Software System Security

Confidentiality

Computer security rests on confidentiality, integrity, and availability

Confidentiality is the state of keeping or being kept secret or private.

Access control mechanisms support confidentiality -> transform data to make it unavailable for those not allowed to see it.

I.e.

- Personal data
- Bank data
- Company confidential data

Integrity

Integrity is the state of data or resources being whole and undivided.

Integrity includes

- data integrity (the content of the information)
- origin integrity (the source of the data, authentication).

Integrity mechanisms fall into

- prevention mechanisms and
- detection mechanisms.

Availability

Availability refers to the ability to use information or resources.

An unavailable system is at least as bad as no system at all.

Example: denial of service (DoS) attacks

Threats

A threat is a **potential** violation of security.

- The violation need not actually occur for there to be a threat.
- If it happens it is called an **attack.**
- Those who execute such actions, or cause them to be executed, are called attackers.

Four classes of threats:

- Disclosure - unauthorized access to information;
- Deception - acceptance of false data;
- Disruption - interruption or prevention of correct operation;
- Usurpation - unauthorized control of some part of a system

Policy and Mechanism

- A **security polic**y is a statement of what is, and what is not allowed.
- A **security mechanism** is a method, tool, or procedure for enforcing a security policy

Goals of Security

- Prevention means that an attack will fail.
- Detection is useful when an attack cannot be prevented.
- Recovery
    - stop the attack and to repair any damage caused by that attack
    - the system continues to function correctly during the attack

Assurance

Assurance requires specific steps to ensure that the computer will function properly.

- The sequence of steps includes detailed specfications of the desired (or undesirable) behavior;
- an analysis of the design of the hardware, software, and other components to show that the system will not violate the specifications;
- arguments or proofs that the implementation, operating procedures, and maintenance procedures will produce the desired behavior.

Specification

A specification is a (formal or informal) statement of the desired functioning of the system.

It can be

- mathematical,
- formal specification language,
- Natural language like English

The specification can be low-level, combining program code with logical and temporal relationships to specify ordering of events. The defining quality is a statement of what the system is allowed to do or what it is not allowed to do.

Design

The design of a system translates the specifications into components that will implement them.

The design is said to satisfy the specifications if, under all relevant circumstances, the design will not permit the system to violate those specifications.

Implementation

Given a design, the implementation creates a system that satisfies that design. If the design also satisfies the specifications, the implementation will also satisfy the specifications.

Correctness

A program is correct if its implementation performs as specified.

Proofs of correctness require each line of source code to be checked for mathematical correctness.

Each line is seen as a function, transforming the input (constrained by preconditions) into some output (constrained by postconditions derived from the function and the preconditions). Each routine is represented by the composition of the functions derived from the lines of code making up the routine. Like those functions, the function corresponding to the routine has inputs and outputs, constrained by preconditions and postconditionfied.

The same techniques can be applied to sets of programs and thus verify the correctness of a system.

Testing

Because formal proofs of correctness are so time-consuming testing have become widespread.

During testing, the tester executes the program (or portions of it) on data to determine if the output is what it should be and to understand how likely the program is to contain an error.

Testing techniques range from supplying input to ensure that all execution paths are exercised to introducing errors into the program and determining how they affect the output to stating specifications and testing the program to see if it satisfies the specifications.

These techniques are considerably simpler than the more formal methods, they do not provide the same degree of assurance that formal methods do. Furthermore, testing relies on test procedures and documentation, errors in either of which could invalidate the testing results.

Assurance techniques do not guarantee correctness or security, they provide a firm basis for assessing what one must trust in order to believe that a system is secure.

Their value is in eliminating possible, and common, sources of error and forcing designers to define precisely what the system is to do.

Cost-Benefit Analysis

In a complex system, the benefits of computer security are weighed against their total cost (including the additional costs incurred if the system is compromised).

If the data or resources cost less, or are of less value, than their protection, adding security mechanisms and procedures is not cost-effective because the data or resources can be reconstructed more cheaply than the protections themselves.

Risk Analysis

To determine whether an asset should be protected, and to what level, requires analysis of the potential threats against that asset and the likelihood that they will materialize.

The level of protection is a function of the probability of an attack occurring and the effects of the attack should it succeed.

If an attack is unlikely, protecting against it typically has a lower priority than protecting against a likely one. If the unlikely attack would cause long delays in the company’s production of widgets but the likely attack would be only a nuisance, then more effort should be put into preventing the unlikely attack.

The situations between these extreme cases are very subjective.

Laws and Customs

- Laws restrict the availability and use of technology and affect procedural controls. Hence, any policy and any selection of mechanisms must take into account legal considerations.

Organizational issues

Security provides no direct financial rewards to the user.

It limits losses, but it also requires the expenditure of resources that could be used elsewhere.

Who is responsible for the security at the company?

Clear chains of responsibility and power must be established,

One common problem security managers face is the lack of people trained in the area of computer security. Another problem is that knowledgeable people are overloaded with work.

Lack of resources is another common problem. Securing a system requires resources as well as people.

- It requires time to design a configuration that will provide an adequate level of security, to implement the configuration, and to administer the system.
- It requires money to purchase products that are needed to build an adequate security system or to pay someone else to design and implement security measures.
- It requires computer resources to implement and execute the security mechanisms and procedures.
- It requires training to ensure that employees understand the importance of security, how to use the security tools, how to interpret the results, and how to implement the nontechnical aspects of the security policy.

Human Issues

The heart of any security system is people. This is particularly true in computer security, which deals mainly with technological controls that can usually be bypassed by human intervention.

Most dangerous attacks come from insiders who are authorized to use the computers.

Untrained personnel also pose a threat to system security.

Many attacks come from social engineering.