Essay 2

Concepts and Terminology for Computer Security

Donald L. Brinkley and Roger R. Schell

This essay introduces many of the concepts and terms most important in gaining an understanding of computer security. It focuses on techniques for achieving access control within computer systems and networks.

The essay begins by defining what is meant by computer security and describing why it is important to constrain the definition to protection that can be meaningfully provided with a significant degree of assurance within computer systems. The theory of computer security — the reference monitor concept — is introduced next through an analogy with security concepts from the world of people and sensitive documents.

Next, the essay develops the presentation of the theory by introducing concepts and terms related to the security policy. Distinctions between discretionary and nondiscretionary access control policies are provided, and supporting policies are introduced. Techniques used for building a secure system based on the principles of the theory are presented, along with methods of usefully verifying the security of a system. The security kernel is presented as a useful, high-assurance realization of the reference monitor concept, and the principles behind designing and implementing one from scratch are discussed. Feasible improvements to the security of an existing operating system, as well as fundamental limitations on those improvements, are described next.

Finally, the reference monitor concept is applied to networks, and cryptography and access control are shown to be useful partners.

This essay concerns concepts and terminology relevant to computer security. However, it is not a glossary. (A glossary typically does not make very interesting reading from beginning to end, and interesting reading is one of our goals for this essay.) This essay certainly does not define all

concepts and terminology relevant to computer security; nor does it address concepts and terminology for communication security and related communication networking technology. It does address concepts and terms that we consider to be the most critical to gain a fundamental understanding of computer security technology — that is, the theory of this technology and something of its implementation.

Our approach in this essay is to focus primarily on explaining concepts critical to understanding computer security. For the basis of the communication about the concepts, we use a set of terms that have been consistently used over a period of time. Alternate terms have been used for some of the concepts we present, and in some cases, the alternate terms are being promulgated by other individuals and/or organizations. In contrast, in Essay 6, we have used terminology specific to a particular document [TCSE85] and to a specific organization (the US Department of Defense). However, in this essay, we have tried to remain clear of a specific set of organizational "utterances," instead preferring to use a set of widely and historically accepted terminology. Specific sources for some of the terms are given where the source is thought to be historically significant.

The next few sections focus on identifying the domain of discourse for the concepts and terminology discussed in the rest of the essay. They clarify what we mean by just one term — computer security.

Considerations for computer security

There are many characterizations of computer security. The one we use is related to the term information technology security. Information technology security is defined in a document [ITSE91] created by the European Community, which has gained some recent international acceptance. The document [ITSE91] defines information technology (IT) security to include the following:

- Confidentiality. Prevention of unauthorized disclosure of information.
- Integrity. Prevention of unauthorized modification of information.
- Availability. Prevention of unauthorized withholding of information or resources.

Essay 1 describes four broad areas of computer misuse: theft of computational resources, disruption of computational services, unauthorized information disclosure, and unauthorized information modification. These four areas correspond to threats to IT security. The first two categories correspond to threats to availability; the third corresponds to a threat to confidentiality; and the fourth to the integrity of the information. (Note that in this essay, theft of computational resources is considered a threat

to availability of resources since it fundamentally results in the withholding of the stolen resources from those who are paying to use them.)

Integrity, which is also traditionally referred to as data integrity [TNI87], means that information is modified only by those who have the right to do so. However, integrity has meanings other than the meaning used here. These alternate meanings vary greatly from such a broad definition as "soundness" to the definition of system integrity, meaning that the hardware and software generally operate as expected. Program integrity means that programs can be invoked only by programs that are lower in integrity. This arrangement is intended to prevent corruption (that is, by unauthorized modification) of higher integrity programs [SHIR81] by lower integrity programs (for example, by viruses or other Trojan horses that might be in them). It has been shown [SCHE86] that program integrity is just a special case of the data integrity described here. Although in other contexts integrity may be used differently, throughout this essay, integrity is used with the first meaning given above — that of data integrity. Information is modified only by those who have the right to do so.

Distinctions among availability, confidentiality, and integrity

When considering the costs and benefits of protection against the list of threats to IT security given above, the distinctions among confidentiality and integrity and other system properties such as availability are important.

Availability differs in kind from the other two components of IT security. One difference acknowledged in the definition of IT security given in the EC document [ITSE91] is that availability pertains to both information and resources, such as computer systems themselves. On the other hand, confidentiality and integrity pertain only to information itself.

In a further distinction from availability, consider that confidentiality and integrity can be enforced by preventing illicit access to the information under protection (that is, access control in the computer context). In contrast, availability cannot be provided by access control within a computer system. Rather, the key objective of availability is that information or resources should not be withheld [ITSE91].

This distinction has very fundamental implications for the protection feasible against threats to availability, in contrast to the protection feasible against threats to confidentiality and integrity. This is easy to see, since, for example, a process may in general consume resources in a manner that may prevent other processes from accessing those resources when needed. The observation that a "runaway process" can waste resources, even in a system which implements access controls, was made by Butler Lampson as early as 1971 [LAMP71]. Twenty years later, Lampson [LAMP91] stated more flatly that access controls provide a foundation for confidentiality and integrity, but are less useful for availability.

We see then that there are very many things that can affect the availability of a system; in fact, it is not possible to identify all the factors that may affect availability. It is the unboundedness of the possible causes of a loss of availability that leads to the conclusion that it is not possible to verify to a high degree of assurance that a system possesses the quality of availability. However, it is possible to verify to a high degree of assurance that a system possesses qualities of confidentiality and integrity through the dependable enforcement of access controls.

Furthermore, consider the problem of malicious software. Essay 1 characterized the growth of malicious software. It is clear that as the use of commercial off-the-shelf software products or any other software of unknown pedigree grows, so do the opportunities for insertion of malicious software. This is because there are now many more points in the software life cycle, including during distribution as well as development, at which malicious software can be incorporated. However, even in the face of malicious software, we can obtain, for a reasonable cost, the benefit of meaningful assurance that a useful form of access control will continue to be enforced (as we see in a later section on mandatory access control policy). In contrast, current technology does not allow us to obtain at any cost, in the face of malicious software, this kind of assurance for other system properties such as availability. This is not because one cannot design a system that enhances availability, but rather because one cannot be sure the system will meet any particular level of availability in the face of malicious software. We shall see later that this means it is possible to provide a reference monitor for confidentiality and integrity, but not for availability.

The reason for this is the existence of a very basic distinction between confidentiality and integrity and other system properties, such as availability. The distinction is that confidentiality and integrity can be characterized in terms of properties that are precisely defined, global, and persistent. Confidentiality and integrity can be specified for a particular system in a way that allows one to know, beyond the shadow of a doubt, whether or not the system enforces those properties.

We understand that this is an incredibly strong statement which may be surprising to some. However, there is a set of mathematical tools in computer science that gives one the confidence to make such a statement. The characteristic ability to specify these properties for a particular system in a manner that allows one to positively know that they are enforced is known in the jargon of computer science as “being computable.” It is not essential that a reader of this essay understand computability to understand the remainder of the essay. However, the next paragraph offers a very brief discussion of the implications of the computability of confidentiality and integrity for those readers who are interested. Others may wish to skip to the next section.

As implied by the above statements, confidentiality and integrity can be specified for a particular system such that whether or not that system

enforces those properties is computable. Being computable means basically that one can specify an algorithm that can be used in a mechanical way to determine the result. This is particularly significant since computers can only execute algorithms and can therefore only dependably perform computable functions! This means that one can program a computer in such a way as to dependably determine whether it enforces confidentiality and integrity. On the other hand, whether a given system meets criteria of availability, reliability, safety, and other such properties is fundamentally "noncomputable," meaning that it is impossible to determine whether a computer's program enforces these properties, given their existing definitions. In fact, it is generally noncomputable to determine whether an arbitrary protection system enforces particular properties [HARR76]. It is fortunate that a particularly useful form of confidentiality and integrity (for example, mandatory access control) constitutes a special case whose enforcement within a protection system can be proven, as will be discussed subsequently in this essay.

Meaning of computer security

In the real world of information that people care about, it is highly beneficial to treat confidentiality and integrity separately from other system properties, including availability. Information can be very dependably protected from unauthorized modification or disclosure in the face of a large range of threats; that is, confidentiality and integrity can be provided with high assurance of enforcement. Availability cannot. If confidentiality and integrity were rolled into the same class as availability, an important and very sharp distinction between assurance that is feasible in the two cases would be lost. We feel that this would not serve well the readers of this essay who could benefit from the ability to provide access control to information with a high degree of assurance, despite the lesser assurance possible for other properties such as availability.

To clarify the importance of assurance, recall the danger described in Essay 1 that was associated with misplaced trust in a supposedly "secure" system — Enigma. The danger of misplaced trust in technology, of false assurance in the Enigma case, was a serious contributor to the loss of World War II by the Germans. False assurance is a danger that is avoidable by only trusting technology that is demonstrably trustworthy.

Because of these fundamental differences, we say that confidentiality and integrity are the two components of IT security in a computer system that make up computer security (that is, computer security is the subset of IT security that addresses security of information in a computer against threats to confidentiality and integrity, but which does not address availability). Therefore, computer security, as used in this essay, may be provided by the methods used for access control within a computer system.

The remainder of this essay is divided into major sections that present concepts and terminology related to

- the theory of computer security,
- an important aspect of that theory — the security policy,
- methods of building a secure system based on the principles of the theory, and
- application of the theory to networks.

Theory of computer security

As noted above, the threats to computer security can be countered by providing access control over information on a computer to ensure that only specifically authorized users are allowed access. What we desire is a set of methods that make it possible to build a relatively small part of the system in such a way that one can even allow a clever attacker who uses malicious software to build the rest of the system and its applications, and it will still be secure. The theory of computer security gives us this.

Understanding computer security involves understanding three fundamental notions:

1. a security policy, stating the laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information;
2. the functionality of internal mechanisms to enforce that security policy; and
3. assurance that the mechanisms do enforce the security policy.

Now we introduce these three important notions and describe how they pertain to the reference monitor concept, which provides a set of principles that can be applied to the design or selection of security features and to their implementation in ways that afford a high degree of resistance to malicious software. We begin with an example from the world of people and sensitive documents to illustrate the requirements of any information security system. We then introduce the reference monitor concept as we apply it in designing secure computer systems.

An example: Protecting sensitive documents. If we had a collection of extremely sensitive documents — perhaps corporate plans and strategies or classified national security information — we might go to extreme lengths to protect that collection. Thinking about the measures that we might take to provide such protection will help us find an intuitive basis for the reference monitor concept.

Restricting access. Since we are talking about documents (presumably ink on paper), we can most naturally think about locking them up. So we buy something like a bank vault to hold our little library of priceless secrets. But the documents still have to be used, so we have to provide a way for authorized people to get at them and read them. Now we have to put a door in our vault and provide some set of controls over who can and who cannot go in.

We can place a guard post in front of our vault door and staff it with a team of extremely vigilant and trustworthy guards. These guards can surely exercise control over who goes in and who goes out, but they will need some set of criteria for determining who is authorized for such access. We can solve this problem by providing the guards with a list that specifies only those individuals we have authorized (in the national security case, those we have cleared) for access to the vault and its secrets. Now the guards know who may and who may not enter.

We are not finished, though. For when an individual shows up at the guard post and requests entry, the guards need some way to check that the person is not claiming a fake identity. We might simply rely on the guards' powers of recognition, or we can invent a variety of measures to provide the guards with the information they need. We can give each authorized individual a badge or pass and direct the guards to check the badge against the appearance of a valid badge. Perhaps we can store each individual's photograph, or even fingerprints, and associate them in the authorization list with individuals' names. This information is necessary to authenticate the identification of the individual in a reliable way. We can invent schemes of almost limitless cost and complexity to help the guards assure themselves that they are admitting only authorized people to the vault.

Finally, we might want to check up on the guards to make sure that only those users on the authorization list are being admitted to the vault to use the document collection — to ensure individual accountability for the guards' work. So we can add to our basic protection scheme a log that must be signed by both guard and visitor to give us a clear record of visits to our vault.

Of course, we must not only have a good security system; we must also implement it correctly. If a guard is subject to subversion or if our vault has walls of paper rather than steel, the security we provide will not be very effective. The extent to which we must worry about such matters will depend on the sensitivity of the information and on the threat we perceive. Perhaps we will put moderately sensitive documents in a locked room with an unarmed guard, and very sensitive ones in a real bank-style vault with armed guards.

The basic scheme outlined here is not too different from some that are actually used to protect very sensitive documents. If the users of the documents are at remote locations and too busy to come to our vault,

perhaps we will send the guards to them. Then we have a system similar to that used to handle "Ultra" information before and during World War II [WINT74]. The point is that the basic scheme is simple, comprehensible, and secure.

When access rights vary. Our scenario of a vault full of sensitive documents differs from reality in (at least) one very important respect. When people or organizations go to such lengths to protect sensitive information, it is unlikely that they will simply put it in a room and give authorized visitors unrestricted access to the room. Rather, different people are likely to have access to different documents, and the document protection system will be required to recognize and enforce this sort of distinction. We may also want the document protection system to enforce some control over the use (or misuse) of the documents by authorized users. We will consider some of these issues in the paragraphs below.

In thinking about a document library in a vault that enforces "fine-grained" protection, we can at least start with the basic concepts that were introduced above. A would-be user will appear at the reading room, identify himself (or herself), and have his identity checked by the guards on duty. Now, however, the user will not simply be admitted to the library for unrestricted access. Instead, he will request access to a specific document or set of documents. The guard will check the user's access in some sort of list and, assuming all is in order, give the user both the documents requested and a place to work on them. Perhaps our library or reading room is divided into individual carrels to which authorized users take their documents. If several people must work together, they may be assigned a closed conference room in which they may work with documents that all are authorized to see.

This extension is a very crucial one. For now, instead of admitting one or more users for unconstrained access to the entire collection, our library grants individual users access to individual documents based on their authorization. Not only that, we also have a mechanism (the carrels or reading room) for ensuring that the access rights of individuals are enforced and that a user reading document A is prevented from gaining (inadvertently or deliberately) access to document B. The access rights are defined by the authorization list and enforced on a document-by-document basis. Some of the documents may have, associated with them on the authorization list, user access rights that are identical with those associated with other documents in the collection. These equivalence classes of document access rights define a notion called access class (also referred to outside this essay by other names, including "classification," "clearance," "security level," and "security class"). An access class is an equivalence class for the sensitivity of information and the authorization of people who share common access rights to the information in that class. An important observation about access classes is that the notion

provides the basis for "fine-grained" protection, such as we are discussing. A document may have more or less sensitivity than another, or its sensitivity may differ from the other's in a noncomparable way, such as might be the case for unrelated documents in totally different fields.

In defining both our basic and enhanced scenarios, we have ignored the question, "Where does the authorization list come from?" We cannot yet deal fully with this question, but presumably the same authority that established the library also defined a mechanism by which some people can establish or change the list. To do so, they have to communicate the updated list to the guard force in some manner, and the mechanism that they use to identify themselves to the guards is probably similar to that used by the ordinary users of the library. It is even possible that some of the users of the library are themselves allowed to modify the authorization lists for some subset of the documents in the library and that the rules enforced by the guards handle this.

We have also ignored until now the practical question of what users of the library do with the information that they have access to. One possibility is that they leave their notes, extracts, and so on in the library. In this scheme, each user may be assigned a file folder for his or her notes, and the folder may be locked up by the guard force from one visit to the next. If a user takes information away from the library, the guards will probably attempt to check that the information can legitimately be removed, and perhaps that its access class is marked on the copy (so the user will assume due care in handling it). It is entirely possible that only a few people will be allowed to remove any written information from our library and then only under controlled circumstances.

If our library is to enforce access restriction at the level of the individual document, it can also collect a more detailed record of users' accesses to documents. The log that we mentioned above can be expanded to include documents accessed and individual users' actions.

The library scenario outlined here may sound unlikely. However, some government classified document libraries work almost exactly this way. The mode of operation outlined is not terribly inconvenient once the users and guards become accustomed to it. Such formal libraries typically do a good job of protecting the information entrusted to them, while making it available to the people who need to work with it. We shall see that these libraries also provide a fairly good model for the reference monitor that is implemented in a secure computer system.

The reference monitor. The reference monitor provides the underlying "security theory" for conceptualizing the idea of protection, thereby permitting one to focus attention only on those aspects of the system that are relevant to security. As we shall see, the reference monitor concept for the computer applies equally well to the design of a document library like the one we have just discussed.

The reference monitor [ANDE72] is an abstraction that allows active entities called subjects to make reference to passive entities called objects, based on a set of current access authorizations. The reference monitor is interposed between the subjects and objects. The reference monitor makes reference to an authorization database and reports information used to support an audit trail (similar to the "log" described above) that records operations which have been attempted or allowed.

At an abstract level, the reference monitor supports two classes of functions: reference functions and authorization functions [SCHE74]. Both are controlled by the current access authorization data in the authorization database. The authorization functions allow subjects to change the authorizations in the authorization database. The reference functions control the ability to access information. The utility of the reference monitor concept is independent of the specific rules that make up the access control policy. That is, the reference monitor is not defined by the access control policy, nor does the reference monitor define the access control policy.

The reference functions are defined in terms of only two generic access modes — observe and modify. The equivalents of these abstract access modes in a computer are read and write; therefore, we will use these terms. These are the only access modes for which one can be certain of the enforcement of access control; that is, these are the only access modes for which enforcement of access control policy can be verified. Read and write are fundamentally the only two types of access to computer memory, since, at the level of the hardware "chips" that implement the computer, even operations such as instruction execution begin as read and/or write operations. These two access modes provide the basis for describing the rules for access (that is, the access control policy or the access control aspects of the security policy.

With the following examples we try to clarify why other less primitive modes of access used in computers are not suitable for defining the access control policy. For an access mode to be suitable for this role, one must be able to verify that access control policy rules that are specified in terms of the particular access mode are enforced. For example, some computers support an append access mode. One might wish to build a system to enforce access control policy rules that allow a subject to append some objects but not to read or write them. One would like to be able to verify that the system enforces those rules. However, at the most primitive level in a computer, append relies on a read of some control information to determine where to write the information being appended. Thus, it would not be possible to build a system in such a way as to allow one to verify that access control policy rules that allow one to append but not to read or write will not result in undesired read or write accesses.

As another example, consider instruction execution as an access mode for defining access control policy. For execution to be suitable for this purpose, the following must be true: If an access control policy states that

execution of an object (for example, a program file) is authorized for a particular subject, but read access to that object is not authorized for that subject, we must be able to verify that read access is not possible. However, it is easy to see a specific case in which the access control policy cannot be enforced — it is not generally possible to know that executing a program (which is desired to be "execute-only") will not "leak" information from the program and thus allow undesired read access. D.E. Denning [DENN76, DENN82] has described ways in which executing a program may result in information leaking out of the program. This means that it is not possible to specify that read access is not permitted for an object to which execute access is permitted. Therefore, execute access is not sufficiently primitive to define access control policy in a verifiable way.

It should now be clear how a reference monitor implementation in a computer is related to the document library that we described above. In the document library, the users are our subjects or active entities. They make access to passive documents that correspond to the objects of the reference monitor. The authorization list that defines access to the library itself governs what subjects are known to the library.

Note, as a detail, that the action taken by the guards in the library to authenticate the identification of the individual seeking entry to the library is not itself a function of the reference monitor. Rather, it is a trusted function which is implemented outside the reference monitor. Another trusted function that may be implemented outside the reference monitor is the construction of the audit trail (mentioned above) from information reported by the reference monitor. Recall that the reference monitor contains only reference functions and authorization functions. In a later section we give additional information about the roles of authentication and audit in supporting the reference monitor's functions.

The reference monitor's authorization database corresponds to the library's augmented authorization list that identifies which users may see each document. The reference monitor's reliance exclusively on the two access modes — read and write — corresponds to the library guards' exclusive reliance on controls for what documents users are allowed to read and what notes users may remove from the library. Fortunately, read and write mean the same thing in the computer that they do in the document library. In the library, as in the reference monitor, there is an authorization function that changes the authorization database, and there are reference functions for reading or writing documents. As noted, the library, like the reference monitor, can generate data for an audit trail that reflects those operations that have occurred or been attempted. The guards, walls, doors, and internal partitions (carrels, reading rooms, and so on) of the library are all reflected by the abstraction of the reference monitor.

The reference monitor implementation in a computer system must meet a set of requirements that are also met by components of our document

library. These requirements were first identified by J.P. Anderson [ANDE72] and have been historically referred to as completeness, isolation, and verifiability:

- Completeness. The reference monitor must be invoked on every reference by a subject to an object.
- Isolation. The reference monitor and its database must be protected from unauthorized alteration.
- Verifiability. The reference monitor must be small, well-structured, simple, and understandable so that it can be completely analyzed, tested, and verified to perform its functions properly.

A review of the document library against these three requirements for a reference monitor will be instructive.

As to completeness, we presume that a user of the library cannot gain access to the collection by walking through a wall or around a guard. Note, however, that the guards do not necessarily have to watch the user directly through every moment of his or her use of a document. Our library is designed so that a user in a carrel with a document is still adequately restricted from gaining unauthorized access to other documents.

As to isolation, the library must be designed so that an interloper cannot replace a guard, drill though a wall, or replace the authorization database or other key reference monitor databases.

Finally, the procedures of the library must be simple enough so that they can be reviewed or inspected, thus meeting the requirement for verifiability. If the library system allows a user, for example, to check out a document at one desk and then carry it across a parking lot unobserved to get to a reading room, there is adequate opportunity for mischief, even though all the doors are locked and all the guards who are present are conscientious. The design of the security procedures themselves must be simple and sound, or the provision of more guards and thicker walls will be useless.

The reference monitor and the computer system. Before we leave this introduction of the reference monitor concept, we will tie it to the world of computer systems, and then to the classes of computer misuse techniques that we introduced in Essay 1.

The correspondence between reference monitor components and components of the computer system is reasonably clear: The subjects are the active entities in the computer system that operate on information on behalf of the system's users. The subjects are processes executing in a particular domain (see below for definition) in a computer system (that is, a <process, domain> pair). Most of the subjects are acting out the wishes of an individual whose identification has been authenticated by passing something like a password, using some means of reliable communication

between the individual and the portion of the system performing the identification. The means of ensuring reliable communication between a human and the portion of the system performing identification (and certain other functions such as security administration) is called a trusted path. The topics of identification, authentication, and trusted path are explored more fully in a later section.

The objects hold the information that the subjects may access. A domain of a process is defined to be the set of objects that the process currently has the right to access according to each access mode. As noted above, two primitive access modes, read and write, are the basis for describing the access control policy. While we shall be concerned with many kinds of objects in general, we can think of objects as well-defined portions of memory within the computer, such as segments. Files, records, and other types of information repositories can be built from these primitive objects, but access control is provided by the reference monitor on the basis of the primitive objects over which it has total control. As mentioned earlier, the reference monitor controls access to them by controlling the primitive operations on them — the ability to read and write them.

There is another type of resource in the computer that needs to be tied to the reference monitor concept but that we have not yet mentioned — the device or communication channel. For clarity, we will include communication channel within the notion of device and use this term throughout. A device is the means whereby information is imported to or exported from the computer system — that is, it is the means for input/output. Note that by devices, we mean things that are actually under the control of and logically part of a computer system (for example, a controller connected to the computer's bus or a disk drive). We do not mean a separate "dumb" peripheral unit such as a dumb terminal or dumb printer, and not the actual storage media such as a tape or disk platter. Devices may be considered objects under certain circumstances, but they must be considered subjects under other circumstances. We will return to the topic of devices in a later section, when we discuss networks.

The authorization database specifies those circumstances under which a subject may or may not gain access to objects. There are many ways of specifying authorization in a computer system. We can think of authorization databases associated with each object in the computer system (called a "list-oriented" implementation [SALT75, WILK72]) or with each subject (called a "ticket-oriented" or "capabilities" implementation). Regardless of how authorization is represented, the reference monitor ensures that only authorized accesses occur.

The audit trail records what security-relevant operations have actually occurred in the computer system. These include introduction of objects into the domain of a process acting on behalf of a user (for example, file open), deletion of objects, and so on. For each security-relevant event captured in the audit trail, the audit record includes such information as

the date and time of the event, the user who initiated the event, the type of event, and success or failure of the event. Note that while the reference monitor generates some of the information for the audit trail, it may not be the only source for audit trail information.

Finally, the reference monitor itself is that most primitive portion of the computer system that we rely on to control access. For the purposes of this essay, we shall think of implementing the reference monitor with a subset of a computer's operating system and hardware. We shall find that, to be efficient, the operating system software needs the assistance of computer hardware that is well suited to the task of providing security.

This last suggestion — that we can implement the reference monitor with a subset of a computer's operating system and hardware — will be especially important in our discussions of secure systems. A security kernel is defined as the hardware and software that implement the reference monitor. (In a specific context where the hardware is fixed, security kernel is sometimes used in reference to just the software.) The implication of the term security kernel is that we can design a hardware/software mechanism that meets exactly the requirements for a reference monitor. In particular, such a mechanism must be complete, isolated, and verifiable. While a computer operating system of the usual sort may attempt to meet the reference monitor requirements to some extent, it will normally fall short to some degree. Only by building a mechanism that is explicitly designed to meet the reference monitor requirements can we achieve a high degree of assurance in the security of a computer system. No alternative technical foundation has yet been identified.

Using the reference monitor. We can now turn, as promised, to the classes of computer misuse techniques introduced in Essay 1. The first class of computer misuse techniques resulting in unauthorized disclosure or modification is human error. This class can best be countered by a program of security consciousness; intensive user education; frequent training, retraining, and reminders; and conscientious system administration and operation. The reference monitor can prevent some forms of this class of misuse through the enforcement of access control using access classes. For example, an operator may be prevented from accidentally mounting the wrong tape if the access class of the tape does not meet the requirements specified in the access control policy enforced by the reference monitor. However, the reference monitor most often does not help or hinder this class of misuse.

If we are concerned about the second class, user abuse of authority, we must design a mechanism that meets our security requirements at the user interface and attempts to constrain the users or detect those times when they go astray. Implementing some of the reference monitor functions in an application program may be appropriate in these cases, though this would not give us a verifiable reference monitor. A functional

implementation of some of the reference monitor may be sufficient in this case since, by the definition of this class of abuse given in Essay 1, our irresponsible user is not involved in probing (or else that user's actions would belong in a different class of computer misuse techniques). Therefore, we know that this irresponsible user we have hypothesized will not write a program to bypass the controls we have supplied.

If we are concerned about the threat of direct probing or probing with malicious software, we can probably implement our reference monitor functions in the operating system or within a subset of the operating system. Of course, we may have to pay more attention to security features than have most operating systems today, and we shall also have to use and manage the system with considerable attention to security. But an operating system that is designed with considerable attention to security and very well managed can be quite effective against probing.

If we are worried about penetration or subversion of security mechanisms, we had better go shopping for a security kernel. Not only does such a mechanism incorporate the security features we will need, it also provides (especially by its attention to compactness and verifiability) a high degree of assurance that the design and implementation are complete and that malicious software attacks will not succeed. Furthermore, its compactness and verifiability provide a significant degree of inspectability and assurance that its implementation has not been exposed to subversion. Other mechanisms, such as cryptography, can be used for detecting (after the fact) whether software or data has been modified (as discussed in a later section), but the security kernel is the only method proven effective at countering the threats of penetration and subversion of mechanism, and thus it is the only method effective at preventing illicit access to information under protection.

Computer security and security policy

In our discussion of a document library, we mentioned an authorization list or roster that determined which individuals could enter the library at all, and which documents they could see. External laws, rules, and regulations establish how, when, and what access by people is to be permitted. We do not expect the guards (or walls) of our library to determine who may and who may not enter. Instead, the organization that established the library in the first place also defined a security policy specifying who may enter and who may not. This section provides an introduction to the notion of a security policy and its enforcement in a computer system.

A useful security policy is quite general. It typically does not specify by name that certain people may or may not have access to certain information. Instead, it may state that the holders of certain positions have the authority to gain access to certain information. It may allow the hold-

ers of other positions to grant individuals access to information within some scope or set of checks and balances. A security policy may also state requirements that people must meet for access to information, as in the case of security clearances for access to classified national security information.

The Executive Branch of the US government (as well as branches of other governments) has a general security policy for the handling of sensitive information. This security policy involves giving an access class called a "security classification" to sensitive information and a clearance to individuals authorized to access it. No individual is granted access to information classified higher than that individual's clearance. (For example, since "Top Secret" is higher than "Secret," an individual with a "Secret" clearance is not permitted access to "Top Secret" information.) However, possession of a clearance at or higher than the classification of the information alone is not enough to gain access — that individual also must have a "need-to-know" the information, as judged by someone who already has access to the information.

To better understand how a general security policy such as this is enforced when computer systems are operating in different environments, consider three different modes of secure computing used in the Department of Defense: dedicated, system high, and multilevel.

In a simple computation environment, protection or security is enforced by physical means external to the computer (fences, guards, and so on) in a dedicated mode of operation. In this mode, all users allowed access to the system are cleared for the highest level of information contained in the system and have a need-to-know for all the information in the system (that is, it is dedicated to processing for users with a uniform need-to-know for this information at a given single security level). All users, equipment, and information reside within this protective boundary or security perimete. Everything within the security perimeter is considered benign. The computer system is not expected to seriously "defend" information from any of its users because they are considered nonmalicious by virtue of their security clearances and need-to-know.

In another environment (called the system high mode), the computer not only provides computation but must internally provide mechanisms that separate information from users. This is because not all users of the system have a need-to-know for all the information it contains (but all are cleared for the highest level of information in the system).

In yet another environment (called the multilevel mode), the computer must internally provide mechanisms that distinguish levels of information and user authorization (that is, clearance and need-to-know). In this case, not all users of the system are cleared for the highest level of information contained in the system, nor do all users have a need-to-know for all the information contained in the system.

Here, the computer system must protect the information from the user who is not cleared for it and his possibly malicious software. In effect, the computer system must become part of the security perimeter. The internal protection mechanisms must "assume the roles" of the guards, fences, and so on, that are indicative of the external security perimeter. Anything outside the security perimeter (including software) should be considered suspicious, since it may be malicious.

Clearly, for a computer to operate in the system high or multilevel mode, in which it is responsible for enforcing a portion of the security policy, the security policy must be translated into rules for handling sensitive information on a computer. This translation is not always clear since the security policy is expressed in terms of persons accessing information and not in terms of computer processes (accessing files or segments or bytes). The security policy does not address how a computer may provide both computation and protection.

Thus, one of the first steps in building a secure computer system is to interpret the security policy to be enforced (for example, as described by Lunt et al. [LUNT88a]) in a way that allows it to apply to the internal entities of the computer system. A security policy is interpreted in terms of the permissible access modes (for example, read or write) between the active entities — subjects — and the passive entities — objects — to establish a technical security policy (or a "technical policy" [TDI91]) for the system. We therefore call the specific translation of a security policy into terms implemented on a computer the technical security policy, as distinct from the security policy stated in terms of people accessing information. To build a secure computer system, it is essential to have a technical security policy that is complete and precisely defined and interpreted.

It is adequate to characterize the access controlrequirements of a technical security policy in terms of the set of subjects to be controlled, the set of objects to be protected, and all the rules concerning the access of subjects to objects to be enforced by the system. The basic security-relevant operation available to subjects is a request to access a particular object in a particular access mode. In response to such a request, the secure system may either grant or deny access.

To decide whether a particular request for access is to be granted or denied, the system must make a decision as to whether the requested access is consistent with the access control policy to be enforced. Although actual mechanisms typically function on the basis of accesses that are to be permitted, it is useful to think of a policy abstractly as accesses that are to be prohibited. Therefore, consider an access control policy as a list of ordered triples <*s*, *o*, *m*> of accesses that must be prohibited (where *s* is a particular subject, *o* is a particular object, and *m* is a particular access mode). This list of triples completely specifies the behavior of the access control policy's reference functions. For instance, if the triple <*x*, myfile, read> appears in the list, subject *x* may not be given read access to object

myfile. The convention of representing the abstract access control policy as a list of prohibited accesses is useful because it enables the rules for verifying correct enforcement of the policy to be specified positively and completely. It is also particularly useful in composing access control policies belonging to different components in a network, as we show in a later section. (For access control policies expressed in this way, the composed access control policy is just the union of the access control policies of the components.)

A basic principle of computer security is that a given system can only be said to be "secure" with respect to some specific security policy, stated in terms of controlling access of persons to information. It is critical to understand the distinction between security policy (or technical security policy as defined above) and security mechanisms that enforce the security policy within a given computer system. For example, mechanisms might include type enforcement [BOEB85], segmentation, or protection rings [SCHR72]. These are all mechanisms that may be used within a computer system to help enforce a security policy that controls access of persons to information, but none of these is itself a security policy. Such mechanisms provide functionality that enables the implementation of access control within the computer system, but they do not directly represent rules in the security policy world of persons and information. It has been shown [HARR76, SHIR81] that in general for any given security mechanism, there are security policies that the mechanism is not sufficient to enforce. Thus the mechanism is molded by the security policy that it is designed to support. To understand the danger of mistaking security mechanisms for security policy, consider that some existing systems impose security mechanisms on users, but it is not at all clear what the security policy is that is being enforced. (Examples include the Unix "setuid" and "setgid" mechanisms [LEVI89].) This creates the illusion of security, without providing real security.

As we noted earlier, the reference monitor concept is not defined by the security policy, nor does it define the security policy. The reference monitor concept is compatible with a broad range of security policies that can be considered in two classes: access control policies and supporting policies. Access control policy is that portion of the security policy that specifies the rules for access control that are necessary for the security policy to be enforced (as will be described in later sections). Supporting policy is that part which specifies the rules for associating humans with the actions which subjects take as surrogates for them in computers to access controlled information (as will also be described later).

The access control policies in turn fall into two classes: discretionary and mandatory. These two classes were originally referred to as discretionary and nondiscretionary, and, as described in the following excerpt [SALT75], both have historically been considered necessary for commercial as well as military security:

> We may characterize [one] control pattern as discretionary implying that a user may, at his own discretion, determine who is authorized to access the objects he creates. In a variety of situations, discretionary control may not be acceptable and must be limited or prohibited. For example, the manager of a new department developing a new product line may want to "compartmentalize" his department's use of the company computer system to ensure that only those employees with a need to know have access to information about the new product. The manager thus desires to apply the principle of least privilege. Similarly, the marketing manager may wish to compartmentalize all use of the company computer for calculating product prices, since pricing policy may be sensitive. Either manager may consider it not acceptable that any individual employee within his department can abridge the compartmentalization decision merely by changing an access control list on an object he creates. The manager has a need to limit the use of discretionary controls by his employees. Any limits he imposes on authorization are controls that are out of the hands of the employees, and are viewed by them as nondiscretionary. Similar constraints are imposed in military security applications, in which not only isolated compartments are required, but also nested sensitivity levels (for example, top secret, secret, and confidential) that must be modeled in the authorization mechanics of the computer system. Nondiscretionary controls may need to be imposed in addition to or instead of discretionary controls. For example, the department manager may be prepared to allow his employees to adjust their access control lists any way they wish, within the constraint that no one outside the department is ever given access. In that case, both nondiscretionary and discretionary controls apply.

More recently, nondiscretionary has been called mandatory [TCSE85], but the meaning has been retained: Mandatory is still the complement of discretionary. For reasons that will become clearer below, protection against malicious software is offered only by an implementation of the reference monitor concept enforcing mandatory access control policies, though the reference monitor paradigm of subjects, objects, authorization functions, and reference functions is also used for discretionary access control.

In general, one cannot a priori simply assert whether an arbitrary access control policy is mandatory or discretionary. However, it is clear that some access control policies cannot be mandatory (we will see why in the next section). The more appropriate question is whether the protection against malicious software that is uniquely possible with the high assurance enforcement of a mandatory access control policy is needed for a particular aspect of the security policy. The problem then becomes one of

expressing that aspect of the security policy in a way that maintains the properties of a mandatory access control policy. This too is described in the next section.

As a practical matter, the choice between mandatory and discretionary access control policies to support a particular security policy is, in most cases, tied to the penalty for which one would be liable if one violated the policy in the "paper world" — if no computers were being used. If the person responsible for protecting the information could get into "real trouble" (for example, lose a job, get sued, be placed in jail, or even be severely reprimanded) for violating the policy in the paper world, then a mandatory access control policy should be used to protect the information in the computer.

Mandatory access control policy. A mandatory access control policy provides an overriding constraint on the access of subjects to objects, with high assurance of protection possible, even in the face of Trojan horses and other forms of malicious software, as described in Essay 1. In terms of the reference monitor concept, the idea is that we can affix a label to objects to reflect the access class of the information they hold. We can correspondingly affix a label to subjects to reflect the equivalence class of object sensitivity that the subject can access. The reference monitor compares the labels on subjects and objects, and grants a subject access, per the requested access mode, to an object only if the result of the comparison indicates that the access is proper.

Note that the preceding paragraph identifies the mapping between our two "worlds":

1. The world independent of computers, of people attempting to access information on paper.
2. The world of computers with objects that are repositories for information and subjects that act as surrogates for users in the attempt to access information in objects.

As noted above, the label associated with an object indicates the access class of the information that the object holds. The label associated with a subject that acts as a surrogate for a user indicates the authorization of the user — the access class of the information the user is authorized to access (for example, the user's clearance). Earlier we identified subjects as processes executing in a particular domain. In many systems, there is a single label associated with each process since, in these systems, there is a single domain per process. However, in some systems [SCHE85a, THOM90], each process may have a number of domains (and correspondingly, a number of subjects) simultaneously, each of which has a separate label. (Incidentally, these separate domains within a single proc-

ess are typically implemented by a mechanism called protection rings [SCHR72].) Finally, the access modes used in the computer are the same as the fundamental access modes in the world independent of computers, of people attempting to access information on paper — read and write.

Mandatory access control policies can provide protection against unauthorized modification of information (integrity) as well as protection against unauthorized disclosure (confidentiality). The labels in a specific mandatory access control policy can be selected to accomplish many different purposes for integrity and confidentiality. For example, they can reflect the US government's security policy for confidentiality mentioned earlier, utilizing hierarchical classifications and security clearances (for example, Secret, Top Secret). They can reflect a corporate security policy [LIPN82, LEE88, SHOC88] (for example, Public, Proprietary for Confidentiality or Technical, Management for Integrity). They can also reflect a partitioning of activities into separate spheres or compartments, with different individuals authorized access to information in different areas (for example, Project A, Project B).

Abstractly, in the list of triples that specifies a particular mandatory access control policy, there is an entry for each subject, object, and access mode set (read or write) for which access should not be granted. In other words, if the mandatory access control policy requires that the label associated with the subject be "higher" than that associated with the object in order to grant read access, there are triples for each subject, object pair for which the third element in the triple is read and for which the subject's label is not "higher" than the object's. For a different mandatory access control policy, there would be a different list of triples.

Mandatory access control policies operate by partitioning the sensitivity of objects and the authorizations of subjects into access classes (which correspond to the labels mentioned above). The key to the power and effectiveness of mandatory access control policies is the verifiable restriction on the flow of information from one access class to another. Briefly, a mandatory access control policy reflects a set of rules for comparing access classes. Depending on the security policy being enforced, some flows are allowed and others forbidden. The distinguishing qualities of mandatory access control policies are that they are global and persistent within some universe of discourse; these qualities enable verifiability of the reference monitor implementations that enforce them.

In this context, "global" means that particular information has the same sensitivity wherever it is; "persistent" means that particular information has the same sensitivity at all times. In other words, the subject and object labels are "tranquil"; they do not change. For an access control policy to be global and persistent, the set of access classes (or labels) must form what is termed in mathematics a "partial order." This means that any members of the set can be compared by using a relation usually called dominate, written ">=" and meaning something like "greater than or

equal to." For any two distinct members x and y of a partially ordered set, x dominates y, y dominates x, or x and y are noncomparable. (Most existing implementations of mandatory access control policies use a particular type of partially ordered set called a lattice. The distinctions between a partial order and a lattice are not particularly important for this essay, so we will not further discuss lattices.)

The partially ordered set of labels and the resulting restriction on the flow of information provide a tool of sufficient power to defend against even malicious software, described in Essay 1. The members of a set that forms a partial order can be compared by the dominate relation in a manner that satisfies three standard mathematical conditions:

1. reflexivity,
2. antisymmetry, and
3. transitivity.

We can say that for a particular set (for example, of labels) and the relation >=, these three conditions mean respectively that, for all x, y, and z in the set,

1. $x >= x$,
2. $x >= y$ and $y >=$ x implies $x = y$, and
3. $x >= y$ and $y >= z$ implies $x >= z$.

If any of these three conditions of a partially ordered set of labels is relaxed in an access control policy, either the global or persistent quality is destroyed, rendering the access control policy fundamentally vulnerable to Trojan horses. For that reason, arbitrary "tags" cannot be used as mandatory access control policy labels. For example, consider a violation of reflexivity — consider an access control policy in which all subjects and objects have either the label "Sensitive" or the label "Public" and which specifies that Public subjects can access Public objects except on odd Tuesdays. That is an example of the label Public not always dominating itself — a violation of reflexivity and of the "persistent" quality. For another example, if the access control policy specifies that Public subjects can access Sensitive objects only on weekends, the label Sensitive would dominate Public and on weekends Public would dominate Sensitive. But since Sensitive and Public are not equal, we lack antisymmetry, and the quality of persistence is again violated, leaving the opportunity for the access control policy to be circumvented. Similarly, if the access control policy uses the label "Proprietary" in addition to "Sensitive" and "Public," and if Proprietary subjects can access Sensitive objects, while Sensitive subjects can access Public objects, but if there are some Public objects that Proprietary subjects cannot access, then the access control policy lacks transitivity and the "global" quality is not met.

As we have noted, an access control policy which does not use labels that conform to the qualities of a partially ordered set for all its subjects and objects is not a mandatory access control policy — it is a discretionary access control policy. Each of the access control policies given as invalid mandatory access control policies in the preceding paragraph is a perfectly valid discretionary access control policy. The basic definitions themselves lead to an important, unavoidable conclusion. Any access control policy is either mandatory or discretionary; there is no gray area between. Furthermore, it is a mandatory access control policy if, and only if, it can be represented by a partially ordered set of access classes; otherwise, it is discretionary. Remember that the key distinction between these two forms of access control policy is the protection against malicious software that is possible with each.

We should note that if more than one mandatory access control policy must be enforced simultaneously within a system, then each subject and object may have a label associated with it for each mandatory access control policy. In this case, the labels may have no relationship with each other. Such is the case for the enforcement of mandatory access control policies for both confidentiality and integrity. For example, an object may have a "Secret" label for confidentiality but a "Junk" label for integrity. Another object may have a "Secret" label for confidentiality but a "High Integrity" label for integrity. The only requirement for these labels is that the set of labels for each mandatory access control policy must be partially ordered. Obviously, the overall decision of whether or not to grant access depends on the proper dominates relationship between the subject and object labels for each set of labels. However, this is easy, since the mathematics tells us that the product of partially ordered sets is another partially ordered set. In other words, since each set of labels is partially ordered, all the sets of labels can be combined into a single set, making the comparison of numerous labels a single operation. This technique has been used in practice in a commercial product to greatly simplify and improve the efficiency of the enforcement of mandatory access control policy [THOM90].

An example of a mandatory access control policy and its computer implementation with a partially ordered set of labels may make the discussion more comprehensible at this point. Let us divide an organization into two divisions — perhaps Marketing and Engineering. The employees in each division are allowed access to the information for their own division, but only top management is allowed access to both access classes. We might mark all the Marketing information in our computer with a label "M" and all the Engineering information "E." Employees of the Marketing and Engineering Divisions would be represented in the computer by processes labeled "M" and "E," respectively.

Our secure operating system would allow any process to read information with the same label it possessed or with no label at all. However, no process could remove or change the label on a file, and no process could

write any information that did not have the same label as the process. Thus, when a Marketing person was using the computer, he or she could read "M" information or unlabeled information at will. But any information that the process wrote would be labeled "M." Neither the employee nor a Trojan horse could communicate information from one access class to the other.

A scenario that allows no communication at all sounds fairly useless. We shall later expand on this basic scenario to suggest ways in which users can share information and to discuss the access authorizations of our organization's top management. For this introduction, though, let it suffice to say that mandatory access control policies provide a powerful and flexible tool [BELL91] for controlling the flow of information among individuals, and a basic tool for the design of secure computer systems.

Discretionary access control policy. Discretionary access control policies are so named because they allow the subjects in a computer system to specify who shall have access to information at their own discretion. In a system that incorporates both mandatory and discretionary access control policies, the discretionary access control policy serves to provide a finer granularity within (but cannot substitute for) the mandatory access control policy. For example, the military need-to-know security policy in which each individual has a responsibility to determine that another has a valid requirement for information, even though the other has a clearance for the information, is a common discretionary access control policy. In other cases, allowability of access within a discretionary access control policy may be based on the content or context of the information to be accessed or on the role of the user at the time of the access request — or it may involve complex conditions for determining allowable access. In contrast to mandatory access control policies, it need not be global or persistent. Alternatively, a system may incorporate only a discretionary access control policy if the mandatory access control policy is degenerate so that all subjects and objects belong to just a single (implicit) access class. This is the case for the system high mode of operation discussed earlier.

A common example of a discretionary access control policy implementation is the ability of a computer user or a process which that user has executed to designate specific individuals as being authorized access to a given file. Many operating systems provide protection bit masks (for example, "owner," "group," and "world"), access control lists, or file passwords as mechanisms to support some form of discretionary access control policy.

As is the case with mandatory access control policies, we can talk about the abstract list of triples that specifies a particular discretionary access control policy. As with mandatory access control policies, for different discretionary access control policies, there are different lists of triples. With a particular discretionary access control policy, there is an entry for each sub-

ject, object, and access mode set for which access should not be granted. However, unlike the limitations on the access modes relevant to mandatory access control policies (that is, read and write), the access modes for a particular discretionary access control policy may be any set of functions. In other words, a particular discretionary access control policy may control not only static read and write access by subjects to objects but also, for example, read-on-every-other-Friday or read-only-if-another-object-has not-been-read, or any other content- or context-dependent rules. (The direct correspondence of security policy access modes to primitive controls — read and write — within a computer system is not important for discretionary access control policies as it is for mandatory access control policies because of the inherent limitations of discretionary access control policies, as is illustrated in the following paragraph.)

A discretionary access control policy is useful in some environments, but it will not defend against Trojan horses or other forms of malicious software such as may be used to perform probing, penetration, or subversion attacks, as described in Essay 1. This can be seen by considering a Trojan horse hidden in a useful program. The example Trojan horse is designed to make a copy, in a directory where the copy is not likely to be noticed right away, of all of the files that belong to a user who runs the program that are marked for reading only by that user. This copy is made readable by some other user who would not be intended to have access to the files. In contrast, consider a mandatory access control policy intended to provide confidentiality. Since the label is attached to any copy which is made and since the Trojan horse cannot change the label, the Trojan horse cannot give a user access to any file in a manner contrary to the mandatory access control policy. In other words, a mandatory access control policy does not prevent a copy from being made by a Trojan horse executing in a process with the same label (for example) as the file, but it does prevent the file's label from changing and prevents access to the file on a global and persistent basis. Discretionary access control policies offer no real protection against even such simply designed malicious software.

Supporting policy. In addition to the access control policies (mandatory and discretionary), there are additional security requirements relating to the accountability of individuals for their security-relevant actions in the computer system. These requirements make up supporting policy [TNI87]. Supporting policy fundamentally "supports" the tie of people in access control policies, about people accessing information, to subjects acting as surrogates for people in computers. Supporting policy provides an environment for ensuring individual accountability for the enforcement and monitoring of the access control policies. In contrast to access control policy, which associates directly with the "theory of computer security" — the reference monitor concept — there is no corresponding "theory" that helps one verify the implementation of supporting policy. Fortunately, it is possi-

ble to analyze and test software performing supporting policy functions to reasonably conclude that it functions properly. In contrast, as we have said, it is not possible to do this for an implementation of access control policy.

Supporting policy includes two subcategories: identification/authentication policy and audit policy. The former supports the access control policies by specifying the requirements for authenticating the identity of an individual prior to allowing subjects to act as surrogates for that individual in attempting access. Identification/authentication policy provides the basis for the labels that are used in enforcing the mandatory access control policy to be associated with subjects acting as surrogates in the computer for individuals. In other words, it determines whether subjects may act as surrogates for a particular individual and what label is associated with such subjects. It also provides the basis for the membership of individuals in a group and more generally for controls on subjects consistent with the discretionary access control policy. Further, it provides the basis for recording the identity of the individual causing an auditable action to be performed by a subject acting as the user's surrogate.

Audit policy provides the basis for the recording of those security-relevant events that can be uniquely associated with an individual. The objective is to provide accountability for the security-relevant actions of individual users. We do not have much more to say in this essay about audit policy. The following paragraphs expand a bit on identification and authentication and other aspects of accountability, as supporting policy considerations.

Identification and authentication overview. Identification is a rather straightforward notion. Our summary of identification is simply this: The secure computer system should associate subjects with the identities of individual users and have the option of making authorization decisions or recording an audit trail on the basis of those individual identities. This is in order to be able to trace back security-relevant actions on the computer to some individual. The question arises, "How do we know the identity is correct?" The answer to this question is the province of authentication.

When we discuss authentication, we are concerned with providing the system with some basis for confirming that the user's identity is as claimed. For example, authentication is commonly implemented with some sort of password scheme. The classic definition of authentication measures presents a taxonomy of something one has, something one is, or something one knows. In addition to password schemes (know), there are other methods, such as the use of badge readers (have), challenge/response calculator-like devices (have), smart cards (have), fingerprint readers (are), palm readers (are), and retinal scanners (are).

All of the authentication schemes attempt to provide a reason to believe that the individual who is claiming an identity is in fact the person claimed. All do so by provoking the occurrence of some event that would be much less likely if the person were not the one claimed, and all are probabilistic. The last point is critical. No matter how refined the password scheme or sophisticated the fingerprint reader, there is still a residual probability that one can "fool" it by luck or by cunning. Longer passwords and better fingerprint readers may reduce the probability of an error, but they cannot reduce it to zero. As with the guards in our document library, there is a chance that the authentication scheme will be fooled. For this reason, topics such as "password management" should be examined in greater detail by anyone implementing an authentication scheme based on passwords. However, such topics will not be discussed further in this essay.

There is another side to authentication in a secure computer system whose very existence may be a surprise to the reader. This side deals with the need to authenticate the system to the user.

Authenticating the system. To motivate the need to authenticate the computer system to its user [SALT75], we will again start with a "war story." Suppose we can write a program that will clear the screen of a display terminal and sit waiting for a user to type something. When the program detects a carriage return, it will respond with the string of characters that resembles the system's prompt for the login identifier (for example, "USERNAME:"). If a user types any string, the program will respond with the string the system uses to prompt for the password (for example, "PASSWORD:"), and, if appropriate, it will direct the terminal to cease printing the characters that the user types. After receiving the new string and a carriage return, the program will type some suitable error message and terminate, leaving the unsuspecting user with a real unassigned terminal. Of course, the program was a Trojan horse that just captured the user's authentication information and stored it some place where the attacker responsible for the Trojan horse can later retrieve it and use it to log in as the user who was the victim.

The scenario presented above is a simple way to capture an unsuspecting user's password. It can be executed more or less easily on almost any time-shared computer system that relies on passwords. This sort of attack is logically the same as one in which a separate computer intercepts communications between the user's terminal and computer and steals the password [SALT75]. The possibility of executing the scheme is directly traceable to the lack of an authentication mechanism that serves to authenticate the computer system to the user. As we have just shown, the lack of such a mechanism can have serious consequences.

If we wish to eliminate the possibility of writing a "password grabber" of the sort proposed, we must develop a sort of reverse authentication

mechanism. Simply, what we would like to do is have some action that the secure system can take and a password grabber cannot. This can be accomplished by what is called a trusted path.

To implement a trusted path, we can provide a unique action that the user can take to communicate with the secure system. The user initiates the exchange, but is guaranteed that his or her action will result in a response from the trusted part of the secure system. For example, many current systems are guaranteed to respond when a terminal is powered off and then back on. This, or other hardware-supported measures such as pressing the break key, can be used when a terminal is directly connected to the system in a form of authentication of the system that is well suited to initiating the login dialogue.

Of course, this method violates the concept of "programming generality." For while the ideals of computer system design might direct that we always allow a program to intercept and interpret or filter the actions or responses of a user or another program, in the case of the secure system, we must have a class of action or response that cannot be filtered. In particular, it is important to the notion of a trusted path that no code outside the trusted part of the secure system may execute as part of the trusted path.

As mentioned, the password grabber is an instance of a Trojan horse. Once a valid <username, password> pair has been captured, the attacker can use it to establish a false identification as the victim. A trusted path can provide a means (for example, a reserved "secure attention key" on the terminal keyboard) whereby the user can ensure that communication is with the trusted part of the secure system before the username and password are typed. The trusted path is used at other times when a positive communication between a user and the secure system is needed, such as for performing security administration on the secure system (for example, entering user clearances, setting the access class label or range on devices) or downgrading a file (if such a capability is implemented).

Security policy in the document library. We might consider our document library once more and make the examples of authorization and partitioning a little more concrete. Suppose that the library contains documents classified "Secret" and "Top Secret." Individuals who have been granted access to the library have either a Secret or a Top Secret clearance, and the authorization list at the guard desk lists the clearance for each authorized individual.

The library is divided into a Secret area and a Top Secret area. While holders of a Secret clearance are restricted to the Secret area, those with a Top Secret clearance are allowed to enter either. Users of the library are normally forbidden to remove documents, and any notes they take are marked with the access class of the material they have been reading (and retained in the library for their later use). Secret notes may be carried to

the Top Secret area. Documents may be removed or classification markings modified only under the control of elaborate procedures authorized by specially selected people. The partitioning of people and areas, and the restriction on removal and reclassification of documents, implement our library's mandatory access control policy.

Documents are stored in the two areas under the control of custodians who are part of the guard force. Each document has an access control list that identifies those cleared individuals who may have access to it. In addition, each document has designated "owners," typically library users who may add names of people to the access control list. The combination of access control lists, custodians, and owners implements a discretionary access control policy that operates within the constraints imposed by the mandatory access control policy.

An audit trail is kept to record the comings, goings, and accesses of individuals, as required by the audit policy.

Indirect access. Any discussion of security policy would be incomplete if it did not address the often-overlooked topic of indirect access. Indirect access is access that occurs outside the security perimeter of the secure computer system. Indirect access is particularly important precisely because it is often overlooked in establishing a security policy for a computer system. Often, the security policy of a so-called secure system may contribute to illicit indirect access; conversely, the security policy of a secure system can contribute to preventing illicit indirect access.

Of course, indirect access is not always overlooked. A description [TECH85] of what is meant by "clearances of system users" says,

> System users include not only those users with direct connections to the system but also those users without direct connections who might receive output or generate input that is not reliably reviewed for classification by a responsible individual.

This statement indicates a security concern not only with controlling the sensitivity of information that the system is processing but also with controlling the sensitivity of information the system is importing or exporting. Note that this illustrates the key to the techniques already partially described for defending against Trojan horses; the key lies not in stopping the Trojan horse from reading the information being protected, but rather in preventing that information from exfiltration.

Now we look at several examples of indirect access. Consider our document library and the security policy described in the preceding paragraphs. Recall that, in the library, documents are marked with their access class and that there are elaborate procedures for removal of documents from the library. Suppose that, for whatever reason, the procedures for removing materials from the library include changing their

access class markings from Secret and Top Secret to Restricted and Very Restricted. (This is not as preposterous as it sounds; in some environments, the very names of the classification markings are sensitive, and code words are used outside of the "document library" instead of the real names.) If these documents are now taken to a different library in which the markings Restricted and Very Restricted do not mean the same things as in the original library, users of the new library may access them who would never have been granted access to them in the original library. In this manner, illicit indirect access might result.

There are many ways other than ambiguous security markings, as in the above example, in which illicit indirect access might occur. For example, there is a so-called secure system in use today that utilizes two types of access class markings. One is a label used by the system to implement its mandatory access control; the other is "advisory," not necessarily reflecting the full sensitivity of the information. The "advisory" marking's accuracy may be destroyed by malicious software that places into the object information more sensitive than that indicated by the "advisory" marking and yet, as enforced by the system, no more sensitive than the label used for the mandatory access control. Malicious software capable of performing this feat is quite simple to develop. In such a way, the "advisory" marking may no longer reflect the sensitivity of the information it marks.

The "advisory" marking may be printed on hard copy by the "secure system." Even though the specification for this system says that the "advisory" marking should not be used in determining whether to export information outside the system, in reality this practice is not always followed. In practice, the hard copy bearing the "advisory" marking is sometimes handled outside the computer in a manner consistent with the "advisory" marking but inconsistent with the mandatory access control label. Clearly, this marking is then the basis for decisions regarding the possession and access of this hard copy, quite possibly resulting in illicit indirect access. This illustrates again the "historical lesson" expounded in Essay 1 that misplaced confidence or false assurance of security is worse than if no security at all were provided.

Illicit indirect access may similarly be effected electronically if not carefully considered and prevented. This is trivially possible in the system just cited if, instead of generating hard copy, the system makes decisions about exporting information out of a communication port (for example, to a network or other computer). As a practical matter, it is impossible to detect all possible ways such electronic indirect access might occur. For example, consider a computer system operating in the system high mode described earlier, in which messages are created by users of the system. The sensitivity of these messages is recorded in the "header" of the message by the user who creates it, but, since the system is operating in the system high mode, the system does not enforce a mandatory access control policy based on the message sensitivity reflected in the header. Given

that, as is the case for system high systems, each of the users on the system is cleared for all of the information on the system (but may not have the need-to-know for all of it), this design may operate with satisfactory security for quite some time. However, if this system is now connected to a network of other systems in a manner in which it is desired to enforce a mandatory access control policy over the messages, it would be a major mistake to trust the message sensitivity from the message header provided by the system high system to suffice as the mandatory access control policy label. This is true, of course, because of the ability of a Trojan horse on the system high system to modify the sensitivity in the header of potentially any message on that system — since that system implements only a discretionary access control policy.

As a final example of electronically effected illicit indirect access, consider a system that processes both sensitive and nonsensitive information. In this system, it is intended that no information will leave the system without undergoing review by a human reviewer to determine whether that information is sensitive or not. Of course, this security strategy can face a high risk of failure, since it pits the reviewer's skill and stamina against the cleverness of the designer of a Trojan horse. All the designer of the Trojan horse has to do to succeed is to find one way of "sneaking information past" the reviewer without detection. In practice, this is usually easy.

The specific method chosen by an attacker would vary based on the format for presenting the information to the reviewer and the media on which the information being reviewed is stored while being exported from the system. For example, if information is presented to the reviewer in a digital image, it is very simple to contaminate the digital image without detection by the reviewer. The interested reader is encouraged to read an insightful paper by Kurak and McHugh [KURA92]. It clearly shows that "image downgrading based on visual display of the image to be downgraded not be performed if there is any threat of image contamination by Trojan horse programs." As another example, if the information were being exported by the use of a magnetic tape, parts of the magnetic tape that are not easily humanly readable (perhaps "header" or unused bits of each byte) could be used to sneak information past the reviewer. Even if paper is used for exporting the information, sneaking information past the reviewer could be as simple as modulating the use of spaces that separate words in the information being exported.

The solution to the problems of indirect access lies in ensuring that the labels used to enforce a mandatory access control policy are reliable both inside and outside the system. This is possible only if the labels are maintained by an implementation of a trusted computing base, as described in the following section. For human interfaces, such as documents, deliberately and carefully choosing the human-readable access class markings for information exported from the secure system and used

outside the system's security perimeter helps to guard against illicit indirect access by making such choices explicit during security policy development. Similarly, for electronic interfaces, a protocol is needed for unambiguously associating a label with all imported and exported information. Only through such explicit designs can the specter of false assurance be prevented.

Building a secure system

The primary focus of this section is the mechanization of security policies by a subset of an operating system called a trusted computing base (TCB) [TCSE85] and methods of achieving degrees of assurance that the mechanisms are correctly enforcing the security policy. (Note that this essay does not address specific evaluation criteria or evaluation classes [TCSE85], which are discussed in Essay 6.) These methods make it possible to build a relatively small part of the system (the TCB) in such a way that one can even allow a clever attacker to build the rest of the system and it will still be secure.

The TCB is defined to be the totality of protection mechanisms within a computer system — including hardware, firmware, and software — the combination of which is responsible for enforcing a security policy. Any software outside the TCB may be malicious software. If a security kernel is implemented in a TCB, then it is the most privileged part of the TCB, and it implements the reference monitor. The ability of a TCB to enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrators of parameters (for example, a user's clearance or the access class label for devices) related to the security policy. The input of these parameters is generally called security administration, and the specially privileged users responsible for the accuracy of this information are called security administrators. Entry of the parameters by the security administrators is an example of exercising the authorization functions.

The role of a security policy model. Once we have the reference monitor concept and a suitable security policy in hand, we are faced with the problem of building a secure system or TCB, as it is called. In other words, we have the problem of implementing the concept in a real hardware/software mechanism that enforces the security policy. In beginning this task of implementation, we shall find that a crucial role in the assurance of security policy enforcement is played by a model of the security policy of our TCB. The security policy model allows us to make the leap from security policy to real TCB by providing an intermediate step: a formal description of the functions that the TCB will perform. By "formal" we mean that the model must be a precise and complete mathematical statement which

can be proven self-consistent — such statements are usually presented using predicate calculus.

This intermediate step offered by the security policy model bridges between two constrained universes of discourse — the computer-independent world of people attempting to access information, and the world of computers. We have already discussed the roles of subjects as surrogates for users and objects as containers of information. In a security policy model, we have a set of operations or rules that model the reference functions and the authorization functions that define the access of subjects to objects. The security policy model takes as a "given" (that is, does not model) certain initial state information — including certain information typically entered into a TCB by security administrators, as mentioned above. It also takes as a given the proper operation of the TCB to correctly implement the supporting policies, including the proper operation of the TCB to identify individuals authorized to perform as security administrators or as users of the system.

The security policy defines the behavior desired of the TCB but does not directly dictate the functions to be performed by the TCB. In fact, nothing in the security policy itself explicitly relates to the notion of a TCB or a computer system at all (remember that we just applied a "computer security" paradigm to a document library). We proceed by precisely (formally) defining the functions of the TCB. We must, of course, find functions such that the behavior exhibited by the security policy model complies with the security policy. The security policy model can be viewed as a formalization and particularization of the reference monitor, providing its reference and authorization functions. Note that current practice is for the security policy model to encompass only the access control policies of the TCB; it does not itself represent the supporting policies. As stated above, a security policy model takes as a given the proper operation of the TCB to correctly implement the supporting policies. History has shown that enforcement of the access control policy is one of the hardest things in a computer system to get right. It is fortunate that the technology has developed to allow this tool to be used to substantially increase the assurance of access control policy enforcement.

If the security policy model encompasses the discretionary access control policy, then it must represent authorization functions whereby a subject can, for some object, grant access to another specified subject. This function could, for example, correspond to adding a user's name to an access control list. If there is no explicit discretionary access control policy in the model, then the controls on access within the model reflect only the mandatory access control policy.

By the very definition of mandatory access control policy, in general, subjects cannot modify mandatory security authorizations. The access class labels for the initial objects are supplied from "outside" the TCB as part of its initial configuration, typically by security administrators. The security

policy model may, however, include functions for creating and deleting objects, and for controlling the access class for new objects and subjects.

There is an exception, by the way, to the notion that in a mandatory access control policy, subjects cannot modify mandatory security authorizations. That exception is for trusted subjects. Trusted subjects are subjects outside the reference monitor, but within the TCB, that can read and write objects at different access classes, in a manner that would normally be prevented by the reference monitor. Trusted subjects are used, for example, in downgrading a file in TCBs, which permits sanitization of information, or they may be used simply in the course of implementing a TCB.

The use of trusted subjects is distinguished from the more general extension of privilege to read and write computer resources outside the reference monitor (for example, in "privileged" processes), as is implemented in some systems. In building a TCB, it is important to recognize the sharp distinctions that exist between privileged processes and trusted processes (that is, processes containing a trusted subject). Privileged processes are those that are capable of affecting access control decisions or other functions of the reference monitor. Since they can potentially affect any aspect of the reference monitor, the entire security of the system depends on the behavior of these privileged processes as much as any other part of the reference monitor. In other words, the reference monitor's security perimeter is extended to encompass privileged processes. This is in contrast to trusted processes that operate within a range of access classes but do not affect access control decisions. The difference in the level of system exposure that accompanies privileged processes from that which accompanies trusted processes is clearly considerable.

For example, in TCBs like Multics [SCHR72] and the Gemini Trusted Network Processor [THOM90], there are multiple protection rings where trusted processes can be placed, and therefore their actions can be constrained from affecting the rest of the TCB. However, there is no such thing in the typical Unix architecture. The usual choice for implementing trusted processes with Unix is to effectively modify the Unix kernel by implementing privileged code. However, this is a problem, since even if the Unix code is well-modularized and understandable, it is generally unreasonable to expect writers of privileged processes to understand the security implications of so fundamental a modification. Further, note that some implementations provide a "privilege bit" intended to allow one to build things like a trusted process, but this is the moral equivalent of extending the Unix kernel out into the trusted process. If one were to use this "privilege" mechanism, one would not be able to state any well-formed security properties about it. In contrast, building a trusted process on a base with protection rings, such as those mentioned above, allows one to make the trusted process somewhat independent from the rest of the TCB, thereby providing a valid basis for security in the system. Since trusted processes contain subjects that are outside the reference

monitor and their behavior is constrained, their effects are not usually explicitly included in the security policy model for a mandatory access control policy. However, the effects of privileged processes must be included in the security policy model since their behavior is far less constrained.

While a number of security policy models have been developed, by far the most widely used has been the Bell-LaPadula security policy model [BELL73]. The Bell-LaPadula security policy model is an abstract model of the behavior of a TCB. It provides a framework within which a security policy can be formally represented as mandatory access control policy and discretionary access control policy. The Bell-LaPadula security policy model is security policy-independent. It has been used to represent a set of rules to enforce security policies whose primary objective is confidentiality, and it has also been used to represent a set of rules to enforce security policies whose primary objective is integrity. Each time it is used, it must be interpreted for a specific security policy and a specific system [BELL75, BIBA77, LUNT88a].

Each such interpretation of this model mathematically represents the state of a TCB and prescribes the criteria for a secure state with respect to the requirements of a mandatory access control policy and a discretionary access control policy. The TCB modeled is defined to be in a secure state only if no subject can access information that it is not authorized to access. Each interpretation of the Bell-LaPadula security policy model defines a set of functions or rules for changing the state of the TCB and for permitting subjects to reference objects. It is mathematically proven that the rules preserve the property that the new state of the TCB is still secure — no subject can access information that it is not authorized to access.

Bell and LaPadula presented an interpretation of their security policy model for the Multics system and for a security policy aimed at confidentiality [BELL75]. The key properties preserved in this interpretation for mandatory access control are the "simple security property" and the "*-property." The simple security property for confidentiality stipulates that the label of a subject must dominate the label of an object in order for that subject to get read access to that object. The *-property for confidentiality stipulates that the label of a subject must be dominated by the label of an object in order for that subject to get write access to that object. It is easy to see why these properties prevent information from flowing "downward" from an object labeled with a particular confidentiality access class (for example, Top Secret) to one lower (for example, Unclassified). In this way, confidentiality Trojan horses are rendered ineffective.

Biba's interpretation [BIBA77] of the Bell-LaPadula security policy model for the Multics system and for a security policy aimed at integrity is quite like Bell and LaPadula's [BELL75], except that the simple security property and *-property are changed in an interesting way. The equivalent of the simple security property stipulates that for integrity the label of a subject must be dominated by the label of an object in order for that subject to

get read access to that object. The *-property for integrity stipulates that the label of a subject must dominate the label of an object in order for that subject to get write access to that object. It is easy to see why these properties prevent information from flowing "upward" from an object labeled with a particular integrity access class (for example, Junk) to one higher (for example, High Integrity). In this way, integrity Trojan horses are rendered ineffective.

Shockley [SHOC88] has pointed out that Biba's security policy model interpretation [BIBA77] is capable of satisfying commercial security requirements, as well as the more frequently seen military requirements. In particular, he said that the Clark/Wilson integrity policy that is claimed to be "an accurate representation of what the business and commercial data processing community means by the term 'integrity'...in commercial data processing" can be represented using Biba's techniques [BIBA77].

The power of the security policy model comes from the fact that a suitable set of rules has been developed. It has been inductively proven that if the initial state is secure, these rules can never produce a state that is not secure. That is, a set of sufficient (but not always necessary) conditions is assured by the rules. This power ensures that, if the security policy model indeed represents that behavior of a TCB, then no use of that TCB can cause a violation of the mandatory access control policy.

The security policy model dictates what must and what need not be included in the TCB of the secure system. Thus it has a strong impact on the design of any TCB that attempts to follow the security policy model's requirements rigorously.

Two key issues must be emphasized in closing this subject because they have been an occasional source of misunderstanding. First, the security policy model must be a valid representation of the behavior with respect to information protection of the entire system. Merely modeling distinct computer functions with respect to individual assertions about a protection mechanism provides little indication of overall system security and can even be misleading. Second, the security policy model must include a proven security theorem, which establishes that the security policy model's behavior always complies with the security requirements for the security policy of interest. As stated above, the security policy model takes as a given certain initial state information — including certain information typically entered into a TCB by security administrators. It also takes as a given the proper operation of the TCB to implement the supporting policies, and in particular that subjects faithfully represent the access classes of the individuals for which they are acting as surrogates. With this input, which it considers a satisfactory basis for establishing a secure initial state, the security theorem which is proven is that no matter which rule of the model is operated, all future states will also be secure. This proof of what is called the Basic Security Theorem is the key to the power and utility of this technique. We leave the topic of security policy models with

one guiding principle: A security policy model without a proven security theorem is like a fire bucket with a large hole in its bottom — it is the sort of thing that can give you a warm feeling, but it is probably not what you really want. A false sense of assurance is a dangerous thing.

Design, specification, and implementation. The security policy model defines the functions the TCB must provide but does not specify the design for the TCB. The next step after the specification and interpretation of a security policy model is the specification of the interface to the TCB. The specification defines a set of subroutines (that is, TCB calls, similar to operating system calls) and hardware operations that implement the operations of the security policy model which provide access to the resources contained in the model's objects. The TCB contains internal databases that represent the security policy model's state. These internal databases are used by the TCB (in particular, by the reference monitor) to create the abstractions of subjects and objects that are entities outside the TCB.

When we reach the point of specifying the TCB's interface functions, we are at the point where our efforts reflect the level of system security, or assurance, that we are trying to attain. As we mentioned in the introduction to the reference monitor concept, if we are interested only in preventing probing we will probably elect to make a significant portion of or all of our operating system serve as the implementation of the reference monitor functions. On the other hand, if our system must also be resistant to penetration and subversion, we will need to have a much smaller subset which can be verified — a security kernel [AMES83, SCHE84a] that implements a true reference monitor. Gasser [GASS88] describes security kernel implementation strategies, noting that the security kernel is the single most often used technique for building a highly secure operating system.

While it might seem foolish to contemplate less than the best, and thus to build a system that would not resist malicious software attacks, there are reasons for electing to do so. Probably the best are cost and compatibility. Most existing operating systems can be modified to implement a portion of the reference monitor concept and get improved security at modest cost. Going further and actually modifying the system to incorporate a security kernel is, in general, as costly as reimplementing the system from scratch. If not engineered carefully, such modifications may have a noticeable impact on system performance (though it is quite possible to build an operating system with a security kernel in a way that has quite good performance [SCHE85a]). On the other hand, if a development is planned to build a new operating system "from scratch," one gains significant benefits from incorporating a security kernel, for in this case the incremental costs in development and performance are likely to be relatively modest. In addition, significant savings in life-cycle software maintenance costs are associated with the rigorous software engineering

practices used in building a security kernel. Furthermore, the stable interface of a minimized security kernel appears from a software maintenance point of view to be an extension of the hardware interface, often resulting in a maintenance philosophy much more akin to hardware than to software.

For these reasons, we now consider the steps required to build a security kernel and a TCB *without* a security kernel. Because the latter approach represents a relaxation of requirements and a set of compromises, it is in some ways easier to deal with the security kernel case. We thus present the steps to a security kernel design first and then describe the compromises that one might make in developing a TCB that lacks a security kernel.

The security kernel. In the case of a security kernel, we will design a mechanism that performs a carefully selected subset of operating system functions to implement at least a mandatory access control policy. (Note that because a discretionary access control policy is fundamentally incapable of preventing Trojan horses, as described earlier, there is little need to strive for as high a level of assurance of enforcement as for a mandatory access control policy and therefore little need to include enforcement of a discretionary access control policy inside the security kernel.)

The reference monitor concept requires that every reference by a subject to an object be mediated and subject to the system's security policy and security policy model. To simplify a security kernel, we take steps to minimize its complexity, and we may minimize the number of different types of objects it supports.

The security kernel functions form its interface with the rest of the operating system. It turns out that, to meet the constraints of the security policy model for the mandatory access control policy, the security kernel interface will provide a pure virtual environment for the rest of the system. This requirement results from the need to eliminate (sometimes subtle) methods of circumventing the security policy model's restrictions on the flow of information from one access class to another. Frequently, specifications of the interface are expressed in a formal language capable of being analyzed by a computer program and of supporting a formal proof that the specifications conform to the requirements of the security policy model. Such specifications are termed formal top-level specifications (FTLSs). A system whose interface has been specified and proven in this manner is a system for which one has significantly increased assurance of its ability to enforce the security policy.

As with any design effort, preparing the specifications is a creative activity molded by the peculiar design goals (other than security) of the system. Typically, one proceeds iteratively, testing the design against such requirements as performance, functional richness, use of the underlying hardware, and compatibility with other software systems. Ana-

lytical tools supplement intuition in the effort to ensure that the resulting specification complies with the overriding security requirements of the security policy model.

Beneath the set of security kernel interface functions is a set of computer code, databases, and hardware structures. The selection of secure functions at the security kernel interface is of little benefit to assurance if, for example, the underlying software turns out to be a hundred thousand lines of incomprehensible assembly language "spaghetti code." The security kernel itself is structured in a series of layers (for example, SASS Kernel [SCHE83]), each performing a distinct set of services either for processes outside the security kernel or for other security kernel layers, and each depending only on the services of lower security kernel layers. A layered security kernel provides an opportunity to develop an informal proof sketch of each layer's sufficiency to meet the requirements of the next higher layer, based on an implementation specification for each layer. This proof sketch for succeedingly higher layers of the security kernel, if performed, would provide a clear correspondence of the layers in the design to the FTLS and hence to the security policy model. The correspondence of the layered implementation to the security policy model is also facilitated by breaking the implementation design into modules that have a clean, clear abstraction that allows for information hiding within modules in the manner described by Parnas [PARN72a, PARN72b]. The design of such a modular, layered security kernel is not an easy exercise, but it is by no means impossible. It pays rich dividends in providing assurance that the security kernel software in fact implements the primitives at the interface that were shown to be consistent with the security policy model.

The security kernel software will both control and be supported by the associated processor hardware. The requirements for hardware structures to support a security kernel derive from the basic requirements for a reference monitor: completeness, isolation, and verifiability. In addition, the need for well-defined subjects and objects is reflected in the choice of hardware to support a security kernel.

The key requirements are the following:

1. Hardware support for a notion of process and support for rapid change from process to process.
2. Hardware support for some sort of objects and for protection of those objects. In practice, this requirement is satisfied by a segmented virtual memory system and provisions for security kernel control over input/output operations.
3. Hardware support for the protection of the security kernel and its databases by some sort of protection domain or protection ring [SCHR72] mechanism.

The implementation of the security kernel programs typically involves the use of some form of structured coding and a higher level language. The choice of language and standards is influenced by efficiency as well as security since the security kernel is the heart of an operating system.

This brief summary of the security kernel design process has taken us from the high-level requirements of security policy and a security policy model to the implementation of the security kernel using a programming language. As the process of security kernel development proceeds, progressively more design detail is provided. At each step, though, there must be a correspondence between the detailed design produced and the requirements identified by the earlier steps. Layering, modularization, abstraction, and information hiding within the security kernel and documentation in the form of an implementation specification facilitate a demonstration of this correspondence. In this way, one can develop running code that corresponds to the requirements of a security policy model and security policy. There are formal tools available that support the demonstration of the correspondence between the FTLS and the security policy model, and there are informal techniques for demonstrating the correspondence between the code and the FTLS. However, even without mechanical tools, the discipline and structure of the security kernel design process provide a high degree of assurance about the security of the resulting system.

Improving an operating system. When we think about making an operating system or a subset of an operating system — a TCB — mimic the behavior of a reference monitor, we are frequently interested in making after-the-fact improvements to an existing system. For this reason, we often refer to such an effort as security enhancement and to the product as a security-enhanced system. The approach used to develop a security-enhanced system begins with identifying a security policy and security policy model, and specifying the interface of the security-enhanced system to the outside world (now an interface to the users and application programs). Instead of doing a layered design and structured implementation of the entire system "from the interface down to the hardware," we now modify the code of the operating system to implement the interface specification. However, in so doing, we accept that we will not be able to completely separate protection-critical components within the operating system from those that are not protection-critical. This places a fundamental limit on what one can know about the behavior of such a security-enhanced system and thus places a limit on the degree of assurance of security policy enforcement that is possible using one.

As noted in the previous section, to simplify a security kernel, we take steps to minimize its complexity, and we may minimize the number of types of objects it supports. However, in contrast to that approach, in developing the interface specification for the security-enhanced system we are talking about in this section, we may take additional short cuts.

For example, an existing operating system is likely to support a vast number of object types, and it may be impossible to apply controls to them all. We may leave some uncontrolled by the security policy model and, in doing so, leave in the system a much higher level of vulnerability to Trojan horse attack than a security kernel would suffer.

While the code of the security-enhanced system will be as clean and error-free as good practice can make it, it will not have the assurance advantages of the structured design and implementation that a security kernel has. It will be much more likely to contain residual errors in design and implementation (or even to contain malicious software), and it will be more likely to be subject to malicious software attacks from the outside.

The above is not to diminish the value of such security-enhanced systems. The primary value of these systems lies in their resistance to probing and in their improvement over a more conventional system against this threat. They typically provide a very robust environment and may have enhanced penetration resistance in addition to a rich set of security features. For that reason, and because of the feasibility of developing them with modest effort, security-enhanced systems offer an attractive option for those with limited needs for assured security.

TCB subsets. It is sometimes beneficial to develop a TCB and evaluate its security in parts, rather than as a monolith. A TCB subset [TDI91] is an extension of the reference monitor concept that enforces some access control policy on subjects that may attempt to gain access to a set of objects under the subset's control. The access control policy enforced by a TCB subset is a subset of the overall access control policy of the TCB. Access control policies of distinct TCB subsets are enforced by distinct domains within the TCB.

A TCB subset must meet the three requirements of a reference monitor: completeness, isolation, and verifiability. In fact, a reference monitor is a TCB subset. The primary difference is that a TCB subset may have an interface to a more primitive mechanism, which is also a TCB subset (enforcing a less restrictive access control policy). In the degenerate case of a single, monolithic TCB (for example, the sort of system described above as a security-enhanced system), there is a single TCB subset — which includes the portion of the system performing the reference monitor functions and the hardware. A TCB partition (more frequently called a network TCB partition or NTCB partition [TNI87]) that enforces an access control policy is a TCB subset which does not depend on another TCB subset. A TCB partition is thus in direct control of a particular, well-defined subset of subjects, objects, and hardware of a processing component of a particular networked system. NTCB partitions are discussed further in a later section.

As noted, the reference monitor implements the least restrictive mandatory access control policy in a TCB. Other TCB subsets within that TCB may implement other more restrictive access control policies, including more

restrictive mandatory access control policies and discretionary access control policies. These TCB subsets are, like supporting policy implementations, within the TCB but outside the reference monitor.

The topic of TCB subsets is only briefly introduced here. Additional information about TCB subsets may be found in the paper [SHOC87] where the notion of TCB subsets was first formalized.

Demonstrating security. The discussions above have touched briefly on the tools that we can use to convince ourselves that a system is secure. We next give a little more of a feeling for the steps that we can take to achieve assurance of system security. The quest for system security is old enough to have a body of history. We mentioned in Essay 1 the iterative "fix and test" or "penetrate and patch" approach that some took at a time when computer security was in its infancy. We will begin this discussion with a few comments on how "not to do it" and then summarize the methods that can lead to effective assurance of system security.

Testing, penetration, and reading the code. In the early days of computer security, advocates of secure systems tried to follow a path of searching for ways to penetrate the systems' controls, often relying on malicious software in the same way a potential attacker could launch a probing, penetration, or subversion attack, as described in Essay 1. Their plan was that, failing to penetrate, they could plausibly argue that there was no way to penetrate since no way was known (to them). In this scenario, if a security hole is found, it can be patched before the argument for security is made. Obviously, this argument suffers from both theoretical and practical difficulties.

One presumes that one could test all possible programs to find any that led to a security penetration. If possible, this method of exhaustion would be effective, but it is far beyond the realm of feasibility. For any real computer it would take so long that before the evaluation was finished the sun would literally have burned out! Thus any evaluation conducted by exhaustion must be so incomplete as to be ludicrous.

Practically speaking, the effort spent in penetrate and patch techniques yields poor marginal return in terms of security. Experience has shown the following conclusions to be true:

1. New penetrators tend to find new holes — even after previous teams have found all that they could. It seems unlikely that a real, malicious attacker would fail to involve new people.
2. Holes do not generally result from rank stupidity but from human oversight in dealing with a difficult design problem. Thus the fixes themselves are likely to be flawed.
3. It does not take a highly specialized expert to penetrate system security. It is true that most computer professionals do not know

ways to penetrate the systems they use; they want to do a job, not interfere with it. Yet when given the assignment, even junior and inexperienced professionals have consistently succeeded in penetration.

The real difficulty of achieving security by penetrate and patch techniques is precisely the difficulty of finding errors in a program by testing. A test can demonstrate the presence of an error but, short of exhaustion of every possible combination of inputs, it cannot demonstrate the absence of errors. Much of the weakness of security in existing systems results, in the first instance, from the fact that these systems were designed to perform a function and tested to assure that they did so correctly. The function to be performed had the nature of "the right answer" rather than secure operation.

A failure of system security results from the failure by a designer or implementer to anticipate a "functional" requirement for security, and to build and test a system to meet that requirement. Security is a fundamentally "negative" requirement stating that, for all possible applications, unauthorized access will not be granted. When examining the system, the penetrator tests a different function — namely, that there exists at least one way in which unauthorized access to information will be granted. There are many ways of succeeding with this function.

In the limit, the problem of security is that the designer must search out every way to penetrate security and correct all; the penetrator is really interested in finding and using only one. This is an unbalanced "game of wits" in which the attacker has a substantial advantage.

The key to successful functional testing lies in factoring the system's structure into the selection of test cases, and in designing the system to support testing. In a real sense, this is the approach we take with a security kernel: designing the system so that, by means of a proven security policy model, security becomes a positive requirement rather than the absence of errors, and structuring the system to support analysis and testing. Since testing security becomes testing for positive requirements, it is possible to test in the same manner as in the usual software testing scenario. In contrast to the strategy of testing for all possible penetration approaches, testing a security kernel is quite doable.

TCB design and security verification. When we discussed design and specification of a security kernel, we presented the process of system design, beginning with security policy and culminating in security kernel code. We also indicated that formal security kernel verification tools could follow the same path. In the paragraphs below, we summarize the process of security kernel verification, then indicate how we might apply some of the same steps to the security evaluation of the TCB of a security-enhanced operating system.

The security policy model is the linchpin of verification. It translates the fundamentally negative requirements of security mentioned in the previous section into positive properties that can be verified. The security policy model bridges between the people-oriented world of security policy and the computer-oriented world of the reference monitor abstractions of subjects and objects. The step from security policy to security policy model is necessarily an informal one — at some point we must make the translation from English "legalese" to formal, mathematical language. By taking this step at the level of the security policy model, we reduce the complexity of the mechanism that we must review in an unstructured way. In the case of the Bell-LaPadula security policy model, while there is a fair amount of formalism to be dealt with, the basic objectives and approach are relatively simple. Numerous readers of the security policy model have convinced themselves that it is a formal statement of the objectives that it claims to support. It is easy to see that a security policy model that is simple and abstract is necessary to the verification process.

Once the security policy model is accepted, it provides a formal basis for the security evaluation of the rest of the TCB. We can state our TCB's interface specifications in a formal language and verify the correspondence between security policy model and interface specification. We can also specify the security kernel interface in the formal language and verify that the system implemented is the same as the security kernel whose interface has been specified. The specifications of the security kernel implementation are presented module by module for each layer and layer by layer in a structure corresponding to the security kernel software itself. This enables a proof sketch of each module's sufficiency to meet its specification and each layer's sufficiency to meet the requirements of succeedingly higher layers, all the way up to the interface, as described earlier.

The security kernel implementation specification supports the proof sketch of each corresponding module of security kernel code. The technology of program verification does not practically allow for formal code verification of large real programs. However, there are a few key factors that contribute to the success of this approach for providing a meaningful verification of the implementation. The decomposition of the implementation design into "Parnas modules" with a clean, clear abstraction that allows for information hiding, and strict layering of the modules, both make formal verification more thinkable and offer a structured way to read the code and review its correctness informally. In fact, it is information hiding and strict layering that provide the real basis for progressively developing a proof sketch of the code's sufficiency to meet the interface specification, module by module and layer by layer, as mentioned in the previous paragraph. Our experience shows that the difficulty of reviewing operating system code for correct and secure operation results from both the complexity of the code and the lack of any clear definition of what the

code is supposed to be doing. The implementation specifications for a security kernel support the determination of what the code should be doing.

Beyond the higher level code are machine language and hardware implementation issues. Verification technology is not up to these issues yet. Again, however, the structure of the security kernel and its specifications at least offer meaningful support for a partially informal but effective assessment of security.

There is one more aspect of assessing security that we would like to address. It should also include a determination of whether the implementation of the reference monitor contains covert channels. Covert channels are flows of information between access classes in a manner that is counter to the mandatory access control policy portion of the security policy but allowed by the implementation. It is important to recognize that valid security policy models do not have covert channels; covert channels are an implementation phenomenon. A security policy model that allows information to flow in a manner counter to the security policy is a flawed security policy model [LEVI90].

Recall we observed earlier that the security policy model represents the state of the TCB. Covert channels operate because this state information is sometimes passed to subjects outside the reference monitor. The two types of covert channels — covert storage channels and covert timing channels — differ by the manner in which the state information is passed. In covert storage channels, information is passed out of the reference monitor through the value of an exception or error code. In covert timing channels, information is passed out of the reference monitor through a delay (that is, a measurable change in response time) observable by a subject. The only other way besides through covert channels in which information could flow through a reference monitor in a manner counter to the security policy would be through the use of a flawed security policy model, as described in the previous paragraph.

For an example of a covert storage channel, consider two subjects with Trojan horses operating at different access classes ("Marketing" and "Accounting") such that the security policy which the reference monitor is enforcing forbids any information to flow between the two. If, for example, one of the state variables in the reference monitor is used to indicate "disk_full," then information can easily flow between these two subjects. The Trojan horse in one of the subjects can signal a single bit of information to the other by filling the disk (or not filling it) at an agreed-upon time or after an agreed-upon condition, at which time the Trojan horse in the other subject checks to see whether the disk is full (for example, by trying to write to the disk). This signaling can be done repeatedly and rapidly. The maximum rate at which the signaling of a particular covert channel can be accomplished is called its bandwidth.

A similar exploitation scenario can be constructed to illustrate covert timing channels. Suppose that instead of the "disk_full" error condition,

our Trojan horses were to use the amount of time required for a read from the disk to signal a bit. Suppose the Trojan horses in both subjects know how long a normal disk access takes. To signal a bit of information, the "writer" Trojan horse need only make sure the disk is busy enough to slow down the "reader" Trojan horse's disk access. Again, this can be done repeatedly and rapidly, and one can calculate or measure the bandwidth.

One might ask, "Doesn't the existence of covert channels destroy the assurance of enforcement of the mandatory access control policy we worked so hard to achieve?" Actually, while uncorrected, covert channel bandwidths can be quite high, though they may be difficult to exploit at the highest rates because of "noise." But there are many techniques available for reducing the bandwidths. In fact, security kernels are available today that can be configured to have no covert storage channels [THOM90]. The bandwidths of covert timing channels can be reduced in the most brute-force way by slowing down the functions that allow them to operate if it is detected that a particular bandwidth threshold is about to be exceeded. This can be mechanically accomplished by inserting delays wherever the functions are called. Of course, this can have an adverse impact on performance, but it permits stronger enforcement of the mandatory access control policy.

Finally, we promised to include in this discussion a comment about the problem of assessing the security of the TCB of a security-enhanced operating system. In such a system, we are concerned about features at the TCB interface as well as about having as "correct" an implementation as we can. The security policy model supports a review of the interface to the TCB in this case. We may prepare a formal or English specification of the TCB's interface, then review it for inappropriate ways that a subject can gain access to an object. We will probably supplement the interface analysis with a penetration test to help make sure that no obvious flaws remain. We may also look for covert channels and even work to reduce particular covert storage channels. However, experience tells us that the lack of a structured — layered and modular — implementation in a system which is not designed to be subjected to the level of analysis of a security kernel results in a system that is subject to penetration (as well as subversion of security mechanism), as defined in Essay 1. Such a system is of some value, though, since it can offer a relatively high degree of resistance to probing.

For further reading. More information on building a secure computer system can be found in a readable first book [GASS88] on this subject that presents difficult and subtle topics clearly, while achieving a good depth of coverage.

Special considerations for networks

As noted in Essay 1, networks present greater accessibility to potential penetrators by providing more ways of accessing the systems (through other interconnected systems, through dial-up terminals, through taps into the communication lines used by the networks themselves, and so on). Obviously, networks are strongly dependent on computer systems to provide the networks' services. The resources being shared within computer systems connected to or implementing the network provide numerous opportunities for sensitive information to "leak" out or be illicitly modified from within. The networks themselves provide no fundamentally new vulnerabilities, except through the enhanced accessibility to the computer systems that protect the information. The enhanced accessibility facilitates the exfiltration of information, which, as noted earlier, may aid the successful use of Trojan horses.

The reference monitor concept scales very nicely to networks that possess a coherent network security architecture and design — in other words, coherent security policies, security objectives, and protocols. In a network context, we speak of the trusted computing base or TCB as a network trusted computing base or NTCB. The computer systems that implement the switching and other processing within the network, as well as the client systems of the network, provide the opportunities for sensitive information to leak out or be illicitly modified, as described above. The NTCB prevents such leakage and illicit modification.

The enhanced accessibility of the connections between the computers or switching elements of the network system is countered most often through the use of cryptography, either alone or in combination with other security services (for example, physical protection of the wire) and security supporting protocols. The enhanced accessibility of one computer system from another is countered through applying the reference monitor concept to the computer systems that make up the network system.

The unique aspects of building secure networks are described in the following two sections. The first section applies the theory of computer security to networks with a coherent network security architecture, in the form of an NTCB [TNI87]. (The section contains some actual excerpts from the "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria" [TNI87] to explain the concepts.) The second section describes some of the fundamental access control problems that must be addressed in implementing cryptography within a computer system. We will not otherwise delve into the technology of cryptography or security supporting protocols in this essay.

Network trusted computing base. As stated previously, this section applies the theory of computer security — the reference monitor concept — to networks. In so doing, it also expands somewhat on the application of the

reference monitor concept in stand-alone computing environments — specifically, in the area of input/output devices. So, note that the information presented here about devices also pertains to stand-alone computing environments; it is presented here as a convenience to the discussion of networks and the NTCB.

One of the first steps in the design of a secure network is the development of coherent network security architecture and design. The network security architecture addresses the security policies, objectives, and protocols of the network. The network security design specifies the interfaces and services that must be incorporated into the network for it to be secure, relative to its security policy.

The overall network security policy is the security policy of the NTCB. It may be decomposed into security policy elements that are allocated to appropriate components and used as the bases for the security policies for those components. This technique is referred to as partitioning a network's security policy into component security policies or NTCB partitions [SCHE85]. As used in this essay, a component is a collection of hardware, firmware, and software that performs a specific security function in a network and that contains an NTCB partition. A component is defined recursively, in that a component may consist of other components. The distinguishing quality of a component is that it has a security policy partition allocated to it — induced on it by the overall network security policy. The recursive definition of component is based on the fundamental concept that it contains an NTCB partition, a recursively defined concept based on the "partition" of mathematics. (As in mathematics, if a set is partitioned and if one of the partitions is further partitioned, the result is still partitions of the original set.) Such recursive definitions of component and NTCB partition are very useful in actual practice, since, once the security of a component is evaluated, the results of that evaluation may be directly reused in evaluating the security of a component which contains it. (See the section "Evaluation by parts" in Essay 6 for a further discussion of applying the strategy of "divide and conquer" by separately evaluating NTCB partitions and using the results to evaluate more complex components within which they may be contained.)

Therefore, one of the powerful design tools provided by this approach is the ability to compose a set of components, and then treat the result as a single componenthaving a security policy made up of the set of allocated security policies and defined network interfaces. Once a composition is performed, the resulting component may be considered in a variety of network security architectures without having to reconsider the original components or the process of composition.

Much of the usefulness of this approach lies in its general applicability. For example, a host computer system that implements a full set of user services may be treated as a component for purposes of composing the computer system into the network system.

Components can include entire subnetworks within an overall network, encryption systems, local area networks, digital PABX systems, packet switched or circuit switched systems, and virtual machines running on a virtual machine monitor (VMM) on a single computer system, when analyzed as a network. The ability to view a set of virtual machines as distinct network components is a powerful tool that allows varied security policies to be implemented in virtual machine components on top of a VMM component. It is an efficient technique in that if the underlying VMM component has already been examined and found to enforce a security policy required for the network system, it does not have to be reexamined after the composition. In the case of network components implemented as virtual machines, the interface between the VMM component and the virtual machine components is the set of functions provided by the VMM component. An example of the use of this technique is described elsewhere [IRVI91].

Abstractly, the reference monitor for a partitioned NTCB is realized as a collection of security kernels for individual components. To obtain the required levels of assurance that each such security kernel enforces its security policy, a formal security policy model is formulated for each such component. However, it would be too restrictive to require that the formal security policy model for each security kernel be the same, or that an overall formal security policy model be formulated for the network. Instead, each formal security policy model is shown by convincing arguments to correctly represent the security policy allocated to the component. Since the overall security policy is stated informally, the convincing arguments are also stated informally, in the same manner as that described in the earlier section "TCB design and security verification."

The formal security policy models of components therefore provide the basis for the security policy exercised by the NTCB over subjects and objects in the entire network. The purpose of the formal security policy model for each component is to serve as a precise starting point in the chain of arguments leading to the sufficient levels of assurance required for each component in the network and ultimately for the whole network. These arguments are easier to make if the formal security policy model has an intuitively attractive resemblance to the abstractions of subject, object, and access properties of the computer system TCB on which the component is implemented.

Thus, the reference monitor subject and object definitions that pertain for a computer system TCB are sufficient also for an NTCB. The reference monitor represents the fundamental security policy enforcement at the individual component level but may not directly represent all of the overall network security policy issues such as the network's connection policy. In many networking environments, the overall network security policy includes controlling the establishment of authorized connections across the network. The access control mediation performed by the components of

these networks enforces the establishment of connections between host computers on the network and provides the basis for a connection-oriented abstraction. Understanding the connection-oriented abstraction for a given network may be essential to understanding that network's overall security policy. The network security architecture describes the linkage between the connection-oriented abstraction and its realization in the individual components of the network.

Subjects in networks are therefore active entities outside the perimeter of the reference monitor in each component; that reference monitor's objects are passive entities that exist in the component which implements that reference monitor. Reference monitor support to ensure control over all the operations of each subject in the network is completely provided within the single NTCB partition on which that subject interfaces to the NTCB. This means that the entire portion of the formal security policy model's "secure state" that may undergo transitions because of the actions of this subject is likewise contained in the same component.

The level of abstraction of the formal security policy model and the set of subjects and objects that are explicitly represented in the formal security policy model are clearly affected by the NTCB partitioning. Subjects and objects are represented explicitly in the formal security policy model for a particular component whose NTCB partition exercises access control over them. Global network security policy elements that are allocated to a component must be represented by the formal security policy model for that component.

Each partition of the NTCB therefore enforces the security policy over all subjects and objects in its component. In a network, the responsibility of an NTCB partition encompasses all security policy functions in its component that would be required of a TCB in a stand-alone system. In particular, subjects and objects in a particular component that are used for communication with other components are under the control of the single component which contains them. Conceptual entities associated with communication between two components, such as sessions, connections, and virtual circuits, may be thought of as having two ends, one in each component, where each end is represented by a local object. Communication is viewed as an operation that copies information from an object at one end of a communication path to an object at the other end. Transient data-carrying entities, such as datagrams and packets, exist either as information within objects at one end of the communication path, or as a pair of objects, one at each end of the communication path.

Access by a subject in one component to information contained in an object in another component requires the creation of a subject in the remote component that acts as a surrogate for the first subject. The security policy must be enforced at the interface of the reference monitor (that is, the mechanism that controls physical processing resources) for each NTCB partition.

Recall the introduction of devices and communication channels and the roles they play in the reference monitor. The only links between NTCB partitions are the various communication channels provided by devices. Devices may import or export information only under the explicit control of the reference monitor in accordance with the security policy.

A basic choice must be made about each device of a TCB or of an NTCB partition allocated to a network component. That choice is

1. whether the information that will be imported or exported through the device at any particular point in time will have its label associated and flowing with it, or
2. whether the access class of the information will be established administratively (that is, whether the security administrator will "tell" the TCB or NTCB partition what the access class of the information is that can flow through that device).

If the access class of the information is associated with a label that flows with the information, that device may be used to import or export information whose access class varies within a range enforced by the reference monitor implementation, and it is called a multilevel device. On the other hand, if the access class of the information that may flow through the device is established administratively and thus cannot vary (until a different access class is administratively established), that device is called a single-level device. It should be explicitly recognized that a device or a communication channel that does not support the transmission of labeled information is, by definition, single-level. In a network component, therefore, the devices coupling the communication channel to the processing nodes may be single-level devices, administratively set to import or export information of the same access class, or they are multilevel devices, capable of handling information of some intersecting access class.

A single-level device may be regarded either as a subject or an object. A multilevel device is regarded as a subject that is within the TCB or NTCB partition (that is, it is a trusted subject — trusted to correctly associate information and labels within some range of access classes). However, as with all subjects, it is outside the reference monitor. For a multilevel device, the range of the subject is the minimum-maximum range of access classes of information that may be transmitted over the device.

To support single-level devices, the TC or NTCB includes a trusted subject or a reliable communication mechanism by which the TCB or NTCB and a security administrato (via a trusted path) can designate the single access class of information imported or exported. A single-level devic also has a range of access classes within which the access class set by the security administrator must fall, but it is important to note that single-level devices have only a single access class at one time.

The allocation of mandatory and discretionary access control policies to different components in a network may require communication between trusted subjects that are part of the NTCB partitions in different components. This communication is normally implemented with a protocol between the subjects as peer entities. The protocols and data which they carry between these subjects that associate the access class with exported information (for multilevel channels) or that designate the single access class of information imported or exported (for single-level channels) require special attention to ensure the NTCB's integrity. These protocols are actually being used to communicate internal NTCB data among NTCB subjects (that is, trusted subjects outside the reference monitor but inside the NTCB) belonging to different NTCB partitions. This data must be protected against external interference or tampering. For example, a cryptoseal (see the next section) or physical means may be used to protect such data exchanged between NTCB partitions.

The need to be concerned about the integrity of internal NTCB data is clear if one considers that labels are among the sort of information communicated between NTCB partitions. Since there is no standard representation for labels communicated between components, each component needs to be able to translate from the form in which the label is communicated to its own internal representation in a highly reliable way.

Subjects outside the NTCB but not direct surrogates for a user (human) are termed internal subjects. Protocol handlers are examples of services that are usually provided by internal subjects. It is important to understand that a key distinction of internal subjects from "regular" subjects is that since they are not acting for users, there is little or no need for a discretionary access control policy to be enforced over them. Similarly, since the purpose of supporting policy is to provide the tie between persons and access control policies, there is no need for supporting polic (for example, identification/authentication and audit policies) to be concerned with them.

Therefore, a crucial step in the design of a network is the allocation of the NTCB security policy to individual components within the network that share information using devices to provide communication channels. This partitioning into components must be done in such a way that all of the following conditions can be easily validated:

1. A subject is confined to a single component throughout its lifetime.
2. A subject may directly access only objects within its component.
3. Every component contains a component reference monitor (or, for security-enhanced systems, the functions of a reference monitor) that mediates all accesses made locally to enforce an overall access control policy for the whole network.
4. All communication channels linking components do not compromise the security of the information entrusted to them. This allows the

conclusion that the total collection of components enforces the network's overall security policy.

Access control and cryptography in networks. Access control and cryptography interrelate in at least two fundamental ways. For one, cryptography may be useful in supporting access control decisions. As stated above, the reference monitor represents the fundamental security policy enforcement at the individual componen level but may not directly satisfy all the overall network security policy requirements such as the network's connection policy. In many networking environments, the overall network security policy includes controlling the establishment of authorized connections across the network and protecting the information during transmission. The access control mediation performed by the components of these networks enforces the establishment of connections between host computers (including, for example, mainframes, workstations, and/or personal computers) on the network. Cryptography may be used (with specific security-enhancing communication protocols) to support the access control decisions (that is, to provide the mechanism to ensure that the host computers may communicate only as authorized by the overall network security policy). Of course, this is no different in principle from the use of cryptography in media encryption. All data on a medium such as a disk or transmission cable is encrypted to eliminate the risk of exposure in case the medium is stolen or tapped into — in other words, to prevent illicit access to the media.

That is all we are going to say specifically about the use of cryptography to enforce access control. Instead, this section focuses on access control issues involved in the implementation of cryptography on a computer system. Cryptography is, of course, in its simplest application, a way of protecting information from disclosure. However, the success of cryptographic techniques fundamentally depends on the protection of certain keys from disclosure and the assured application of the intended keys and algorithms. If the keys that require protection from disclosure are in any way disclosed, the information whose protection depends on those keys can no longer be assumed to be protected. If incorrect keys or algorithms (that is, those supplied by an attacker) are used, the protection is illusory.

It is because resource-sharing computer systems have difficulty with secure simultaneous processing of sensitive and nonsensitive information that computer security has arisen as an important discipline. For that same reason, systems have difficulty with secure simultaneous processing of plaintext, ciphertext, and keys. In a nutshell, this is why depending on cryptography to encrypt and keep separate individual files is usually not a good idea — because, to be secure, the system on which the cryptography is implemented must already have the capability of keeping separate the information of different sensitivities. That means that one can

do no more for access control with cryptography than what a TCB enforcing a mandatory access control policy already supports. Providing individual file encryption is, therefore, quite susceptible to Trojan horses without the presence of a TCB that is enforcing a mandatory access control policy, rendering it also quite susceptible to false assurance. However, use of encryption under the control of a TCB enforcing a mandatory access control policy to protect information on a medium (for example, a disk or a transmission cable) is quite achievable, assuming one has such a TCB [FELL87, THOM90, WEIS92].

Up to this point in this section, we have only explicitly mentioned using cryptography in conjunction with a TCB to provide protection from disclosure. However, it should be noted that cryptography can also be used in conjunction with a TCB enforcing a mandatory access control policy to provide protection from illicit modification.

Most readers are probably familiar with techniques of using parity, a checksum, or some other checkfunction (that is, the result of some algorithmic function carried out on the data) to detect data errors, especially in communication. While such techniques are useful for detecting and sometimes even correcting random or burst errors on a communication line or in memory, they cannot protect against an intruder, with access to the data, illicitly modifying bits of the data. If an intruder can modify bits of the data, he can also recalculate the checkfunction (since the algorithm is typically well known) and insert this modified checkfunction.

The technique used to provide valid protection from illicit modification is called cryptosealing. Cryptoseals themselves are data that represent the result of a function involving cryptography being carried out over the data being protected. Cryptoseals are typically many fewer bits of data (64 to 128 bits) than the information they are protecting. They have the property that if any bit of data (or of the cryptoseal) is modified, the modification can be detected by carrying out the cryptographic function over the data again, using the same key. If the result is different from the cryptoseal, the data was modified. Cryptosealing is a useful technique when physical controls on access to the medium on which the data is held cannot be guaranteed. For example, bank (ATM) cards commonly use a form of cryptoseal to ensure that bank account numbers are not "forged." A number of techniques for generating and using cryptoseals in different applications have been standardized (for example, CCITT X.509 and ANSI X9.9).

The problems with trying to use cryptosealing on a resource-sharing computer system are the same as those noted above for cryptography in general. Similarly with the general case, the problem is secure simultaneous processing of data being cryptosealed, the cryptoseals themselves, and keys. That means that one is subject to the same security problems inherent to resource-sharing computer systems when using cryptography or specifically cryptosealing inside that computer system. In other words,

cryptography and cryptosealing are not substitutes for a TCB enforcing a mandatory access control policy, and, if implemented in a computer system that is not enforcing a mandatory access control policy, there is a significant possibility of false assurance. Again, providing cryptosealing over individual files is susceptible to Trojan horses without the presence of a TCB enforcing a mandatory access control policy. However, use of cryptosealing under the control of a TCB enforcing a mandatory access control policy to protect information on a medium is a quite achievable task, assuming one has such a TCB. The application of this technique to database management systems has been addressed rather extensively [DENN84, DENN85], and the same basic techniques can be used for networks, as overviewed below.

With a TCB that enforces a mandatory access contro policy to implement cryptosealing, an important class of NTCB extension can be built — commonly termed guards. The concept of a guard was first introduced in the early 1980s in a study by Roger Schell for the Korean Air Intelligence System or KAIS [DENN84]. Jim Anderson then used these ideas to develop the recon guard [ANDE81]. Guard is a term sometimes used (imprecisely and improperly) to indicate almost any form of application providing a security function in a network. We use the term precisely and consistently with the historical usage to refer to a technique for implementation of a specific function: communication of information of multiple sensitivities between secure computers by using a system high network, operating with the access class of the most sensitive information being communicated.

The guard method works in the following manner: For a particular transmission of information, two guards are involved. One is associated with the transmitting computer and has a secure communications path to the transmitting computer's TCB, and one is associated with the receiving computer and has a secure communications path to the receiving computer's TCB. The guard associated with the transmitting computer receives the data to be communicated and a label indicating its access class from the transmitting computer's TCB. (Note that the label is often implicit, say, at the single system high level of the transmitting computer.) The transmitting guard then cryptoseals the data to be transmitted, together with a representation of the label. This data may now be exported from the transmitting guard, since the network is at a single level which dominates that of the data, and it can be subsequently received by the receiving guard. The receiving guard now checks to see whether the cryptoseal is intact, indicating that no modification of the data or label has occurred. Assuming the cryptoseal checks out, the receiving guard hands off the data and its label to the receiving TCB for proper disposition based on the validated label. Note that the system high network can be carrying data of various sensitivities from one or more pairs of guards. The guards have now logically implemented a multilevel network out of a system high net-

work, albeit a network providing transmission protection (that is, via cryptography) for up to the most sensitive information.

Some past efforts have used the terms guard and filter almost interchangeably. It is useful, however, to distinguish between these two. In a sense, both devices perform sanitization and/or downgrading from one access class to another in a manner normally counter to the access control policy. They perform this function against data sent to them, preventing unauthorized disclosure based on some policy, and then release the data. The nature of this policy allows us to distinguish between the relatively weak nature of filters and the more robust nature of guards. A policy for a filter may rely on virtually any content or context-dependent criteria for its decision to release the data. An example of a filter is a deviceperforming "dirty word" sanitization, where data containing specific code words or combinations of code words will be deemed unreleasable. Thus, filters can be described as devices that determine the releasability of data by "figuring out" the sensitivity of the data.

The major distinction between filters and guards lies in the outcome of their correct operation. A filter examines data from a system high environment and attempts to downgrade this data to something other than its original system high access class. In the face of malicious software, no claim can be made about whether this downgrade is valid or not, since any number of tricks can be used by an attacker to "sneak information past" the filter, in a manner like that described earlier for illicit indirect access. A guard, on the other hand, bases its sanitization decision on the credible label, which is assigned and checked at a place where it is reliable and which is protected by a cryptoseal. Therefore, even though there are some problems that limit the protection against a determined attacker [DENN85], guards do offer some real protection against malicious software.

Conclusion

We have presented an introduction to the concepts and terminology of computer security and to the reference monitor concept that allows us to build secure systems in an orderly way. We have presented an introductory view of the role of a security policy in the development of a secure system and have outlined the processes of specifying, designing, implementing, and evaluating a TCB. We have also provided a summary of the application of computer security to networks. In all, we have addressed concepts and terms that we consider to be among the most critical to gaining a fundamental understanding of this technology.

The reference monitor concept, if applied properly, is a powerful tool for the control of computer misuse. While human error and user abuse of authority are largely external problems, a TCB is a vehicle for constraining users to their authorized domain and for obtaining data for an audit trail of the users' actions. Application of the reference monitor concept lets

us enhance the security of an operating system and make it essentially immune to probing.

Finally, the consistent application of the reference monitor concept and the system implementation techniques that have grown up with it allows us to develop a security kernel for preventing penetration or subversion of security mechanism with a high degree of assurance. There is a firm technical foundation on which to build a security kernel for a specific system in a manner that yields good performance. The designs and implementations to date have empirically validated the principles needed to use the security kernel without degrading system capabilities [SCHE85a].

Epilogue

The statements above are strong ones. The reader may well ask, "Is all of this real?" In fact, the technology has been applied to real systems. Security-enhanced systems based on the reference monitor concept are in use today in commercial organizations, in the Defense Department, and in other places throughout the world. Several organizations have successfully developed security kernels, including the Gemini Trusted Network Processor (GTNP) from Gemini Computers [THOM90], the Honeywell SCOMP [FRAI83], the XTS-200/STOP from Honeywell Federal Systems, Inc., the Boeing Multilevel Secure Local Area Network Server System [SCHN85], the VAX VMM Security Kernel from Digital Equipment Corp. [KARG91], and Blacker [WEIS92]. The technology that we describe has sufficient credibility to be the basis of the National Computer Security Center's evaluation criteria for TCBs [TCSE85], trusted networks [TNI87], and trusted database management systems [TDI91] (presented in Essay 6), as well as the basis of multinational evaluation criteria [ITSE91].

We hope that this essay has given the reader a sense of the concepts and terminology of the technology that allows us to solve a significant portion of the computer security problem. The moral of this essay is perhaps summarized by the following quotation [SCHE79]: "Do not trust security to technology unless that technology is demonstrably trustworthy, and the absence of demonstrated compromise is absolutely *not* a demonstration of security."

Acknowledgments

We would like to give a special acknowledgment to the contributions of Steven B. Lipner of Trusted Information Systems. Substantial portions of the text as well as several key insights related to the reference monitor are drawn from material he previously prepared and he has graciously permitted us to use in this essay.

In addition, acknowledgment is given to "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," National Com-

puter Security Center [TNI87], from which excerpts have been drawn for incorporation into the discussion of NTCB-related concepts.

We also gratefully acknowledge the substantial constructive comments and suggestions provided by James P. Anderson on an earlier version of this essay.