NONMALICIOUS PROGRAM ERRORS

Being human, programmers and other developers make many mistakes, most of which are unintentional and nonmalicious. Many such errors cause program malfunctions but do not lead to more serious security vulnerabilities.

Malicious Code Can Do Much (Harm)

Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file.

A **Trojan horse** is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect.[1] As an example of a computer Trojan horse,

logic bomb is a class of malicious code that "detonates" or goes off when a specified condition occurs. A **time bomb** is a logic bomb whose trigger is a time or date.

A **trapdoor** or **backdoor** is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges.

A **worm** is a program that spreads copies of itself through a network. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files).

CONTROLS AGAINST PROGRAM THREATS

Developmental Controls

- Specify the system : capturing the requirements and building a model of how the system should work from the users' point of view.
- Design the system: proposing a solution to the problem described by the requirements and building a model of the solution.
- Implement the system: using the design as a blueprint for building a working solution.
- Test the system: to ensure that it meets the requirements and implements the solution as called for in the design.

- Review the system at various stages: to make sure that the end products are consistent with the specification and design models
- Document the system: users can be trained and supported
- Manage the system: to estimate what resources will be needed for development and to track when the system will be done
- Maintain the systems: tracking problems found, changes needed, and changes made, and evaluating their effects on overall quality and functionality

File Protection Mechanism

Several different types of operations can be controlled:

- **Read –** Reading from a file.
- **Write –** Writing or rewriting the file.
- **Execute –** Loading the file and after loading the execution process starts.
- **Append –** Writing the new information to the already existing file, editing must be end at the end of the existing file.
- **Delete –** Deleting the file which is of no use and using its space for the another data.
- **List –** List the name and attributes of the file.

- **AllNone Protection**
- **Group Protection**
- **Individual Permissions**
- **Persistent Permission**
- **Temporary Acquired Permission**
- **Per-Object and Per-User Protection**

User Authentication

is **a process that verifies a person's identity allowing them access to an online service, connected device, or other resource**. Authenticating users occurs differently across services as business logic and risk profiles at enterprises can vary markedly.

Trusted OS design

We say that an operating system is **trusted** if we have confidence that it provides these four services consistently and effectively.

Policy. Every system can be described by its requirements: statements of what the system should do and how it should do it. An operating system's security requirements are a set of well-defined, consistent, and implementable rules that have been clearly and unambiguously expressed.

Model. To create a trusted operating system, the designers must be confident that the proposed system will meet its requirements while protecting appropriate objects and relationships. They usually begin by constructing a model of the environment to be secured.

Design After having selected a security model, designers choose a means to implement it. Thus, the design involves both what the trusted operating system is (that is, its intended functionality) and how it is to be constructed (its implementation).

Trust. Because the operating system plays a central role in enforcing security, we (as developers and users) seek some basis (assurance) for believing that it will meet our expectations. Our trust in the system is rooted in two aspects: features (the operating system has all the necessary functionality needed to enforce the expected security policy) and assurance (the operating system has been implemented in such a way that we have confidence it will enforce the security policy correctly and effectively).