1 Introduction

The computer security problem is an adversary problem: there is an adversary who seeks to misuse the storage, processing, or transmittal of data to gain advantage. The misuse is classified as either unauthorized observation of data, unauthorized or improper modification of data, or denial of service. In denial of service misuse, the adversary seeks to prevent someone from using features of the computer system by monopolizing or tying up the necessary resources.

Thus, a complete solution to the computer security problem must meet the following three requirements:

- **Secrecy or confidentiality**: Protection of information against unauthorized disclosure
- **Integrity**: Prevention of unauthorized or improper modification of information
- **Availability**: Prevention of denial of authorized access to information or services

These three requirements arise in practically all systems. Consider a payroll database in a corporation. It is important that salaries of individual employees are not disclosed to arbitrary users of the database, salaries are modified by only those individuals that are properly authorized, and paychecks are printed on time at the end of each pay period. Similarly, in a military environment, it is important that the target of a missile is not given to an unauthorized user, the target is not arbitrarily modified, and the missile is launched when it is fired.

The computer security problem is solved by maintaining a separation between the users on one hand and the various data and computing resources on the other, thus frustrating misuse. This separation is achieved by decomposing the computer security problem into three subproblems: security policy, mechanism, and assurance. Thus, a complete solution consists of first defining a security policy, then by choosing some mechanism to enforce the policy and, finally, by assuring the soundness of both the policy and the mechanism. Not only each subproblem must be solved, it is important that each solution fit the solutions of the other two subproblems. For
example, it does not make sense to have a security policy that cannot be implemented, nor does it make sense to have a mechanism that is easily bypassed.

2 Security Policy

The security policy elaborates on each of the three generic objectives of security—secrecy, integrity, and availability—in the context of a particular system. Thus, computer security policies are used like requirements; they are the starting point in the development of any system that has security features. The security policy of a system is the basis for the choice of its protection mechanisms and the techniques used to assure its enforcement of the security policy.

Existing security policies tend to focus only on the secrecy requirement of security. Thus, these policies deal with defining what is authorized or, more simply, arriving at a satisfactory definition of the secrecy component.

The choice of a security policy with reasonable consequences is nontrivial and a separate topic in its own right. In fact, security policies are investigated through formal mathematical models. These models have shown, among other things, that the consequences of arbitrary but relatively simple security policies are undecidable and that avoiding this undecidability is nontrivial \[5,7,8\]. To read more about the formal security models, see [3].

All security policies are stated in terms of objects and subjects. This is because in reasoning about security policies, we must be careful about the distinction between users and the processes that act on behalf of the users. Users are human beings that are recognized by the system as users with a unique identity. This is achieved via identification and authentication mechanisms; the familiar example is a user identifier and password.

All system resources are abstractly lumped together as objects and, thus, all activities within a system can be viewed as sequences of operations on objects. In the relational database context, an object may be a relation, a tuple within a relation, or an attribute value within a tuple. More generally, anything that holds data may be an object, such as memory, directories, interprocess messages, network packets, I/O devices, or physical media.

A subject is an abstraction of the active entities that perform computation in the system. Thus, only subjects can access or manipulate objects. In most cases, within the system a subject is usually a process, job, or task, operating on behalf of some user, although at a higher level of abstraction users may be viewed as subjects. A user can have several subjects running in the system on his or her behalf at the same time, but each subject is associated with only a single user. This requirement is important to ensure the accountability of actions in a system.

Although the subject–object paradigm makes a clear distinction between subjects and objects (subjects are active entities, while objects are passive entities), an entity could be both a subject and an object. The only requirement is that if an entity behaves like a subject (respectively, object), it must abide by rules of the model that apply to subjects (respectively, objects).