Chapter 114: Protection (security) models & policy

Carl E. Landwehr, Naval Research Laboratory

Introduction

Assets, vulnerabilities, and threats

Protection becomes an issue for operating systems and networks when they are used to process information or control systems that represent a significant asset to someone. The value of information may vary over time, even when the information doesn't change -- tomorrow's weather forecast is more valuable than yesterday's. The value of the control may also vary: a computer system that controls a nuclear power plant represents a more significant asset than one that controls a microwave oven. We are naturally more concerned about protecting more valuable assets. The vulnerabilities of a system are its weak points -- flaws in its design or implementation that could cause it to behave in ways not intended by its builders or operators. Threats are the agents that, acting against a vulnerability, could cause such misbehavior. Threats can be external or internal to a system; they may be hostile or benign; and they may be structured or unstructured. Lightning usually represents an external, benign, unstructured threat. Random operator errors represent an internal, benign, unstructured threat, and the actions of a subverted operator might represent an internal, hostile, structured threat. Assets that have significant value are more likely to be subject to hostile, structured threats.

Implicit and explicit security policies

"Secure" means, literally, "apart from care." A system is secure if we don't have to worry about it. But exactly what guarantees do we require of a computer system? When computers began to serve several users at once, operating systems needed to protect one user's programs and files from the errors of others. In a military environment, there are requirements for multilevel security that require a system to separate different levels of classified information from users with different clearances. The security policy for a system defines what it means for that system to be "secure"; usually it concerns identification and authentication of users, authorization of actions, and accountability for actions taken. Often, security policies are implicit. Most personal computers, as delivered, do not restrict (or even identify) the individuals who use them, so there is no way to distinguish an authorized access request by a program from an unauthorized one. Nevertheless, users do expect certain properties of their machines -- for example, that running a new piece of commercial software should not cause all of their files to be deleted. When security policies are explicit they often specify the behavior desired of the system as a whole, so the implications for operating system behavior may not be obvious.

Varieties of models

We use "model" here in the sense of the engineer, rather than the logician. A model of a system represents aspects of it that concern us without representing it in complete detail. An architect's model of a house will accurately reflect the shape and spatial relationships of the parts of the dwelling, but it will not include a functioning furnace or be built of materials whose strength is proportional to those to be used in construction. Security models have been used for two fairly distinct purposes: to represent the security policy a system is intended to enforce and to represent the mechanisms in a computer system that are intended to enforce the policy. Policy models typically represent the flow of authorizations and information; mechanism models represent data structures found in operating systems. Some models combine both purposes.

Underlying Principles

Concerns about how a system will behave motivate the development of a security policy for a system. A security model represents the security policy, the structure of a system, or both in a form that allows us to reason about the policy and its enforcement.

SECURITY CONCERNS AND SECURITY POLICY

Information security concerns conventionally focus on preserving the confidentiality, integrity, and availability of information for those authorized access to it. These concerns are usually considered equivalent to preventing unauthorized disclosure, modification, or withholding (denial of service) of protected information. Parker suggests that, particularly in a business context, possible loss scenarios should be the basis for information security concerns [Parker, 1994]. He pairs possession with confidentiality, authenticity with integrity, and utility with denial of service as "elements" of information security. For example, if information is encrypted and the key becomes unavailable, its utility may be lost (assuming cryptanalysis is infeasible) even though it remains available. Authenticity, but not necessarily integrity, is compromised if pirated software is altered so it appears to have been produced by someone else. Possession, but not confidentiality, may be lost if a laptop containing the only copy of encrypted data is stolen and held for ransom.

Within the area of confidentiality, security concerns may also depend on the kind or amount of information disclosed to an individual. A firm that provides accounting or legal services to competing clients may need to prevent employees who have seen information concerning one of its clients from viewing similar information from other, competing clients.

This is sometimes known as building a "Chinese Wall" between sets of employees exposed to different information [Brewer and Nash, 1989]. Preventing the undesired aggregation of information can also be a concern. A bank teller may be permitted to have access to information on individual accounts, but not to summaries derived from all accounts, since these might reveal the bank's overall financial status. In a military context, information about the readiness of an individual unit may not be considered highly sensitive, while the collective readiness of all units would be.

Security concerns can extend beyond the information in a system to the integrity of the system itself. Errant or malicious programs can overwrite data or programs, reformat disks, or, if the computer is part of a control system, damage property and injure people. Programming errors have, in fact, contributed to fatal accidents [Leveson and Turner, 1993]. Concerns such as these typically require attention at the system design level, but their effects can be seen at the level of the operating system.

Security policy is a plan or course of action intended to satisfy a set of security concerns. History teaches that security concerns are unlikely to be satisfied unless they are identified explicitly and a corresponding policy is implemented. A security policy for an organization typically addresses many different security disciplines, including personnel security, administrative security, physical security, and so on. This chapter is concerned with security policies that can be implemented within the scope of a computer system and, more specifically, by an operating system. This focus narrows our concerns significantly.

Access and Authorization Policies

Access and authorization policies specify what behaviors are permitted in terms of the types of access (read, write, execute) one entity (user, process, file) is authorized to have to another. For example, "a user is permitted read, write, or execute access to any file that user creates," or "a process that is permitted to read a file may grant that permission to any other process." Such a policy is called discretionary if it leaves the user with the ability to make some choices (*i.e.*, with some discretion) in administering the policy. A mandatory policy, conversely, prohibits the user from making decisions. The use of mandatory access control policies is typically associated with military applications, where mandatory security dictates that a user not be permitted to read a file classified above his/her clearance. However, the requirement that every user of a commercial system present a valid password in order to gain access to it is also a form of mandatory security policy. Policies often include both mandatory and discretionary parts: discretionary policy permits the Secret-cleared user to decide whether or not to permit other users with Secret or higher clearances to read a Secret file; mandatory policy prohibits the Secret user from placing Secret-labeled data where it could be read by users not cleared for it. Role-based security policies assign privileges to job functions (roles), rather than individuals; the user inherits the privileges of the job assigned. This approach can simplify the management of privileges, since a new privilege associated with a particular job is automatically inherited by all users who occupy the corresponding role.

Information Flow Policies

Restricting the flow of information is often the purpose of access control policies: a class of users is denied access to some physical object (a file, for example) because information contained in that file is not intended to flow to users in that class. Recognizing this fact, and stating the policy directly in terms of constraints on information flow, can simplify and clarify the resulting security requirements. The classic example is the formulation of military security policy as a lattice of security classes, with the requirement that information can only flow upwards in the lattice. Commercial applications seem more often to involve total isolation of different classes (as in the Chinese Wall policy), but when divisions of two competing firms cooperate on a particular project or proposal, for example, it may be more natural to characterize security policy in terms of permitted and prohibited information flows, rather than accesses. Relationships between component suppliers and system integrators may also lead to security policies formulated most naturally in this way.

Integrity and Availability Policies

Although access control and information flow policies have attracted the most attention and have the best developed models and supporting structures in operating systems and networks, two other kinds of security policies are worth noting here: integrity and availability policies. "Integrity" has many meanings to many groups. In the context of operating systems and networks, protecting integrity usually means protecting data from unauthorized or improper modification. Information flow and access control policies may incorporate integrity concerns. For example, an information flow policy that restricts the flow of information among security classes might be reformulated to restrict the flow of information among integrity classes, where a higher integrity class represents information that is more likely to be accurate or correct, and a lower integrity class represents less reliable information. To preserve the integrity of reliable information, we might prevent low integrity information from flowing to higher integrity classes. Similarly, we might protect integrity by controlling read and write accesses to objects with different integrity labels.

Having crucial information available when needed is often more important than preserving its confidentiality or its integrity. Of course, corrupted information may be of little use even if it is available, and relying on compromised information (*e.g.*, a crypto key) can have unpleasant consequences. In the context of operating systems, protecting the availability of information usually means controlling access to resources so that one process cannot indefinitely prevent others from gaining access to it. Systems that serve several users at once usually include precautions against accidental loss of resources but rarely protect against malicious attacks aimed at denial of service. Policies for system availability are rarely distinguished from those for reliability, which typically consider only accidental failures, not malicious behavior.

SECURITY MODELS

Security policy is at root informal, representing a plan or course of action. A security model attempts to capture the security-related behavior and/or structure of a system more formally. It is an abstraction of a system from the standpoint of security and may omit other important aspects that do not affect its security properties. Most security models have focused on confidentiality, rather than integrity or availability.

Policy Models and System Models

A security model may focus either on a system's security policy (*e.g.*, "no individual can view a message for which he lacks clearance") , leaving open questions of system design and implementation, or on the system's structure and operations, its mechanisms (*e.g.*, "a process running on behalf of a given user can only open a file for reading if there is an entry for that user in the file's access control list with an access mode of *read*").   Security policy amounts to the top level security requirement for a system, so we consider security policy models as models of system requirements. Some models attempt to represent both the requirements and the mechanisms; if such a model is used to guide a development, it will constrain the set of possible set of implementations more tightly than a model of the requirements only. It's possible to view the distinction between models of requirement and mechanism simply as a difference in the level of system design being modeled. An application-based model may define and refer to entities visible to users (users, messages, files, input and output devices), while a mechanism-based model may focus on processes, storage blocks, file locks, and so forth. Difficulties arise when security requirements at the application level are difficult to realize with the mechanisms available at lower levels.

Proving Properties of Models

Whether a particular model focuses on the domain of requirements or the domain of implementations, it should provide a structure with a clear relationship to that domain, so that reasoning about the more simple structure of the model carries over to the relevant real-world domain. A good road map provides a simplified representation of a network of streets and roads that permits a driver to reason correctly about possible paths to a particular destination and, consequently, which way to turn at the next intersection. A successful model of a set of security requirements should permit its user to reason about how a system satisfying those requirements will behave. A security model of the implementation domain should support reasoning about the security-related behavior of the modeled class of mechanisms.

Because security models are artificial -- the model consists of abstractions whose properties the modeler is at liberty to define -- it is possible to reason about them more rigorously than one can reason about the real world directly. This is a benefit, in that we can state properties we desire of the model (*e.g.*, that its behavior satisfies certain security constraints) and, potentially, determine with certainty whether the property holds for the model or not. But caution is required: a property that we prove holds for the model still may not hold in the real world, if the intended relationships between the model and the real world are violated. If a road map omits, say, a tunnel of which cartographers were unaware, or that was built after the map was printed, the driver may think he cannot cross a river without going over a bridge, when in fact he could. Likewise, proving that a user can never gain access to a file according to a particular security model is of interest only if the system as implemented conforms to that security model.

The access matrix model [Lampson, 1971, Graham and Denning, 1972] is based on abstraction of operating system protection structures. Its simplicity and generality have led to its wide application over many years, and it continues to be a reference point for system developers and users. It includes three primary components: a set of passive objects (files, devices or other entities implemented by the operating system), a set of active subjects, which may manipulate the objects, and a set of rules governing the manipulation of objects by subjects, including transformations to the access matrix itself. A subject is a process and a domain (a set of constraints within which the process may access certain objects); every subject is also an object, since it can be read or otherwise manipulated by other subjects. The access matrix is a rectangular array with one row per subject and one column per object. The entry for a particular row and column reflects the permitted modes of access between the corresponding subject and object, typically a subset of {read, write, append, execute}. Although the access matrix is never implemented literally as a matrix (it would be very sparse), it provides a basis for reasoning about accesses within a system. *[EDITOR: refer to chapter 115 (Authorization, Access controls, and Intrusion Detection) here?]*

Harrison, Ruzzo, and Ullman [Harrison, Ruzzo, and Ullman, 1976] developed a particular formalization of the access matrix model (the HRU model) and identified the safety property for it: given an initial state for the access matrix and a set of commands, can a particular subject gain a given right *a* to a particular object? A configuration is safe for *a* if there is no sequence of transformations that will cause *a* to be added to an element of the access matrix that does not already contain it. They were able to show that the safety problem for the HRU model is undecidable in general, although it is decidable if the access matrix transformation rules are "mono-operational" -- *i.e.*, if each request to change the access matrix results in only a single change of the form "add/remove a row/column" or "add/remove access right a for a single cell in the matrix." Unfortunately, mono-operational systems exclude most practical security policies, since, for example, they exclude policies that grant the creator of an object particular access rights to that object. The take-grant model [Jones, Lipton, and Snyder 1976], which represents subjects, objects, and accesses with directed graphs, has a linear time algorithm for safety, but although it has been explored formally in considerable detail [Snyder, 1981, Bishop, 1988], it has seen little practical application. Sandhu has developed the schematic protection model and typed access matrix models [Sandhu, 1988, 1992a, 1992b], based more closely on the HRU approach, and has been able to narrow the gap between models for which safety can be proven and those reflecting practical access control policies.

Composability of Security Properties

If we define a security property for a system, show that the property holds for a model of that system, and show that the model accurately represents the system, we can be confident that the behavior of that system will enforce that security property. Unfortunately, we cannot be sure that if we connect two such systems together, the combination will enforce the same security property that the systems did individually. This is referred to as the composability problem: the combination will only be secure if the security property in question is composable, even if each system individually enforces the specified security property. As computers are increasingly interconnected, security threats and vulnerabilities arising from those interconnections are growing. Finding useful, composable, security properties that would provide a basis for connecting systems without introducing new vulnerabilities is a current research topic.

ENFORCEMENT

The concept of a monitor as a system component that controls allocation and use of a particular resource led to the notion of a **reference monitor** as a key component that would control the accesses subjects could gain to objects [Anderson, 1972]. When a subject attempts to gain access to an object (*e.g.*, to open a file for reading), the reference monitor checks the access matrix to see whether the requested access is allowed; if not, the request is denied. Three properties are required of a reference monitor: it must be invoked on every access attempt, it must be tamperproof, and it must be small enough to be subject to thorough testing or verification (*i.e.*, it must be correct). A **security kernel**, the hardware and software realization of the reference monitor, is intended to include all of the security enforcement mechanisms in an operating system and nothing else.

General purpose, efficient security kernels proved hard to realize [Landwehr, 1983], and partly as a consequence, the notion of a **Trusted Computing Base** (TCB) developed. The TCB, like the security kernel, is intended to include all of the security enforcement mechanisms in the system (thus the TCB boundary is the system's security perimeter) but the TCB is not required to be minimal -- it may include non-security-relevant software as well. The Trusted Computer System Evaluation Criteria (TCSEC) [Department of Defense, 1985] are organized around the TCB concept; they were developed to permit different commercial computer system products to be ranked in terms of how well they realize it. To be ranked at any of the top four of the seven levels established by the TCSEC, the system's documentation must include a security policy model and some degree of evidence that the implementation in fact enforces the constraints of that model.

Best Practices

A system may in fact be quite secure, but if we cannot convince others of this fact, they will be unlikely to trust valuable assets to be placed under its control. Security models can be used to define what security means for a given system, to guide the system's designers and developers so that an implementation provides the intended security properties, and to convince others that the system is trustworthy. In this section, we first review basic access and authorization models and then security policies and models concerned with the three traditional security properties: confidentiality, integrity, and availability. For each property, we will describe different models developed to permit a determination as to whether a system that corresponds to the model preserves the property of interest.

ACCESS AND AUTHORIZATION MODELS

The fundamental model for access control is the access matrix model, described above. The modes of access registered in the access matrix at any instant define the authorized accesses between subjects and objects. Because this model corresponds so well both to intuitive notions of controlling the access of people to material objects and to the protection structures implemented by many operating systems, it is widely known and used. We can say that the contents of the access matrix encodes the system's security policy, and that it is up to the operating system's mechanisms to enforce this policy. Separating policy and mechanism is usually desirable in a system design.

But if we want to reason about whether or not some particular subject can ever read or write some particular object, or whether information can leak across some system boundary, the access matrix model is less than ideal. First, the undecidability of the safety problem limits general results. Second, and closely related, is the problem of Trojan horse programs. Programs, not humans, initiate the operations that alter the access matrix, and a program executed by a given subject can generally exercise all the rights of that subject, including granting those rights to other subjects. This means that humans who invoke any Trojan horse program are vulnerable to the complete redistribution of their access rights (within the bounds of the authorization scheme), and, unless we are willing to assert that no user will invoke such a program, necessitates a worst case assumption about propagation of access rights.

To address these two problems, Sandhu has introduced a version of strong typing to access matrices [Sandhu, 1992b]. Each subject and object is assigned an immutable type at its creation, and, with some additional constraints on the structure of commands in the ternary monotonic typed access matrix model, Sandhu shows that its safety is decidable in polynomial time.

A third problem that arises with the access matrix model is the practical difficulty of including all of the relevant objects in the access matrix. In most operating systems, processes, files, and devices may have these kinds of access controls associated with them, but there are often other shared, user-visible data structures and resources (such as file names, file locks, system clocks) that do not. If the security policy calls for constraining information flow from one set of objects to another in an environment that may include Trojan horses, these uncontrolled resources can act as covert channels [Lampson, 1973], permitting violations of the security policy even though the controls listed in the access matrix are correctly enforced. The shared resource matrix methodology [Kemmerer, 1983] provides a practical approach to identifying such channels.

CONFIDENTIALITY MODELS

Because protecting sensitive information against disclosure is a major concern of military security, and because protecting confidentiality seems to be a more tractable problem than maintaining integrity and availability, confidentiality models dominate security modeling research literature. The best known and most widely applied of these is the Bell-LaPadula model, which is closely based on the access matrix model, to which it adds the military security lattice, some additional data structures, and a set of definitions and rules [Bell and LaPadula, 1975]. Its goal is to establish a formal structure in which even a Trojan horse with access to sensitive data would be unable to leak those data to a user or program that did not already have access to them.

Military Security Structure

Military security classifications are characterized by a linearly ordered set of sensitivity levels (*e.g.*, confidential, secret, top secret) together with a set of compartments. Compartments typically correspond to information on a certain topic or from a certain source that requires special protection. The security label for a particular document or paragraph will include a sensitivity level and a (possibly empty) set of compartments. To be authorized access to a particular document, the user must have a clearance for both the sensitivity level and all of the compartments listed in the label. A sensitivity level and a set of compartments denotes a security level. In general, information is allowed to flow from one security level to another if the sensitivity level of the source is lower than that of the destination (*e.g.*, from confidential to secret) and if the compartment set of the destination level includes all of the compartments of the source level. Recognizing that (unclassified, no compartments) is the lowest possible security level, from which any information can flow upwards, and (top secret, all compartments) is the highest level, which can receive information from any other level, we see that the set of security levels together with the flow relation forms a lattice [Denning, 1976].

Bell LaPadula Model

The Bell-LaPadula model associates a security level with each subject and object and identifies a set of access modes that subjects may have to objects, including read, write, and execute. When a subject attempts to read an object, the access matrix is checked, but an additional check is required to determine whether the subject's security level is greater than or equal to that of the object to be read (since it is permissible for a secret level subject to read a confidential level object, for example). Unless the access matrix check (referred to as discretionary access control or DAC) and the security level check for reading, (referred to as the simple security property of mandatory access control (MAC), or "no read-up") both succeed, access is denied. If a subject requests write access to an object, the request is checked against the access matrix for DAC permission, and it is checked to be sure that the security level of the object to be written is equal to the current security level of the subject (referred to as the *-property, or "no write-down"). This check prevents, for example, a secret level subject from writing secret data into a confidential level object.

The argument that systems conforming to this model preserve security proceeds by identifying a secure state of a system as one in which no users have accesses to objects that would violate the security policy. The so-called Basic Security Theorem (BST) asserts that if a system starts in a secure state and proceeds to new states only through a set of possible transitions, each of which guarantees termination in a secure state if it is initiated from a secure state, then the system will never reach an insecure state. Two noteworthy objections have been raised about this formalization of confidentiality. First, the BST holds regardless of the characterization of a secure state -- if we defined a secure state to any state in which all objects could be read or written by all subjects, we could still prove the BST. Consequently, we cannot rely on the fact that the BST has been proven to assure ourselves that we have found an intuitively correct definition of secure state [McLean, 1985]. Second, some intuitively insecure systems in fact satisfy the particular definition of security given by Bell and LaPadula [McLean, 1990]. It is possible to construct an operation that satisfies the stated security properties but changes the security levels of subjects and objects between states. Thus, while no accesses violate the security policy in either the state before or after the transition, the transition itself has changed the system state in a way that is intuitively insecure, yet not prohibited. These problems have not prevented the model's use as engineering guidance in many system developments, however.

A major challenge in applying this model rigorously is the covert channel problem mentioned earlier. Although the Bell-LaPadula model is defined only in terms of access controls, its purpose is to enforce an information flow policy -- information should not flow from higher security levels to lower ones by any path. Finding all of the potential data structures and hardware devices in a system that could be used to convey information between security domains is a difficult but necessary part of assuring that an implementation based on access control correctly realizes an information flow policy.

Non-Interference Model

The non-interference model has also been used in system developments as a basis for showing that a system enforces confidentiality ([Goguen and Meseguer, 1982, 1984], based on earlier work by Feiertag [Feiertag, *et. al., 1977*]). This model focuses on capturing the confidentiality requirement rather than the mechanisms required of the implementation. By focusing on the properties required to preserve confidentiality at the system specification level, it achieves a cleaner and more intuitive definition of security and permits the implementer more flexibility in realizing the requirements.

This model describes a system in terms of the sequence of system calls (trace) invoked by users at different security levels. System output is defined as a function of the system's input history, the user requesting the output, and the user's clearance. If a user with a given clearance sees the same output at any point in the system's history that he would see if none of the inputs from users with higher clearances had occurred (*i.e.*, if their inputs were purged from the trace), then the system is secure. That is, the higher level inputs do not interfere in any way with the behavior observed by lower level users; hence the model's name.

Goguen and Meseguer developed a set of "unwinding conditions" to help apply this model in a state machine context more commonly used in system specification and development. Proving that the unwinding conditions hold for the state machine's transition functions establishes non-interference. This approach has the benefit that covert storage channels are identified in the normal course of developing the unwinding conditions and proving non-interference; no separate effort is required to identify them as it is in developments based on the Bell-LaPadula model.

Discussion

Applying the access matrix model, Bell-LaPadula model, or non-interference model to an operating system development is likely to lead to a stronger system with fewer security flaws than if security is not given explicit consideration. But there are several issues left open by current models and practices. One we have already touched on: covert channels. Sometimes covert channels are shrugged off as a consideration of only academic concern, and, given certain assumptions about the threats a system is subject to and the requirements it must meet, sometimes this attitude may be appropriate. However, if strict separation is desired between information in different domains, then the concern is not merely covert channels, it is channels of any sort that permit information to cross the boundary between domains. Assuming that some such channels exist, the question becomes: what is their capacity, and what information can be transferred over them? The relevant models of information, and information flow, have their roots in Shannon's information theory [Shannon and Weaver, 1963]. Some recent efforts have attempted to model known covert channels in real system designs using information theory [Kang, Moskowitz, and Lee 1995], but this is still a research topic.

All of the models discussed in this section so far are for deterministic systems, where the current state and input uniquely determine the next state. This basis may be appropriate for modeling single computers, but is not fully satisfactory if the security model is to be used to analyze either a network or a system specification, where the current state and input may only determine a set of possible next states (*i.e.* non-deterministic systems). Substantial research effort has been devoted to developing nondeterministic, including probabilistic, models for properties like non-interference, but none of these has been demonstrated in a practical system development. For a good summary, see [McLean, 1994].

Note that the security properties intended to be enforced by Bell-LaPadula and non-interference type models are fundamentally policies of strict separation -- no information is supposed to flow between incommensurate security domains, and information may flow at most upward from lower security levels to higher ones. When models like these began to be developed in the early to mid 1970's, hardware was expensive, and sharing hardware across security levels seemed essential. With the drastic declines in the cost of hardware in the last decade, separating information (if isolation is what is required) by storing it on physically separate computer systems is a much more realistic approach than it once was. Of course, if absolute separation or simple upward flow does not meet the security requirements of the application, physical separation may not work so well, after all. In this case, not only walls, but also gates may be required. The best use of security models like Bell-LaPadula and non-intereference in this case may be to model the "walls," and modeling the gates -- paths between domains that are sometimes opened intentionally -- may require a different approach that can assure that only authorized information type passes through. Indeed, the Bell-LaPadula model countenances such flows via the mechanism of a "trusted subject" that is permitted to violate the *-property but is trusted not to compromise security. Establishing trust in a subject is an activity outside the security model. An early but still relevant approach to organizing a system based on physical separation and encryption rather than trusted software is described by Rushby and Randell [Rushby and Randell, 1983]. More recent work to develop a multilevel secure database system based on physical separation and data replication, and to extend the architecture to provide an MLS distributed computing service is described by Froscher, *et al.* [Froscher, *et al.* 1994].

INTEGRITY MODELS

Integrity models reflect the policies and mechanisms intended to assure that data and programs are modified only in an authorized manner. The access matrix model includes controls on writing to subjects and objects, and these can be used for this purpose. For example, only the system administrator may be granted write access to a file used to record the list of authorized users of the system. Of course, if the system administrator ever invokes a Trojan horse program, it could pass this privilege to any other user, because these controls are discretionary.

Lattice-based Integrity

To limit the ability of Trojan horses to alter data that might be unclassified, yet critical to system operation, Biba [Biba, 1977] introduced a lattice of integrity levels to the Bell-LaPadula model corresponding to the security lattice. The controls imposed on subjects and objects with respect to their integrity levels are intended to prevent high integrity data from being modified based on low integrity data. The integrity level of an object is not necessarily related to its security level. For example, an airline's current flight schedule is readable by all, but should only be written by authorized subjects.

Conversely, actions of any user might append records to the system's audit file, but ability to read the audit file might be limited to system administrators.

In Biba's "strict integrity policy" a subject can only "read up" in integrity and "write down". The system administrator and the list of authorized users would both be assigned high integrity levels, so that even if a Trojan horse granted a lower level user access to the critical file, that user could not modify the file. There is still a vulnerability, however, since the Trojan horse could modify the file directly, instead of trying to pass permissions to another user.

Several systems built to meet military security requirements have included both security and integrity levels for subjects and objects, and some practical policies that use integrity levels have been proposed [Lipner 1982, Shirley and Schell, 1981, Schell and Denning, 1986]. These policies aim primarily at enforcing configuration controls as described in the example above, rather than assessing whether the output of a particular program or activity is of higher integrity than another. Typical integrity levels for a system might be user, programmer, and administrator (from low to high). Integrity compartments can be used to isolate different functional areas. A properly constructed integrity lattice can be used to model conflict-of-interest classes and enforce a Chinese Wall policy, as well [Sandhu,1993].

Commercial Integrity

An integrity model based on traditional controls in commercial systems to prevent fraud and assure accuracy in records has also been proposed [Clark and Wilson, 1987]. This informal model defines Unconstrained Data Items (UDIs), Constrained Data Items (CDIs), Transformation Procedures (TPs), Integrity Verification Procedures (IVPs), and Logs and incorporates the notions of separation of duties and certification. UDIs reflect raw, unchecked information that may be entered into the system. CDIs can be altered only by TPs, which may use UDIs and other CDIs as input. A change to a CDI also appends a record to a Log file. IVPs are run periodically against all of the CDIs to assure that they are in a valid state. There is a set of certification assertions (for humans to assure) and enforcement assertions (for machines to assure), including, for example, that the system must maintain the list of relations of the form $(UserID, TP_i, (CDI_a, CDI_b, ...))$ relating a user, a TP, and the data objects that TP may reference on behalf of that user. The system must ensure that only executions described in one of these relations are performed. A human must assure that this list of relations is subjected to the separation of duty requirement. The Clark-Wilson model provides highly intuitive structure in which particular applications can be developed, but assuring that an implementation conforms syntactically to the structure of the Clark-Wilson model by itself can do little to assure that the system behaves as intended. In this way, the model is similar to the access matrix model: it provides a structure for expressing a policy; a human must determine whether enforcement of that particular policy will have the desired results.

AVAILABILITY MODELS

Assuring "availability" or providing "assured service" is often included in lists of security concerns, perhaps because those phrases are brief and positive, but our real concern is to prevent service from being denied by a malicious attack. Work in the fields of dependability and fault-tolerance addresses availability in the positive sense, where system availability may be measured as mean time between failures divided by the sum of mean time to failure and mean time to repair. But these calculations are usually based on failures occurring randomly according to some model and not as the result of malicious actions.

To represent the relevant aspects of preventing denial of service, it is necessary to characterize the agreement between the service provider and the user. Further, the type of resource or service being provided will have to enter into the formulation of the model [Dobson, 1991].

In the 1980's, Yu and Gligor developed a specification and verification method for preventing denial of service in the absence of failures and integrity violations [Yu and Gligor, 1990]. A user agreement in their model is a set of constraints on the sequences of calls the user may make to the shared service. A user who violates the agreement cannot expect to obtain the intended service, but should not be able to interfere with the service guarantees made to other users who abide by their agreements. The Finite Waiting Time (FWT) policy they propose incorporates the user agreements, a fairness policy (a user will not be blocked forever if that user has many opportunities to make progress), and a simultaneity policy (a user will eventually have all the opportunities needed to make progress, provided that the user agreements can be satisfied). They apply their temporal-logic-based specification and verification method to a general resource allocator and prove that all users eventually make progress and receive intended service.

Building on this work, Millen has defined a denial-of-service protection base (DPB) analogous to the concept of a trusted computing base (TCB) [Millen, 1993, 1994]. The TCB is that part of a system that is responsible for the enforcement of confidentiality and integrity properties; the DPB must offer those services whose loss would be viewed as a denial of service. Millen develops this concept as a state machine, drawing on earlier models of resource allocation in operating systems. A DPB is characterized by a resource monitor, a waiting time policy, and user agreements, and it must satisfy two conditions: each benign process will make progress in accordance with the waiting time policy (progress), and no non-CPU resource is revoked from a benign process until its time requirement is zero (patience). A resource monitor is built on a set of processes and resource types, each of which has a fixed capacity. Processes may request allocations of particular amounts of particular resources, including the CPU. The state of the resource monitor is given by the current allocation matrix, which determines how many units of each resource type are allocated to each process, the time vector, the space requirement matrix, and the time requirement matrix. The time vector records the real time at which each process was activitated or deactivated. The explicit treatment of time in the model permits the definition of waiting time policies based on Maximum Waiting Time (MWT) and Probabilistic Waiting Times (PWT) in addition to the FWT policy of Yu and Gligor.

Despite this progress in developing denial of service models, it must be acknowledged that these are still research topics. None of these have been applied to production systems as yet, though with increasing concern about denial of service attacks, particularly against networks, these models, or their successors, may soon find application.

Research Issues and Summary

Models can represent a protection or security policy, the mechanisms that implement a policy, or both. They can provide a basis for reasoning about how a system is intended to behave or how it behaves in fact. Models have been used effectively to guide the design and implementation of systems, to help users understand how a system is intended to work, and to raise the confidence of evaluators and certifiers that a system is safe to operate.

Unless a system is known to contain or control a significant asset, it may not be the target of malicious threats and its vulnerabilities may remain untried. Recognizing security concerns, including loss of possession or confidentiality, integrity or authenticity, and utility or service, is the first step in forming a security policy. As a course of action intended to satisfy a set of security concerns, a security policy is usually stated informally. It may define mandatory controls, discretionary controls, or controls based on users' roles. It may call for two-person control of particular functions or operations in order to secure a system against an attack by a single individual, or for a "Chinese Wall" to separate employees with knowledge of competing clients' records. An access control policy constrains the objects a user may read, write, or execute, while an information flow policy controls whether the data read from one object is permitted to affect data written to another.

Models are typically stated more formally than policies. It is sometimes possible to prove that a formally expressed model has some desired property, such as barring certain kinds of accesses or certain undesired information flows. Protection models represent the protected objects in a system, how users or subjects (their proxies in the computer system) may request access to them, how access decisions are made, and how the rules governing access decisions may be altered. The access matrix model is the primary example of a protection model. Although the safety property (determining whether or not a given subject can ever gain a particular access to a given object) is undecidable for this model, its ability to represent a wide variety of security policies and the degree to which its structures match those commonly found in operating systems have kept it in wide use.

Security models have a broader scope than protection models; they may represent the security policy requirements of maintaining confidentiality, integrity, and availability in the face of malicious attacks, possibly including hostile software. The Bell-LaPadula model was developed to capture confidentiality requirements as reflected in the lattice of security levels defined by military classification schemes. Systems built to conform to it implement both mandatory access controls, which normal users cannot modify, and discretionary controls, like those represented in the access matrix model, which users can change. The mandatory controls are intended to preserve confidentiality of information across security levels even if users execute Trojan horse programs. The non-interference model has similar objectives to the Bell-LaPadula model, but it represents a system in terms of its response to a sequence of calls (a trace). If a user at a given security level sees the same system behavior whether all inputs from higher level users are purged from the system trace or not, then the non-interference property holds and the system is considered secure. Systems have been built both to conform to the non-interference model and to the Bell-LaPadula model. Covert channels, mechanisms that conform to the security model but permit information to be transmitted in violation of security policy, are an implementation issue for both models, although a system based on the non-interference should not require the separate analysis of storage channels that a Bell-LaPadula based implementations do.

Integrity models have received much less attention than confidentiality models, not because people are less concerned about integrity, but because it has many, often application-specific, interpretations. Biba's integrity model defines a lattice of integrity levels analogous to the lattice of security levels and controls the modification of data in a way analogous to the Bell-LaPadula model controls its disclosure. Implementations of this model have been used to enforce configuration control policies. The less formal Clark-Wilson model reflects integrity controls found in commercial systems organized around transactions.

Least developed are availability models, better termed models for denial of service prevention. Both the Yu-Gligor and Millen models require formalizing agreements between the user and the system as to what constitutes delivery of service. Without such agreements the basis for asserting that service has (or has not) been denied is lacking. Neither model, however, has seen significant application.

Confidentiality and integrity controls are typically implemented by some form of reference monitor, which validates each reference by a subject to an object. A Trusted Computing Base (TCB) comprises all of the security enforcement mechanisms in a computer system, and the system's security perimeter is the boundary of the TCB. Millen's model for preventing denial of service introduces the notion of a denial of service protection base (DPB) analogous to the TCB.

All of the models discussed here have been developed to reflect the requirements and behavior of a single computer system, yet we live in a world of increasingly interconnected machines. Not surprisingly, many current research issues in this field stem from this difference. These include the development of models that can be applied to nondeterministic specifications, the identification of security properties that are "composable" (*i.e.*, if the property holds for two components, then it will also hold for a correctly connected combination of the two), and the development of principles for allocating security requirements among components (decomposition).

Current models also tend to reflect absolute, rather than relative, notions of security. A policy model prescribes that information flow either is or is not permitted between two entities or security levels. In practice, some limited amount of information flow may be tolerable. The tools of information theory are beginning to be applied to quantifying such flows in practical systems.

The representation of security policies oriented toward commercial applications, where integrity and denial of service concerns typically outweigh confidentiality, should lead to improved models in those areas. Assuring that a system actually enforces a specified policy, and that it continues to do so through a series of releases and upgrades, continues to be a difficult problem. Finally, distributed architectures that support freely moving software "agents" that may ship back not only images and text, but also programs in one form or another to be executed, raise interesting security questions as to how one can be sure that such programs cannot cause damage when executed.

Defining Terms

Access matrix: matrix with a row for each subject and a column for each subject and each object. Records authorized access modes (such as read, write, execute) subjects may have to objects. Represents the current access authorization policy, but never implemented directly as an array because it would be too sparse.

Covert channel: a channel used to pass information not originally intended for that purpose. Usually applied to channels that permit two cooperating processes to pass information in violation of the security policy, but without violating the system's access controls.   For example, if one process suddenly begins to impose heavy performance demands on a system, it may cause responses to other processes' requests to be delayed. This change in delay can be used as a signal.

Discretionary access controls (DAC): Access controls that may be modified by ordinary subjects. The access controls implemented by typical file systems are discretionary, in that the owner of the file can grant other users/subjects the right to read, write, or execute the file. A Trojan horse executed by an unwitting subject can alter the discretionary access controls on the associated user's files.

Mandatory access controls (MAC): Access controls that cannot be modified by ordinary subjects. In military security, mandatory access controls are intended to prevent a user (or a Trojan horse executing on that user's behalf) from moving secret data to an unclassified file, where uncleared users might read it. Special "trusted" subjects used by system administrators may be permitted to violate these constraints in limited ways.

Object: a passive entity within a system, such as a file, but any entity that can be used to record or transmit data may be considered an object. The mapping of the entities implemented by the system to the subjects and objects of a security model is a crucial step in assuring that the system's behavior will conform to the model.

Role-based access controls: access controls associated a user's job-related responsibilities, rather than with individual identity or security clearance. Like mandatory controls, in that a subject may not arbitrarily pass role-based privileges to other users. Provides a convenient way to organize authorizations in relation to job assignments, so that if one individual must assume another's job temporarily or permanently, the individual inherits the role-based authorizations associated with the new assignment.

Reference monitor: a system component that enforces system security policy by checking all accesses initiated by subjects to objects and permitting only those that are consistent with the policy. To be effective, the reference monitor must be tamperproof, enforced on every access, and it must not make mistakes.

Safety problem: to determine whether or not a given subject may at some time obtain some specific access authorization to some specified object. For many straightforward security models, the general safety problem is undecidable.

Security policy: a plan or course of action intended to satisfy a set of security concerns. In the context of operating systems, a security policy typically seeks to ensure that data are protected against unauthorized disclosure, modification, or withholding. Usually expressed informally, in natural language, they are sometimes formalized. Security policies should be explicit, but often are not.

Security model: an abstraction representing security policy, system mechanisms that can enforce it, or both. May be used to inform users how the system behaves, guide the system's design and implementation, and reason about its properties.

Subject: an active entity within a system, such as a process or device. Forms a unit for assigning authorizations, usually associated with (acting on behalf of) some user. Subjects are also objects.

Trusted computing base (TCB): the collection of all mechanisms in a system that enforce security policy. The TCB boundary represents the security perimeter of the system.

References

Anderson, J. P. 1972. Computer security technology planning study, Vol I. ESD-TR-73-51, ESD/AFSC, (NTIS AD-758 206) Hanscom AFB, Bedford, MA.

Bell, D. E. and LaPadula, L. J. 1975. Secure computer system: unified exposition and MULTICS interpretation. MTR-2997, MITRE Corp., Bedford, MA.

Biba, K. J. 1977. Integrity considerations for secure computer systems. MTR-3153 (NTIS AD A039324), MITRE Corp., Bedford, MA.

Bishop, M. 1988. Theft of information in the take-grant protection model. In *Computer Security Foundations Workshop*, p. 194-218. IEEE CS Press, Los Alamitos, CA.

Brewer, D. F. C., and Nash, M. J. 1989. The Chinese Wall security policy. In *Proc. 1989 IEEE Symp. Security and Privacy*, p. 215-228. IEEE CS Press, Los Alamitos, CA.

Clark, D. D. and Wilson, D. R. 1987. A comparison of commercial and military security policies. In *Proc. 1987 IEEE Symp. Security and Privacy*, p. 184-194. IEEE CS Press, Los Alamitos, CA.

Denning, D. E. 1976. A lattice model of secure information flow. *Comm. ACM* 19(5):236-243.

Dobson, J. 1992. Information and denial of service. In *Database Security V, IFIP Trans A-6*, ed. C. Landwehr and S. Jajodia, p. 21-46. North-Holland, New York, NY.

Department of Defense, U.S. 1985. *Trusted Computer System Evaluation Criteria.* DoD 5200.28-STD, Fort Meade, MD.

Feiertag, R. J., Levitt, K. N., and Robinson, L. Proving multilevel security of a system design. In *Proc. 6th ACM Symp. on Operating Systems Principles, ACM SIGOPS Operating System Rev*. 11(5):57-65.

Froscher, J. N., Kang, M. H., McDermott, J., Costich, O., and Landwehr, C. E. 1994. A practical approach to high assurance multilevel secure computing service. In *Proc. Tenth Annual Computer Security Applications Conf.*, p.2-11. IEEE CS Press, Los Alamitos, CA.

Goguen, J. and Meseguer, J. 1982. Security policies and security models. In *Proc. 1982 IEEE Symp. on Security and Privacy*, p. 11-20. IEEE CS Press, Los Alamitos, CA.

Goguen, J. and Meseguer, J. 1984. Unwinding and inference control. In *Proc. 1984 IEEE Symp. on Security and Privacy*, p. 75-86. IEEE CS Press, Los Alamitos, CA.

Graham, G. S., and Denning, P. J. 1972. Protection -- principles and practice. In *Proc. 1972 AFIPS Spring Jt. Computer Conf*, p. 417-429. AFIPS Press, Arlington VA.

Harrison, M. A. , Ruzzo, W. L. and Ullman, J. D. Protection in operating systems. *Comm. ACM* 19 (8): 461-471.

Jones, A. K., Lipton, R. J., and Snyder, L. 1976. A linear time algorithm for deciding security. In *Proc. 17th IEEE Symp. on the Foundations of Computer Sci.* p. 337-366. Houston, TX.

Kang, M. H., Moskowitz, I. S. and Lee, D.C. 1995. A network version of the Pump. In. *Proc. of 1995 IEEE Symp. on Security and Privacy,* p. 144-154. IEEE CS Press, Los Alamitos, CA.

Kemmerer, R. A. 1983. Shared resource matrix methodology: an approach to identifying storage and timing channels. *ACM Trans. on Comp. Sys.* 1 (3):256-277.

Lampson, B. W. 1971. Protection. In *5th Princeton Symp. Information Sci. and Sys.*, p. 437-443. Reprinted in *ACM Op. Sys. Rev.* 8(1)18-24.

Lampson, B. W. 1973. A note on the confinement problem. *Comm. ACM* 16(10):613-615.

Landwehr, C. E. 1983. Best available technologies for computer security. *IEEE Computer* 16(7):86-100.

Leveson, N., and Turner, C. S. 1993. An investigation of the Therac-25 accidents. *IEEE Computer* 26(7):18-41.

Lipner, S. B. 1982. Nondiscretionary controls for commercial applications. In *Proc. 1982 IEEE Symp. on Security and Privacy*, p. 2-10. IEEE CS Press, Los Alamitos, CA.

McLean, J. 1985. A comment on the 'basic security theorem' of Bell and LaPadula. *Information Proc. Lett.* 20(2):67-70.

McLean, J. 1990. The specification and modeling of computer security. *IEEE Computer* 23(1):9-16

McLean, J. 1994. Security models. In *Encyclopedia of Software Engineering*, ed. J. Marciniak, p. 1136-1145. Wiley Press., New York, NY

Millen, J. K.1993. A resource allocation model for denial of service protection. *J. Computer Security* 2(2,3)89-106.

Millen, J. K. 1994. Denial of service: a perspective. In *Dependable Computing for Critical Applications 4*, ed. F. Cristian, G. LeLann, T. Lunt, p. 91-93. Springer Verlag, New York, NY.

Parker, D. B. 1994. Demonstrating the elements of information security with threats. In *Proc. 17th Nat'l Comp. Sec. Conf.,* p.421-430. NIST, Gaithersburg, MD.

Rushby, J. M. and Randell, B. 1983. A distributed secure system. *IEEE Computer* 16(7):55-67.

Sandhu, R. S. 1988. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *J. ACM* 35(2):404-432.

Sandhu, R. S. 1992a. Expressive power of the schematic protection model. *J. Computer Security* 1(1):59-98.

Sandhu, R. S. 1992b. The typed access matrix model. In *Proc. 1992 IEEE Symp. on Res. in Security and Privacy*, p. 122-136. IEEE CS Press, Los Alamitos, CA.

Sandhu, R. S. 1993. Lattice-based access control models. *IEEE Computer* 26(11):9-19.

Schell, R. R., and Denning, D. E. 1986. Integrity in trusted database systems. In *Proc. 9th Nat'l Comp. Sec. Conf.,* p. 30-36. NIST, Gaithersburg, MD.

Shannon, C. E., and Weaver, W. 1963. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL.

Shirley, L. J., and Schell, R. R. 1981. Mechanism sufficiency validation by assignment. In *Proc. 1981 IEEE Symp. on Security and Privacy*, p. 26-32. IEEE CS Press, Los Alamitos, CA.

Snyder, L. 1981. Theft and conspiracy in the take-grant model. *J. of Computer and Sys. Sci.* 23(3):333-347.

Yu, C.-F. and Gligor, V. G. 1990. A specification and verification method for preventing denial of service. *IEEE Trans. on Software Engineering* 16(6):581-592.

Further Information

Pfleeger's textbook, *Security in Computing*, provides a comprehensive introduction. Denning's *Cryptography and Data Security* is still a valuable source, and the most comprehensive treatment of how to build an operating system to meet security requirements is Gasser's *Building A Secure Operating System*. Those new to the field may also find *Computers at Risk*, a report available from the National Research Council, a good introduction to computer security issues generally; it includes quite a bit of tutorial material along with its research recommendations..

Much of the key literature in this field is published in the proceedings of annual conferences. The proceedings of the IEEE Symposium on Security and Privacy, the Annual Computer Security Applications Conference, the smaller IEEE Workshop on the Foundations of Computer Security, the European Symposium on Research in Computer Security (ESORICS), and the ACM's Conference on Computer and Communications Security will reflect the latest research results and applications related to security modeling. Relevant literature can also be found in the *Computer Security Journal, IEEE Transactions on Software Engineering,* and *Computers and Security*.

Finally, national and international standards for evaluating the security of computer systems (the U.S. *Trusted Computer System Evaluation Criteria*, the *Canadian Trusted Computer Product Evaluation Criteria*, and the European *Information Technology Security Evaluation Criteria*),
contain relevant information for developers of secure systems.