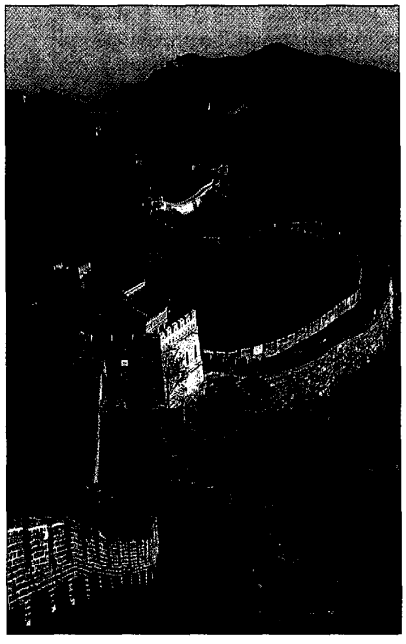

Lattice-Based Access Control Models

Ravi S. Sandhu, George Mason University

Although developed for the defense sector, lattice-based access controls can be used in most circumstances where information flow is critical. They are a key component of computer security.

System architects and users recognized the need for information security with the advent of the first multiuser computer systems. This need gained significance as computer systems evolved from isolated mainframes behind guarded doors to interconnected and decentralized open configurations.

Information security has three separate but interrelated objectives:

(1) *confidentiality* (or *secrecy*), related to disclosure of information,
(2) *integrity*, related to modification of information, and
(3) *availability*, related to denial of access to information.

These objectives appear in practically every information system. In a payroll system, for example, confidentiality is concerned with preventing an employee from finding out the boss's salary; integrity, with preventing an employee from changing his or her own salary; and availability, with ensuring that paychecks are printed on time. Similarly, in a military command and control complex, confidentiality is concerned with preventing the enemy from determining the target coordinates of a missile; integrity, with preventing the enemy from altering the target coordinates; and availability, with ensuring that the missile is launched when the order is given.

Bell and LaPadula developed lattice-based access control models to deal with information flow in computer systems. Information flow is clearly central to confidentiality and also applies to integrity to some extent. But its relationship to availability is tenuous at best. Hence, these models are primarily concerned with confidentiality and can deal with some aspects of integrity.

Bell, Biba, LaPadula, and Denning performed the basic research in this area in the 1970s. Since then, models have been implemented in a number of systems, mostly driven by the needs of the US defense sector and its allies. The theory and concepts are, however, applicable to almost any situation in which information flow is a concern. The commercial sector has unique policies that concern information flow.

0018-9162/93/1100-0009$03.00 © 1993 IEEE

Lattice-based access control is one of the essential ingredients of computer security. This article describes a number of models developed in this context and examines their underlying theoretical and conceptual foundations.

Information flow policies

Information flow policies are concerned with the flow of information from one security class to another. In a system, information actually flows from one object to another. The models I discuss treat "object" as an undefined primitive concept. An object can be informally defined as a container of information. Typical examples of objects are files and directories in an operating system, and relations and tuples in a database.

Information flow is usually controlled by assigning every object a *security class*, also called a *security label*. Whenever information flows from object x to object y, there is an accompanying information flow from the security class of x to the security class of y. Henceforth, when I talk about information flowing from security class A to security class B, visualize information flowing from an object labeled A to an object labeled B.

Denning defined the concept of an information flow policy[1] as follows:

Definition 1 [Information flow policy] — A triple $< SC, \rightarrow, \oplus >$ where SC is a set of security classes, $\rightarrow \subseteq SC \times SC$ is a binary can-flow relation on SC, and $\oplus : SC \times SC \rightarrow SC$ is a binary class-combining or join operator on SC.

All three components of an information flow policy are fixed; they don't change with time. This definition allows objects to be created and destroyed dynamically (as one would expect in useful systems). Security classes, however, cannot be created or destroyed dynamically.

It is convenient to use infix notation for the can-flow relation, so that $A \rightarrow B$ means the same as $(A, B) \in \rightarrow$; that is, information can flow from A to B. We also write $A \not\rightarrow B$ to mean $(A, B) \notin \rightarrow$; that is, information cannot flow from A to B. In other words, information can flow from security class A to security class B under a given policy if and only if $A \rightarrow B$. (It might be more appropriate to call this relation may-flow rather than can-flow, since the connotation is that the indicated flow is permitted under the given policy. I opt to retain Denning's original terminology.)

Similarly, infix notation is used for the join operator; that is, $A \oplus B = C$ means the same as $\oplus(A, B) = C$. The join operator specifies how to label information obtained by combining information from two security classes. Thus, $A \oplus B = C$ tells us that objects that contain information from security classes A and B should be labeled with the security class C.

A trivial example of an information flow policy is one in which no information flow is allowed from one security class to a different security class. (Note that information flow from a security class to itself cannot be prevented and therefore must always be allowed. After all, information contained in an object *flows* to that object, thereby resulting in information flow from the security class of the object to itself.) This trivial policy of isolated security classes is formally stated as follows:

Example 1 [Isolated classes] — $SC = \{A_1, \ldots, A_n\}$; for $i = 1 \ldots n$ we have $A_i \rightarrow A_i$ and $A_i \oplus A_i = A_i$; and for $i, j = 1 \ldots n$, $i \neq j$ we have $A_i \not\rightarrow A_j$ and $A_i \oplus A_j$ is undefined.

The simplest form of a nontrivial information flow policy occurs when there are only two security classes called, for example, H (for high) and L (for low); all flows are allowed except that from high to low. In other words, high information is more sensitive than low information. This is stated formally as

Example 2 [High-low policy] — $SC = \{H, L\}$, and $\rightarrow = \{(H, H), (L, L), (L, H)\}$. Equivalently, in infix notation, $H \rightarrow H$, $L \rightarrow L$, $L \rightarrow H$, and $H \not\rightarrow L$. The join operator is defined as follows: $H \oplus H = H$, $L \oplus H = H$, $H \oplus L = H$, and $L \oplus L = L$.

This policy is represented by the Hasse diagram in Figure 1a in which the can-flow relation is understood to be directed upward. Reflexive flows from H to H and from L to L are implied but not explicitly shown. The other Hasse diagrams in Figure 1 represent information flow policies that I discuss in this article. In these diagrams, transitive edges, such as from L to H in Figure 1b, are implied but not explicitly shown.

Denning showed that under certain assumptions, an information flow policy forms a finite lattice:

Definition 2 [Denning's axioms] —

(1) The set of security classes SC is finite.
(2) The can-flow relation $\rightarrow$ is a partial order on SC.
(3) SC has a lower bound with respect to $\rightarrow$.
(4) The join operator $\oplus$ is a totally defined least upper bound operator.

It can be shown that Denning's axioms imply the existence of a greatest lower bound operator, which in turn implies the existence of an upper bound with respect to $\rightarrow$. Example 2 satisfies Denning's axioms, whereas Example 1 does not, specifically failing to satisfy axioms 3 and 4. I will illustrate how

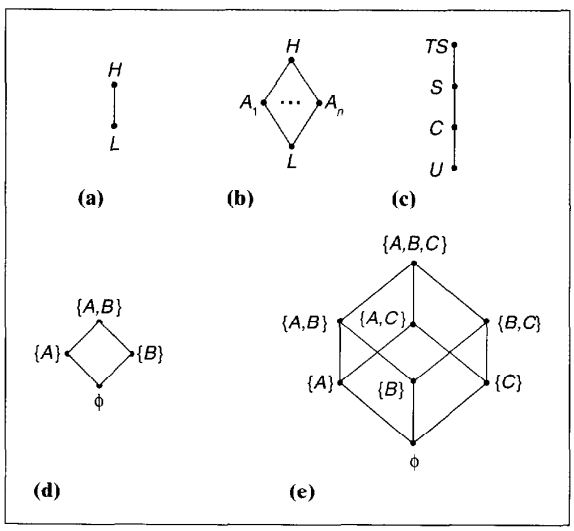

Figure 1. Hasse diagrams for certain information flow policies.

Example 1 can be extended to form a lattice. Note that although this article focuses on policies that satisfy Denning's axioms, there are legitimate information flow policies that do not satisfy these axioms.

Denning's first axiom. This requires that the set of security classes be finite and needs little justification. Keep in mind that the axiom applies to security classes and not to the objects in a system. Denning's axioms enable objects to be created and destroyed dynamically, with no bound on the number of objects that can be created.

Denning's second axiom. This states that $\rightarrow$ is a partial order on SC. A partial order is a reflexive, transitive, and antisymmetric binary relation. Above, we saw the need for reflexivity in the context of Example 1, whereby $A \rightarrow A$ for all $A \in SC$. Transitivity requires that if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$; that is, if indirect information flow is possible from A to C via B, then we should allow direct information flow from A to C. This is a very reasonable requirement in most situations. (There are, however, situations in which indirect flow should not imply direct flow. For example, suppose we wish to allow transfer of information from H to L but only if mediated by a sanitizing process with security class SAN. We then have $H \rightarrow SAN$, $SAN \rightarrow L$, but $H \not\rightarrow L$. These situations are typically handled as exceptions falling outside the normal lattice framework of information flow. Such nontransitive information flows can be enforced using the concepts of *type enforcement* and *assured pipelines*.[2] Nontransitive information flow policies can also be expressed in the *typed matrix access model*.[3])

Antisymmetry requires that if $A \rightarrow B$ and $B \rightarrow A$, then $A = B$. Given the reflexive and transitive requirements, antisymmetry merely eliminates redundant security classes. In other words, there is no point in having two different security labels if objects with these labels are restricted to having exactly the same information flows.

Denning's third axiom. This requires that SC have a lower bound L (for system low), that is, $L \rightarrow A$ for all $A \in SC$. This axiom acknowledges the existence of public information in the system. Public information allows for desirable features such as public bulletin boards and databases, which users expect to find in any modern computer system. From a theoretical perspective, one can argue that information from constants should be allowed to flow to any other object; therefore, constants should be labeled L. An example of such a constant would be version information about the operating system. Version information is necessary for the correct operation of certain programs and should be publicly available. Note that the policy of Example 1 does not have a lower bound.

Denning's fourth axiom. This is the most subtle. There are actually two parts to it. First, the join operator is required to be totally defined; that is, $A \oplus B$ is defined for every pair of security classes from SC. This means that it is possible to combine information from any two security classes and give the result a label. In Example 1, this property was not satisfied; that is, $A_i \oplus A_j$ was undefined for $i \neq j$. To bring Example 1 into line with Denning's axioms, we can introduce a new security class H (for system-high security) and define $A_i \oplus A_j = H$ for $i \neq j$. By introducing L and H, we can modify Example 1 as follows:

Example 3 [Bounded isolated classes] — $SC = \{A_1, \ldots, A_n, L, H\}$; $L \rightarrow L$, $L \rightarrow H$, $H \rightarrow H$, and for $i = 1 \ldots n$ we have $L \rightarrow A_i$, $A_i \rightarrow A_i$, $A_i \rightarrow H$; for $i = 1 \ldots, n$ we have $A_i \oplus A_i = A_i$, $A_i \oplus H = H$, and $A_i \oplus L = A_i$; and for $i, j = 1 \ldots n, i \neq j$ we have $A_i \oplus A_j = H$.

The Hasse diagram in Figure 1b shows this policy. The can-flow relation goes upward in the figure along the edges shown. (Recall that reflexive edges, such as from A_i to A_i, and transitive edges, such as from L to H, are implied but not explicitly shown.) System-high objects have a practical role in that information about the global state of the system can only go in objects labeled H; such information could be crucial for proper system administration and audit. On the other hand, this example also suggests that in some situations it might be more appropriate to use partially ordered labels than to strive for a complete lattice.

The second part of Denning's fourth axiom states that the join operator is a least upper bound. This means that for all $A, B, C \in SC$, we have property 1, which is $A \rightarrow A \oplus B$ and $B \rightarrow A \oplus B$, and property 2 if $A \rightarrow C$ and $B \rightarrow C$ then $A \oplus B \rightarrow C$. Property 1 follows from the intuition underlying the join operator; that is, $A \oplus B$ is the label on information collectively obtained from A and B. Therefore, information from A, as well as from B, should be able to flow to $A \oplus B$. Property 2 stipulates that if information can flow individually from A and B to C, then information obtained by combining information from A and B should also be able to flow to C. This is a reasonable requirement, somewhat analogous to the transitivity property in Denning's second axiom.

An important consequence of Denning's fourth axiom is that the join operator can be applied to any number of security classes. This is because least upper bound is an associative and commutative operator. Thus, we can compute $A_1 \oplus A_2 \oplus \ldots \oplus A_n$ to be the least upper bound of $\{A_1, A_2, \ldots, A_n\}$. In this manner, we can label the result of combining information from any number of security classes.

Finally, note that the security literature is usually cast in terms of the inverse of the can-flow relation, defined as follows:

Definition 3 [Dominance] — $A \geq B$ (read as A dominates B) if and only if $B \rightarrow A$. The strictly dominates relation $>$ is defined by $A > B$ if and only if $A \geq B$ and $A \neq B$. We say that A and B are comparable if $A \geq B$ or $B \geq A$; otherwise A and B are incomparable.

The strictly dominates relation has the following significance: If $A > B$ then $A \not\rightarrow B$, but $B \rightarrow A$. In other words, A is more sensitive than B.

The military lattice

The simplest examples of nontrivial information flow policies occur when the can-flow relation is a total or linear ordering of the security classes. The most common examples of totally ordered security classes are the TS (for top secret), S (for secret), C (for confidential), and U (for unclassified) sensitivity levels encountered in the military and government sectors (see Figure 1c). In general, we can have any number of totally ordered security classes. (In the security literature, a total or linear ordering is often called a hierarchical ordering, but in this article I avoid using that expression, since it is some-

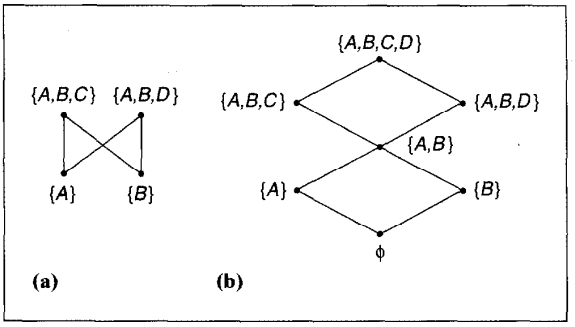

Figure 2. Embedding a partial order in a lattice.

times understood to mean a tree-like ordering.)

Note that ≥ (or dominance) is a total ordering if and only if its inverse → (or can-flow) is a total ordering. Moreover, there are no incomparable security classes in a total ordering. The definition of $A \oplus B$ is then simply the maximum of A and B with respect to the dominance relation. In other words, when information from two security classes is combined, the label of the higher result of the two classes is used for the result (for example, $S \oplus U = S$).

Figure 1d shows a partially ordered lattice. The security classes are obtained as the power set (that is, set of all subsets) of $\{A, B\}$. Say A denotes salary information and B denotes medical information in a personnel database. The system-low class is the empty set, which can have public information but no salary or medical information. The security labels $\{A\}$ and $\{B\}$ are singleton sets, respectively corresponding to salary information and medical information. When salary information and medical information are combined, the result must be labeled $\{A, B\}$. Note that $\{A\}$ and $\{B\}$ are incomparable and that $\{A\} \oplus \{B\} = \{A, B\}$. In this policy, can-flow is identical to the subset relation, dominance is identical to superset, and join is the set union of the labels. Such a lattice is called a *subset lattice*. In the military and government sectors, the individual set elements (that is, A and B) are known as *categories*; the security classes (that is, sets of categories) are known as *compartments*.

Figure 1e shows a subset lattice with three categories A, B, and C that might denote salary, medical, and educational information, respectively. In this case, the security classes $\{A\}$ and $\{B\}$ have two upper bounds, namely, $\{A, B\}$ and $\{A, B, C\}$, with $\{A, B\}$ being the least upper bound.

Similarly, subset lattices of any size can be defined. Since there are 2^n subsets of a set of size n, there is an exponential increase in the number of security classes as the number of categories increases. In practice, only a decreasingly small fraction of the security classes actually would be employed for large n.

Selecting an arbitrary subset of a lattice will not necessarily yield a lattice. For example, the partial order of Figure 2a results by selecting these four security classes from the subset lattice on $\{A, B, C, D\}$. The partial order of Figure 2a fails to be a lattice for two reasons. First, it is missing the system-low and system-high security classes. Second, the two upper bounds of $\{A\}$ and $\{B\}$ are incomparable; hence there is no least upper bound of $\{A\}$ and $\{B\}$. By filling in these missing security classes, we can extend the partial order of Figure 2a to obtain the lattice of Figure 2b. Such a construction is always possible for any partial order; that is, every partial order can be embedded in a lattice by including additional security classes.

The two lattices considered above (that is, the totally ordered lattice and the subset lattice) are often combined. This is particularly true in the military and government sectors, where this structure is laid down by law. (The system-high security clearance is sometimes not given to any individual in systems that contain highly sensitive information.) Each security class has two components: one from the totally ordered security lattice of Figure 1c, and the second from a subset lattice on some number of categories. One label is said to dominate another if each component of the first label dominates the corresponding component of the second. For example, $< TS, \{A\} >$ dominates $< S, \{A\} >$ but is incomparable to $< S, \{B\} >$. The join of two labels is similarly defined as the pairwise join of the individual components (for example, $< TS, \{A\} > \oplus < S, \{B\} > = < TS, \{A, B\} >$). It is easy to see that the result is a lattice. In fact, it is known as the *product lattice* of the two underlying lattices. This example illustrates the general property that the product of two lattices is a lattice.

It is possible to generate very large lattices in this manner. As mentioned previously, only a small subset of the entire lattice realistically would be used in such cases. Smith[4] describes an actual lattice based on common practice in the military. This lattice consists of the four linearly ordered security levels $TS > S > C > U$, and eight categories $\{A, K, L, Q, W, X, Y, Z\}$ corresponding to, say, eight different projects in the system. Smith's lattice (see Figure 3) has 21 labels from a possible space of $4 \times 2^8 = 1{,}024$ labels. The 21 labels actually used do constitute a lattice. Except for the system-high security class, combinations of categories occur only in twos and threes. In addition, the use of categories occurs mostly above top secret, and none occur below secret.

Access control models

The active entities in a system are usually processes executing programs on behalf of users. Therefore, information flow between objects, and thereby between security classes, is carried out by processes. Information can potentially flow from every object that a process reads to every object that it writes. In the absence of knowledge about what a given program does, we must assume the worst case and say that wherever there is a potential for information flow, the flow actually occurs. Said another way, we must be conservative and ensure that programs simply do not have the ability to cause information flows contrary to the given policy. Before showing how the Bell-LaPadula model[5] addresses this objective, I introduce some basic abstractions for access control models.

To understand access control and computer security, we must first understand the distinction between users and subjects. This distinction is fundamental but often is treated imprecisely, leading to undue confusion about the objectives of computer security.

We understand that a user is a human being. And we assume that each human

being known to the system is recognized as a unique user. Stated another way, the unique human named Jane Doe cannot have more than one user identity in the system. If Jane Doe is not an authorized user of the system, she has no user identity. Conversely, if she is an authorized user, she is known by exactly one user identity, say, JDoe.

Clearly this assumption can be enforced only by adequate administrative controls, which we assume are in place. This assumption is not required for some of the policies considered in this article. At the same time, it is crucial for policies such as Lipner's integrity lattice and Chinese Walls, which are discussed later. This assumption is often violated in current systems. (It is worth observing that the converse requirement, that each user identifier in the system be associated with exactly one human being, is critical to maintaining strict accountability. The use of shared user-identifiers to facilitate sharing is usually applied only because the system lacks convenient facilities for such sharing. In a properly designed system, there should be no need for this artifice.)

We understand a subject to be a process in the system; that is, a subject is a program in execution. Each subject is associated with a single user, with the subject executing on the user's behalf. In general, a user can have many subjects concurrently running in the system. Every time a user logs in to the system, the user does so as a particular subject. (Note that access control models assume that identification and authentication of users takes place in a secure and correct manner, and are concerned with what happens afterward.)

Different subjects associated with the same user can obtain different sets of access rights. Suppose that top-secret user John logs in at the secret level. In user-subject terminology, we interpret this as follows: First, there is a unique user, John, cleared to top secret; second, John can have subjects executing on his behalf at every level dominated by top secret. For now, assume that each subject runs at a fixed security level. (Below, we see that the security level of subjects can be changed in some models.)

The access rights of subjects to objects in a system are conceptually represented by an *access matrix*.[6] This matrix has a row for every subject and a column for every object. A subject can also be an object in the system; for example, one process can execute suspend and resume operations on another process.

In general, all subjects are also objects, but not all objects are subjects. The cell for row s and column o is denoted by $[s, o]$ and contains a set of access rights specifying the authorization of subject s to perform operations on object o. For example, $read \in [s, o]$ authorizes s to read o. Operations authorized by the access matrix are the only ones that can be executed. In the access matrix, all users are also regarded as subjects in their own right. A subject retains the access rights of the user, even when the user is not engaged in any activity in the system. The access matrix is usually sparse and is stored in a system using access control lists, capabilities, relations, or another data structure suitable for efficient sparse-matrix storage.

The access matrix is a dynamic entity, and its individual cells can be modified by subjects. For example, if subject s is the owner of object o (that is, $own \in [s, o]$), then s typically can modify the contents of all cells in the column corresponding to o. In this case, the owner of an object has complete discretion regarding access to the owned object by other subjects. Such access controls are said to be *discretionary*.

The access matrix also changes with the addition and deletion of subjects and objects. (The typical access control on files and directories, provided by popular multiuser operating systems on the basis of protection bits, is an example of discretionary access control. Another example of discretionary access control is the access control on relations and parts of relations provided by popular relational database management systems.)

By themselves, discretionary access controls are inadequate for enforcing information flow policies. Their basic problem is that they provide no constraint on copying information from one object to another. (There are also other, more subtle problems with discretionary access controls, notably concerning the so-called safety problem[3] for propagation of access rights.)

Suppose that Tom, Dick, and Harry are users and that Tom has a confiden-

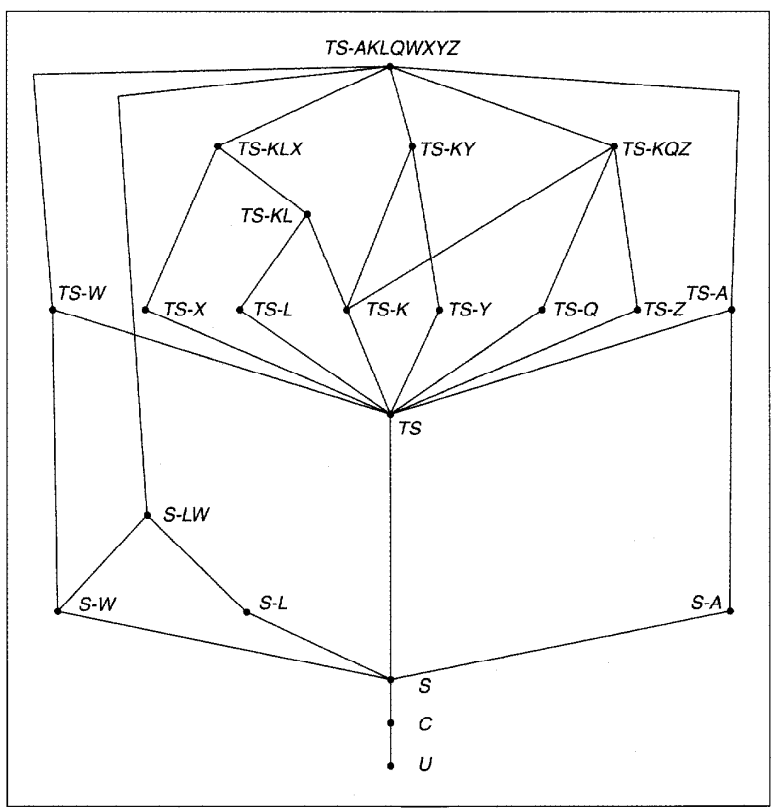

Figure 3. Smith's lattice.

tial file Private that he wants Dick to read but doesn't want Harry to read. Tom can authorize Dick to read the file by entering *read* in [Dick, Private]. (Assume that Dick does not thereby have the authority to grant the *read* right for Private to other users, such as Harry.) Dick can easily subvert Tom's intention by creating a new file called Copy-of-Private and copying the contents of Private into it. As the creator of Copy-of-Private, Dick has the authority to grant *read* access for it to any user, including Harry; that is, Dick can enter *read* in [Harry, Copy-of-Private]. Then, for all practical purposes, Harry can read Private as long as Dick keeps Copy-of-Private reasonably up to date with respect to Private.

This situation is actually worse than the above scenario indicates. So far, I have portrayed Dick as a cooperative participant in this process. Now suppose Dick is Tom's trusted confidant and would not deliberately subvert Tom's intentions regarding the Private file. However, Dick uses an ingenious text editor that Harry gave to him. This editor provides all the editing services that Dick needs. In addition, Harry has also programmed the editor to create the Copy-of-Private file and to grant Harry the right to read Copy-of-Private. This kind of *Trojan horse* software performs normal functions expected by its user, but also engages in surreptitious activities to subvert security. A similar Trojan horse that Harry created and Tom executed could actually grant Harry the privilege to directly read Private. We can require that all software run on the system be free of Trojan horses, but this is hardly practical, particularly if we wish to guarantee this freedom with a high degree of assurance. The solution is to impose mandatory controls that cannot be bypassed, even by Trojan horses.

The Bell-LaPadula model

Bell and LaPadula[5] formalized the concept of mandatory access controls, defining a model commonly bearing their names. Numerous variations of the model have since been published. Consequently, the exact meaning of the Bell-LaPadula model is not clear.

In this article, I take a minimalist approach and define a model, called BLP, that is generally the smallest model capturing the essential access control properties I want to illustrate. The notation and precise formulation of the rules of BLP are substantially different from those of the original Bell-LaPadula model. BLP is more in line with the formulations authors of more recent literature have used.

Mandatory access control policy is expressed in terms of security labels attached to subjects and objects.

The key idea in BLP is to augment discretionary access controls with mandatory access controls to enforce information flow policies. BLP takes a two-step approach to access control. First is a discretionary access matrix D, whose contents can be modified by subjects (in a way we don't need to specify here). However, authorization in D is not sufficient for an operation to be carried out. Second, the operation must be authorized by the mandatory access control policy, over which users have no control.

Mandatory access control policy is expressed in terms of security labels attached to subjects and objects. A label on an object is called a *security classification*, while a label on a user is called a *security clearance*. A user labeled secret can run the same program, such as a text editor, as a subject labeled secret or as a subject labeled unclassified. Even though both subjects run the same program on behalf of the same user, they obtain different privileges due to their security labels. It is usually assumed that once assigned, the security labels on subjects and objects cannot be changed (except by the security officer). This assumption, known as *tranquility*, can be relaxed in a secure manner (see below).

With λ signifying the security label of the indicated subject or object, the specific mandatory access BLP rules are as follows:

- Simple-security property: Subject s can read object o only if $\lambda(s) \geq \lambda(o)$.
- ★-property (read as star-property): Subject s can write object o only if $\lambda(s) \leq \lambda(o)$.

Read access implies a flow from object to subject, hence the requirement that $\lambda(s) \geq \lambda(o)$ or equivalently $\lambda(o) \rightarrow \lambda(s)$ (that is, $\lambda(o)$ can flow to $\lambda(s)$). Write access conversely implies a flow from subject to object, hence the requirement that $\lambda(s) \leq \lambda(o)$ or equivalently $\lambda(s) \rightarrow \lambda(o)$ (that is, $\lambda(s)$ can flow to $\lambda(o)$). Write access is interpreted here as write only. In some models, write access is interpreted to mean read and write, with append access provided for write only.

These properties are stated in terms of read and write operations. In a real system, additional operations will exist (for example, create and destroy objects). Considering read and write suffices to illustrate the main points. For example, create and destroy are also constrained by the ★-property because they modify the state of the object in question. Mandatory controls are formulated as "only if" conditions; that is, they are necessary but not sufficient for the indicated access. In operational terms, we can visualize the mandatory controls as kicking in only after the checks embodied in the discretionary access matrix D have been satisfied. If D does not authorize the operation, we do not need to check the mandatory controls, since the operation will be rejected anyway. Equivalently, the mandatory controls can be checked first, followed by a check of the discretionary controls.

The simple-security requirement applies to humans and programs equally, and the need for it is self-evident. On the other hand, the ★-property is not applied to human users, but rather to programs. Human users are trusted not to leak information. A secret user can write an unclassified document because we assume that he or she will put only unclassified information in it. Programs, on the other hand, are not trusted because they can have embedded Trojan horses. The ★-property prohibits a program running at the secret level from writing to unclassified objects, even if it is permitted to do so by discretionary access controls. A user labeled secret who wishes to write an unclassified object must log in as an unclassified subject.

A curious aspect of the ★-property is

that an unclassified subject can write a secret file. This means that secret data can be destroyed or damaged, perhaps accidentally, by unclassified subjects. To prevent this integrity problem, a modified ★-property is sometimes used that requires $\lambda(s) = \lambda(o)$; that is, subjects can write at their own level but cannot "write up."

Here's how these properties impact the previous Tom, Dick, and Harry Trojan horse example. Suppose Tom and Dick are secret users and Harry is an unclassified user. Tom and Dick can have secret and unclassified subjects, while Harry can have only unclassified subjects. Now, let Tom create the secret file Private via a secret subject. The simple-security property will prevent Harry's subjects from being able to directly read the file Private. The simple-security and ★-properties will ensure that Harry's subjects cannot surreptitiously read Copy-of-Private because Copy-of-Private will either be labeled secret (or above) or will not contain any information from Private. Specifically, if Dick's Trojan horse-infected secret subject creates Copy-of-Private, it will be a secret (or above) file and Harry's unclassified subjects will not be able to read it. On the other hand, Dick's unclassified subject running the Trojan horse cannot read Private and copy it to Copy-of-Private. BLP mandatory access controls only prevent information flow between security classes. Thus, if Harry was a secret user like Tom and Dick, these controls would not solve the problem.

Unfortunately, mandatory controls do not solve the Trojan horse problem completely. A secret subject is prevented from writing directly to unclassified objects. There are, however, other ways of communicating information to unclassified subjects. For example, the secret subject can acquire large amounts of memory in the system. This fact can be detected by an unclassified subject that can observe how much memory is available. Even if the unclassified subject is prevented from directly observing the amount of free memory, it can do so indirectly by making a request for a large amount of memory itself. Granting or denying this request will convey some information about free memory to the unclassified subject. The load on the CPU can be similarly modulated to communicate information.

This kind of indirect method of communication is called a *covert channel*.[7] Covert channels present a formidable problem for enforcement of information flow policies. They are difficult to detect, and once they are detected they are difficult to close without incurring significant performance penalties. Covert channels tend to be noisy due to interference by the activity of other subjects in the system. Nevertheless, standard coding techniques for communication on noisy channels can be used by Trojan horses to achieve error-free communication. The resulting data rates can be as high as several million bits per second if efforts are not made to mitigate the channels.

Once covert channels are detected, they are difficult to close without incurring significant performance penalties.

The covert channel problem is outside the scope of lattice-based access control models such as Bell-LaPadula. These models seek to prevent insecure information flows via objects that are explicitly intended for interprocess data sharing and communication. The problem of avoiding, mitigating, or tolerating covert channels is considered an implementation and engineering issue, requiring analysis of the system architecture, design, and code (see Proctor and Neumann[8]). Another class of models, called *information flow models*, attempts to deal with all information flows uniformly.[9]

Above I noted that the tranquility assumption requires that security labels of subjects and objects not change. This assumption can be relaxed in several different ways, some securely and others insecurely (that is, they introduce information flow contrary to the given lattice). Suppose we allow a subject s to change the security label of object o from $\lambda(o)$ to $\lambda'(o)$, with the stipulation that $\lambda(s) = \lambda(o)$, and $\lambda'(o) > \lambda(o)$; for example, an unclassified subject upgrades the label of a file from unclassified to secret. Such a change is secure, because it causes no information flow from secret to unclassified. Now suppose we allow a secret subject to perform the same change; that is, we replace the stipulation that $\lambda(s) = \lambda(o)$ with $\lambda(s) > \lambda(o)$. This latter case is insecure, because upgrading an unclassified file to secret by a secret subject will make this file disappear from the view of unclassified subjects, thereby opening a means for communicating information from secret to unclassified; this could be exploited by Trojan horses. A secret user can securely upgrade an unclassified file to secret by logging in as an unclassified subject.

The Biba model and duality

Confidentiality considerations motivated the mandatory controls in the Bell-LaPadula model. Biba proposed that similar controls could be formulated for integrity.[10] The basic concept in Biba's model is that low-integrity information should not be allowed to flow to high-integrity objects, whereas the opposite is acceptable. Biba proposed several ways to use mandatory controls for integrity objectives. The best known of these is called *strict integrity*.

In the usual formulation of the Biba model, high integrity is placed toward the top of the lattice of security labels and low integrity at the bottom. With this formulation, the permitted flow of information is from top to bottom, directly opposite that of the Bell-LaPadula model and Denning's axioms. This led Biba to propose the following mandatory controls, in analogy with the mandatory controls of the BLP model (ω denotes the integrity label of a subject or object):

- Simple-integrity property: Subject s can read object o only if $\omega(s) \leq \omega(o)$.
- Integrity ★-property: Subject s can write object o only if $\omega(s) \geq \omega(o)$.

These properties are said to be duals of the corresponding properties in BLP.

There is nothing intrinsic about placing high integrity at the top of the lattice (or placing high confidentiality at the top, for that matter). Top and bottom are relative terms coined for convenience and have no absolute significance. But information flow in the Biba model can be brought into line with the BLP model by simply saying that low integrity is at the top of the lattice and high integrity at the bottom. This forces us to invert

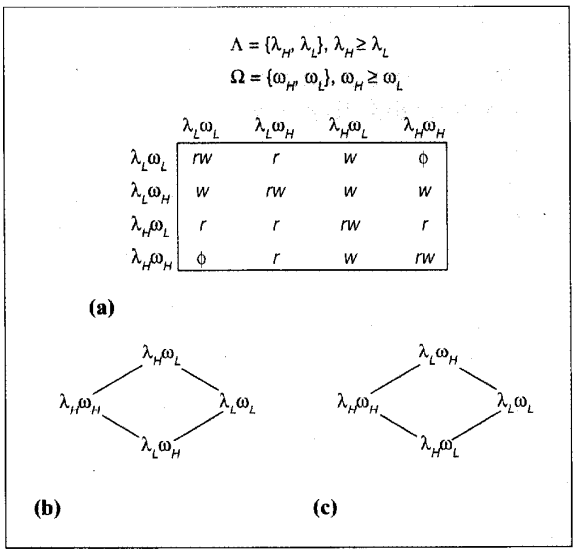

Figure 4. Example of equivalence: (a) composite model (and accompanying table stating resulting mandatory controls); (b) equivalent BLP lattice (allowed flows are upward); and (c) equivalent Biba lattice (allowed flows are downward).

the dominance relation in the Biba model so that low integrity dominates high integrity.

With this viewpoint, we can use the mandatory controls of the BLP model to enforce the information flows required by the Biba model. This situation is symmetrical. We could equally well invert the BLP (and Denning) lattices to put low confidentiality at the top and high confidentiality at the bottom, and employ the mandatory controls of Biba to enforce the information flows.

There is no fundamental difference between the Biba and BLP models. Both are concerned with information flow in a lattice of security classes, with information flow allowed only in one direction in the lattice. The BLP model allows information flow upward in the lattice, and the Biba model allows it downward. Since direction is relative, a system that can support one of these models can also support the other (given some straightforward remapping of labels to invert the dominance relation as needed).

It is often suggested that the BLP and Biba models could be combined in situations where both confidentiality and integrity are of concern. If a single label is used for both confidentiality and integrity, the models impose conflicting constraints. One adverse factor to combining them is that a subject can read or write only those objects that have exactly the same security label as the subject. This amounts to the trivial policy of no information flows between security classes as discussed in Example 1.

A more useful situation features independent confidentiality and integrity labels. As such, each security class consists of two labels: a confidentiality label λ and an integrity label ω, with BLP mandatory controls applied to the former and Biba controls to the latter. Let $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ be a lattice of confidentiality labels, and let $\Omega = \{\omega_1, \ldots, \omega_q\}$ be a lattice of integrity labels. Assume that in both lattices high confidentiality and high integrity are at the top, as proposed in the original models. The combined mandatory controls are

- Subject s can read object o only if $\lambda(s) > \lambda(o)$ and $\omega(s) \leq \omega(o)$.
- Subject s can write object o only if $\lambda(s) \leq \lambda(o)$ and $\omega(s) \geq \omega(o)$.

This popular composite model has been implemented in several operating system, database, and network products specifically built to meet requirements of the military sector. This model amounts to the simultaneous application of two lattices, with information flow occurring in opposite directions (going upward in the confidentiality lattice and downward in the integrity lattice). However, we can invert the integrity lattice and view the composite model as the simultaneous application of two lattices with information flow going upward in both of them. This gives us a product of two lattices, which is mathematically one lattice. Hence, the composite model can be reduced to a single lattice (see Figure 4).

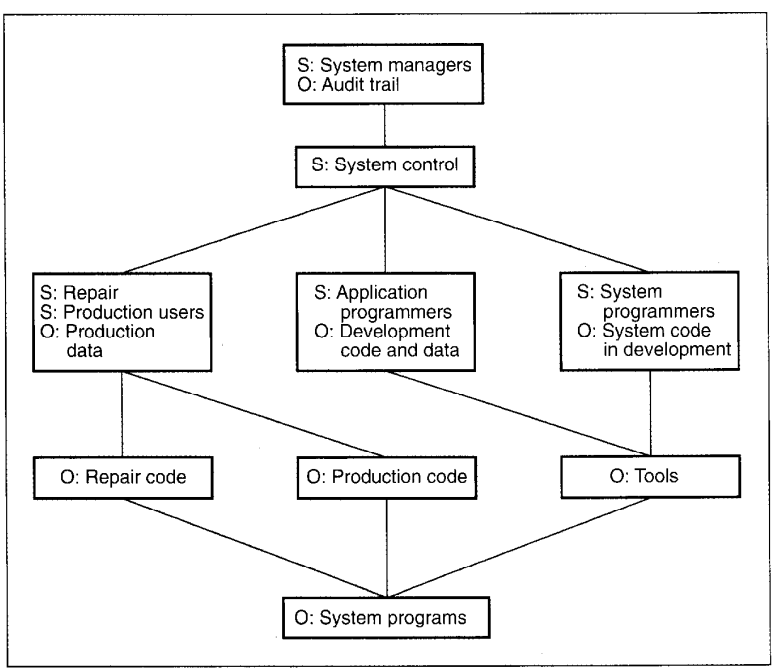

Figure 5. Lipner's composite model as a BLP lattice. Each box represents a label. Entries in each box specify subjects (prefixed S:) and objects (prefixed O:) that are assigned to that label. Allowed flows are upward.

Figure 4a shows two lattices Λ and Ω to which BLP and Biba controls, respectively, are applied. The accompanying table shows the resulting mandatory controls. Each entry specifies the maximum access that a subject with a label on the row can have to an object with a label on the column. In this table, *r* denotes read access, *w* denotes write access, *rw* denotes both read and write access, and ϕ denotes no access.

The same mandatory controls are enforced by the BLP lattice of Figure 4b. Here, subjects labeled $\lambda_H\omega_H$ cannot read objects labeled $\lambda_H\omega_L$ because of the ω-component of their labels. At the same time, they cannot write objects labeled $\lambda_L\omega_H$ because of the λ-component. They cannot read objects labeled $\lambda_L\omega_L$ because of the ω-component, and they cannot write to these objects because of the λ-component. We can similarly interpret other relationships in this lattice. Also note that the top element of the lattice has high confidentiality but low integrity, whereas the bottom element has low confidentiality but high integrity. The same mandatory controls are enforced by the Biba lattice of Figure 4c, obtained by inverting Figure 4b. In short, the mandatory controls expressed by the three formulations of Figure 4 are precisely equivalent. Which mandatory controls a system enforces doesn't matter — the net effect is identical.

Lipner[11] gave us another example of the simultaneous use of confidentiality and integrity labels. Lipner constructed a composite lattice for possible application in a conventional data processing environment. The Lipner lattice consists of three integrity levels, two integrity categories, two confidentiality levels, and three confidentiality categories. This results in $3\times2^2\times2\times2^3=192$ distinct labels, all of which could be instantiated as a single BLP lattice, as I have argued. However, Lipner instantiates subjects and objects with only nine distinct labels, which are related by the BLP lattice of Figure 5. In this lattice, system programs have the highest integrity, whereas audit trail has the highest confidentiality. Audit trail can be read only by system managers. It can be written by all subjects (in an append-only mode).

Lipner also imposes the additional constraint that production users can execute only production code. Also, no individual can be both an application programmer and a production user. It is easier to enforce this restriction if each human being is required to have a unique user identity in the system, as suggested in the section on access control models.

Finally, in contradiction to the ★-property, system control subjects are allowed to "write down." Such additional constraints and relaxations of the lattice model appear to be necessary for the application that Lipner considered. The point is not so much to discuss the adequacy of the lattice model for integrity applications but to emphasize that lattice-based information flow policies that combine several lattices can be cast within a single lattice.

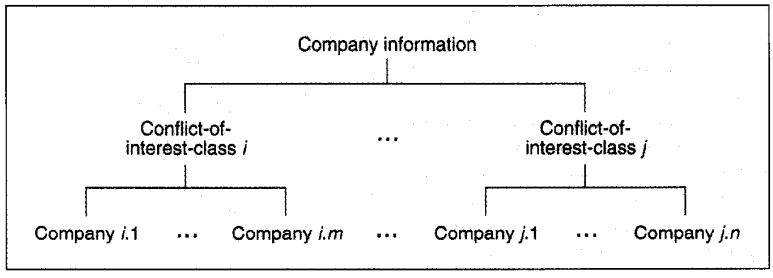

Figure 6. Company information in the Chinese Wall policy.

The Chinese Wall lattice

The Chinese Wall policy that Brewer and Nash[12] identified arises in the segment of the commercial sector that provides consulting services to other companies. In a 1992 paper, I presented a lattice-based access control model for enforcing this policy.[13] The objective of the policy is to prevent information flows that result in a conflict of interest for individual consultants.

Consultants deal with confidential company information for their clients. But, a consultant should not have access to information about, say, two banks or two oil companies because such information creates a conflict of interest in the consultant's analysis and is a disservice to clients. Insider information about two similar types of companies also presents the potential for consultants to use such knowledge for personal profit.

The Chinese Wall policy has a dynamic aspect to it. New consultants start with no mandatory restriction on their access rights. Say a consultant accesses information about bank A. Thereafter, this consultant is mandatorily denied access to information about any other bank. (This denial should persist long enough to avoid a conflict of interest. To simplify this discussion, assume that this denial is forever.) However, there are still no mandatory restrictions regarding the consultant's access to an oil company, an insurance company, and so forth.

It is useful to distinguish *public* information from *company* information. Public information involves desirable features such as public bulletin boards, electronic mail, and public databases; has no mandatory reading controls; and can have discretionary access controls restricting who can read different public items. (For simplicity's sake in this article, I ignore discretionary and additional mandatory controls, which coexist with the Chinese Wall policy. Such additional controls can also apply to company information and are similarly ignored.)

As Figure 6 shows, company information is categorized into mutually disjoint conflict-of-interest classes. Each company belongs to one conflict-of-interest class. The Chinese Wall policy requires that a consultant not be able to read information for more than one company in any given conflict-of-interest class. This policy applies uniformly to users and subjects.

The policy for writing public or company information is derived from its consequence on providing possible indirect read access contrary to mandatory read controls. In this respect, users and subjects (possibly infected with Trojan horses) must be treated differently. The policy for writing is essentially the same as the Bell-LaPadula ★-property. To make this statement meaningful, we need to define a lattice of labels.

Say there are n conflict-of-interest classes: $COI_1, COI_2, \ldots, COI_n$ each with m_i companies, so that $COI_i = \{1, 2, \ldots,$

$m_i\}$, for $i = 1, 2, \ldots, n$. We propose to label each object in the system with the companies from which it contains information. Thus, an object that contains information from bank A and oil company OC is labeled {bank A, oil company OC}. Assume that banks and oil companies are distinct conflict-of-interest classes. Then, labels such as {bank A, bank B, oil company OC} are clearly contrary to the Chinese Wall policy. We prohibit such labels in our model by defining a security label as an n-element vector $[i_1, i_2, \ldots, i_n]$, where each $i_k \in COI_k$ or $i_k = \perp$ for $k = 1 \ldots n$. (The symbol $\perp$ is read as null.) An object labeled $[i_1, i_2, \ldots, i_n]$ is interpreted as (possibly) containing information from company i_1 of COI_1, company i_2 of COI_2, and so on. When an element of the label is $\perp$ rather than an integer, the object cannot have information from any company in the corresponding COI class. For example, an object that contains information from company 7 in COI_2 and company 5 in COI_4 is labeled $[\perp, 7, \perp, 5, \perp, \ldots, \perp]$.

The dominance relation among labels is defined as follows: $l_1 \geq l_2$ provided l_1 and l_2 agree wherever $l_2 \neq \perp$. For example, $[1, 3, 2] \geq [1, 3, \perp]$, $[1, 3, 1] \geq [\perp, \perp, 1]$ while $[1, 3, 2]$ and $[1, 2, 3]$ are incomparable. To be precise, let $l[i_k]$ denote the i_k-th element of label l. Then $l_1 \geq l_2$ if $l_1[i_k] = l_2[i_k] \vee l_2[i_k] = \perp$, for all $k = 1 \ldots n$.

The label with all null elements naturally corresponds to public information. There is, however, no naturally occurring system-high label. In fact, such a label is contrary to the Chinese Wall policy. We can introduce a special label Syshigh for this purpose, not assigning that name to any subject in the system. By definition, Syshigh dominates all other labels. (Alternately, we can recognize that the Chinese Wall policy does not quite fit within a lattice and requires the Syshigh class to be eliminated. However, recall that when I discussed Example 3 above, I said the need for system-high objects for system administration and audit purposes could be crucial.)

The class-combining join operator must be defined to complete the lattice structure. We say that two labels l_1 and l_2 are compatible if wherever they disagree at least one of them is $\perp$, that is, $l_1[i_k] = l_2[i_k]$ or $l_1[i_k] = \perp$ or $l_2[i_k] = \perp$ for all $k = 1 \ldots n$. Note that if l_1 dominates l_2 then l_1 and l_2 are compatible. In other words, comparable labels are compatible.

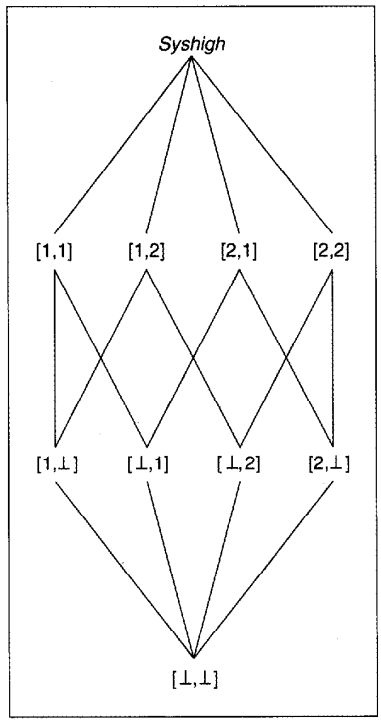

Figure 7. Example of a Chinese Wall lattice.

Incomparable labels, on the other hand, might or might not be compatible. For instance, $[1,3,2]$ and $[1,2,3]$ are incompatible, while $[1,\perp,2]$ and $[1,2,\perp]$ are compatible. Incompatible labels cannot be legitimately combined under the Chinese Wall policy. This is expressed by the requirement that if l_1 is incompatible with l_2 then $l_1 \oplus l_2$ = Syshigh. For compatible labels the i_k-th element of the join is computed as follows: $(l_1 \oplus l_2)[i_k]$ = **if** $l_1[i_k] \neq \perp$ **then** $l_1[i_k]$ **else** $l_2[i_k]$. For example, $[1, \perp, 2] \oplus [1, 2, \perp] = [1, 2, 2]$. Finally, the join of any label with Syshigh is Syshigh.

Given this lattice structure, here's how the Chinese Wall policy can be enforced. I describe the solution in the context of the specific lattice of Figure 7, which contains two conflict-of-interest classes with two companies in each class. The solution is completely general, however, and applies to any Chinese Wall lattice. Every object in the system bears one of the labels in Figure 7. (I discussed the interpretation of these labels previously.) Objects labeled Syshigh violate the Chinese Wall policy by combining information from more than one company in the same COI class. These objects are inaccessible in the system, since no user will be cleared to Syshigh.

Next, consider labels on users and subjects. We treat the clearance of a user as a high-water mark that can float up in the lattice but not down. A newly enrolled user in the system is assigned the clearance $[\perp, \perp]$. (This assumes that the user is entering the system with a "clean slate." A user with prior exposure to company information in some other system should enter with an appropriate clearance reflecting the extent of the prior exposure.) As the user reads various company information, the user's clearance floats up in the lattice. For example, by reading information about company 1 in conflict-of-interest-class 1, the user's clearance is modified to $[1, \perp]$. Reading information about company 2 in conflict-of-interest-class 2 further modifies the user's clearance to [1, 2]. This floating up of a user's clearance is allowed as long as the clearance does not float up to Syshigh. Operations that would force the user's clearance to Syshigh are thereby prohibited.

The ability to float a user's clearance upward addresses the dynamic requirement of the Chinese Wall policy. The floating clearance keeps track of a user's read operations in the system. It is also important to ensure that a consultant cannot be known by two (or more) user identities in the system. Otherwise, each user identity could obtain information about different companies in the same conflict-of-interest class.

The exact manner in which a user's clearance is allowed to float up is not specified in the model, since there are numerous alternatives. If users have complete freedom in this respect, the proposed read access could be specified at login. The system could then create a suitable subject for that user session.

On the other hand, one might constrain this by discretionary access controls. For instance, a user might be allowed to read only company information that the user's boss assigns. In this case, the float-up of a user's clearance is effected by some other user. (A complete treatment would require models such as the typed access matrix.[3])

With each user we associate a set of subjects whose labels are dominated by the user's clearance. Thus, if user Jane has the clearance $[1, 1]$, she could create the following subjects associated with

her: Jane.[1, 1], Jane.[1, ⊥], Jane.[⊥, 1], and Jane.[⊥, ⊥]. Each of the subjects corresponds to the label with which she wishes to log in on a given session. (More generally, a user might be allowed to open several windows in a single login session, with each window associated with its own subject.) Each subject has a fixed label that does not change.

The floating up of a user's clearance corresponds with the ability to create subjects with new labels for that user. For example, when Jane has the clearance [1, ⊥], she can create subjects with labels [1, ⊥] and [⊥, ⊥]. When Jane's clearance floats up to [1, 1], she acquires the ability to create subjects with labels [1, 1] and [⊥, 1].

Each subject has a fixed label, and each subject created by that subject inherits that label; that is, subject creation is allowed only at the label of the creating subject. A subject's label remains fixed for the life of that subject.

All read and write operations in the system are carried out by subjects. These subjects are constrained by the familiar simple-security and ★-properties of the Bell-LaPadula model. Suppose Jane logs in as a subject with label [1, ⊥]. All subjects created during that session will inherit the label [1, ⊥]. This will allow these subjects to read public objects labeled [⊥, ⊥], to read and write company objects labeled [1, ⊥], and to write objects with labels [1, 1], [1, 2], and Syshigh.

Although lattice-based access controls were initially developed for military sector, they can be applied in almost any situation where information flow is of concern. The commercial sector has largely ignored lattice-based access controls, possibly due to their genesis in confidentiality policies for the military and government. However, as I have argued in this article, lattice-based controls are relevant to integrity policies in commercial data processing, as well as for confidentiality policies that are unique to the commercial sector.

Lattice-based access control is a key ingredient of information systems security as we understand it today. At the same time, the lattice-based approach does not provide a complete solution for information flow policies, let alone for security policies in general. Albeit a very important one, the lattice-based approach is but one ingredient of information systems security. ■

Acknowledgments

I thank the anonymous referees for identifying several subtle errors, omissions, and ambiguities, thereby helping me to significantly improve the final draft of this article. This work was partially supported by National Science Foundation Grant No. CCR-9202270 and National Security Agency Contract No. MDA904-92-C-5141. I am grateful to Dan Atkinson, Nathaniel Macon, Howard Stainer, and Mike Ware for making this work possible.

References

1. D.E. Denning, "A Lattice Model of Secure Information Flow," *Comm. ACM*, Vol. 19, No. 5, May 1976, pp. 236-243.
2. W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *Proc. Eighth NBS-DOD Nat'l Computer Security Conf.*, US Govt. Printing Office, Washington, D.C., 1985, pp. 18-27.
3. R.S. Sandhu, "The Typed Access Matrix Model," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 2825, 1992, pp. 122-136.
4. G.W. Smith, *The Modeling and Representation of Security Semantics for Database Applications*, PhD thesis, George Mason Univ., Fairfax, Va., 1990.
5. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," Mitre Corp. Report No. M74-244, Bedford, Mass., 1975. (Also available through Nat'l Technical Information Service, Springfield, Va., Report No. NTIS AD-771543.)
6. B.W. Lampson, "Protection," *Fifth Princeton Symp. Information Science and Systems*, Princeton Univ., Princeton, N.J., 1971, pp. 437-443. Reprinted in *ACM Operating Systems Rev.*, Vol. 8, No. 1, Jan. 1974, pp. 18-24.
7. B.W. Lampson, "A Note on the Confinement Problem," *Comm. ACM*, Vol. 16, No. 10, Oct. 1973, pp. 613-615.
8. N.E. Proctor and P.G. Neumann, "Architectural Implications of Covert Channels," *Proc. 15th NIST-NCSC Nat'l Computer Security Conf.*, US Govt. Printing Office, Washington, D.C., 1992, pp. 28-43.
9. J.A. Gougen and J. Meseguer, "Security Policies and Security Models," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 410 (microfiche only), 1982, pp. 11-20.
10. K.J. Biba, "Integrity Considerations for Secure Computer Systems," Mitre Corp. Report TR-3153, Bedford, Mass., 1977. (Also available through Nat'l Technical Information Service, Springfield, Va., Report No. NTIS AD-A039324.)
11. S.B. Lipner, "Nondiscretionary Controls for Commercial Applications," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 410 (microfiche only), 1982, pp. 2-10.
12. D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 1939 (microfiche only), 1989, pp. 215-228.
13. R.S. Sandhu, "A Lattice Interpretation of the Chinese Wall Policy," *Proc. 15th NIST-NCSC Nat'l Computer Security Conf.*, US Govt. Printing Office, Washington, D.C., 1992, pp. 329-339.

Ravi S. Sandhu is associate chair of the Information and Software Systems Engineering Department at George Mason University, where he teaches several graduate-level security courses. His research interest is information systems security, particularly regarding database management systems, distributed systems, and formal models. He has published more than 60 papers on computer security.

Sandhu received a BTech degree in electrical engineering from the Indian Institute of Technology in Bombay, an MTech degree in computer technology from IIT in Delhi, and MS and PhD degrees in computer science from Rutgers University. He is a senior member of IEEE and a member of the IEEE Computer Society.

Readers can contact Sandhu at the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030, electronic mail sandhu@isse.gmu.edu or sandhu@gmu.edu.