Authentication, Access Control, and Intrusion Detection*

Ravi S. Sandhu† and Pierangela Samarati‡

1 Introduction

An important requirement of any information management system is to protect information against improper disclosure or modification (known as confidentiality and integrity respectively). Three mutually supportive technologies are used to achieve this goal. Authentication, access control and audit together provide the foundation for information and system security as follows. *Authentication* establishes the identity of one party to another. Most commonly authentication establishes the identity of a user to some part of the system typically by means of a password. More generally, authentication can be computer-to-computer or process-to-process and mutual in both directions. *Access control* determines what one party will allow another to do with respect to resources and objects mediated by the former. Access control usually requires authentication as a prerequisite. The *audit* process gathers data about activity in the system and analyzes it to discover security violations or diagnose their cause. Analysis can occur off-line after the fact or it can occur on-line more or less in real time. In the latter case the process is usually called *intrusion detection*. This chapter discusses the scope and characteristics of these security controls.

Figure 1 is a logical picture of these security services and their interactions. Access control constrains what a user can do directly, as well what programs executing on behalf of the user are allowed to do. Access control is concerned with limiting the activity of legitimate users who have been successfully authenticated. It is enforced by a reference monitor which mediates every

*Part of this paper appeared under the title "Access Control: Principles and Practice" in *IEEE Communications,* vol. 32, no. 9, pp. 40-48 [SS94].

†ISSE Department, MS 4A4, George Mason University, Fairfax, VA 22030. Voice: +1 703-993-1659, Fax: +1 703-993-1638, email: sandhu@isse.gmu.edu, http://www.isse.gmu.edu/faculty/sandhu

‡Dipartmento di Scienze dell'Informazione, Universitá degli Studi di Milano, Via Comelico 39/41, 20135, Milano, Italy. Voice: +39-2-55006257, Fax: +39-2-55006253, Email: samarati@dsi.unimi.it

attempted access by a user (or program executing on behalf of that user) to objects in the system. The reference monitor consults an authorization database in order to determine if the user attempting to do an operation is actually authorized to perform that operation. Authorizations in this database are administered and maintained by a security administrator. The administrator sets these authorizations on the basis of the security policy of the organization. Users may also be able to modify some portion of the authorization database, for instance, to set permissions for their personal files. Auditing monitors and keeps a record of relevant activity in the system.

Figure 1 is a logical picture and should not be interpreted literally. For instance, as we will see later, the authorization database is often stored with the objects being protected by the reference monitor rather than in a physically separate area. The picture is also somewhat idealized in that the separation between authentication, access control, auditing and administration services may not always be so clear cut. This separation is considered highly desirable but is not always faithfully implemented in every system.

It is important to make a clear distinction between authentication and access control. Correctly establishing the identity of the user is the responsibility of the authentication service. Access control assumes that identity of the user has been successfully verified prior to enforcement of access control via a reference monitor. The effectiveness of the access control rests on a proper user identification and on the correctness of the authorizations governing the reference monitor.

It is also important to understand that access control is not a complete solution for securing a system. It must be coupled with auditing. Audit controls concern a posteriori analysis of all the requests and activities of users in the system. Auditing requires the recording of all user requests and activities for their later analysis. Audit controls are useful both as deterrent against misbehavior as well as a means to analyze the users behavior in using the system to find out about possible attempted or actual violations. Auditing can also be useful for determining possible flaws in the security system. Finally, auditing is essential to ensure that authorized users do not misuse their privileges. In other words, to hold users accountable for their actions. Note that effective auditing requires that good authentication be in place, otherwise it is not possible to reliably attribute activity to individual users. Effective auditing also requires good access control, otherwise the audit records can themselves be modified by an attacker.

These three technologies are interrelated and mutually supportive. In the following sections we respectively discuss authentication, access control and auditing and intrusion detection.

2 Authentication

Authentication is in many ways the most primary security service on which other security services depend. Without good authentication there is little point in focusing attention on strong access control or strong intrusion detection. The reader is surely familiar with the process of signing on to a computer system by providing an identifier and a password. In this most familiar form authentication establishes the identity of a human user to a computer. In a networked environment authentication becomes more difficult. An attacker who observes network traffic can replay authentication protocols to masquerade as a legitimate user.

More generally, authentication establishes the identity of one computer to another. Often, authentication is required to be performed in both directions. This is certainly true when two computers are engaged in communication as peers. Even in a client-server situation mutual authentication is useful. Similarly, authentication of a computer to a user is also useful to prevent against spoofing attacks in which one computer masquerades as another (perhaps to capture user identifiers and passwords).

Often we need a combination of user-to-computer and computer-to-computer authentication. Roughly speaking, user-to-computer authentication is required to establish identity of the user to a workstation and computer-to-computer authentication is required for establishing the identity of the workstation acting on behalf of the user to a server on the system (and vice versa). In distributed systems authentication must be maintained through the life of a conversation between two computers. Authentication needs to be integrated into each packet of data that is communicated. Integrity of the contents of each packet, and perhaps confidentiality of contents, must also be ensured.

Our focus in this chapter is on user-to-computer authentication. User-to-computer authentication can be based on one or more of the following:

- something the user knows, such as a password,
- something the user possesses, such as a credit-card sized cryptographic token or smart card, or
- something the user is, exhibited in a biometric signature such as a fingerprint or voice-print.

We now discuss these in turn.

2.1 Authentication by Passwords

Password-based authentication is the most common technique but it has significant problems. A well-known vulnerability of passwords is that they can be guessed, especially since users are prone to selecting weak passwords. A password can be snooped by simply observing a user keying it in. Users often need to provide their password when someone else is in a position to observe it as it is keyed in. Such compromise can occur without the user even being aware of it. It is also hard for users to remember too many passwords, especially for services that are rarely used. Nevertheless, because of low cost and low technology requirements, passwords are likely to be around for some time to come.

An intrinsic problem with passwords is that they can be shared, which breaks down accountability in the system. It is all to easy for a user to give their password to another user. Sometimes poor system design actually encourages password sharing because there may be no other convenient means of delegating permissions of one user to another (even though the security policy allows the delegation).

Password management is required to prod users to regularly change their passwords, to select good ones and to protect them with care. Excessive password management makes adversaries of users and security administrators which can be counter-productive. Many systems can configure a maximum lifetime for a password. Interestingly, many systems also have a minimum lifetime for a password. This has come about to prevent users from reusing a previous password when prompted to change their password after its maximum life has expired. The system keeps a history of, say, eight most recently used passwords for each user. When asked to change the current password the user can change it eight times to flush the history and then resume reuse of the same password. The response is to disallow frequent changes to a user's password!

Passwords are often used to generate cryptographic keys which are further used for encryption or other cryptographic transformations. Encrypting data with keys derived from passwords is vulnerable to so-called dictionary attacks. Suppose the attacker has access to known plaintext, that is the attacker knows the encrypted and plaintext versions of data which was encrypted using a key derived from a user's password. Instead of trying all possible keys to find the right one, the attacker instead tries keys generated from a list of, say, twenty thousand likely passwords (known as a dictionary). The former search is usually computationally infeasible while the latter can be accomplished in a matter of hours using commonplace workstations. These attacks have been frequently demonstrated and are a very real threat.

Operating systems typically store a user's password by using it as a key to some cryptographic transformation. Access to the so-called encrypted passwords provides the attacker the necessary known plaintext for a dictionary attack. The Unix system actually makes these encrypted passwords available in a publicly readable file. Recent versions of Unix are increasingly using shadow passwords by which this data is stored in files private to the authentication system. In networked systems known plaintext is often visible in the network authentications protocols.

Poor passwords can be detected by off-line dictionary attacks conducted by the security administrators. Proactive password checking can be applied when a user changes his or her password. This can be achieved by looking up a large dictionary. Such dictionaries can be very big (tens of megabytes) and may need to be replicated at multiple locations. They can themselves pose a security hazard. Statistical techniques for proactive password checking have been proposed as an alternative [DG93].

Selecting random passwords for users is not user friendly and also poses a password distribution problem. Some systems generate pronounceable passwords for users because these are easier to remember. In principle this is a sound idea but some of the earlier recommended methods for generating pronounceable passwords have been shown to be insecure [GD94]. It is also possible to generate a sequence of one-time passwords which are used one-by-one in sequence without ever being reused. Human beings are not expected to remember these and must instead write them down or store them on laptop hard disks or removable media.

2.2 Token-Based Authentication

A token is a credit-card size device that the user carries around. Each token has a unique private cryptographic key stored within it, used to establish the token's identity via a challenge-response handshake. The party establishing the authentication issues a challenge to which a response is computed using the token's private key. The challenge is keyed into the token by the user and the response displayed by the token is again keyed by the user into the workstation to be communicated to the authenticating party. Alternately, the workstation can be equipped with a reader that can directly interact with the token eliminating the need for the user to key in the challenge and response. Sometimes the challenge is implicitly taken to be the current time, so only the response needs to returned (this assumes appropriately accurate synchronization of clocks).

The private key should never leave the token. Attempts to break the token open to recover the key should cause the key to be destroyed. Achieving this is in face of a determined adversary is a difficult task. Use of the token itself requires authentication, otherwise the token can be surreptitiously used by an intruder or stolen and used prior to discovery of the theft. User-to-token authentication is usually based on passwords in the form of a PIN (personal identification number).

Token-based authentication is much stronger than password-based authentication, and is often called strong as opposed to weak authentication. However, it is the token that is authenticated rather than the user. The token can be shared with other users by providing the PIN, so it is vulnerable to loss of accountability. Of course, only one user at a time can physically possess the token.

Tokens can used secret key or public key cryptosystems. With secret key systems the computer authenticating the token needs to know the secret key that is embedded in the token. This presents the usual key distribution problem for secret-key cryptography. With public-key cryptography a token can be authenticated by a computer which has had no prior contact with the user's token. The public key used to verify the response to a challenge can be obtained used public-key certificates. Public-key based tokens have scalability advantages that in the long run should make them the dominant technique for authentication in large systems. However, the computational and bandwidth requirements are generally greater for public versus secret key systems. Token-based authentication is a technical reality today, but it still lacks major market penetration and does cost money.

2.3 Biometric Authentication

Biometric authentication has been used for some time for high-end applications. The biometric signature should be different every time, for example, voice-print check of a different challenge phrase on each occasion. Alternately, the biometric signature should require an active input, for example, dynamics of handwritten signatures. Simply repeating the same phrase every time or using a fixed signature such as a fingerprint is vulnerable to replay attacks. Biometric authentication often requires cumbersome equipment which is best suited for fixed installations such as entry into a building or room.

Technically the best combination would be user-to-token biometric authentication, followed by mutual cryptographic authentication between the token and system services. This combination may emerge sooner than one might imagine. Deployment of such technology in a large scale is certain to raise social and political debate. Unforgeable biometric authentication could result in significant loss of privacy for individuals. Some of the privacy issues may have technical solutions while others may be inherently impossible.

2.4 Authentication in Distributed Systems

In distributed systems authentication is required repeatedly as the user uses multiple services. Each service needs authentication, and we might want mutual authentication in each case. In practice this process starts with a user supplying a password to the workstation which can then act on the user's behalf. This password should never be disclosed in plaintext on the network. Typically the password is converted to a cryptographic key which is then used to perform challenge-response authentication with servers in the system. To minimize exposure of the user password, and the long-term key derived from it, the password is converted into a short-term key which is retained on the workstation while the long-term user secrets are discarded. In effect these systems use the desktop workstation as a "token" for authentication with the rest of the network. Trojan Horse software in the workstation can, of course, compromise the user's long-term secrets.

The basic principles outlined above have been implemented in actual systems in an amazing variety of ways. Many of the early implementations are susceptible to dictionary attacks. Now that the general nature and ease of a dictionary attack are understood we are seeing systems which avoid these attacks or at least attempt to make them more difficult. For details on actual systems we refer the reader to [KPS95, Neu94, WL92].

3 Access Control

In this section we describe access control. We introduce the concept of an access matrix and discuss implementation alternatives. Then we explain discretionary, mandatory and role-based access control policies. Finally, we discuss issues in administration of authorizations.

3.1 The Access Control Matrix

Security practitioners have developed a number of abstractions over the years in dealing with access control. Perhaps the most fundamental of these is the realization that all resources controlled by a computer system can be represented by data stored in objects (e.g., files). Therefore protection of objects is the crucial requirement, which in turn facilitates protection of other resources controlled via the computer system. (Of course, these resources must also be physically protected so they cannot be manipulated directly bypassing the access controls of the computer system.)

Activity in the system is initiated by entities known as subjects. Subjects are typically users or programs executing on behalf of users. A user may sign on to the system as different subjects on different occasions, depending on the privileges the users wishes to exercise in a given session. For example, a user working on two different projects may sign on for purpose of working on one project or the other. We then have two subjects corresponding to this user, depending on the project the user is currently working on.

A subtle point that is often overlooked is that subjects can themselves be objects. A subject can create additional subjects in order to accomplish its task. The children subjects may be executing on various computers in a network. The parent subject will usually be able to suspend or terminate its children as appropriate. The fact that subjects can be objects corresponds to the observation that the initiator of one operation can be the target of another. (In network parlance subjects are often called initiators, and objects called targets.)

The subject-object distinction is basic to access control. Subjects initiate actions or operations on objects. These actions are permitted or denied in accord with the authorizations established in the system. Authorization is expressed in terms of access rights or access modes. The meaning of access rights depends upon the object in question. For files the typical access rights are Read, Write, Execute and Own. The meaning of the first three of these is self evident. Ownership is concerned with controlling who can change the access permissions for the file. An object such as a bank account may have access rights Inquiry, Credit and Debit corresponding to the basic operations that can be performed on an account. These operations would be implemented by application programs, whereas for a file the operations would typically be provided by the Operating System.

The access matrix is a conceptual model which specifies the rights that each subject possesses for each object. There is a row in this matrix for each subject, and a column for each object. Each cell of the matrix specifies the access authorized for the subject in the row to the object in the column. The task of access control is to ensure that only those operations authorized by the access matrix actually get executed. This is achieved by means of a reference monitor, which is responsible for mediating all attempted operations by subjects on objects. Note that the access matrix model clearly separates the problem of authentication from that of authorization.

An example of an access matrix is shown in figure 2, where the rights R and W denote read and write respectively, and the other rights are as discussed above. The subjects shown here are John, Alice and Bob. There are four files and two accounts. This matrix specifies that, for example, John is the owner of File 3 and can read and write that file, but John has no access to File 2 or File 4. The precise meaning of ownership varies from one system to another. Usually the owner of a file is authorized to grant other users access to the file, as well as revoke access. Since John owns File 1, he can give Alice the R right and Bob the R and W rights as shown in figure 2. John can later revoke one or more of these rights at his discretion.

The access rights for the accounts illustrate how access can be controlled in terms of abstract operations implemented by application programs. The Inquiry operation is similar to read in that it retrieves information but does not change it. Both the Credit and Debit operations will involve reading the previous account balance, adjusting it as appropriate and writing it back. The programs which implement these operations require read and write access to the account data. Users, however, are not allowed to read and write the account object directly. They can manipulate account objects only indirectly via application programs which implement the Debit and Credit operations.

Also note that there is no Own right for accounts. Objects such as bank accounts do not really have an owner who can determine the access of other subjects to the account. Clearly the user who establishes the account at the bank should not be the one to decide who can access the account. Within the bank different officials can access the account on basis of their job functions in the organization.

3.2 Implementation Approaches

In a large system the access matrix will be enormous in size, and most of its cells are likely to be empty. Accordingly the access matrix is very rarely implemented as a matrix. We now discuss some common approaches to implementing the access matrix in practical systems.

3.2.1 Access Control Lists

A popular approach to implementing the access matrix is by means of Access Control Lists (ACLs). Each object is associated with an ACL, indicating for each subject in the system the accesses the subject is authorized to execute on the object. This approach corresponds to storing the matrix by columns. ACLs corresponding to the access matrix of figure 2 are shown in figure 3. Essentially the access matrix column for File 1 is stored in association with File 1, and so on.

By looking at an object's ACL it is easy to determine which modes of access subjects are currently authorized for that object. In other words ACLs provide for convenient access review with respect to an object. It is also easy to revoke all access to an object by replacing the existing ACL with an empty one. On the other hand determining all the accesses that a subject has is difficult in an ACL-based system. It is necessary to examine the ACL of every object in the system to do access review with respect to a subject. Similarly if all accesses of a subject need to be revoked all ACLs must be visited one by one. (In practice revocation of all accesses of a subject is often done by deleting the user account corresponding to that subject. This is acceptable if a user is leaving an organization. However, if a user is reassigned within the organization it would be more convenient to retain the account and change its privileges to reflect the changed assignment of the user.)

Many systems allow group names to occur in ACLs. For example, an entry such as (ISSE, R) can authorize all members of the ISSE group to read a file. Several popular Operating Systems, such as Unix and VMS, implement an abbreviated form of ACLs in which a small number, often only one or two, group names can occur in the ACL. Individual subject names are not allowed. With this approach the ACL has a small fixed size so it can be stored using a few bits associated with the file. At the other extreme there are a number of access control packages which allow complicated rules in ACLs to limit when and how the access can be invoked. These rules can be applied to individual users or to all users who match a pattern defined in terms of user names or other user attributes.

3.2.2 Capabilities

Capabilities are a dual approach to ACLs. Each subject is associated with a list, called the capability list, indicating for each object in the system, the accesses the subject is authorized to execute on the object. This approach corresponds to storing the access matrix by rows. Figure 4 shows capability lists for the files in figure 2. In a capability list approach it is easy to review all accesses that a subject is authorized to perform, by simply examining the subject's capability list. However, determination of all subjects who can access a particular object requires examination of each and every subject's capability list. A number of capability-based computer systems were developed in the 1970s, but did not prove to be commercially successful. Modern operating systems typically take the ACL-based approach.

It is possible to combine ACLs and capabilities. Possession of a capability is sufficient for a subject to obtain access authorized by that capability. In a distributed system this approach has the advantage that repeated authentication of the subject is not required. This allows a subject to be authenticated once, obtain its capabilities and then present these capabilities to obtain services from various servers in the system. Each server may further use ACLs to provide finer-grained access control.

3.2.3 Authorization Relations

We have seen that ACL- and capability-based approaches have dual advantages and disadvantages with respect to access review. There are representations of the access matrix which do not favor one aspect of access review over the other. For example, the access matrix can be represented by an authorization relation (or table) as shown in figure 5. Each row, or tuple, of this table specifies one access right of a subject to an object. Thus, John's accesses to File 1 require three rows. If this table is sorted by subject, we get the effect of capability lists. If it is sorted by object we get the effect of ACLs. Relational database management systems typically use such a representation.

3.3 Access Control Policies

In access control systems a distinction is generally made between policies and mechanisms. Policies are high level guidelines which determine how accesses are controlled and access decisions determined. Mechanisms are low level software and hardware functions which can be configured to implement a policy. Security researchers have sought to develop access control mechanisms which are largely independent of the policy for which they could be used. This is a desirable goal to allow reuse of mechanisms to serve a variety of security purposes. Often, the same mechanisms can be used in support of secrecy, integrity or availability objectives. On the other hand, sometimes the policy alternatives are so many and diverse that system implementors feel compelled choose one in preference to the others.

In general, there do not exist policies which are "better" than others. Rather there exist policies which ensure more protection than others. However, not all systems have the same protection requirements. Policies suitable for a given system may not be suitable for another. For instance, very strict access control policies, which are crucial to some systems may be inappropriate for environments where users require greater flexibility. The choice of access control policy depends on the particular characteristics of the environment to be protected.

We will now discuss three different policies which commonly occur in computer systems as follows:

- classical discretionary policies,
- classical mandatory policies, and
- the emerging role-based policies.

We have added the qualifier "classical" to the first two of these to reflect the fact that these have been recognized by security researchers and practitioners for a long time. However, in recent years there is increasing consensus that there are legitimate policies which have aspects of both of these. Role-based policies are an example of this fact.

It should be noted that access control policies are not necessarily exclusive. Different policies can be combined to provide a more suitable protection system. This is indicated in figure 6. Each of the three inner circles represents a policy which allows a subset of all possible accesses. When the policies are combined only the intersection of their accesses is allowed. Such combination of policies is relatively straightforward so long as there are no conflicts where one policy asserts that a particular access *must* be allowed while another one prohibits it. Such conflicts between policies need to be reconciled by negotiations at an appropriate level of management.

3.3.1 Classical Discretionary Policies

Discretionary protection policies govern the access of users to the information on the basis of the user's identity and authorizations (or rules) that specify, for each user (or group of users) and each object in the system, the access modes (e.g., read, write, or execute) the user is allowed on the object. Each request of a user to access an object is checked against the specified authorizations. If there exists an authorization stating that the user can access the object in the specific mode, the access is granted, otherwise it is denied.

The flexibility of discretionary policies makes them suitable for a variety of systems and applications. For these reasons, they have been widely used in a variety of implementations, especially in the commercial and industrial environments.

However, discretionary access control policies have the drawback that they do not provide real assurance on the flow of information in a system. It is easy to bypass the access restrictions stated through the authorizations. For example, a user who is able to read data can pass it to other users not authorized to read it without the cognizance of the owner. The reason is that discretionary policies do not impose any restriction on the usage of information by a user once the user has got it, i.e., dissemination of information is not controlled. By contrast dissemination of information is controlled in mandatory systems by preventing flow of information from high-level objects to low-level objects.

Discretionary access control policies based on explicitly specified authorization are said to be closed, in that the default decision of the reference monitor is denial. Similar policies, called open policies, could also be applied by specifying denials instead of permissions. In this case, for each user and each object of the system, the access modes the user is forbidden on the object are specified. Each access request by a user is checked against the specified (negative) authorizations and granted only if no authorizations denying the access exist. The use of positive and negative authorizations can be combined, allowing the specification of both the accesses to be authorized as well as the accesses to be denied to the users. The interaction of positive and negative authorizations can become extremely complicated [BSJ93].

3.3.2 Classical Mandatory Policies

Mandatory policies govern access on the basis of classification of subjects and objects in the system. Each user and each object in the system is assigned a security level. The security level associated with an object reflects the sensitivity of the information contained in the object, i.e, the potential damage which could result from unauthorized disclosure of the information. The security level associated with a user, also called clearance, reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. In the simplest case, the security level is an element of a hierarchical ordered set. In the military and civilian government arenas, the hierarchical set generally consists of Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), where TS > S > C > U. Each security level is said to dominate itself and all others below it in this hierarchy.

Access to an object by a subject is granted only if some relationship (depending on the type of access) is satisfied between the security levels associated with the two. In particular, the following two principles are required to hold.

Read down A subject's clearance must dominate the security level of the object being read.

Write up A subject's clearance must be dominated by the security level of the object being written.

Satisfaction of these principles prevents information in high level objects (i.e., more sensitive) to flow to objects at lower levels. The effect of these rules is illustrated in figure 7. In such a system information can only flow upwards or within the same security class.

It is important to understand the relationship between users and subjects in this context. Let us say that the human user Jane is cleared to S, and assume she always signs on to the system as an S subject (i.e., a subject with clearance S). Jane's subjects are prevented from reading TS objects by the read-down rule. The write-up rule, however, has two aspects that seem at first sight contrary to expectation.

- Firstly, Jane's S subjects can write a TS object (even though they cannot read it). In particular, they can overwrite existing TS data and therefore destroy it. Due to this integrity concern, many systems for mandatory access control do not allow write up; but limit writing to the same level as the subject. At the same time write up does allow Jane's S subjects to send electronic mail to TS subjects, and can have its benefits.

- Secondly, Jane's S subjects cannot write C or U data. This means, for example, that Jane can never send electronic mail to C or U users. This is contrary to what happens in the paper world, where S users can write memos to C and U users. This seeming contradiction is easily eliminated by allowing Jane to sign to the system as a C, or U, subject as appropriate. During these sessions she can send electronic mail to C or, U and C, subjects respectively.

In other words a user can sign on to the system as a subject at any level dominated by the user's clearance. Why then bother to impose the write-up rule? The main reason is to prevent malicious software from leaking secrets downward from S to U. Users are trusted not to leak such information, but the programs they execute do not merit the same degree of trust. For example, when Jane signs onto the system at U level her subjects cannot read S objects, and thereby cannot leak data from S to U. The write-up rule also prevents users from inadvertently leaking information from high to low.

In additional to hierarchical security levels, categories (e.g., Crypto, NATO, Nuclear) can also be associated with objects and subjects. In this case the classification labels associated with each subject and each object consists of a pair composed of a security level and a set of categories. The set of categories associated with a user reflect the specific areas in which the user operates. The set of categories associated with an object reflect the area to which information contained in objects are referred. The consideration of categories provides a finer grained security classification. In military parlance categories enforce restriction on the basis of the need-to-know principle, i.e., a subject should be only given those accesses which are required to carry out the subject's responsibilities.

Mandatory access control can as well be applied for the protection of information integrity. For example, the integrity levels could be Crucial (C), Important (I), and Unknown (U). The integrity level associated with an object reflects the degree of trust that can be placed in the information stored in the object, and the potential damage that could result from unauthorized modification of the information. The integrity level associated with a user reflects the user's trustworthiness for inserting, modifying or deleting data and programs at that level. Principles similar to those stated for secrecy are required to hold, as follows.

Read up A subject's integrity level must be dominated by the integrity level of the object being read.

Write down A subject's integrity level must dominate the integrity level of the object being written.

Satisfaction of these principles safeguard integrity by preventing information stored in low objects (and therefore less reliable) to flow to high objects. This is illustrated in figure 8. Controlling information flow in this manner is but one aspect of achieving integrity. Integrity in general requires additional mechanisms, as discussed in [CFMS94, San94].

Note that the only difference between figures 7 and 8 is the direction of information flow, being bottom to top in the former case and top to bottom in the latter. In other words both cases are concerned with one-directional information flow. The essence of classical mandatory controls is one-directional information flow in a lattice of security labels. For further discussion on this topic see [San93].

3.3.3 Role-Based Policies

The discretionary and mandatory policies discussed above have been recognized in official standards, notably the so-called Orange Book of the U.S. Department of Defense. A good introduction to the Orange Book and its evaluation procedures is given in [Cho92].

There has been a strong feeling among security researchers and practitioners that many practical requirements are not covered by these classical discretionary and mandatory policies. Mandatory policies rise from rigid environments, like those of the military. Discretionary policies rise from cooperative yet autonomous requirements, like those of academic researchers. Neither requirement satisfies the needs of most commercial enterprises. Orange Book discretionary policy is too weak for effective control of information assets, whereas Orange Book mandatory policy is focused on the US Government policy for confidentiality of classified information. (In practice the military often finds Orange Book mandatory policies to be too rigid and subverts them.)

Several alternatives to classical discretionary and mandatory policies have been proposed. These policies allow the specification of authorizations to be granted to users (or groups) on objects like in the discretionary approach, together with the possibility of specifying restrictions (like in the mandatory approach) on the assignment or on the use of such authorizations. One of the promising avenues which is receiving growing attention is that of role-based access control [FK92, SCFY96].

Role-based policies regulate the access of users to the information on the basis of the activities the users execute in the system. Role based policies require the identification of roles in the system. A role can be defined as a set of actions and responsibilities associated with a particular working activity. Then, instead of specifying all the accesses each users is allowed to execute, access authorizations on objects are specified for roles. Users are given authorizations to adopt roles. A recent study by NIST confirms that roles are a useful approach for many commercial and government organizations [FGL93].

The user playing a role is allowed to execute all accesses for which the role is authorized. In general a user can take on different roles on different occasions. Also the same role can be played by several users, perhaps simultaneously. Some proposals for role-based access control allow a user to exercise multiple roles at the same time. Other proposals limit the user to only one role at a time, or recognize that some roles can be jointly exercised while others must be adopted in exclusion to one another. As yet there are no standards in this arena, so it is likely that different approaches will be pursued in different systems.

The role-based approach has several advantages. Some of these are discussed below.

- **Authorization management:** Role-based policies benefit from a logical independence in specifying user authorizations by breaking this task into two parts, one which assigns users to roles and one which assigns access rights for objects to roles. This greatly simplifies security management. For instance, suppose a user responsibilities change, say, due to a promotion. The user's current roles can be taken away and new roles assigned as appropriate for the new responsibilities. If all authorization is directly between users and objects, it becomes necessary to revoke all existing access rights of the user and assign new ones. This is a cumbersome and time-consuming task.

- **Hierarchical roles:** In many applications there is a natural hierarchy of roles, based on the familiar principles of generalization and specialization. An examples is shown in figure 9. Here the roles of hardware and software engineer are specializations of the engineer role. A user assigned to the role of software engineer (or hardware engineer) will also inherit privileges and permissions assigned to the more general role of engineer. The role of supervising engineer similarly inherits privileges and permissions from both software-engineer and hardware-engineer roles. Hierarchical roles further simplify authorization management.

- **Least privilege:** Roles allow a user to sign on with the least privilege required for the particular task at hand. Users authorized to powerful roles do not need to exercise them until those privileges are actually needed. This minimizes the danger of damage due to inadvertent errors or by intruders masquerading as legitimate users.

- **Separation of duties:** Separation of duties refer to the principle that no user should be given enough privileges to misuse the system on their own. For example, the person authorizing a paycheck should not also be the one who can prepare them. Separation of duties can be enforced either statically (by defining conflicting roles, i.e., roles which cannot be executed by the same user) or dynamically (by enforcing the control at access time). An example of dynamic separation of duty is the two-person rule. The first user to execute a two-person operation can be any authorized user, whereas the second user can be any authorized user different from the first.

- **Object classes:** Role-based policies provides a classification of users according to the activities they execute. Analogously, a classification should be provided for objects. For example, generally a clerk will need to have access to the bank accounts, and a secretary will have access to the letters and memos (or some subset of them). Objects could be classified according to their type (e.g., letters, manuals) or to their application area (e.g., commercial letters, advertising letters). Access authorizations of roles should then be on the basis of object classes, not specific objects. For example, a secretary role can be given the authorization to read and write the entire class of letters, instead of giving it explicit authorization for each single letter. This approach has the advantage of making authorization administration much easier and better controlled. Moreover, the accesses authorized on each object are automatically determined according to the type of the object without need of specifying authorizations upon each object creation.

3.4 Administration of Authorization

Administrative policies determine who is authorized to modify the allowed accesses. This is one of the most important, and least understood, aspects of access controls.

In mandatory access control the allowed accesses are determined entirely on basis of the security classification of subjects and objects. Security levels are assigned to users by the security administrator. Security levels of objects are determined by the system on the basis of the levels of the users creating them. The security administrator is typically the only one who can change security levels of subjects or objects. The administrative policy is therefore very simple.

Discretionary access control permits a wide range of administrative policies. Some of these are described below.

- **Centralized:** A single authorizer (or group) is allowed to grant and revoke authorizations to the users.

- **Hierarchical:** A central authorizer is responsible for assigning administrative responsibilities to other administrators. The administrators can then grant and revoke access authorizations to the users of the system. Hierarchical administration can be applied, for example, according to the organization chart.

- **Cooperative:** Special authorizations on given resources cannot be granted by a single authorizer but needs cooperation of several authorizers.

- **Ownership:** A user is considered the owner of the objects he/she creates. The owner can grant and revoke access rights for other users to that object.

- **Decentralized:** In decentralized administration the owner of an object can also grant other users the privilege of administering authorizations on the object.

Within each of these there are many possible variations.

Role-based access control has a similar wide range of possible administrative policies. In this case roles can also be used to manage and control the administrative mechanisms.

Delegation of administrative authority is an important area in which existing access control systems are deficient. In large distributed systems centralized administration of access rights is infeasible. Some existing systems allow administrative authority for a specified subset of the objects to be delegated by the central security administrator to other security administrators. For example, authority to administer objects in a particular region can be granted to the regional security administrator. This allows delegation of administrative authority in a selective piecemeal manner.

However, there is a dimension of selectivity that is largely ignored in existing systems. For instance, it may be desirable that the regional security administrator be limited to granting access to these objects only to employees who work in that region. Control over the regional administrators can be centrally administered, but they can have considerable autonomy within their regions. This process of delegation can be repeated within each region to set up sub-regions and so on.

4 Auditing and Intrusion Detection

Auditing consists of examination of the history of events in a system in order to determine whether and how security violations have occurred or been attempted. Auditing requires registration or logging of users requests and activities for later examination. Audit data is recorded in an *audit trail*, or *audit log*. The nature and format of this data varies from system to system.

Information which should be recorded for each event includes the subject requesting the access, the object to be accessed, the operation requested, the time of the request, perhaps the location from which the requested originated, the response of the access control system, the amount of resources (CPU time, I/O, memory, etc.) used, whether the operation succeeded or, if not, the reason for the failure, and so on.

In particular actions requested by privileged users, such as the system and the security administrators, should be logged. This serves as a deterrent against misuse of of powerful privileges by the administrators, and also as a means for detecting operating must be controlled (the old problem of "guarding the guardian"). The second reason is that it allows to control penetrations in which the attacker gains a privileged status.

Audit data can become voluminous very quickly and searching for security violations in such a mass of data is a difficult task. Of course, audit data cannot reveal all violations because some may not be apparent in even a very careful analysis of audit records. Sophisticated penetrators can spread out their activities over a relatively long period of time thus making detection more difficult. In some cases, audit analysis is executed only if violations are suspected or their effects are visible since the system shows an anomalous or erroneous behavior, such as continuous insufficient memory, slow processing, or non accessibility of certain files. Even in this case, often, only a limited amount of audit data, namely those which may be connected with the suspected violation, are examined. Sometimes the first clue to a security violation is some real world event which indicates that information has been compromised. The may happen long after the computer penetration occurred. Similarly, security violations may result in Trojan Horses or viruses being implanted whose activity may not be trigerred long after the original event.

4.1 Intrusion Detection Systems

Recent research has focussed on the development of automated tools to help or even to carry out auditing controls. Automated tools can be used to screen and reduce audit data that needs to be reviewed by humans. These tools can also organize audit data to produce summaries and measures needed in the analysis. This data reduction process can, for instance, produce short summaries of user behaviors, anomalous events, or security incidents. The auditors can then to go over summaries instead than examining each single event recorded. Another class of automated tools is represented by the so-called *intrusion detection systems*. The purpose of these tools is to not only to automate audit data acquisition and reduction but also its analysis. Some of the more ambitious efforts attempt to perform intrusion detection in real time.

Intrusion detection systems can be classified as *passive* or *active*. Passive systems, generally operating off-line, analyze the audit data and bring possible intrusions or violations to the attention of the auditor who then takes appropriate actions (see Figure 10). Active systems analyze audit data in real time. Besides bringing violations to the attention of the auditor, these systems may take immediate protective response on the system (see Figure 11). The protective response can be executed post-facto, after the violation has occurred, or preemptively, to avoid the violation being perpetrated to complete. This latter possibility depends on the ability of the system to foresee violations. Protective responses include killing the suspected process, disconnecting the user, disabling privileges, or disabling user accounts. The response may be determined in total autonomy by the intrusion detection system or through interactions with the auditors.

Different approaches have been proposed for building intrusion detection systems. No single approach can be considered satisfactory with respect to different kinds of penetrations and violations that can occur. Each approach is appropriate for detecting a specific subset of violations. Moreover, each approach presents some pros and cons determined by the violations that can or cannot be controlled and by the amount and complexity of information necessary for its application. We now discuss the main intrusion detection approaches that have been attempted.

4.1.1 Threshold-Based Approach

The threshold-based approach is based on the assumption that the exploitation of system vulnerabilities involves abnormal use of the system itself. For instance, an attempt to break into a system can require trying several user accounts and passwords. An attempt to discover protected information can imply several, often denied, browsing operations through protected directories. A process infected by a virus can require an abnormal amount of memory or CPU resources.

Threshold-based systems typically control occurrences of specific events over a given period of time with respect to predefined allowable thresholds established by the security officer. For instance, more than three unsuccessful attempts to login to a given account with the wrong password may indicate an attempt to penetrate that account. Multiple unsuccessful attempts to log in the system, using different accounts, concentrated in a short period of time, may suggest an attempt to break in.

Thresholds can also be established with respect to authorized operations to detect improper use of resources. For instance, a threshold can specify that print requests totaling more than a certain number of pages a day coming from the administrative office is to be considered suspicious. This misuse can be symptomatic of different kinds of violations such as the relatively benign misuse of the resource for personal use or a more serious attempt to print out working data for disclosure to the competition.

The threshold-based approach is limited by the fact that many violations occur without implying overuse of system resources. A further drawback of this approach is that it requires prior knowledge of how violations are reflected in terms of abnormal system use. Determining such connections and establishing appropriate thresholds is not always possible.

4.1.2 Anomaly-Based Approach

Like threshold-based controls, anomaly-based controls are based on the assumption that violations involve abnormal use of the system. However, while threshold-based systems define abnormal use with respect to pre-specified fixed acceptable thresholds, anomaly-based systems define abnormal use as a use that is significantly different from that normally observed. In this approach, the intrusion detection system observes the behavior of the users in the target system and define profiles, i.e., statistical measures, reflecting the normal behavior of the users. Profiles can be defined with respect to different aspects to be controlled such as the number of events in a user session, the time elapsed between events in a user session, the amount of resources consumed over a certain period of time or during execution of certain programs.

Construction of profiles from raw audit data is guided by rules that can be specified with respect to single users, objects, or actions, as well as to classes of these. For instance, rules can state that profiles should be defined with respect to the number of pages printed every day by each user in the administration office, the number of resources per session and per day consumed by each user, the time elapsed between two login sessions for each single user, some "habit" measures such as the time and the location from which a user generally logs in and the time the connections last. As users operate in the system, the intrusion detection system learns their behaviors with respect to the different profiles thus defining what is "normal," and adapting the profiles to changes. Whenever a significant deviation occurs for a profile, an alarm is raised.

Statistical models that can be used include the *operational model*, the *mean and standard deviation model*, and *time series model*. With the operational model an anomaly is raised when an observation exceeds a given acceptable threshold. This is similar to the threshold-based approach. With the mean and standard deviation model an anomaly occurs when the observation falls outside an allowed confidence interval around the mean. For instance, an alarm can be raised if the CPU time consumed during a session for a user falls much below or above the CPU time generally consumed by the same user. With the time series model an anomaly is raised when an event occurs at a given time at which the probability of its occurring is too low. For instance, a remote night-hour login request by a user who has never connected off-hours or from outside the building may be considered suspicious.

The main advantage of the anomaly detection approach is that it does not require any a priori knowledge of the target system or of possible flaws from which the system may suffer. However, like the threshold-based approach, it can only detect violations that involve anomalous use. Moreover, some legitimate users may have a very erratic behavior (e.g., logging on and off at different hours or from different locations varying their activity daily). For such users no normal behavior can be actually established and misuse by them as well as by masqueraders exploiting their accounts would go undetected. The approach is also vulnerable from insiders who, knowing that behavior profiles are being defined, may either behave in a "bad" way from the beginning or slowly vary their behavior, going from "good" to "bad", thus convincing the system that the "bad" behavior is normal.

4.1.3 Rule-Based Approach

In the rule-based approach, rules are used to analyze audit data for suspicious behavior, independently from users' behavioral patterns. Rules describe what is suspicious on the basis of known past intrusions or known system vulnerabilities. This approach is generally enforced by means of expert systems encoding knowledge of the security experts about past intrusions in terms of sequences of events or observable properties characterizing violations.

For instance, a rule can specify that a sequence of browsing operations (e.g., `cd`, `ls`, and `more` commands in a Unix environment) coming off-hours from a remote location may be symptomatic of an intrusion. Rules can also identify suspicious sequences of actions. For example the withdrawal of a large amount of money from an account and its deposit back few days later may be considered suspicious.

The rule-based approach can detect violations that do not necessarily imply abnormal use of resources. Its main limitation is that the expert knowledge encoded in the rules encompasses only known system vulnerabilities and attack scenarios or suspicious events. The system can therefore be penetrated by attackers employing new techniques.

4.1.4 Model-Based Reasoning Approach

The model-based reasoning approach is based on the definition, by the security officers, of models of proscribed intrusion activities [GL91]. Proscribed activities are expressed by means of sequences of user behaviors (single events or observable measures), called scenarios.

Each component of a scenario is therefore a high-level observation on the system and does not necessarily correspond to an audit record (which contains information at a lower level of specification). From these high-level specifications, the intrusion detection system generates, on the basis of specified rules, the corresponding sequences of actions at the level of the audit records. Each audit record produced on the observation of the system is controlled against the specified scenarios to determine if a violation is being carried out. Audit data reduction and analysis can be modeled in such a way that only events relevant to specific scenarios corresponding to intrusions probably being carried out are examined. When the probability of a given scenario being followed passes a specified threshold an alarm is raised informing the auditor of the suspected violation.

The basis of this approach is essentially the same as the rule-based approach, the main difference being the way in which controls are specified. While in the rule-based approach the security officer must explicitly specify the control rules in terms of the audit data, in the model-based approach the security officer only specifies the scenario in terms of high-level observable properties. This constitutes the main advantage of this approach which allows the security officer to reason in terms of high-level abstractions rather than audit records. It is the task of the system to translate the scenarios into corresponding rules governing data reduction and analysis.

Like the rule-based approach, this approach can control only violations whose perpetration scenario (i.e., actions necessary to fulfill them) are known. By contrast, violations exploiting unknown vulnerabilities or not yet tried cannot be detected.

4.1.5 State Transition-Based Approach

In the state transition-based approach a violation is modeled as a sequence of actions starting from an initial state to a final compromised state [IKP95]. A state is a snapshot of the target system representing the values of all volatile, semi-permanent, and permanent memory locations. Between the initial and the final states there are a number of intermediate states and corresponding transitions. State transitions correspond to key actions necessary to carry out the violation. Actions do not necessarily correspond to commands issued by users but instead refer to how state changes within the system are achieved. A single command may produce multiple actions. Each state is characterized as a set of assertions evaluating whether certain conditions are verified in the system. For instance, assertions can check whether a user is the owner of an object or has some privileges on it, whether the user who caused the last two transitions is the same user, or whether the file to which an action is referred is a particular file. Actions corresponding to state transitions are accesses to files (e.g., read, write, or execute operations) or actions modifying the permissions associated with files (e.g., changes of owners or authorizations) or some of the files' characteristics (e.g., rename operations).

As the users operate in the system, state transitions caused by them are determined. Whenever a state transition causes a final compromised state to be reached, an alarm is raised. The state transition-based approach can also be applied in a real-time active system to prevent users from executing operations that would cause a transition to a compromised state.

The state transition-based approach is based on the same concepts as the rule-based approach and therefore suffers from the same limitations, i.e., only violations whose scenarios are known can be detected. Moreover, it can be used to control only those violations that produce visible changes to the system state. Like the model-based approach, the state transition-based approach provides the advantage of requiring only high level specifications, leaving the system the task of mapping state transitions into audit records and producing the corresponding control rules. Moreover, since a state transition can be matched by different operations at the audit record level, a single state transition specification can be used to represent different variations of a violation scenario (i.e., involving different operations but causing the same effects on the system).

4.1.6 Other Approaches

Other approaches have been proposed to complement authentication and access control to prevent violations to happen or to detect their occurrence.

One approach consists in preventing, rather than detecting, intrusions. In this class are tester programs that evaluate the system for common weaknesses often exploited by intruders and password checker programs that prevent users from choosing weak or obvious passwords (which may represent an easy target for intruders).

Another approach consists in substituting known bugged commands, generally used as trap doors by intruders, with programs that simulate the commands' execution while at the same time sending an alarm to the attention of the auditor. Other trap programs for intruders are represented by fake user accounts with "magic" passwords that raise an alarm when they are used.

Other approaches aim at detecting or preventing execution of Trojan Horses and viruses. Solutions adopted for this include integrity checking tools that search for unauthorized changes to files and mechanisms controlling program executions against specifications of allowable program behavior in terms of operations and data flows.

Yet another intrusion detection approach is represented by the so-called keystroke latencies control. The idea behind the approach is that the elapsed time between keystrokes for regularly typed strings is quite consistent for each user. Keystroke latencies control can be used to cope against masqueraders. Moreover, they can also be used for authentication by controlling the time elapsed between the keystrokes when typing the password.

More recent research has interested intrusion detection at the network level [MHL94]. Analysis is performed on network traffic instead than on commands (or their corresponding low level operations) issued on a system. Anomalies can then be determined, for example, on the basis of the probability of the occurrence of the monitored connections being too low or on the basis of the behavior of the connections. In particular, traffic is controlled against profiles of expected traffic specified in terms of expected paths (i.e., connections between systems) and service profiles.

4.2 Audit Control Issues

There are several issues that must be considered when employing intrusion detection techniques to identify security violations. These issues arise independent of the specific intrusion detection approach being utilized.

The task of generating audit records can be left to either the target system being monitored or to the intrusion detection system. In the former case, the audit information generated by the system may need to be converted to a form understandable by the intrusion detection system. Many operating systems and database systems provide some audit information. However, this information is often not appropriate for security controls since it may contain data not relevant for detecting intrusions and omits details needed for identifying violations. Moreover, the audit mechanism of the target system may itself be vulnerable to a penetrator who might be able to bypass auditing or modify the audit log. Thus, a stronger and more appropriate audit trail might be required for effective intrusion detection.

Another important issue that must be addressed is the retention of audit data. Since the quantity of audit data generated every day can be enormous, policies must be specified that determine when historical data can be discarded.

Audit events can be recorded at different granularity. Events can be recorded at the system command level, at the level of each system call, at the application level, at the network level, or at the level of each keystroke. Auditing at the application and command level has the advantage of producing high level traces which can be more easily correlated, especially by humans (who would get lost in low level details). However, the actual effect of the execution of a command or application on the system may be not reflected in the audit records, and therefore cannot be analyzed. Moreover, auditing at such a high level can be circumvented by users exploiting alias mechanisms or by directly issuing lower level commands. Recording at lower levels overcomes this drawback at the price of maintaining a greater number of audit records (a single user command may correspond to several low level operations) whose examination by humans (or automated tools) becomes therefore more complicated.

Different approaches can be taken with respect to the time at which the audit data is recorded and, in case of real-time analysis, evaluated. For instance, the information that a user has required execution of a process can be passed to the intrusion detection system at the time the execution is required or at the time it is completed. The former approach has the advantage of allowing timely detection and, therefore, a prompt response to stop the violation. The latter approach has the advantage of providing more complete information about the event being monitored (information on resources used or time elapsed can be provided only after the process has completed) and therefore allows more complete analysis.

Audit data recording or analysis can be carried out indiscriminately or selectively, namely on specific events, such as events concerning specific subjects, objects, or operations, or occurring at particular time or situation. For instance, audit analysis can be performed only on operations on objects containing sensitive information, on actions executed off-hours (nights and weekends) or from remote locations, on actions denied by the access control mechanisms, or on actions required by mistrusted users.

Different approaches can be taken with respect to the time at which audit control should be performed. Real-time intrusion detection system enforce control in real time, i.e., analyze each event at the time of its occurrence. Real-time analysis of data brings the great advantage of timely detection of violations. However, due to the great amount of data to analyze and analysis to perform, real-time controls are generally performed only on specific data, leaving a more thorough analysis to be performed off-line. Approaches that can be taken include the following.

- *Period-driven.* Audit control is executed periodically. For example, every night the audit data produced during the working day are examined.

- *Session-driven.* Audit control on a user's session is performed when a close session command is issued.
- *Event-driven.* Audit control is executed upon occurrence of certain events. For instance, if a user attempts to enter a protected directory, audit over the user's previous and/or subsequent actions are executed.
- *Request-driven.* Audit control is executed upon explicit request of the security officer.

The intrusion detection system may reside either on the target computer system or on a separate machine. This latter solution is generally preferable since it does not impact the target systems performance and protects audit information and control from attacks perpetrated on the target system. On the other hand, audit data must be communicated to the intrusion detection machine which itself could be a source of vulnerability.

A major issue in employing an intrusion detection system is privacy. Monitoring user behavior, even if intended for defensive purposes, introduces a sort of "Big Brother" situation where a centralized monitor is watching everybody's behavior. This may be considered an invasion of individual privacy. It also raises concerns that audited information may be used improperly, for example, as a means for controlling employee performances.

5 Conclusion

Authentication, access control and audit and intrusion detection together provide the foundations for building systems which can store and process information with confidentiality and integrity. Authentication is the primary security service. Access control builds directly upon it. By and large access control assumes authentication has been successfully accomplished. Strong authentication supports good auditing because operations can then be traced to the user who caused them to occur. There is a mutual interdependence between these three technologies which can be often ignored by security practitioners and researchers. We need a coordinated approach which combines the strong points of each of these technologies, rather than treating these as separate independent disciplines.

Acknowledgment

The work of Ravi Sandhu is partly supported by grant CCR-9503560 from the National Science Foundation and contract MDA904-94-C-61?? from the National Security Agency at George Mason University.

References

[BSJ93] Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. Authorizations in relational database management systems. In *1st ACM Conference on Computer and Communications Security*, pages 130–139, Fairfax, VA, November 3-5 1993.

[CFMS94] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison Wesley, 1994.

[Cho92] Santosh Chokhani. Trusted products evaluation. *Communications of the ACM*, 35(7):64–76, July 1992.

[DG93] Chris Davies and Ravi Ganesan. Bapasswd: A new proactive password checker. In *16th NIST-NCSC National Computer Security Conference*, pages 1–15, 1993.

[FGL93] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. An examination of federal and commercial access control policy needs. In *NIST-NCSC National Computer Security Conference*, pages 107–116, Baltimore, MD, September 20-23 1993.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, October 13-16 1992.

[GD94] Ravi Ganesan and Chris Davies. A new attack on random pronouncable password generators. In *17th NIST-NCSC National Computer Security Conference*, pages 184–197, 1994.

[GL91] T.D. Garvey and T. Lunt. Model-based intrusion detection. In *Proc. 14th Nat. Computer Security Conference*, pages 372–385, Washington, DC, October 1991.

[IKP95] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transaction on Software Engineering*, 21(3):222–232, March 1995.

[KPS95] Charles Kaufman, Radia Perlman, and Mike Speciner. *Network Security*. Prentice Hall, 1995.

[MHL94] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, May/June 1994.

[Neu94] B. Clifford Neuman. Using Kerberos for authentication on computer networks. *IEEE Communications*, 32(9), 1994.

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[San94] Ravi S. Sandhu. On five definitions of data integrity. In T. Keefe and C.E. Landwehr, editors, *Database Security VII: Status and Prospects*, pages 257–267. North-Holland, 1994.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SS94] Ravi S. Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9):40–48, 1994.

[WL92] Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39–52, January 1992.

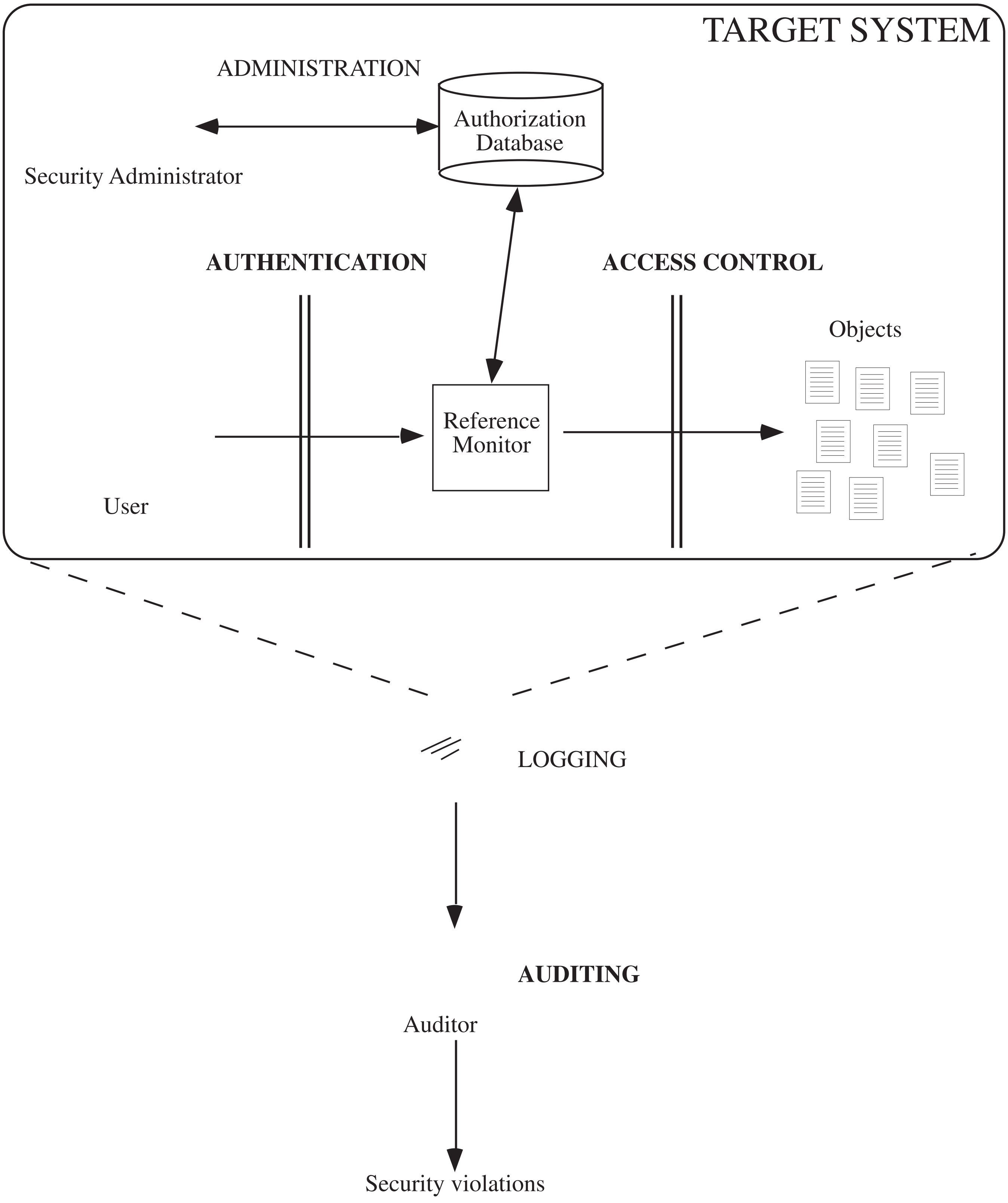

Figure 1: Access Control and Other Security Services

	File 1	File 2	File 3	File 4	Account 1	Account 2
John	Own R W		Own R W		Inquiry Credit	
Alice	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Bob	R W	R		Own R W		Inquiry Debit

Figure 2: An Access Matrix

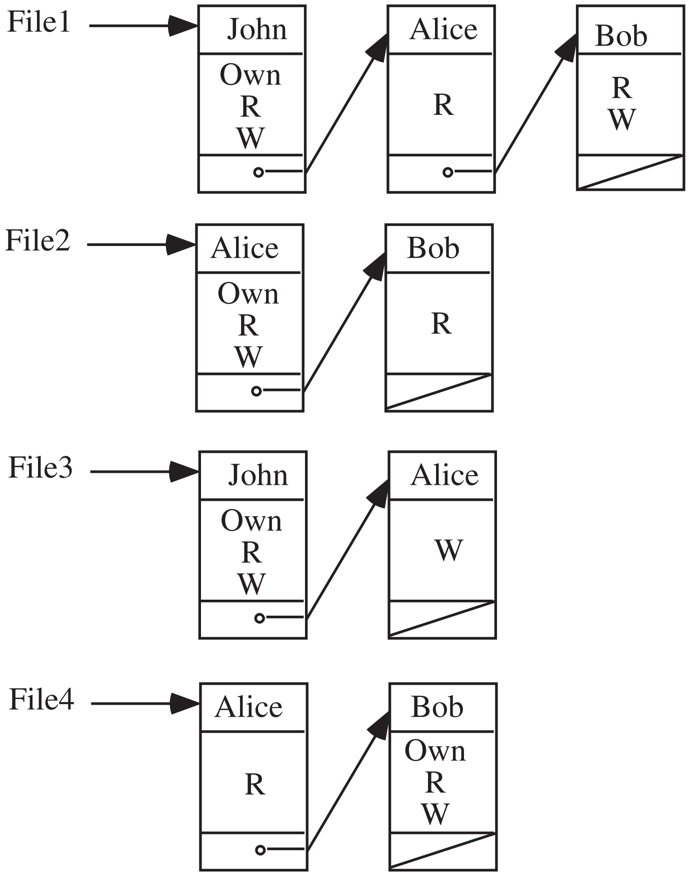

Figure 3: Access Control Lists for Files in Figure 1

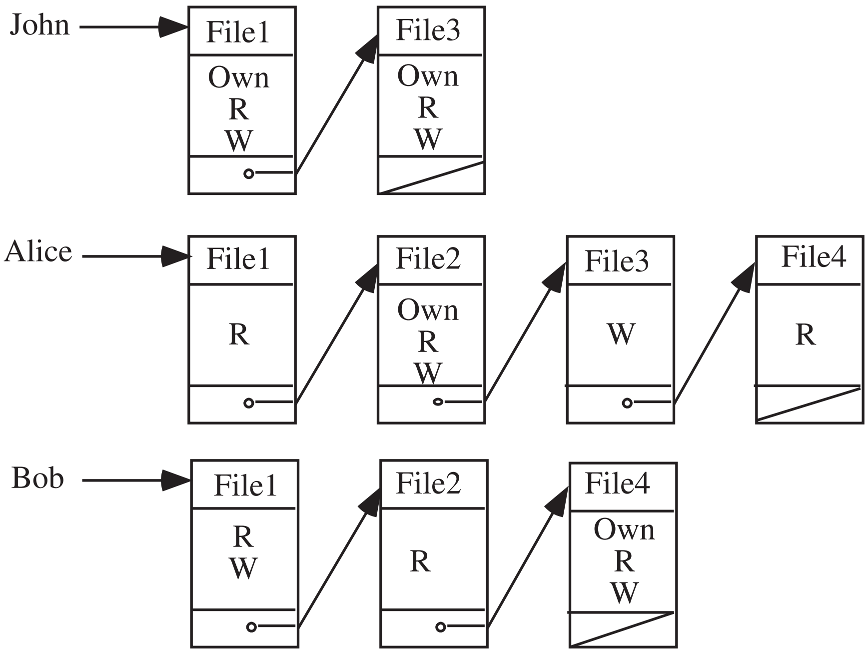

Figure 4: Capability Lists for Files in Figure 1

Subject	Access mode	Object
John	Own	File 1
John	R	File 1
John	W	File 1
John	Own	File 3
John	R	File 3
John	W	File 3
Alice	R	File 1
Alice	Own	File 2
Alice	R	File 2
Alice	W	File 2
Alice	W	File 3
Alice	R	File 4
Bob	R	File 1
Bob	W	File 1
Bob	R	File 2
Bob	Own	File 4
Bob	R	File 4
Bob	W	File 4

Figure 5: Authorization Relation for Files in Figure 1

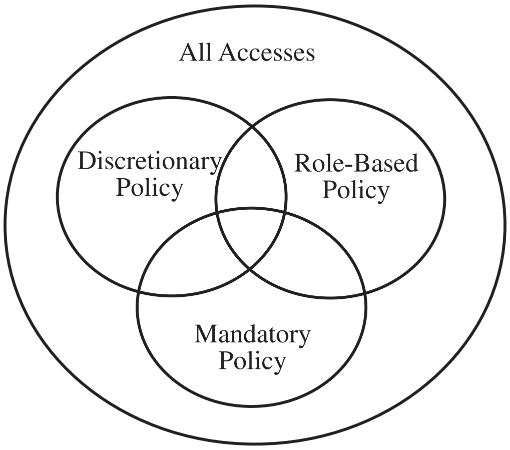

Figure 6: Multiple Access Control Policies

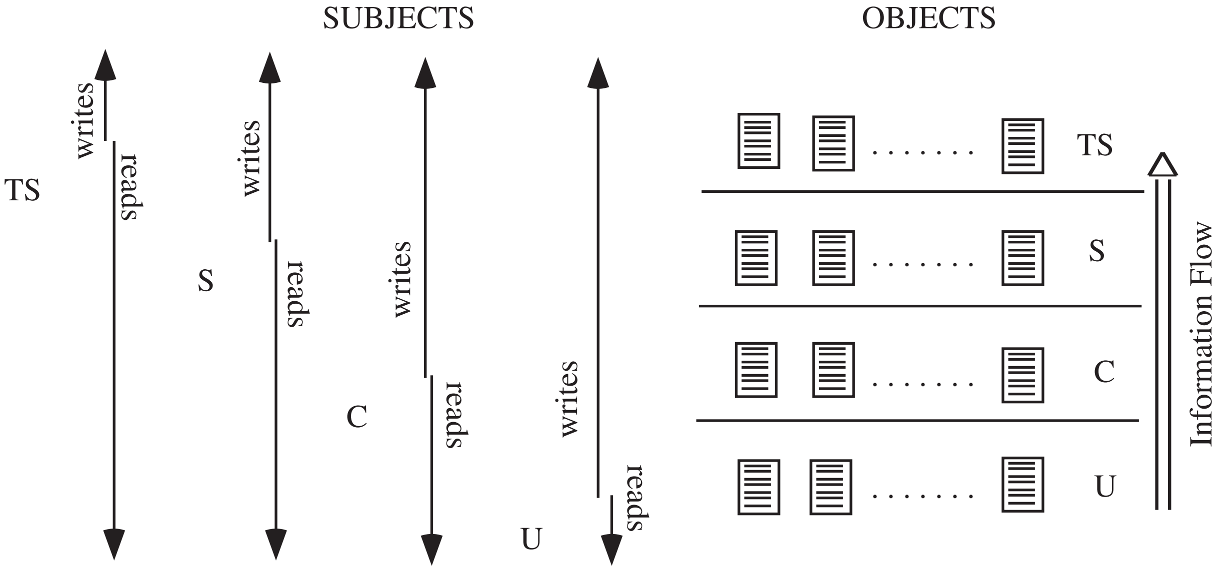

Figure 7: Controlling Information Flow for Secrecy

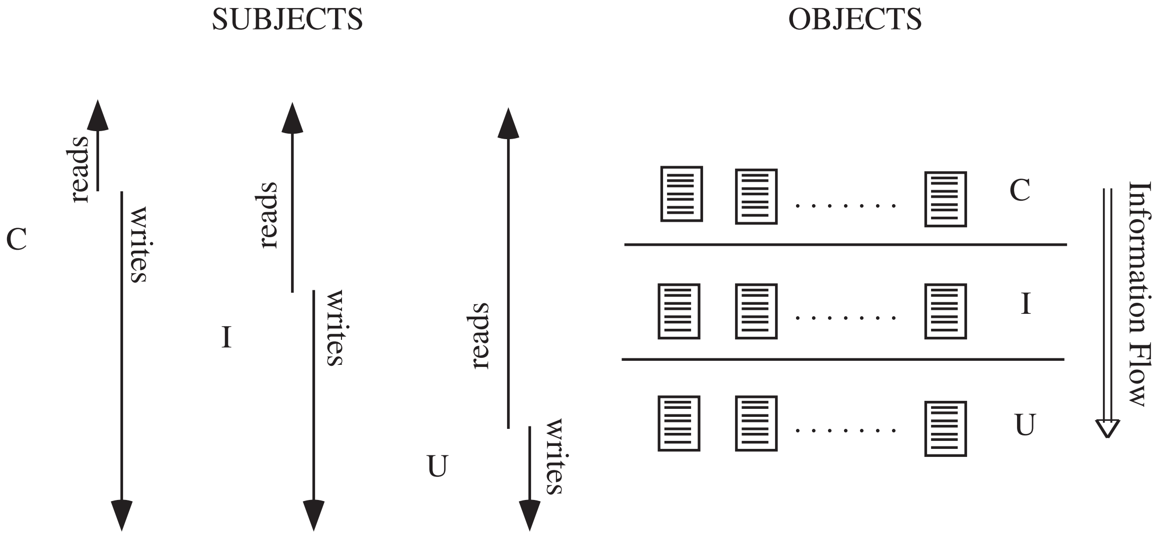

Figure 8: Controlling Information Flow for Integrity

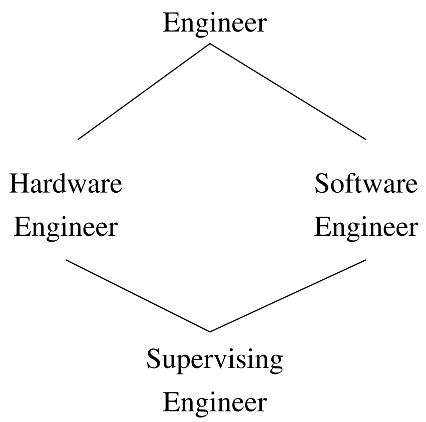

Figure 9: A Role Inheritance Hierarchy

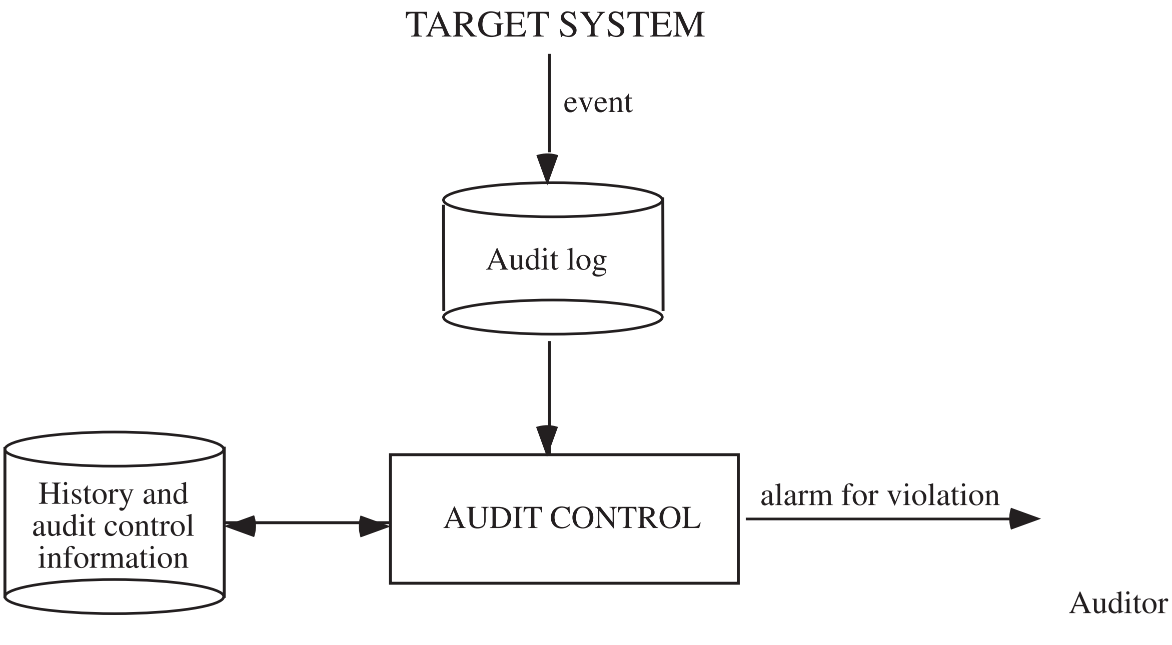

Figure 10: Passive Intrusion Detection

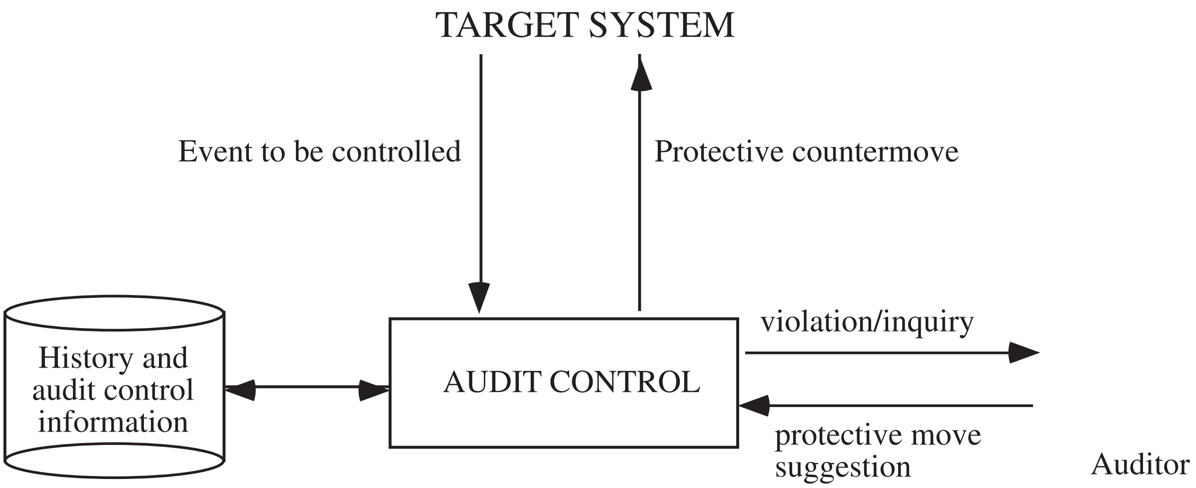

Figure 11: Active Intrusion Detection