IEEE Computer, Volume 26, Number 11, November 1993, pages 9-19 (Cover article).

Lattice-Based Access Control Models[1]

Ravi S. Sandhu

Center for Secure Information Systems
&
Department of Information and
Software Systems Engineering
George Mason University
Fairfax, Virginia 22030

email: sandhu@sitevax.gmu.edu
voice: 703-993-1659
fax: 703-993-1638

Abstract The objective of this article is to give a tutorial on lattice-based access control models for computer security. The paper begins with a review of Denning's axioms for information flow policies, which provide a theoretical foundation for these models. The structure of security labels in the military and government sectors, and the resulting lattice is discussed. This is followed by a review of the Bell-LaPadula model, which enforces information flow policies by means of its simple-security and $\star$-properties. It is noted that information flow through covert channels is beyond the scope of such access controls. Variations of the Bell-LaPadula model are considered. The paper next discusses the Biba integrity model, examining its relationship to the Bell-LaPadula model. The paper then reviews the Chinese Wall policy, which arises in a segment of the commercial sector. It is shown how this policy can be enforced in a lattice framework.

Keywords: security, access control, lattice model, confidentiality, integrity

[1]This work was partially supported by National Science Foundation grant CCR-9202270 and National Security Agency contract MDA904-92-C-5141. The paper has been accepted for publication in *IEEE Computer*, and is provided here for early dissemination of its contents.

1 INTRODUCTION

The need for information security was recognized with the advent of the first multi-user computer systems. This need has become more and more significant, as computer systems have evolved from isolated mainframes behind guarded doors to interconnected and decentralized open systems.

The objective of information security can be divided into three separate, but interrelated, areas as follows: *confidentiality* (or *secrecy*) is concerned with disclosure of information, *integrity* is concerned with modification of information, and *availability* is concerned with denial of access to information. These three objectives arise in practically every information system. For example, in a payroll system confidentiality is concerned with preventing an employee from finding out the boss's salary; integrity is concerned with preventing an employee from changing his or her salary; and availability is concerned with ensuring that the paychecks are printed on time. Similarly, in a military command and control system confidentiality is concerned with preventing the enemy from determining the target coordinates of a missile; integrity is concerned with preventing the enemy from altering the target coordinates; and availability is concerned with ensuring that the missile does get launched when the order is given.

Lattice-based access control models were developed to deal with information flow in computer systems. Information flow is clearly central to confidentiality. As we will see it also applies to integrity to some extent. Its relationship to availability is tenuous at best. Thus, these models are primarily concerned with confidentiality and can deal with some aspects of integrity.

The basic work in this area was done in the 1970s. Since then these models have been implemented in a number of systems, mostly driven by the needs of the Defense sector. The theory and concepts are, however, applicable to almost any situation in which information flow is a concern. We will see there are policies unique to the Commercial sector which have such concerns.

Lattice-based access control is one of the essential ingredients of computer security as we understand it today. We discuss a number of models developed in this context, and examine their underlying theoretical and conceptual foundations.

2 INFORMATION FLOW POLICIES

Information flow policies are concerned with flow of information from one security class to another. In a system, information actually flows from one *object* to another. The models we will be discussing, 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 typically controlled by assigning every object a *security class*, also called the object's *security label*. Whenever information flows from object a to object b, there is also information flow from the security class of a to the security class of b. Henceforth, when we speak of information flow from security class X to security class Y, the reader can

visualize this as information flow from an object labeled X to an object labeled Y.

The concept of an information flow policy was formally defined by Denning [5] as follows.

Definition 1 [Information Flow Policy] An *information flow policy* is a triple $< \mathcal{SC}, \rightarrow, \oplus >$ where $\mathcal{SC}$ is a set of *security classes*, $\rightarrow \subseteq \mathcal{SC} \times \mathcal{SC}$ is a binary *can-flow relation* on $\mathcal{SC}$, and $\oplus : \mathcal{SC} \times \mathcal{SC} \rightarrow \mathcal{SC}$ is a binary *class-combining* or *join operator* on $\mathcal{SC}$. □

It is understood that all three components of an information flow policy are fixed, and do not change with time. Note that this definition allows objects to be created and destroyed dynamically (as one would expect to do 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 $(A, B) \in \rightarrow$, i.e., information can flow from A to B. We also write $A \not\rightarrow B$ to mean $(A, B) \notin \rightarrow$, i.e., 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. We have chosen to retain the original terminology of [5].)

Similarly, infix notation will be used for the join operator, i.e., $A \oplus B = C$ 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 which 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] $\mathcal{SC} = \{A_1, \ldots, A_n\}$; 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 example of a non-trivial information flow policy occurs when there are just two security classes, called H (for high) and L (for low), with all flows allowed excepting that from high to low. In other words high information is more sensitive than low information. A formal statement is given below.

Example 2 [High-Low Policy] $\mathcal{SC} = \{H, L\}$, 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 of figure 1(a). The can-flow relation is understood to be directed upwards in this figure. The reflexive flows from H to H and L to L are implied but not explicitly shown. The other Hasse diagrams in figure 1 also represent information flow policies which will be discussed in this paper. In these diagrams transitive edges, such as from L to H in figure 1(b), are implied but not explicitly shown.

Denning [5] showed that under certain assumptions, given below, an information flow

policy forms a finite lattice.

Definition 2 [Denning's Axioms]

1. The set of security classes $\mathcal{SC}$ is finite.
2. The can-flow relation $\rightarrow$ is a partial order on $\mathcal{SC}$.
3. $\mathcal{SC}$ 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 [5] 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. We will see how example 1 can be extended to form a lattice. We note that, although our focus in this paper is on policies which satisfy Denning's axioms, we will see there are legitimate information flow policies which do not satisfy these axioms.

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

Denning's second axiom states that $\rightarrow$ is a partial order on $\mathcal{SC}$. A partial order is a reflexive, transitive and anti-symmetric binary relation. We have already seen the need for reflexivity in context of example 1, whereby $A \rightarrow A$ for all $A \in \mathcal{SC}$. Transitivity requires that if $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$, i.e., 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.[1] Finally, anti-symmetry requires that if $A \rightarrow B$ and $B \rightarrow A$ then $A = B$. Given the reflexive and transitive requirements, anti-symmetry merely eliminates redundant security classes. In other words, there is no point having two different security labels if objects with these labels are restricted to having exactly the same information flows.

Denning's third axiom requires that $\mathcal{SC}$ has a lower bound L (for system low), i.e., $L \rightarrow A$ for all $A \in \mathcal{SC}$. 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; so constants should be labeled L. A concrete example of such a constant would be version information about the operating system. Version information is necessary

[1] 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 (high) to L (low) but only if mediated by a sanitizing process with security class SAN. We then have H ! SAN, SAN ! L, but H 6! L. These situations are typically handled as exceptions falling outside the normal lattice framework of information flow. It should be noted that such non-transitive information flows can be enforced using the concepts of *type enforcement* and *assured pipelines* discussed in [3]. Non-transitive information flow policies can also be expressed in the type-based access control model of [11].

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 is the most subtle. There are actually two parts to it. Firstly the join operator is required to be totally defined, i.e., $A \oplus B$ is defined for every pair of security classes from $\mathcal{SC}$. This means that it is possible to combine information from any two security classes and give the result a label. In example 1 we saw a situation in which this property was not satisfied, i.e., $A_i \oplus A_j$ 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) and define $A_i \oplus A_j = H$ for $i \neq j$. By introducing L and H example 1 is modified as follows.

Example 3 [Bounded Isolated Classes] $\mathcal{SC} = \{A_1, \ldots, A_n, L, H\}$; $L \rightarrow L$, $L \rightarrow H$, $H \rightarrow H$, 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 $A_i \oplus A_j = H$. □

This policy is depicted by the Hasse diagram of figure 1(b). The can-flow relation goes upwards 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.) There is a practical role for system-high objects in that information about the global state of the system can only go in objects labeled H, and such information could be crucial for proper system administration and audit. On the other hand, this example also suggests that in some situations it may be more appropriate to use partially-ordered labels, rather than striving 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 $A, B, C \in \mathcal{SC}$ we have (i) $A \rightarrow A \oplus B$ and $B \rightarrow A \oplus B$, and (ii) if $A \rightarrow C$ and $B \rightarrow C$ then $A \oplus B \rightarrow C$. Property (i) follows from the intuition underlying the join operator, i.e., $A \oplus B$ 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 (ii) 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, we 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.

3 EXAMPLES

The simplest examples of non-trivial information flow policies occur when the can-flow relation is a total or linear ordering of the security classes. The most common example 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. This policy is depicted in figure 1(c). In general we can have any number of totally ordered security classes. (A total or linear ordering is often called a hierarchical ordering in the security literature. In this paper we avoid using the term "hierarchical ordering", since it is sometimes understood to mean a tree-like ordering.)

Note that $\geq$ is a total ordering if and only if its inverse $\rightarrow$ is a total ordering. Moreover, in a total ordering there are no incomparable security classes. 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 result is the higher one of the two classes being combined, e.g., $S \oplus U = S$.

Figure 1(d) depicts a partially ordered lattice. The security classes are obtained as the power-set (i.e., set of all subsets) of $\{A, B\}$. To be concrete, say that 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 and medical information is 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 (i.e., A and B) are known as *categories*; while the security classes (i.e., sets of categories) are known as *compartments*.

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

We can similarly define subset lattices of any size. 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 vanishingly small fraction of the security classes would be actually employed for large n.

It should be noted that selecting an arbitrary subset of a lattice will not necessarily yield a lattice. For example, the partial order of figure 2(a) results by selecting these four security classes from the subset lattice on $\{A,\ B,\ C,\ D\}$. The partial order of figure 2(a) fails to be a lattice for two reasons. Firstly, it is missing the system low and system high security classes. Secondly, the two upper bounds of $\{A\}$ and $\{B\}$ are incomparable, and therefore 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 1(a) to obtain the lattice of figure 1(b). It is always possible to do such a construction for any partial order, i.e., every partial order can be embedded in a lattice by including additional security classes, if so desired.

The two lattices we have considered above, i.e., the totally ordered lattice and the subset lattice, are often combined. This is particularly so in the military and government sectors, where this structure is laid down by law. (Note that the system high security clearance is sometimes not given to any individual in systems which handle the most sensitive information.) Each security class has two components: one from the totally ordered security lattice of figure 1, 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, e.g., $<TS, \{A\}> \oplus <S, \{B\}> = <TS, \{A, B\}>$. 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 earlier, in such cases realistically only a small subset of the entire lattice would be used. A concrete illustration of this was given by Smith [13], who describes a actual lattice based on common practice in the military. This security classes consist of the four linearly ordered security levels $TS > S > C > U$, and 8 categories $\{A, K, L, Q, W, X, Y, Z\}$ corresponding to say 8 different projects in the system. Smith's lattice is shown in figure 3. It has 21 labels from a possible space of $4 \times 2^8 = 1024$ labels. It is interesting that the 21 labels which are actually used do constitute a lattice. Note that, except for the system high security class, combinations of categories occur only in two's and three's. Also the use of categories occurs mostly above top-secret (TS), and none occurs below secret (S).

4 ACCESS CONTROL MODELS

So far we have defined what an information flow policy is, and have seen that Denning's axioms imply that we have a lattice of security labels. We now consider how to enforce an information flow policy in a system. 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. There is a potential for information 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. In other words we must be conservative, and ensure that programs simply do not have the ability to cause information flows contrary to the given policy. We will shortly see how the Bell-LaPadula model [1] addresses this objective. First we introduce some basic abstractions for access control models.

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

We understand a user to be a human being. We assume that each human being known to the system is recognized as a unique user. In other words the unique human being 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 we consider in this paper. At the same time, it is crucial for policies such as Lipner's integrity lattice and Chinese Walls, discussed later in the paper. It is also something that 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 in order to maintain strict accountability. The use of shared user identifiers to facilitate sharing is usually resorted to 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, i.e., a subject is a program in execution. Each subject is associated with a single user on behalf of whom the subject executes. In general a user may have many subjects concurrently running on the user's behalf in the system. Every time a user logs in to the system it is 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 after that.) Different subjects associated with the same user may obtain different sets of access rights. Consider the statement that the top-secret user John logs in at the secret level. In the user-subject terminology we interpret this statement as follows: firstly, there is a unique user John, cleared to top-secret; secondly, John can have subjects executing on his behalf at every level dominated by top-secret. For now, let us assume that each subject runs at a fixed security level. (We will see later that the security level of subjects can be changed in some models.)

To summarize: each authorized human user is known as a unique user to the system who can have several subjects executing on the user's behalf, but each subject is associated with only one user. In general, subjects can create other subjects in the system. Nevertheless, each subject has a unique user as an ancestor, on behalf of whom the subject executes its operations.

The access rights of subjects to objects in a system are conceptually represented by an *access matrix* [7]. The matrix has a row for every subject and a column for every object. A subject can also be an object in the system, e.g., a process may have suspend and resume operations executed on it by some other process. In general, subjects are viewed as a subset of the objects. 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. Only those operations which are authorized by the access matrix can be executed. For purpose of the access matrix, every user is also regarded as a subject in its own right. This 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 some other data structure suitable for efficient storage of a sparse matrix.

The access matrix is a dynamic entity. The individual cells in the access matrix can be modified by subjects. For example, if subject s is the owner of object o (i.e., $own \in [s, o]$)

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 by other subjects to the owned object. Such access controls are said to be *discretionary*. The access matrix also changes due to addition and deletion of subjects and objects. (The typical access control on files and directories, provided by popular multi-user operating systems, on the basis of protection bits, is an example of discretionary access control. Similarly, the access control on relations and parts of relations provided by popular relational database management systems, is also an example of discretionary access control.)

Discretionary access controls are by themselves inadequate to enforce information flow policies. The basic problem with discretionary access controls is that there is no constraint on copying information from one object to another. (There are other more subtle problems with discretionary access controls, notably concerning the so-called safety problem for propagation of access rights. For a discussion of the safety problem see [11].)

To be concrete say that Tom, Dick and Harry are users. Suppose Tom has a confidential file PRIVATE, which he desires to be read by Dick but not by Harry. Tom can authorize Dick to read this 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.) Tom's intention can be easily subverted by Dick. Dick creates a new file, call it COPY-OF-PRIVATE, into which he copies the contents of PRIVATE. As the creator of COPY-OF-PRIVATE, Dick has the authority to grant read access for it to any user, including Harry, i.e., Dick can enter *read* in [Harry,COPY-OF-PRIVATE]. Then, for all practical purposes, Harry can read PRIVATE, so long as Dick keeps COPY-OF-PRIVATE reasonably up to date with respect to PRIVATE.

The situation is actually worse than the above scenario indicates. So far, we have portrayed Dick as a cooperative participant in this process. Now suppose that Dick is a trusted confidant of Tom, and would not deliberately subvert Tom's intentions regarding the PRIVATE file. However, Dick uses a fancy text editor supplied to him by Harry. This editor provides all the editing services that Dick needs. In addition Harry has also programmed it to create the COPY-OF-PRIVATE file, and to grant Harry the right to read COPY-OF-PRIVATE. Such software is said to be a *Trojan Horse*, because in addition to the normal functions expected by its user it also engages in surreptitious actions to subvert security. Note that a similar Trojan Horse executed by Tom could actually grant Harry the privilege to directly read PRIVATE.

In summary, even if the users are trusted not to deliberately breach security we have to contend with Trojan Horses which have been programmed to deliberately do so. We can require that all software that is run on the system is free of Trojan Horses. But this is hardly a practical option, particularly if we wish to guarantee this freedom with a high degree of assurance. The solution is to impose mandatory controls which cannot be bypassed, even by Trojan Horses.

5 THE BELL-LAPADULA MODEL

The concept of mandatory access controls was first formalized by Bell and LaPadula [1]. They defined a model, commonly called the Bell-LaPadula model. Since its original publication numerous variations of this model have been published. Consequently it has become somewhat unclear, exactly what one means by the Bell-LaPadula model.

In this paper we will take a minimalist approach, and define a model, called BLP, which is roughly speaking the "smallest" model which captures the essential access control properties we wish to illustrate. The reader should note that the notation and precise formulation of the rules of BLP are substantially different from the original Bell-LaPadula model [1]. BLP is more in line with the formulations used by authors in the more recent literature.

The key idea in BLP is to augment discretionary access controls with *mandatory access controls*, so as to enforce information flow policies. BLP takes a two step approach to access control. First there is a discretionary access matrix D, the contents of which can be modified by subjects (in some manner which we do not need to specify). However, authorization in D is not sufficient for an operation to be carried out. In addition, the operation must also be authorized by the mandatory access control policy, over which users have no control.

The 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*. It is important to understand that a Secret user may run the same program, such as a text editor, as a Secret subject or as an Unclassified subject. 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 the security labels on subjects and objects, once assigned, cannot be changed (except by the security officer). This last assumption, that security labels do not change, is known as *tranquility*. We will see later how this assumption can be relaxed in a secure manner.

The specific mandatory access rules given in BLP are as follows, where λ signifies the security label of the indicated subject or object.

- *Simple-Security Property*: Subject s can read object o only if $\lambda(s) \geq \lambda(o)$.
- $\star$*-Property*: Subject s can write object o only if $\lambda(s) \leq \lambda(o)$. (The $\star$-property is pronounced as the star-property.)

Read access implies a flow from object to subject, hence the requirement that $\lambda(s) \geq \lambda(o)$, or equivalently $\lambda(o) \rightarrow \lambda(s)$. Write access conversely implies a flow from subject to object, hence the requirement that $\lambda(s) \leq \lambda(o)$, or equivalently $\lambda(s) \rightarrow \lambda(o)$. Note that 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 the read and write operations. In a real system there will be additional operations, e.g., create and destroy objects. It suffices to consider read and write to illustrate the main points. For example, create and destroy are also constrained by the $\star$-property because they modify the state of the object in question. Note that the mandatory controls are formulated as "only if" conditions, i.e., 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 (or vice versa). If D does not authorize the operation, we do not need to check the mandatory controls since the operation will anyway be rejected (or vice versa).

The simple-security requirement applies equally to humans and programs, and its need is self-evident. The $\star$-property on the other hand 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 it is assumed that he or she will only put Unclassified information in it. Programs, on the other hand, are not trusted because they may have Trojan Horses embedded in them. A program running at the Secret level is therefore not allowed to write to Unclassified objects by the $\star$-property, even if it is permitted to do so by the discretionary access controls. A Secret user who wishes to write an Unclassified object must log in as an Unclassified subject.

A curious aspect of the $\star$-property is that an Unclassified subject can write a Secret file. This means that Secret data may be destroyed or damaged, perhaps accidentally, by Unclassified subjects. To avoid this integrity problem a modified $\star$-property is sometimes used, which requires $\lambda(s) = \lambda(o)$, i.e., subjects can write at their own level but cannot "write up."

Let us see how these properties impact on the Tom, Dick and Harry Trojan Horse example discussed above. Suppose that Tom and Dick are Secret users, whereas Harry is an Unclassified user. Tom and Dick can have Secret and Unclassified subjects, while Harry can only have Unclassified subjects. 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 $\star$-properties will ensure that Harry's subjects cannot surreptiously 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 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. Note that the 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 which is able to 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 denial of 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.

Such indirect methods of communication are called *covert channels* [8]. Covert channels present a formidable problem for enforcement of information flow policies. They are difficult

to detect, and once detected are difficult to close without incurring significant performance penalties. Covert channels do 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 employed by the Trojan Horses to achieve error-free communication. The resulting data rates can be as high as several million bits per second, if efforts have not been made to mitigate these channels.

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 which are explicitly intended for inter-process 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 [10], for example). It should be mentioned that there is another class of models, called *information flow models*, which attempt to deal with all information flows in an uniform manner [6]. Discussion of these models is beyond the scope of this paper.

Let us now revisit the tranquility assumption, which requires that security labels of subjects and objects do not change. This assumption can be relaxed in several different ways, some of which are secure while others are insecure (i.e., 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 $\lambda'(o) > \lambda(o)$, e.g., 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, i.e., we replace the stipulation that $\lambda(s) = \lambda(o)$ by $\lambda(s) > \lambda(o)$. This latter case is insecure because upgrade of 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 (which could be exploited by Trojan Horses). Note that a Secret user can securely upgrade an Unclassified file to Secret, by logging in as an Unclassified subject in order to make this change.

6 THE BIBA MODEL AND DUALITY

The mandatory controls in the Bell-LaPadula model were motivated by confidentiality considerations. Biba proposed that similar controls could be formulated for purpose of integrity [2]. 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 different ways that mandatory controls could be used for integrity objectives. We limit our discussion to the best known of these, called *strict integrity*.

In the usual formulation of the Biba model, high integrity is placed towards 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. This is in opposite direction to the permitted flow of information in the Bell-LaPadula model (or in Denning's axioms as stated earlier). This fact led Biba to propose the following mandatory controls, in analogy with the mandatory controls of the BLP model. Here 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 $\star$-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.

Now there is nothing intrinsic about placing high integrity at the top of the lattice (or for that matter placing high confidentiality at the top). After all, top and bottom are relative terms coined for convenience, and have no absolute significance. But then information flow in the Biba model can be brought into line with the BLP model, by the simple expedient of saying that low integrity is at the top of the lattice and high integrity at the bottom. This requires us to invert 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. Note that the situation here is a symmetrical one. 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.

The point of the above discussion is that there is no fundamental difference between the Biba and BLP models. Both models 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, whereas the Biba model allows it downward. Since direction is relative, a system which can enforce one of these models can also enforce 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 can be combined in situations where both confidentiality and integrity are of concern. If a single label is used for both confidentiality and integrity, then the two models impose conflicting constraints. The consequence of combining them is that a subject can read or write only those objects which have exactly the same security label as the subject. This amounts to the trivial policy of no information flows between security classes discussed in example 1. A more useful situation is one in which there are independent confidentiality and integrity labels. In other words 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. To be precise let $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ be a lattice of confidentiality labels, and $\Omega = \{\omega_1, \ldots, \omega_q\}$ 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 as follows.

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

We call this the *composite model*. It is a popular model and has been implemented in several operating system, database and network products (specifically built to meet requirements of the military sector). Now the composite model amounts to the simultaneous

application of two lattices, in which information flow occurs in opposite directions (going upwards in the confidentiality lattice and downwards in the integrity lattice). However, by our discussion above we can invert the integrity lattice, and view the composite model as the simultaneous application of two lattices with information flow going upwards in both of them. But then we have a product of two lattices, which is itself mathematically a lattice. In other words the composite model can be reduced to a single lattice.

We illustrate this general result in figure 4. Figure 4(a) shows two lattices Λ and Ω to which BLP and Biba controls are respectively applied. The resulting mandatory controls are stated in the accompanying table. Each entry specifies the maximum access that a subject with label on the row can have to an object with label on the column. Exactly the same mandatory controls are enforced by the BLP lattice of figure 4(b). In this lattice subjects labeled $\lambda_{\mathrm{H}}\omega_{\mathrm{H}}$ cannot read objects labeled $\lambda_{\mathrm{H}}\omega_{\mathrm{L}}$ because of the ω-component of their labels. At the same time they cannot write objects labeled $\lambda_{\mathrm{L}}\omega_{\mathrm{H}}$ because of the λ-component. Finally they cannot read objects labeled $\lambda_{\mathrm{L}}\omega_{\mathrm{L}}$ because of the ω-component, and cannot write to these objects because of the λ-component. Other relationships in this lattice can be similarly interpreted. 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. Now exactly the same mandatory controls are also enforced by the Biba lattice of figure 4(c) obtained by inverting figure 4(b). In short the mandatory controls expressed by the three formulations of figure 4 are precisely equivalent, and it does not matter which one a system enforces. The net effect is identical.

Another example of the simultaneous use of confidentiality and integrity labels is from Lipner [9], who constructed a composite lattice for possible application in a conventional data processing environment. Lipner's lattice consists of 3 integrity levels, 2 integrity categories, 2 confidentiality levels and 3 confidentiality categories. This results in a total of $3 \times 2^2 \times 2 \times 2^3 = 192$ distinct labels, all of which could be instantiated as a single BLP lattice as we have argued. However, Lipner instantiates subjects and objects with only 9 distinct labels, which turn out to be related by the BLP lattice of figure 5. Note that, in this lattice, System Programs have the highest integrity whereas the Audit Trail has highest confidentiality. The 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 only execute Production Code. It is further required that 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 section 4. Finally, System Control subjects are allowed to "write down" in contradiction to the $\star$-property. Such additional constraints and relaxations of the lattice model appear to be necessary for the application considered by Lipner. The point here is not so much to discuss the adequacy of the lattice model for integrity applications, but rather to emphasize that lattice-based information flow policies which combine several lattices can be cast within a single lattice.

7 THE CHINESE WALL LATTICE

The Chinese Wall policy was identified by Brewer and Nash [4]. It arises in the segment of the commercial sector which provides consulting services to other companies. A lattice-based access control model for enforcing this policy was given by Sandhu [12]. The objective of the Chinese Wall policy is to prevent information flows which cause conflict of interest for individual consultants. Consultants naturally have to deal with confidential company information for their clients. An individual consultant should not have access to information about two banks or information about two oil companies, etc., because such information creates conflict of interest in the consultant's analysis and disservice to the clients. Insider information about companies of the same type also presents the potential for consultants to abuse such knowledge for personal profit. The Chinese Wall policy has a dynamic aspect to it. A new consultant starts with no mandatory restriction on his or her access rights. Now say that the consultant accesses information about bank A. Thereafter that consultant is mandatorily denied access to information about any other bank. (This denial should be for a sufficient length of time to avoid conflict of interest. To simplify the discussion we will assume that this denial is forever.) There are, however, still no mandatory restrictions regarding that consultant's access to oil companies, insurance companies, etc.

It is useful to distinguish *public* information from *company* information. Public information allows for desirable features such as public bulletin boards, electronic mail and public databases. There are no mandatory controls on reading public information. There may be discretionary access controls restricting who can read different public items. For simplicity, we ignore discretionary controls, as well as additional mandatory controls, which coexist with the Chinese Wall policy. Such additional controls can also apply to company information, and will be similarly ignored.

Company information is categorized into mutually disjoint conflict of interest classes, as shown in figure 6. Each company belongs to exactly one conflict of interest (COI) class. The Chinese Wall policy requires that a consultant should not be able to read information for more than one company in any given COI 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 the mandatory read controls. It is in this respect that users and subjects (possibly infected with Trojan Horses) must be treated differently. The policy for writing is essentially the same as the Bell-LaPadula $\star$-property. To make this statement meaningful we need to define a lattice of labels.

Let there be n conflict of interest classes: COI_1, COI_2, ..., 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 which 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, it means the object cannot have information from any company in the corresponding COI class. For example, an object which contains information from company 7 in COI_2 and company 5 in COI_4 is labeled $[\perp, 7, \perp, 5, \perp, \ldots, \perp]$.

We define the dominance relation among labels 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 which has 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, which we will not assign 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 our remark, in context of example 3, regarding the need for system high objects for system administration and audit purposes.)

To complete the lattice structure it remains to define the class-combining join operator. We say that two labels l_1 and l_2 are *compatible* if wherever they disagree at least one of them is $\perp$, i.e., $l_1[i_k] = l_2[i_k]$ 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. Incomparable labels, on the other hand, may or may 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, let us see how we can enforce the Chinese Wall policy. To be concrete we describe the solution in context of the specific lattice of figure 7, with two conflict of interest classes each with two companies in it. The solution is, however, completely general and applies to any Chinese Wall lattice. Every object in the system is labeled by one of the labels in figure 7. The interpretation of these labels has been discussed earlier. Objects labeled $SYSHIGH$ violate the Chinese Wall policy, in that they combine 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 $SYSHIGH$.

Now let us consider labels on users and subjects. We treat the clearance of a user as a high-water mark which 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 who has had prior exposure to company information in some other system should enter with an appropriate clearance reflecting the extent of this 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, so long as the clearance does not float up to $SYSHIGH$. Operations which would force the user's clearance to $SYSHIGH$ are thereby prohibited. The ability to float a user's clearance upwards addresses the dynamic requirement of the Chinese Wall policy. The floating clearance keeps track of a user's read operations in the system. Also, note that it is important to ensure that a consultant cannot be known by two (or more) user identities in the system. Otherwise, each user identity can 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 the users have complete freedom in this respect, the proposed read access could be specified at the time of 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 may be allowed to read only that company information which the user's boss assigns him or her to. In this case the float up of a user's clearance is effectively done by some other user. Full consideration of such discretionary policies, and their interplay with the mandatory policy is beyond the scope of this paper. A complete treatment would require models such as [11].

With each user we associate a set of subjects, whose labels are dominated by the user's clearance. Thus, if Jane as a user has the clearance $[1, 1]$, she could create the following subjects associated with her: Jane.$[1, 1]$, Jane.$[1, \bot]$, Jane.$[\bot, 1]$ and Jane.$[\bot, \bot]$. Each of these corresponds to the label with which she wishes to login on a given session. (More generally, a user might be allowed to open several windows within a single login session; with each window associated with its own subject.) Each subject has a fixed label which does not change. The floating up of a user's clearance corresponds to the ability to create subjects with new labels for that user. For example, when Jane has the clearance $[1, \bot]$, she can create subjects with labels $[1, \bot]$ and $[\bot, \bot]$. When Jane's clearance floats up to $[1, 1]$, she acquires the ability to create subjects with labels $[1, 1]$ and $[\bot, 1]$.

Each subject has a fixed label. Every subject created by that subject inherits that label, i.e., 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 $\star$-properties of the Bell-LaPadula model. Now suppose that Jane logs in as a subject with label $[1, \bot]$. All subjects created during that session will inherit the label $[1, \bot]$. This will allow these subjects to read public objects labeled $[\bot, \bot]$, to read and write company objects labeled $[1, \bot]$, and write objects with labels $[1, 1]$, $[1, 2]$ and $SYSHIGH$.

8 CONCLUSION

In many situations information flow policies lead to a lattice structure of security classes, due to Denning's axioms. The enforcement of these policies in a computer system, which may have Trojan Horse software in it, can be achieved by the mandatory access controls of Bell-LaPadula (modulo the elimination, mitigation or tolerance of covert channels). Although lattice-based access controls were initially developed for the Defense sector, they have ap-

plication in almost any situation where information flow is of concern. Lattice-based access control is certainly an essential ingredient of computer security as we understand it today. At the same time it must be appreciated that the lattice-based approach does not provide a complete solution for information flow policies, let alone for security policies in general. In other words the lattice-based approach is but one ingredient of computer security, albeit a very important one.

Acknowledgment

The author thanks the anonymous referees for their comments which have greatly improved the accuracy of the paper. Several subtle mistakes, omissions and ambiguities were identified by the referees. Their identification and correction has resulted in a significantly better product. This work was partially supported by National Science Foundation grant CCR-9202270 and National Security Agency contract MDA904-92-C-5141. The author is grateful to Dan Atkinson, Nathaniel Macon, Howard Stainer, and Mike Ware for making this work possible.

References

[1] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Mathematical Foundations and Model." M74-244, Mitre Corporation, Bedford, Massachusetts (1975). (Also available through National Technical Information Service, Springfield, Va., NTIS AD-771543.)

[2] Biba, K.J. "Integrity Considerations for Secure Computer Systems." Mitre TR-3153, Mitre Corporation, Bedford, Massachusetts, (1977). (Also available through National Technical Information Service, Springfield, Va., NTIS AD-A039324.)

[3] Boebert, W.E. and Kain, R.Y. "A Practical Alternative to Hierarchical Integrity Policies." *Proceedings of the 8th NBS-NSA National Computer Security Conference*, 18-27 (1985).

[4] Brewer, D.F.C and Nash, M.J. "The Chinese Wall Security Policy." *Proceedings IEEE Symposium on Security and Privacy*, 215-228 (1989).

[5] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

[6] Gougen, J.A. and Meseguer, J. "Security Policies and Security Models." *IEEE Symposium on Security and Privacy*, 11-20 (1982).

[7] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems*, 437-443 (1971). Reprinted in *ACM Operating Systems Review* 8(1):18-24 (1974).

[8] Lampson, B.W. "A Note on the Confinement Problem." *Communications of ACM* 16(10):613-615 (1973).

[9] Lipner, S.B. "Non-Discretionary Controls for Commercial Applications." *Proceedings IEEE Symposium on Security and Privacy*, 2-10 (1982).

[10] Proctor, N.E. and Neumann, P.G. "Architectural Implications of Covert Channels." *Proceedings 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD, October 1992, pages 28-43.

[11] Sandhu, R.S. "The Typed Access Matrix Model." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992, pages 122-136.

[12] Sandhu, R.S. "A Lattice Interpretation of the Chinese Wall Policy." *Proceedings 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD, October 1992, pages 329-339.

[13] Smith, G.W. *The Modeling and Representation of Security Semantics for Database Applications*. Ph.D. thesis, George Mason University, 1990.

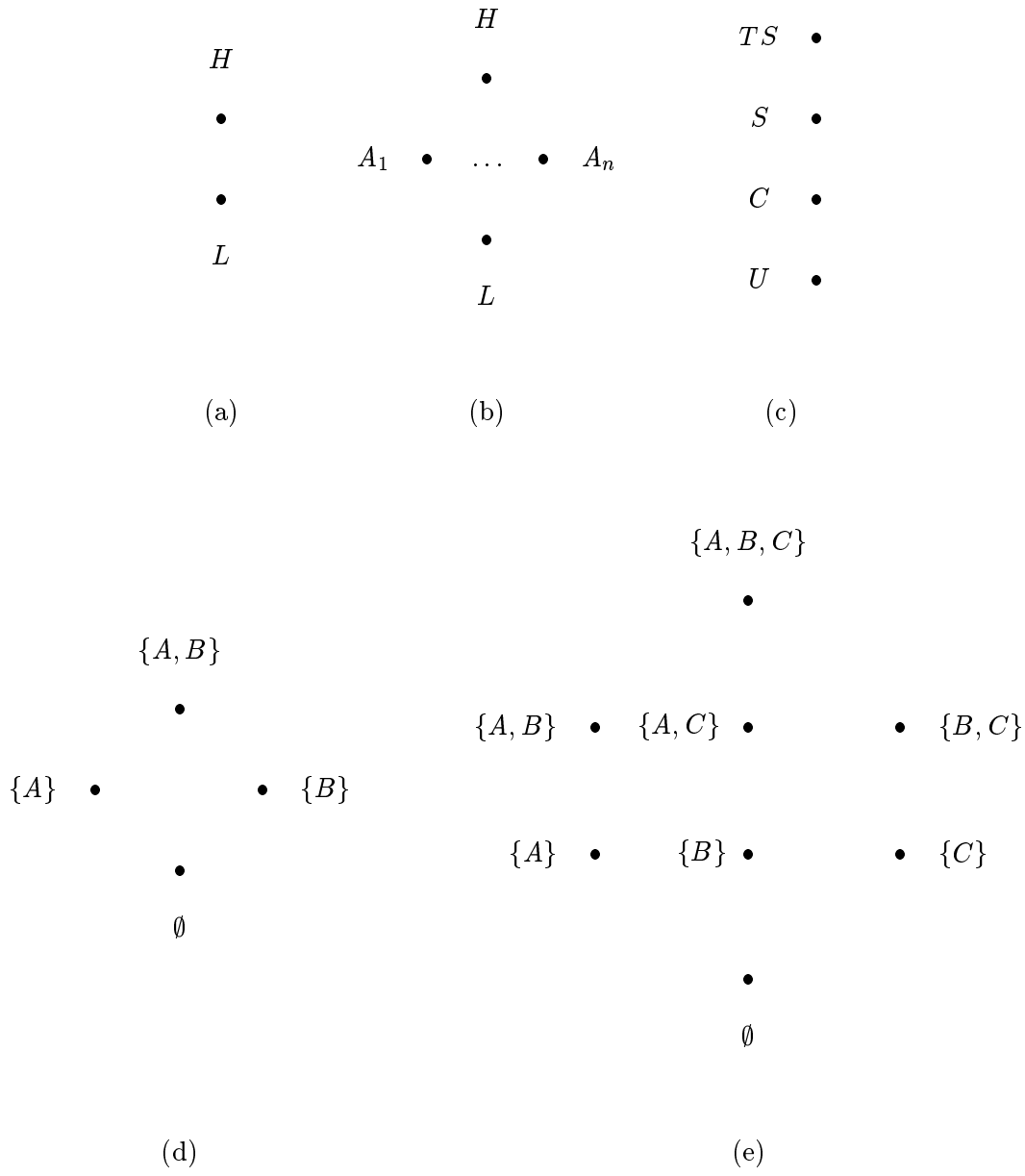


Figure 1: Hasse Diagrams for Some Information Flow Policies

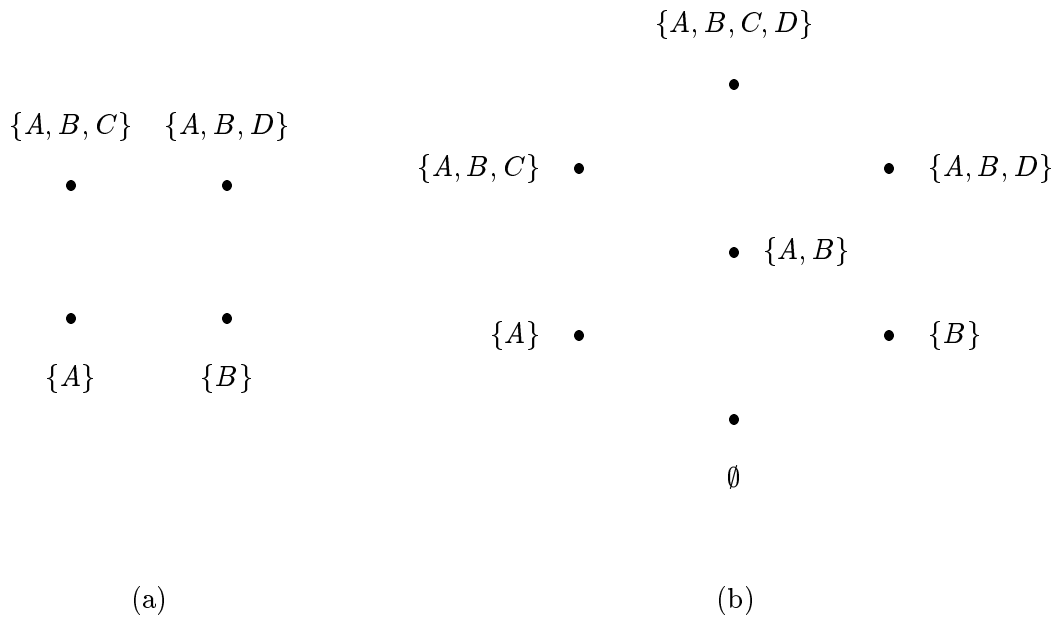


Figure 2: Embedding a Partial Order in a Lattice

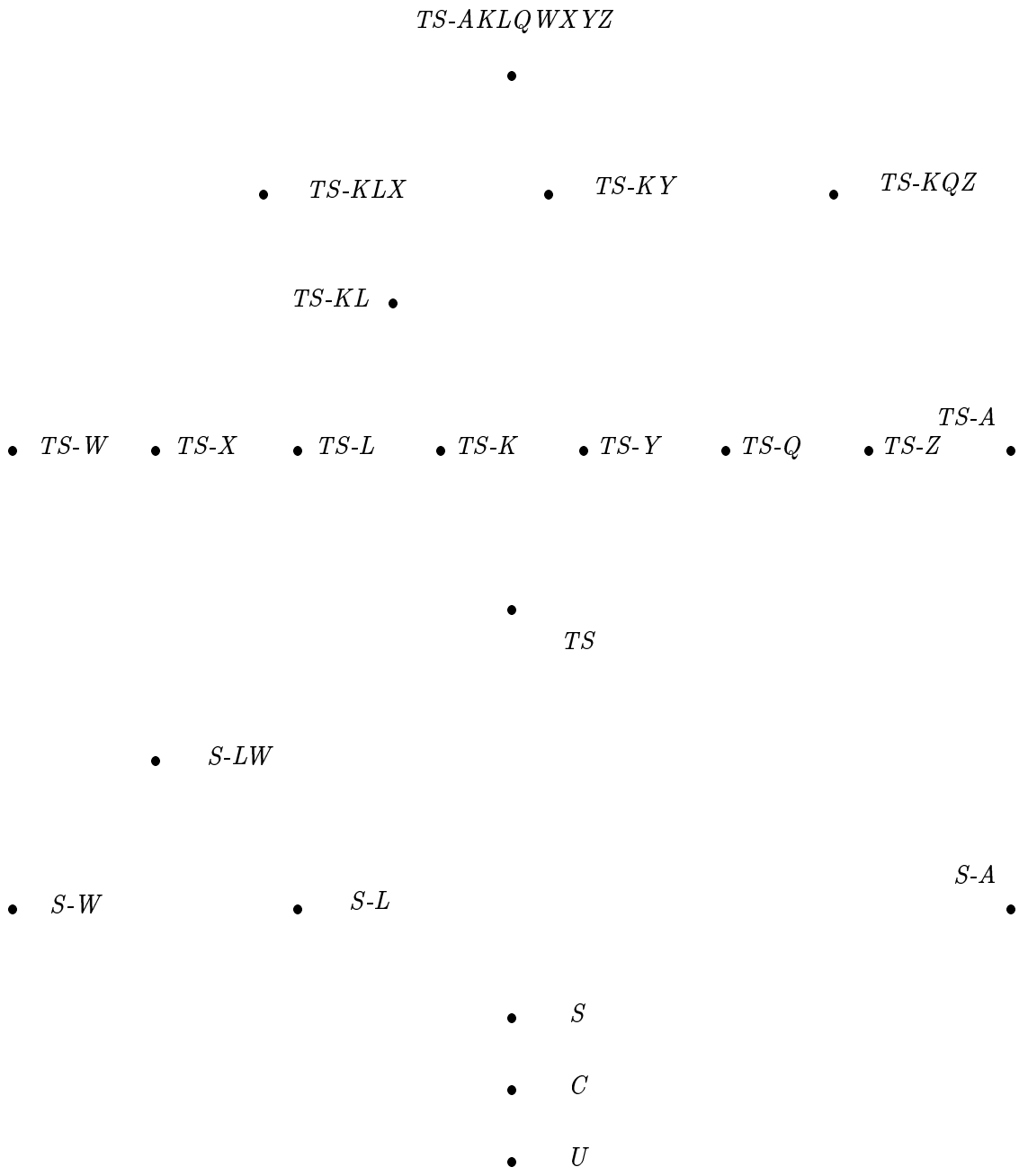
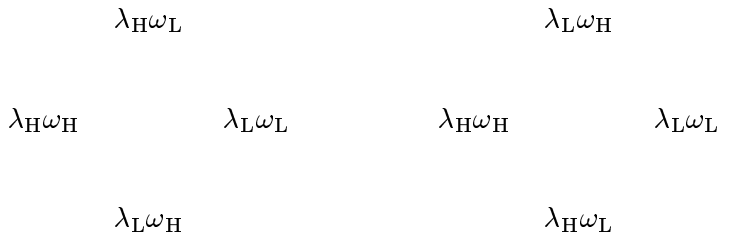


Figure 3: Smith's Lattice

$$\Lambda = \{\lambda_{\mathrm{H}}, \lambda_{\mathrm{L}}\}, \quad \lambda_{\mathrm{H}} \geq \lambda_{\mathrm{L}}$$
$$\Omega = \{\omega_{\mathrm{H}}, \omega_{\mathrm{L}}\}, \quad \omega_{\mathrm{H}} \geq \omega_{\mathrm{L}}$$

	$\lambda_{\mathrm{L}}\omega_{\mathrm{L}}$	$\lambda_{\mathrm{L}}\omega_{\mathrm{H}}$	$\lambda_{\mathrm{H}}\omega_{\mathrm{L}}$	$\lambda_{\mathrm{H}}\omega_{\mathrm{H}}$
$\lambda_{\mathrm{L}}\omega_{\mathrm{L}}$	rw	r	w	ϕ
$\lambda_{\mathrm{L}}\omega_{\mathrm{H}}$	w	rw	w	w
$\lambda_{\mathrm{H}}\omega_{\mathrm{L}}$	r	r	rw	r
$\lambda_{\mathrm{H}}\omega_{\mathrm{H}}$	ϕ	r	w	rw

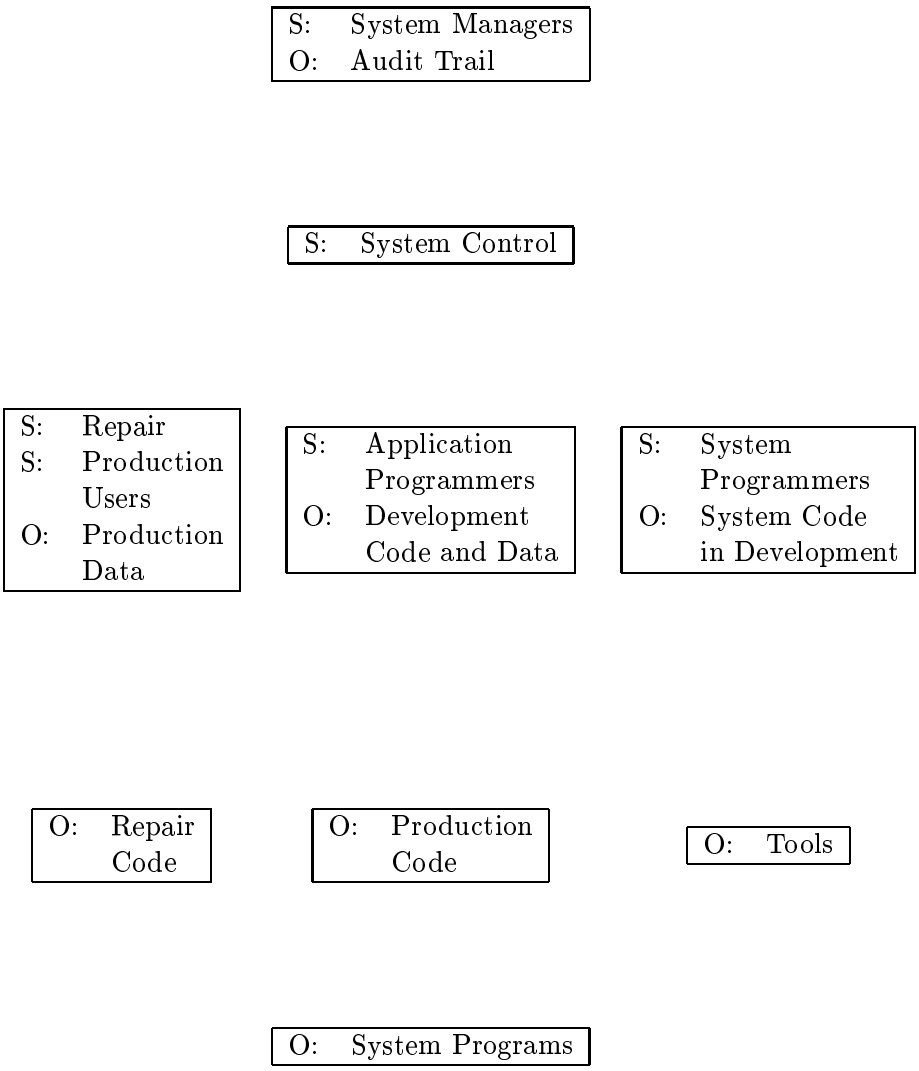
(a) Composite Model



(b) Equivalent BLP Lattice (Allowed flows are upward)

(c) Equivalent Biba Lattice (Allowed flows are downward)

Figure 4: Example of Equivalence



Each box represents a label. Entries in each box specify subjects (prefixed S:) and objects (prefixed O:) who are assigned that label. Allowed flows are upward.

Figure 5: Lipner's Composite Model as a BLP Lattice

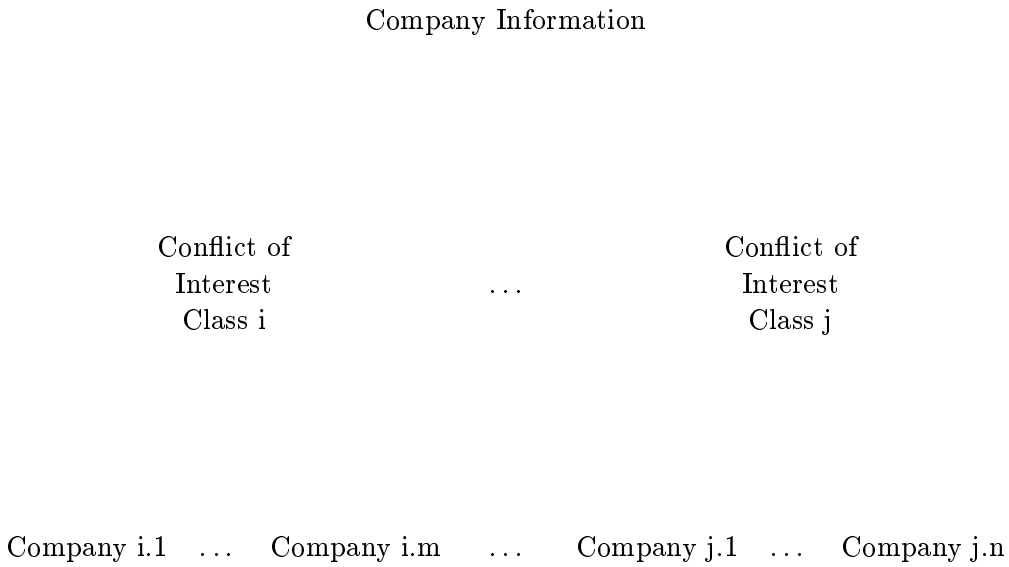


Figure 6: Company Information in the Chinese Wall Policy

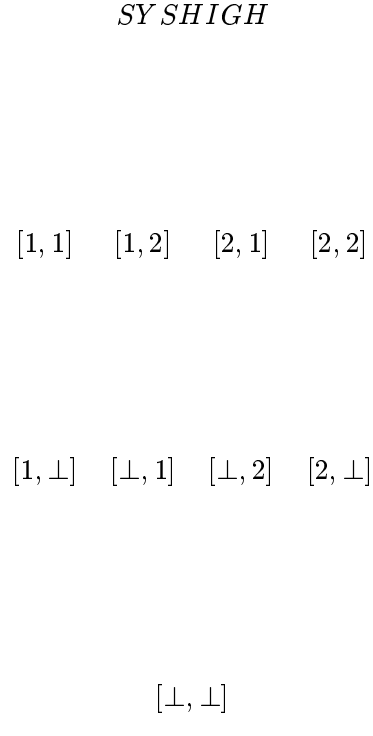


Figure 7: Example of a Chinese Wall Lattice