sizes, etc.

It is very hard to eliminate covert channels completely in a system, and since a covert channel with a high bandwidth constitutes a higher threat than a covert channel with a low bandwidth, most security mechanisms try to reduce the bandwidth of these channels as much as possible. Even a covert channel with a bandwidth as low as 100 baud is in some environments considered to be dangerous [17]. However, actions to limit covert channel bandwidths always limit system performance. For example, in order to avoid the length of messages from being used as an information carrier, all messages can be forced to be of equal length. The problem with this method is that it reduces the available bandwidth of the network as well.

5. ASPECTS OF SECURITY

Traditionally security has been divided into three different aspects: *confidentiality, integrity, availability*. However, when security is integrated with dependability (fig. 1), availability must be seen as a dependability issue and not as a security issue [3][25]. As a result, security is divided into two aspects, *confidentiality* and *integrity* (fig. 5). In a system where dependability is a crucial matter, all objects in a system must be protected against both intentional and unintentional threats. To a user, there is not a very big difference between someone deliberately disconnecting the power to a system, someone doing it by mistake or if a hardware failure disconnects the power, since in all three cases the system will fail to deliver its service.

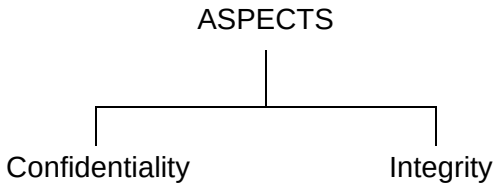

Figure 5: Aspects of security

Sometimes a fourth aspect of security is identified: *preservation of objects*. However, preservation of objects can be dealt with either as an availability issue or as an integrity issue. It is closely related to integrity issues since loss of information is not very different from having it destroyed completely by unauthorized modification.

5.1 Confidentiality

Security was defined as the possibility of a system to protect objects with respect to confidentiality and integrity. Confidentiality is the issue of how to protect objects from unauthorized **release** of information or from unauthorized **use** of system resources such as CPUs, programs or other kinds of equipment. The protection mechanism should make it possible for each individual entity to decide whether his objects should be confidential or not. Also, a good protection mechanism should allow the owner of a system to determine who may and who may not access or use an object, i.e. the protection system must be able to prevent an entity from making confidential information available to other entities. A more detailed discussion of protection mechanisms follows in section 7.

5.2 Integrity

Integrity is the issue of how to **preserve** objects to make them trustworthy, i.e. how to avoid the unauthorized **modification** of objects. Unauthorized users might not be able to read the contents of an object but the protection system must be able to prevent an authorized entity from adding or modifying any parts of the object. Integrity of a resource is the issue of how to preserve the resource and how to protect it from unauthorized modification.

6. FORMS OF SECURITY

An attacker can find several objects to attack in a system in order to obtain a specific piece of information: system software can be attacked; the physical computer installation can be attacked, for example, by theft of magnetic tapes; a legal user may be bribed (i.e. attacked); etc. Thus, there is a need to divide security into different forms which will keep similar security characteristics together. There are some (somewhat disagreeing) approaches to this problem [4][7][43]. In this paper, most of the security forms are similar to ones found elsewhere, but here they are grouped into a completely different hierarchical structure. This structure is based on where in a system vulnerabilities may be found: vulnerabilities in the hardware, vulnerabilities in information or software, and vulnerabilities in the organization that administers the system, as shown in figure 6.

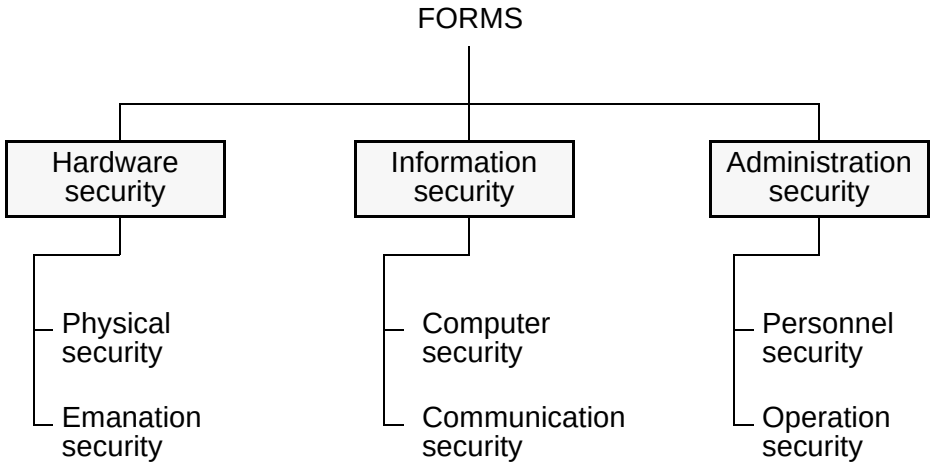

Figure 6: Forms of security

6.1 Hardware-related Security Issues

Hardware security issues deal with protection of objects from vulnerabilities present in the handling of hardware. Hardware security can be divided into physical security and emanation security:

Physical security deals with protection of hardware in the system from external physical threats, such as tampering, theft, earthquakes, water flooding. All equipment handling or containing sensitive information needs to be protected. There must be no possibility for an intruder to access these devices, for example no one must be able to remove a disk containing sensitive information or to install devices to record confidential information. These problems can be solved by locating equipment in an environment secure enough to contain the information handled by the equipment, i.e. physical security deals with how to create and maintain such an environment. Note that physical security deals with protection of objects and not with computer equipment in general. However, the result of protecting sensitive objects can very well result in better protection of equipment as well.

Emanation security deals with protection against emission of signals (i.e. information) from the hardware in the system, for example electromagnetic emission (e.g. from displays), visible emission (displays may be visible through windows) and audio emission (sounds from printers).

6.2 Information-related Security Issues

Information security is protection of objects from vulnerabilities present in the architecture of the system, i.e. vulnerabilities in software, hardware and in the combination of software and hardware. Information security can be divided into computer security and communication security:

Computer security deals with protection of objects against exposures and against attacks making use of vulnerabilities in the system architecture. It deals with a wide variety of problems: how programs inside the computer should act to enforce the security policy; how the access control mechanism should work; what hardware mechanisms the operating system needs, for example virtual memory; what encryption mechanisms to select; etc.

Communication security deals with protection of information during transportation. When objects are transported, either between computers or locally within a computer, an active attack may be undertaken in order to interact with the communication process, for example to modify, retransmit, reorder or destroy information. Also, objects need to be protected against passive attacks and exposures during transmission.

6.3 Administration- related Security Issues

Administration security is protection of objects from vulnerabilities caused by users (i.e. humans) and threats against vulnerabilities in the security organization. Administration security can be divided into personnel security and operation security:

Personnel security deals with protection of objects against attacks from authorized users. Users of a system have access to various objects, and protection mechanisms against users who deliberately abuse their privileges are necessary or at least methods limiting the damages a user may cause. The reasons a user may have to abuse his/her privileges can be several: an external attacker may convince the user to perform an attack (bribes, threats, cheating, etc), there can be a personal gain such as money, it can be an intellectual challenge, it can be to punish the company, etc. Personnel security issues also include protection against exposures arising from authorized users, for example when a user sends confidential mail to the wrong person or when a user forgets to log out.

Important mechanisms used to limit damages are to educate users as to the importance of maintaining security, to restrict privileges to users who have a need to know, to move sensitive information away from certain systems, to have supervisory controls, etc. Note that if a user needs to commit a security violation to access information, this would be an information security issue and not a personnel security issue.

In general, authorized users are a greater threat than external attackers, and an even worse threat is personnel responsible for maintaining security in a system who abuse their privileges. Statistics say that only about 10% of all computer crimes are performed as outside break-ins, 40% are committed by insiders and about 50% by former employees as an act of revenge [20]. Clearly, personnel security issues should highly affect the security mechanisms to implement in a system.

Operation security deals with protection of objects against vulnerabilities present in the organization that maintains security in a system. Operation security regulates how all the other forms of security should be implemented and how the system should be operated. It deals with ways of enforcing the rules stated in the security policy, what actions to take when security violations are detected in the system, what recovery mechanisms to implement, etc. It is important that persons maintaining security are informed about events that

cause security violations in other systems, and that they continuously update and modify their own system to reach the desired level of security, which can be done by regular updating, changes and modifications of the mechanisms enforcing security.

All six forms of security are summarized in the table below. The primary target shown describes what kind of object an attacker may use in the system, i.e. what primary resource to protect in the system.

Note that operation security as described above cannot be a target for attacks. An attacker can very well use this knowledge that a vulnerability in the operation of a system exists, for example the knowledge that faulty software is used and never updated, but the actual attack is directed against another form of security, in this case it is a computer security issue. It is necessary to make this distinction, or else all attacks could be seen as operation-security issues since operation security maintains the overall security in the system. In other words, an attack can not be directed against operation security, but a vulnerability (a deficiency) in operation security may result in an attack against another form of security, or in exposure of information.

Category	Form of security	Aspect	Target for attacks	Example of threat	Primary security mechanisms
Hardware	Physical	Conf+Int	h/w	Theft, modification	h/w
	Emanation	Conf	h/w	Receivers	h/w
Information	Computer	Conf+Int	s/w (h/w)	Abuse of system s/w	s/w (h/w)
	Communication	Conf	inform.	Recording information	encryption, h/w
		Int	s/w	Interaction (with comm)	encryption, s/w
Administration	Personnel	Conf+Int	humans	Authorized users	rules, education
	Operation	Conf+Int	--	Operational mistakes	rules, training

Table 1: Characteristics for different forms of security

The reasons why it can be necessary to protect a computer against theft can be several, and many forms of security can be involved:

1) Protection of information in the computer: a physical security issue
2) Protection of the computer as a resource not to be used by others: a physical security issue.
3) Protection of the equipment because of its monetary value: some other kind of problem, not different from theft of typewriters and other equipment.
4) Depending on the circumstances, it could be an availability issue, a reliability and/or a safety issue.

7. SECURITY MECHANISMS

Security mechanisms are mechanisms used to implement the rules stated in the security policy as shown in figure 3. Security mechanisms can be divided into three categories: prevention, detection and recovery mechanisms [19]. Within each group, there are many security mechanisms available, where each mechanism focuses on a specific kind of threat and deals with a specific form and aspect of security.

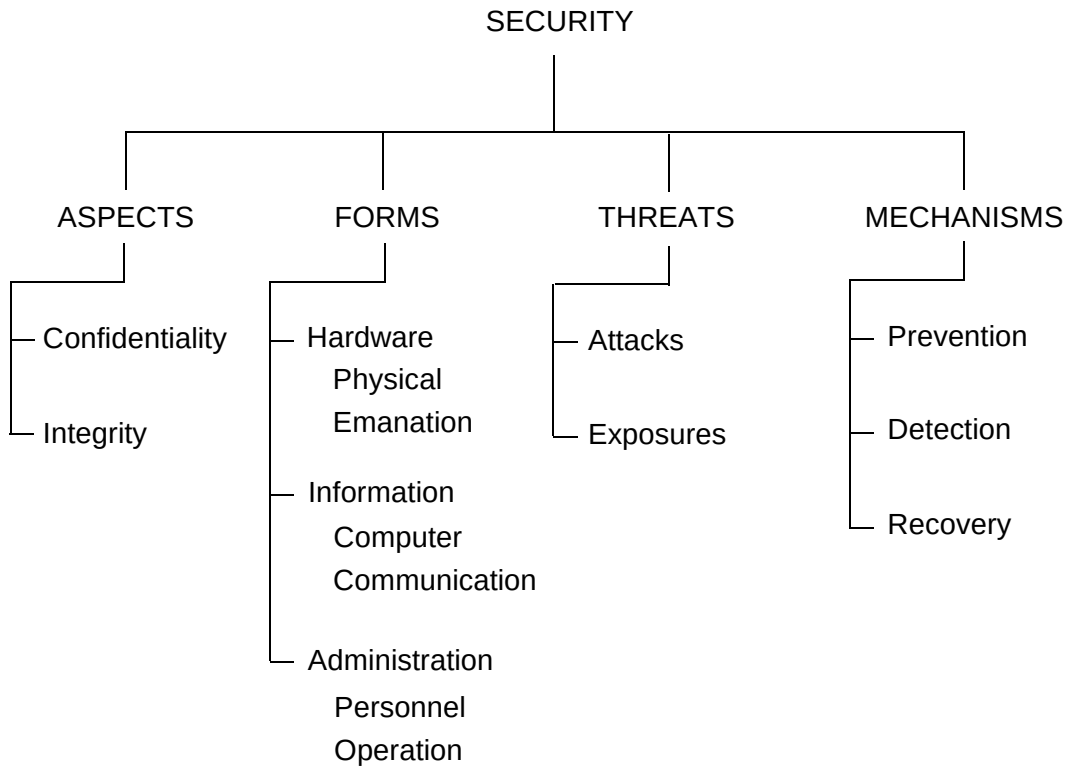

Figure 7: Relations within security

A security **prevention** mechanism is a mechanism enforcing security during the operation of a system by preventing a security violation from occurring, for example a mechanism restricting physical access to a system or the use of access control mechanisms based on encryption to prevent unauthorized users from accessing objects. A **detection** mechanism is used to detect both attempts to violate security and successful security violations, when or after they have occurred in a system. Alarms can be used to detect unauthorized physical accesses and audit trails can be used to detect unusual system activities after they have occurred. A **recovery** mechanism is a mechanism that is used after a security violation has been detected, and is a mechanism that restores the system to a pre security violation state, for example to have backup tapes and to add redundant hardware to a system.

It is also possible to find mechanisms which belong to several of these categories: a program registering all unusual system activities and thus working as a detection mechanism, may also prevent security breaches from occurring simply because it exists. In a system with a total lack of vulnerabilities and where the security prevention mechanisms fully implement all rules stated in the security policy, there would be no need for detection and recovery mechanisms.

The following paragraphs describe in detail some of the most important security mechanisms in use today. The discussion is, for practical reasons, limited to information security issues only. Security mechanisms are identified in the ISO/OSI security addendum [19], however, here we have a somewhat different approach to the problem: only the most important mechanisms are selected but each individual mechanism is discussed more in detail. Also, some mechanisms not present in the OSI addendum (e.g. separation mechanisms) are included here. The following mechanisms are discussed:

- Authentication
- Access control: policies, models and implementations
- Separation mechanisms
- Communication mechanisms: routing control, traffic padding, signatures, etc.
- Detection and recovery mechanisms

7.1 Authentication and Identification Mechanisms

An authentication mechanism makes it possible to uniquely identify entities, which is necessary before other mechanisms can make decisions based on the identity of an entity. Since the authentication mechanism is the basis for most other security mechanisms, it needs to be as secure and robust as possible in order for other entities and security mechanisms to be trusted.

Before the authentication process can start, all entities need a unique **identification**. It must be something which is very hard to counterfeit by other entities, for example something an entity knows or carries such as a key to an encryption algorithm (a password), a smart card or a finger print. The process of checking the identity of an entity is called to **authenticate** the entity. To administer identities, a **security administrator**, who maintains a security information database is needed.

The authentication mechanism is normally mandatory for all entities, since identity must be established before the entity can be granted any rights in a system. In many systems there are mechanisms available to pass the identity of one entity to another entity. For example, when a user logs in to a system, his rights are inherited by one or more processes in the system, such as text editors, mail programs, etc. These processes can be thought of as bearers of the user’s identity.

Authentication can be divided into two types: weak and strong authentication. **Weak or simple authentication** mechanisms are "normal" mechanisms used by most systems, for example the use of a password when a user logs in to a system. **Strong authentication** mechanisms are mechanisms where an entity does not reveal any secrets during the authentication process. This can be achieved, for example, by using asymmetric cryptosystems where only the entity itself knows how to encrypt a message but where all entities knows how to decrypt the message. In this case, an entity never has to reveal the encryption key, i.e. it is a secret not shared with anyone else (see below).

7.1.1 Two Party Authentication

The process where one entity (the client) wants to be authenticated by another entity (the server), can be performed in several ways:

The easiest way is to have the client send his password to the server who checks it against his own records. There are at least two major drawbacks with this method: the client cannot know that he is actually talking with the correct server, and even if he does, it may be possible for a third entity to pick up the password when it is sent to the server. This method is called an **unprotected simple authentication procedure** [9].

An improved method is to use a challenge protocol where the client encrypts for example a random number together with a time-stamp using the its password, and sends it to the server. The server then repeats the same process and verifies that the client actually did know the correct password. This method is insensitive to replays from a third party and can be extended by having the server authorize himself against the client in a similar way. This authentication mechanism is easily built into small "smart-cards" containing a chip which is able to perform the encryption of a random number.

A similar method is to apply a one-way function to the random number, the time-stamp and the password, $f(r, t, p)$, and then to send the result of this computation to the server. The server then repeats the procedure to verify the password. This method is called a *protected* **simple authentication procedure** [9].

The above solutions still have one drawback: when a secret is known by many entities, the security in the system relies on the most insecure entity not to reveal his secrets. A solution to this problem is to use a **strong authentication procedure**, e.g. to use an asymmetric cryptosystem. This cryptosystem has the property that an object (for example a random number) M encrypted with one key ke, must be decrypted with another key, the decryption key kd, to retrieve original contents: $M = D_{kd}(E_{ke}(M))$. By keeping the encryption key ke secret and the decryption key kd public, only one entity can perform the encryption but all entities can do the decryption. This makes it is possible for any entity to validate the contents and origin of a message since there is only one entity in possession of the encryption key, i.e. it is possible for an entity to prove his identity to other entities without revealing any secrets.

7.1.2 Third Party Authentication

The previously mentioned authentication methods still have one drawback: all entities who want to authenticate clients need to know either the secret or the public key for all clients in the system, which in a large system very soon becomes impractical from a management point of view. A possible solution is to have a special entity, a third party, performing the authentication process for all entities in the system. One such commercially available authentication server is Kerberos, originally developed at MIT in the Athena project [40]. It is a central authentication server generating keys for use by individual entities in a network. In Kerberos the security of the authentication process depends on the security of the

authentication server only. Each entity has a private key which it shares with Kerberos. This key (or password) is presented during the authentication process by a challenge protocol. To establish connection with another entity, the following process takes place (figure 8):

- Entity 1 contacts Kerberos and asks to be authenticated (1). This authentication can be performed with one of the previously described mechanisms. If Kerberos accepts the entity, entity 1 gets a ticket (2) that can be used to prove its identity for a special server, the Ticket Granting Server, TGS.
- When entity 1 asks the TGS (3), it receives a session key, a ticket (4), to be used together with another entity, for example together with entity 2.
- The session key can be used as an encryption key or as an identification key when entities are communicating (5).
- If more session keys (tickets) are required for communication with other entities, only the TGS need to be contacted again (3)(4).

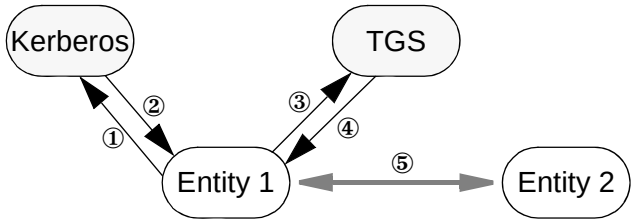

Figure 8: The Kerberos Authentication server

When this scheme is executed, both entities know that only the other entity can possess the same session keys. Also, note that entity 2 may very well deny entity 1 access, since Kerberos is only an authenticating service, giving entities the possibility to uniquely identify themselves.

Unfortunately the identification key is disclosed to the local computer when the authentication takes place, thus this computer can save the key and use it later on. Another potential problem is that Kerberos is in possession of all keys to all entities, and these keys need to be completely secured.

7.2 Access Control Mechanisms

Within a system objects are protected by an **access control mechanism** which mediates all accesses to objects and controls the way in which entities can use them. The basic components of an access control mechanism are entities, objects and access rights. The access rights describe entity privileges and state under what conditions entities can access an object and how these entities are allowed to access the object.

An access control mechanism is implemented in most multi-user operating systems, and there are several demands on a good access control mechanism. Ideally, it should be possible to specify for each object exactly what entities are allowed to access it, and what kind of access each entity is allowed (read, write, delete, create, extend, change protection, etc.) However, existing operating systems do not, for practical reasons, support such a detailed specification. In addition to this, most systems lack methods for specifying access to network objects, for example, mechanisms making it possible to state for individual objects

that an object may always be accessible by entity X unless it is done from system Y. There have been some attempts to solve this problem [37][38], but unfortunately such systems are not widely available.

Also, a good access control system must support **revocation of access rights**. It must be possible to withdraw an entity's rights to access an object. Unfortunately, this is not possible with most operating systems today since an actual implementation of this would imply that the access rights have to be checked each time an object is accessed, for example at each individual read or write operation performed on a file, and not only when a file is opened.

7.2.1 Access Control Policies

Access control can be divided into two policies:

- **Discretionary access control:** the owner of an object has the possibility to protect an object against access from other entities on a need-to-know basis, i.e. an entity can specify what other entities may or may not access an object, and in what ways these entities may access the object.
- **Mandatory access control:** the system always checks an entity's rights to access an object. Neither an entity nor the owner of an object can ever override or change the decision made by the system. Mandatory access control is often required for systems with a high security level [12].

These two policies can be combined to work simultaneously: the entity owning an object can have the possibility to specify what other entities are allowed to access it, but only within certain limits imposed on him by the mandatory access control.

A Trojan Horse (section 4.2.1) may spoof the owner of an object, thus overriding the discretionary access control, but never the mandatory access control. At the same time, a mechanism enforcing mandatory access control is insensitive to user mistakes and makes it possible to make a mathematical description of the information flow in the system, as opposed to the use of discretionary access control, where information flows can be more or less arbitrarily [2] [27].

7.2.2 Access Control Implementations

There are different ways to implement access control mechanisms, for example to use protection groups, access lists, capability lists and lock-key mechanisms. Each mechanism can be used to implement both mandatory and discretionary access control policies:

- Entities and objects can be divided into **groups**, where all entities and all objects belong to one or more of these groups. The access rights to an object are based on group membership of the entity. For example, it is possible to have an implementation where an object can be modified only by entities that are members of the same groups as the object, or it is possible to state that it is enough to be a member of only one of the groups. This last case is the protection mechanism present in most UNIX operating systems today.
- An **access list** is a list associated with an object and contains the names of all entities allowed or not allowed to access the object. The list includes the type of access each entity is entitled to (read, write, create, delete, etc.). Access lists are more general than groups since each individual entity can be specified in an access list.
- A **capability list** is similar to an access list, the difference being that each *entity* is associated with a list describing what objects it is allowed to access. The advantage with capability lists over access lists is that an entity can be granted access to two objects but not at the same time, since this can easily be implemented by allowing entities to choose between different capability lists with different capabilities.
- In a **lock-key mechanism** each object is given a password, a key an entity must be able to present before access is granted to an object. This differs from the other mechanisms where access to an object is based only on the identity of an entity. A lock-key mechanism is used by most operating systems when they determine whether a user should have access to the system or not, but it can also be used to determine access to individual objects within the system.

Combinations of these protection mechanisms are also possible. Groups, for example, can be combined with access lists in order to allow a finer grain of protection than the use of groups alone would allow.

7.2.3 Formal Models for Access Control

A formal security model is a mathematical description of how objects are allowed to be accessed. As a result, it can be used to describe the information flow within a system and to prove that the integrity and/or the confidentiality of objects at all times can and will be preserved.

Two especially important models are described in the following sections: the Bell-LaPadula Model [24] enforcing confidentiality of objects, and the Biba Model [33] enforcing integrity of objects. Both models are based on set theory that defines secure states and transitions within a system.

7.2.4 The Bell-LaPadula Model

In the Bell-LaPadula model entities are divided into security classification levels (for example Unclassified, Confidential, Secret and Top Secret). The Bell-LaPadula model enforces one fundamental rule: no entity should be able to have read-access to an object classified above its own security level, i.e. the entity's security level must be *greater than or equal* to the object's security level. This is called the simple security rule and enforces

"No read-up." To write to an object, the entity's security level must be *less than or equal* to the object's security level. This is called the *-property (the star property) and enforces "No write-down." Together, these rules can be written as:

read:	$SL(Entity) \geq SL(Obj)$	*The simple security rule*
write:	$SL(Entity) \leq SL(Obj)$	*The * property*

where SL denotes the security classification level for an entity or an object.

The Bell-LaPadula model enforces secrecy of objects, since an entity cannot read objects with a higher security classification level than the entity itself. However, since it is possible to write to objects with a higher security classification than the entity itself, it might be possible for an entity to corrupt objects classified above its own security level, and by that violate the integrity of objects.

A serious problem sometimes arises when the *-property is applied to a system: since reports generated by a system often concern objects with a high security classification level, most reports need to be classified at a very high security level. For example, the knowledge of the number of users in a system, may have to be classified as Top Secret information. In general, there is a tendency in a system relying on the Bell-LaPadula model to have an information flow toward higher security classification levels. This makes it necessary to have a special mechanism which is able to declassify information, a mechanism that allows a certain trusted entity to decrease or in any other way change the security classification for objects. Other practical changes to the Bell-LaPadula model can be to allow these trusted entities to talk freely to entities classified below their own security level (violating the *-property), for example to allow a special entity to receive and send mail to other lower classified systems.

When an entity wants to communicate with another entity, the communication should obey the rules above as well. In a networking environment, this is especially cumbersome since there is no possibility to reliably send a message to a higher classified entity, without violating the *-property. This is due to the fact that the receiver is not allowed to send a reply nor even an acknowledgment back to the sender. For local communication taking place within a single computer this may be acceptable, since such a communication can be made (almost) reliable without acknowledgments [41]. Sometimes a solution using a trusted intermediary can be used [28].

7.2.5 The Biba Model

The Biba model is another formal model and it is the counterpart of the Bell-LaPadula model. This model preserves object integrity rather than object confidentiality. It is done by allowing only "Write-down", i.e. it ensures that entities can only write to objects with a lower security classification than the entity itself. This prevents an untrustworthy entity from deliberately creating or modifying objects with higher security classification, which is possible in the Bell-LaPadula model. Also, an entity can only read information from objects with a higher security classification level than the entity itself. Since this model only preserves integrity, it can be described in terms of having **integrity levels** instead of security or sensitivity levels. These rules can be summarized as:

write:	$IL(Entity) \geq IL(Obj)$	*The simple integrity rule*
read:	$IL(Entity) \leq IL(Obj)$	*The * property*

In the Biba model, it is possible for all entities to read any object classified at a higher level than the entity itself, and to send information to all entities working at a lower classification level. Thus it is possible for an entity to deliberately disclose the contents of higher classified objects.

The Bell-LaPadula model and the Biba model can be combined to work together as a single system [8]. One possibility is to force normal entities to follow the Bell LaPadula model, i.e. to have them follow the confidentiality enforcing rules, and to force trusted entities who need to be able to declassify entities to follow the Biba integrity enforcing rules.

7.2.6 The Military Security Model

This model is sometimes referred to as the **lattice model** and sometimes as the **military security model**, since it is used in US military and government specifications for computer security [12]. It is a good example of how a mandatory access control policy can be implemented. The example shown in the figure below, has 9 *non*-hierarchical security levels (A-I) called **compartments**, and four hierarchical security levels (Unclassified, Confidential, Secret and Top Secret) describing an object's **security classification**. The figure contains three objects and, for example, object 2 belongs to compartment F+G+H and is classified as a "Secret" object. In practice, this implies that all objects have some information associated with them, for example labels attached to them that shows their security classification and to what compartment they belong.

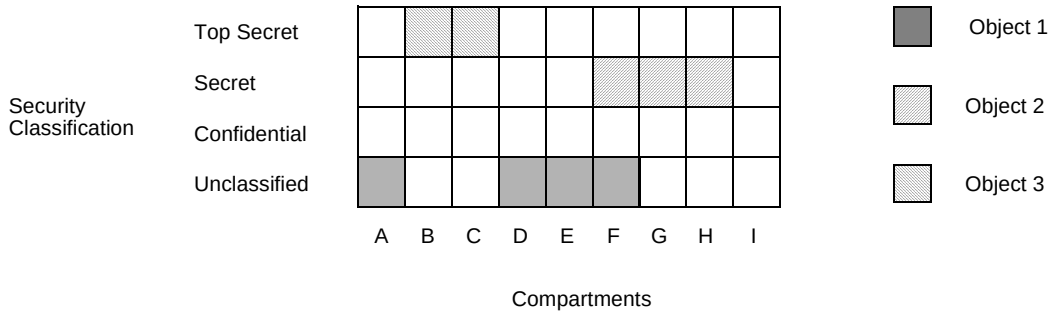

Figure 9: Mandatory access control in the military security model

Both the compartment belonging and the security classification of an object is checked by the mandatory access policy. To be able to access an object, the entity must be a member of all the compartments the object belongs to, for example to access object 2, an entity must (at least) be a member of compartment F+G+H. This can be described as limiting the access to an object to those entities with a *need to know*. Furthermore, only entities allowed to access Top Secret and Secret objects can access object 2 as it is a secret object. Also, the mandatory access control policy makes sure an entity cannot change the security classification or compartment belonging of an object.

The Bell-LaPadula model was the base for the US Department of Defence's "orange book" which describes the lattice model. It can be proven that this model with both hierarchical security levels and with compartment belonging, forms a partial order (a lattice), and the model can be used to describe and to prove how information can flow within a system.

7.2.7 Object Protection in Databases

There is a difference between an operating system and a database protection mechanism: the database object protection mechanism often need to work with a finer granularity, since protection of individual records in a database is not normally offered by operating systems. Thus, a database server may have to implement a protection scheme similar to the operating system, where the database server needs to enforce both integrity and confidentiality protection rules.

Also, each record (object) within the database may need a detailed protection mechanism stating who is and who is not permitted to read, insert, modify, extend, delete, change protection, etc. This detailed specification may not be supported by the operating system, i.e. this is another reason why a database server may have to implement own protection mechanisms. Of course, the database itself can be subject to operating system protection mechanisms at the same time as it maintains its own protection of individual records within the database.

One very specific problem to database systems is the **prevention of information generation**, i.e. how to implement **inference control.** Inference arises when classified information is derived from several other less classified pieces of information, for example the output of a simple operation such as A+B may need a higher security classification level than the individual components A and B have. Inference control can be incorporated into a database server which, depending on specific queries, makes the decisions whether an entity can derive some confidential information from earlier questions and answers or not [17]. However, this is in general very hard to implement. Other solutions to the inference problem can be to add auditing of “unusual” questions issued to the database server.

A similar inference problem occurs when direct questions about an object are rejected because of its classification. However, it may still be possible to ask an indirect question and from the answer derive information about a classified object. For example it may not be possible to ask questions about salaries for individual employees, but a list without names containing all salaries in the company may be possible to get. From this list, it might be possible to derive the salaries for at least some of the employees.

7.3 Separation Mechanisms

Computers handling multilevel objects need a mechanism to separate objects of different security classification levels from each other. There must be no information flow between objects or between objects and entities without permission from the access control system, thus we need to have a mechanism separating objects and entities. This separation can be done in five different ways:

- Physical separation
- Temporal separation
- Cryptographic separation
- Logical separation
- Fragmentation

Note that nothing restricts an actual implementation of a secure system to use any combination of these mechanisms. A system handling objects of only one security classification can be made simpler since it does not need to separate them.

We define a computer connected to a network capable of storing and processing information as a **host**, as opposed to a **system** which can be one or more hosts connected by a network. A host capable of handling objects belonging to several security levels is often referred to as a **multilevel secure host**.

7.3.1 Physical separation

Physical separation is a method suitable for an environment with several individual hosts on a network. The system is divided into different subsystems where each subsystem has a different security classification. For example to have a workstation, dedicated to handling only secret objects, located in a secret environment and only accessible by people authorized to handle secret objects (and whose compartment complies with the usage of the system), is a very desirable solution since it eliminates the need to find a trusted operating system to implement the access control. The disadvantage with this kind of solution is the cost and inefficiency of dedicating hardware to specific tasks, instead of sharing hardware among users.

It is also possible to achieve physical separation by using virtual resources. A multilevel secure host can offer services to several hosts working at different security levels simultaneously. One example of physical separation is the use of a virtual machine, where each user is given their own version of the operating system and of all other programs they are using. Physical separation makes use of hardware and software to create a secure system.

7.3.2 Temporal separation

Temporal separation is another method that is used frequently. Temporal separation uses time to separate objects. When a host needs to change its security level, all old activities are stopped; the machine is completely restored to its initial state (memory, disks, etc.) and the new tasks are started. This method only works if it is possible to find a method to restore the machine safely to its initial state and if the waiting time to reinitialize the machine is acceptable.

7.3.3 Cryptographic separation

Cryptographic separation uses cryptography to separate objects of different security levels. The basic idea is that encrypted information should be of no value to an unauthorized entity. Cryptographic separation seems to be the separation method which is currently being most researched. There are several problems that need to be solved: developing strong, fast and easy encryption/decryption mechanisms, finding easy ways for key distribution among entities, finding where and in what security mechanisms to apply encryption, etc. Cryptographic separation makes it possible to create separate virtual systems within a larger system, where all entities not in possession of a key are isolated from the rest.

7.3.4 Logical separation

Logical separation, sometimes also called **isolation**, is based on the idea that all entities should only be aware of their own environment, i.e. they can only see what they are authorized to see and they should be oblivious of other entities working in the same system. Logical isolation can be implemented by giving processes their own address space and by using virtual memory to separate them from each other.

An example of logical separation is to use reference monitors to separate objects from each other. A **reference monitor** is an abstract machine controlling all accesses to objects within the machine, thus it is a monitor protecting objects. The reference monitor only contains the

necessary protection mechanisms, thus it makes it possible to keep the protection system as small and simple as possible, which in turn, makes it possible to create a system that can be exhaustively tested and which can be formally verified [36].

The actual implementation (or the instantiation) of a reference monitor can be done in several ways: A local database server can have a reference monitor in front of it, checking the identity and access rights for entities before they are granted access to the database. Only if the reference monitor accepts a request, is it passed on to the database server. The entity using the database does not see anything outside its own world, since the reference monitor filters the requests and replies from the database server to contain only those objects the entity is authorized to access.

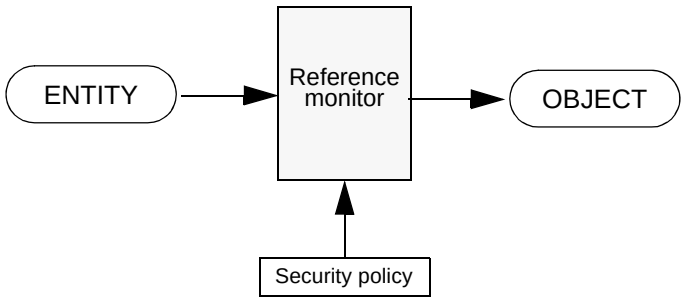

Figure 10: The Reference Monitor Concept

Reference monitors can also be used by computers to check accesses to various objects in the system, instead of having the operating system itself do the checks. A commonly used instantiation of a reference monitor is a **security kernel** or a **trusted computing base (TCB)**. The TCB within a system is made up of all the protection mechanisms that enforce the security policy for that computer, and may consist of both hardware and software. The trust put on the TCB determines the overall trust of the system. By isolating all the protection mechanisms of an operating system into a TCB, the protection system is relatively small and it is easier to analyze the mechanisms used to achieve security, since it may be possible to give formal evidence of a system's ability to behave correctly during given conditions. Also, changes to the operating system and penetrations of the operating system do not have to affect the overall security of the system.

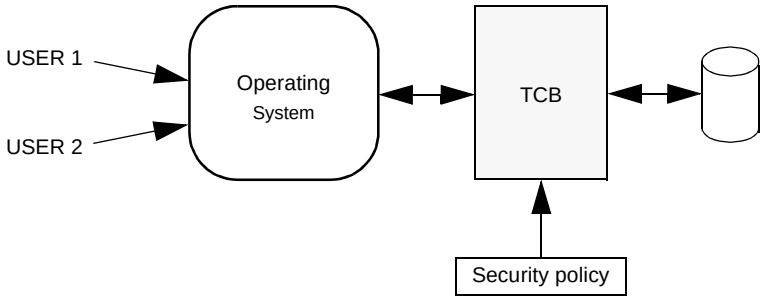

Figure 11: TCB monitoring disk accesses

A reference monitor can also be used between the operating system and a network to check all network traffic. When a reference monitor is implemented as a separate piece of hardware located between a host and the network, this construction is often referred to as a

trusted network interface (TNI). The use of a TNI allows hosts to internally use an untrusted version of Unix but still be protected against network attacks, since the TNIs ensures that only authorized transactions are sent and received by the Unix system. By adding encryption to the network (i.e. to the TNIs), integrity of all objects sent over the network can be guaranteed and the result is the possibility to have reduced physical security requirements for the network cable.

Figure 12 shows three hosts connected to a network, where two hosts are controlled by TNIs toward the network:

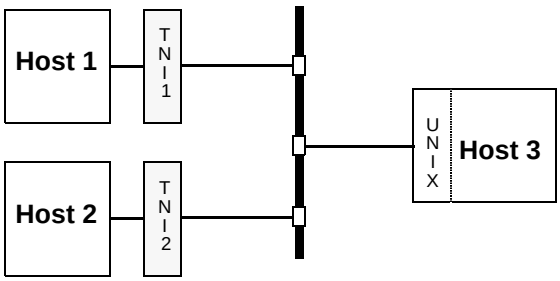

Figure 12: Two hosts with TCBs and one host relying on UNIX®

All network communication by host 1 and host 2 in figure 12, as well as all internal communication between local processes, is controlled by the TNIs (TNI1 and TNI2). When a process on host 1 wishes to send a message to a process on host 2, TNI1 checks what security levels TNI2 is allowed to handle. This has to be done before the communication is established, since a TNI should never receive messages with a higher security classification than it is allowed to handle. As a result of this, all TNIs must know in advance what security levels the other TNIs are allowed to handle, or at least must be able to figure this out, for example to have a security server which can be contacted by the TNIs. If TNI2 is working at the correct security level, TNI1 contacts TNI2 and verifies the security levels for the two processes. In this case, TNI1 trusts TNI2 to give the security level of the process and TNI2 trusts TNI1. If the communication is allowed according to the access control rules (i.e. the security policy), the communication is set up. Note that a TNI needs to make sure that it is at all times talking to the intended TNI and not to someone else, which for example can be assured by cryptographic separation.

When communicating with a system without a TNI, such as with host 3 in the figure, the TNIs must know the security classification for the hosts working without TNIs and know what kind of information the host is allowed to handle. Thus the TNIs know what security classification messages from such hosts must have, no matter what the hosts themselves claim.

If several untrusted hosts are connected to a network and the Bell-LaPadula model is used, these hosts and the network must operate in **system high mode**, i.e. they must operate at the same security level as the highest classified object they may handle or receive from the network, to prevent “write-down” or declassification of objects. Otherwise these systems would have the possibility to send and to receive messages from other untrusted hosts at their own will. Also note that nothing has been done to prevent covert channels in a system like this.

7.3.5 Fragmentation

Fragmentation is a separation mechanism based on dividing information into fragments or small pieces of information, where each fragment should contain little or no information at all. Each fragment should be small enough to be without interest because of lack of information. Fragmentation can be done in several ways with different granularity:

With **location fragmentation**, sensitive objects are divided into intelligible pieces of information where each piece is sent to its own entity for processing. For example, calculations could be done in parallel, having each bit processed by its own entity to prevent each entity to know more than randomly selected bits from the object. Objects can be fragmented on bit-level, but it might be more feasible to use fragmentation on byte-level or even on larger segments.

In **time-fragmentation**, individual pieces of the objects are randomly selected in time, and they are processed by one host. This host only receives fragments of the object (for example one bit at a time) and has no way of telling in what order these bits should be assembled to retrieve the contents of the object.

In the Delta-4 project [8], fragmentation is used to split information (such as files) into fragments and to store these fragments on randomly selected hosts. The contents of each file can be randomly reorganized and even encrypted before it is fragmented. The fragmentation method is constructed in such a way that several fragments stored on different hosts are needed to reconstruct the original contents, thus a security violation in a single host will not violate the security of a whole system. This, together with existing redundancy, will guarantee the confidentiality and integrity of objects. If an extra encryption stage before fragmentation is added, this would be an example where two separation methods are combined to further increase security, in this case cryptographic separation and fragmentation would be combined into one separation mechanism.

7.4 Communication Security Mechanisms

When objects are being transported, it is especially cumbersome to preserve integrity and confidentiality and to eliminate covert channels. When public networks or networks with little or no physical security are used, objects are very vulnerable to both active and passive attacks. An attacker can insert new messages in a network, he can modify, delay, reorder, duplicate or delete messages, etc. The use of end-to-end encryption or fragmentation of objects can, if done correctly, guarantee object integrity and confidentiality. However, there are still problems from passive attacks since at least some information can be received simply by collecting statistics on how objects are being sent between entities. Note that communication problems do not have to involve networks, but can occur within a single host, for example when local entities are sending objects to each other.

In many ways the problems of object transportation are similar to the authentication problems discussed earlier. By adding **key distribution centers** and **access control centers**, such as Kerberos, individual keys for each connection can be established and maintained in a distributed manner. The key distribution centers manage keys and distribute these keys to entities which want to be able to communicate. The access control centers check accesses to objects, e.g. to information and global resources, in the network.

7.4.1 Symmetric Cryptosystems

Both confidentiality and integrity of objects can be preserved by encryption. Before an object is transmitted, it is encrypted with a suitable encryption algorithm. If a symmetric cryptosystem such as DES is used, the same key is used for both encryption and decryption.

A major problem with this encryption algorithm is to find a way to distribute keys to entities, since all entities need one unique encryption key for each entity it needs to communicate with. Yet another complication can be to find a suitable layer for encryption within the ISO/OSI model in order to do it as efficiently and as securely as possible, since it can be done in the physical layer as well as in most higher layers. Each layer gives some advantages but also has some disadvantages.

Note that it is often possible for an attacker to guess some parts of a message. This is important to know, since it eliminates several encryption algorithms because messages often include predictable information such as frame headers, sequence numbers, etc.

7.4.2 Asymmetric Cryptosystems

If we care for only integrity or confidentiality but not both, it is possible to use an asymmetric cryptosystem (section 7.1.1). By keeping the encryption key secret and the decryption key public, it is possible to preserve object integrity. If, on the other hand, the decryption key is secret, it is possible to enforce object confidentiality. The advantage with an asymmetric cryptosystem is that it is enough to give only one private key to each entity, and still allow all entities to communicate.

To make it possible for the receiver of an object to verify the integrity of an object, it is necessary to include some redundant information. This information is used by the receiver to decide whether a message makes sense or not. If the result of the decryption always generates the result 0 or 1, it would not be possible for the receiver to verify the integrity of the object. Redundant information can be created by combining an asymmetric cryptosystem with a **checksum** which can be calculated from the contents of the object. The checksum can be encrypted with an asymmetric cryptosystem and transmitted together with the object or transmitted to the receiver over a different network. The checksum mechanism has the advantage that the object itself does not have to be encrypted, which would save a lot of time and effort in the communication process.

To select a checksum-generating algorithm is a rather complicated process. There are many requirements for such an algorithm, for example, the statistical probability that the checksums for two different texts should be equal, must be zero or almost zero. There are specialized codes for this purpose, for example the Quadratic Congruential Manipulation Detection Code [28][9]. Among others, this code detects permutations, rotations, insertions and deletions of characters within a text. Note that the checksum algorithm should not be a secret in itself. (It is also possible to use a symmetric cryptosystem such as DES to generate the encrypted checksum directly. However, in this case, the effort of generating the checksum may be as big as encrypting the whole message.)

7.4.3 Digital Signatures and Seals

Some systems offer services in order to protect the sender and/or the receiver of information. A typical service offered is the ability to afterwards prove that a specific entity has performed an action, for example transmitted or received a message.

An asymmetric cryptosystem can be used as a **digital signature** to guarantee that a specific entity has created an object or performed a specific action. When an entity creates a digital signature, it cannot later on deny having created it. The signature can be a message (for example a contract or a publicly known number) encrypted with an asymmetric cryptosystem where the encryption key is known only by the entity itself. The entity cannot later on deny having created this object because no one else would be able to perform the encryption.

The following services can be offered by a system [19]:

- Proof of origin of data (digital signature)
- Proof of original content (digital seal)
- Proof of delivery
- Proof of original content received

The first two services protect the receiver and the other two the sender. All four services are called **non-repudiation** services since neither the sender nor the receiver can deny having sent/received a message or deny the contents of a message. The mechanisms implementing this service are similar to the mechanisms described in the previous paragraphs, i.e. with the use of asymmetric cryptosystems.

7.4.4 Routing Control and Traffic Padding

The problems with information collection (passive attacks) and denial of service can be reduced by introducing mechanisms which allow entities to influence traffic routing. If entities have the possibility to bypass certain hosts or parts of a network, an entity refusing to forward messages or an unreliable part of a network can be bypassed. **Routing control** can be used to ensure that sensitive information is not sent through unreliable hosts or networks. It is also possible to use routing control in combination with fragmentation of messages, where fragments are transmitted through different networks and possibly in random order.

To make it harder to collect information from network traffic, **traffic padding** can be added to the system. Traffic padding is a mechanism which adds dummy messages which are transmitted between randomly selected hosts at random intervals. When messages need to be sent, these dummy messages are substituted with real messages, thus an attacker cannot differentiate real traffic from dummy messages on the network. Also, real messages can be sent indirectly to their final destination (i.e. through intermediate hosts) to make use of dummy messages sent between other hosts. This mechanism is often combined with padding messages to a **fixed message length** to give even less information to an observer. The disadvantage of these mechanisms is the possible loss of network bandwidth due to the transmission of dummy messages and increased message lengths. These methods can be valuable tools to prevent messages from being used as covert channels from a Trojan horse.

Another interesting solution to the problems above is to use broadcast or multicast addresses in messages. This can have the advantage of improving the chances of a message delivery.

7.4.5 Secure Protocols

A secure protocol is a protocol which is especially designed to protect the integrity and/or confidentiality of all objects that are transported. In general, it is not necessary to preserve each individual object, but merely to preserve integrity and/or confidentiality of the whole communication process. Also, provisions may be taken to limit the possibilities of passive attacks.

First, there is a need for an authentication mechanism to ensure that communication at all times is performed between the correct entities, and this authentication mechanism needs to be protected against replays from the network. Most likely, a method for distributing encryption keys to entities opening new connections is needed.

Second, a secure protocol needs to be protected against modification of packets, replays of old packets and against lost packets. Replays can be especially cumbersome, since a replay of an old message is performed with a completely valid packet (i.e. a valid checksum and correct encryption). Therefore, some kind of time stamp or sequence number that cannot be forged must be contained in all messages. Also, the encryption algorithm must be able to deal with retransmission of old packets and to deal with duplicates, i.e. it must have a mechanism to handle resynchronization.

Third, the protocol needs to be protected against denial of service attacks and loss of packets. It is necessary for the communicating parties to exchange messages at regular intervals to make sure that all messages are received.

There is a protocol, "secure IP", that conforms to the requirements above but also to the DoD military security model [21]. It is based on a normal network layer protocol (IP) but secure IP adds some security features to it, for example an options field containing the security classification level of a message. There are also rules stating how both messages containing (and not containing) security classification labels should be handled, for example what kind of messages a host is allowed to send and receive.

Secure protocols can be placed in any layer within the ISO/OSI model. Another example of a secure protocol is the OSI File Transfer Access and Management (FTAM) standard, which defines both a protocol and at the same time is a standard for a remote file service. The protocol defines how to perform access control and what attributes a file should have, for example fields indicating what encryption method is being used. Note that the use of secure protocols in a network environment can only solve some security problems but it will not guarantee the overall security of the system. An attacker who wants a specific piece of information from a database server that uses a secure protocol, may perform an indirect attack by breaking the security in one of the hosts that uses the database, and then "convince" this host to talk to the database server with the use of the secure protocol.

7.5 Detection and Recovery Mechanisms

Not all mechanisms are used to prevent security violations. An important group of security mechanisms are those used after a security violation has taken place. Detection mechanisms detect a security violation, and recovery mechanisms restore the system to its state prior to the violation. Ideally, detection mechanisms should detect a security violation immediately and it should give enough information to enable the tracery of the violation a specific entity or user.

In many cases, the knowledge of the existence of such mechanisms can be enough to prevent attacks, for example if an attacker knows that the attack will be detected (even if it is detected at a later date) and traced back to him/her, this may be reason enough for not trying the attack.

An **audit trail** is a log containing security-related events and transactions. It contains information about when, how and by whom a transaction was ordered, thus it is a valuable tool for protecting both objects and the integrity of entities. Audit trails should be used to monitor all sensitive actions, especially those actions that affects the security in the system. The audit trail should be detailed enough to make it possible to trace security violations back to individual users. It can, for example, contain digital signatures from entities ordering transactions, which can be used at a later time to verify (prove) that an entity actually did order a transaction.

8. DISCUSSION

Security for information systems is a very wide and complex subject. This paper creates a structure reflecting and describing the relations between different topics and areas of security. It also discusses the relations between security and the closely related field of dependability, and special concern is given to present a terminology which facilitates the integration of the two areas. The discussion deals with four major topics: threats, aspects of security, forms of security and mechanisms to be used when creating a secure system.

The list of security mechanisms is by no means a complete list, but the list is abstract enough to cover most type of mechanisms and still detailed enough to describe how most protection mechanisms work.

The taxonomy described can be used as an aid when designing a secure system. The various forms and aspects of security as well as the mechanisms described, can be utilized when a specification for security demands in a system is developed. It is also a step toward a consistent terminology for computer security. Both the terminology and the taxonomy shows differences against traditional views, but this is intentional and is necessary as a first step toward a terminology which can be incorporated into the field of dependability as well.

9. REFERENCES

[1] **Marshall D. Abrams, Albert B. Jeng:** Network Security: Protocol reference model and the trusted computer system evaluation criteria, *IEEE Network Magazine, April 1987 - Vol. 1, No. 2.*

[2] **Stanley R. Ames, Morrie Gasser:** Security Kernel Design and Implementation: An Introduction, *Computer, V16, n7, Jul. 83.*

[3] **T. Anderson:** Safe & Secure Computing Systems, *Blackwell Scientific Publications, ISBN 0-632-01819-4.*

[4] **Richard H. Baker:** Computer Security Handbook, 2nd Edition, *TAB Professional and Reference Books, McGraw-Hill Inc, ISBN 0-8306-7592-2, 1991.*

[5] **Steven M. Bellovin:** Security Problems in the TCP/IP Protocol Suite, *Computer Communication Review, Vol. 19, pp. 32-48, April 1989.*

[6] **Steven M. Bellovin, Michael Merritt:** Limitations of the Kerberos Authentication System, *USENIX - Winter 1991.*

[7] **T. Beth, J. Dobson, D. Gollman, A Klar:** Security Evaluation Report: Concepts and Terminology, *EISS-Universität Karslruhe / University of Newcastle upon Tyne, 1990.*

[8] **L. Blain, Y. Deswarte:** An Intrusion-tolerant Security Server for an open Distributed System, *LAAS report no 90085, March 1990, Toulouse, France.*

[9] **C.C.I.T.T.:** Recommendation X.509: The Directory - Authentication Framework, *CCITT, Nov. 1988, ISBN 92-61-03731-3.*

[10] **Flaviu Cristian:** Understand Fault-Tolerant Distributed Systems, *IBM Research Division, Almaden Research Center, San Jose, CA. June 1990.*

[11] **D. W. Davies, W. L. Price:** Security for Computer Networks, *John Wiley & Sons. ISBN 0-41-92137-8*

[12] **Department of Defence:** Trusted Computer System Evaluation Criteria ("orange book"), *CSC-STD-001-83.*

[13] **Rik Farrow:** Security For Superusers, Or How To Break The UNIX System. *UNIX/World May 1986.*

[14] **S. Garfinkel, G. Spafford:** Practical UNIX Security, *O'Reilly and Associates, ISBN 0-937175-72-2*

[15] **F. T. Grampp, R. H. Morris:** UNIX Operating System Security, *AT&T Bell Laboratories Technical Journal, vol. 63, No. 8, October 1984.*

[16] **J. H. Howard, M. L. Kazar:** Scale and Performance of a Distributed File System. *ACM Transactions on Computer Systems, Feb. 1988.*

[17] **D. K. Hsiao:** Database Security Course Module, *Database Security: Status and Prospects, Elsevier Science Publishers B.V, Holland, IFIP WG 11.3, 1989.*

[18] **International Standards Organization:** Data Processing - Open Systems Interconnection - Basic Reference Model, *ISO/IS 7498, Geneva 1983.*

[19] **International Standards Organization:** Information processing systems - Open Systems Interconnection - Basic Reference Model, part 2: Security Architecture 7498/2.

[20] **S. T. Irwin, Tom Bakey:** SURVIVAL: Tips for staying alive in a competitive industry, *Insurance Software Review, Feb/Mar 1989.*

[21] **M. St. John:** Revised IP Security Option, *RFC 1038, January 1988.*

[22] **Joseph, Mark K:** Integration problems in Fault-tolerant, secure computer design, *Dependable Computing for Critical Applications, ISBN 3-211-82249-6, 1991.*

[23] **Karilia, Arto T:** Open Systems Security - an Architectural Framework, *Business Systems R&D, Helsinki, Finland, June 30, 1991.*

[24] **Bell D, LaPadula L:** Secure Computer Systems: Mathematical Foundations and Model, *MITRE Report MTR 2547, Nov. 1973.*

[25] **J.C. Laprie:** Dependability: Basic Concepts and Terminology, *Springer-Verlag*, 1991, ISBN 3-211-82296-8**.**

[26] **Ali Mili:** An introduction to Program Fault Tolerance, *Prentice Hall, ISBN 0-13-493503-9.*

[27] **J. K. Millen:** Operating System Security Verification, *Case Studies in Mathematical Modelling, Pitmann Publishing, Mass., 1981, pp. 335-386.*

[28] **Sead Muftic:** Security Mechanisms for Computer Networks, *Ellis Horwood Ltd, England, ISBN 0-7458-0613-9.*

[29] **N. P. Nelson:** Fault-Tolerant Computing: Fundamental Concepts, *Computer, July 1990, pp. 19-25.*

[30] **Nelson, Welch, Osterhout:** Caching in the Sprite Network File System. *ACM Transactions on Computer Systems, Feb. 1988.*

[31] **D. M. Nessett:** Factors Affecting Distributed System Security, *IEEEE Symp. on Security & Privacy, 1986, pp. 204 - 222.*

[32] **Peder C. Nørgaard:** User Experience with Security in a Wide-area TCP/IP Environment, *EUUG Conference Proceedings September 1989.*

[33] **Charles P. Pfleeger:** Security In Computing. *Prentice Hall International, Inc. ISBN* 0-13-799016-2.

[34] **Dennis M. Richie:** On the Security of UNIX, *Unix Programmers's Manual, section 2, AT&T Bell Labs.*

[35] **John Rushby, Brian Randell:** A Distributed Secure System, *Computer, V.16, n 7, Jul. 1983.*

[36] **J. Saltzer:** Protection and the Control Information Sharing in MULTICS. *Communications of the ACM, vol.17, n 7, Jul. 1974.*

[37] **M. Satyanarayanan:** Integrating Security in a Large Distributed System. *ACM Transactions on Computer Systems, vol. 7, No. 3, August 1989.*

[38] **Robert W. Shirey:** Defence Data Network Security Architecture, *The MITRE Corporation, McLean, VA 22102 - 3481.*

[39] **Eugene H. Spafford:** The Internet Worm: Crises and Aftermath, *Comm. of the ACM, V. 32, n 6, June 1989.*

[40] **J. G. Steiner, C. Neuman, J. I. Schiller:** Kerberos, An Authentication Service for Open Network Systems. *Project Athena, MIT, March 30, 1988.*

[41] **Stephen T. Walker:** Network Security Overview, *IEEE 1985 Symp. on Security and Privacy, April 1985.*

[42] **Patrick H. Wood, Stephen G. Kochan:** UNIX System Security, *Hayden Book Company, Indianapolis, Indiana, 1987.*

[43] **X/Open:** X/Open Security Guide, *Prentice Hall, ISBN 0-13-972142-8.*