PART Two

In the second part of the book, I describe a large number of applications of secure systems, many of which introduce particular protection concepts or technologies.

There are four successive themes. Chapters 7 through 9 look at conventional computer security issues, and by discussing what one is trying to do and how it's done in different environments—the military, banks, and healthcare—I introduce security policy models, which set out the protection concepts that real systems try to implement. I also introduce the first detailed case studies in these chapters. An example is the worldwide network of automatic teller machines, which illustrates many of the problems of transferring familiar protection properties from a bank branch to a global distributed environment using cryptography.

Chapters 10 through 15 look at the hardware engineering aspects of information security. This includes biometrics, the design of various tokens such as smartcards, tamper resistance and tamper evidentness, emission security, and seals. New applications that illustrate the technologies are described, ranging from electronic warfare and nuclear weapons control to taximeters, truck speed limiters, and prepayment gas meters.

The third theme is attacks on networks. I start off in Chapter 16 by covering electronic and information warfare, as these activities give some of the more extreme examples and show how far techniques of denial, deception, and exploitation can be taken by a resourceful opponent under severe operational pressure. This chapter also gives a view of surveillance and intrusion from the point of view of police forces and intelligence agencies, and introduces a number of new concepts, such as anonymity and traffic analysis. We then study the lessons of history by examining frauds on phone systems and on applications that rely on them in Chapter 17. This sets the scene for a discussion in Chapter 18 of attacks on computer networks and defensive technologies such as firewalls and intrusion detection.

The fourth theme is electronic commerce, which I tackle in Chapters 19 and 20. The most high-profile applications are schemes for protecting credit card transactions on the Net, such as SSL/TLS; they are also used for other applications such as medical image distribution. They introduce the debate about public key infrastructures. In addition, I consider mechanisms for copyright protection, specifically, pay-TV, DVD, and copyright watermarking.

One reason for this ordering is to give the chapters a logical progression. Thus, for example, I discuss frauds against magnetic stripe bank cards before going on to describe the smartcards that may replace them and the pay-TV systems that actually use smartcards today. That said, sometimes a neat linear ordering isn't possible, as a particular technology has evolved through a number of iterations involving more than one application. In that case, I try to describe it in a case history.

Finally, to keep the book manageable for readers who will use it primarily as a reference rather than as a textbook, I have put the more technical material toward the end of each chapter or section. That way, if you get lost at a first reading, you can just skip to the next section and carry on.

CHAPTER

7

Multilevel Security

At times, in the name of national security, secrecy has put that very security in harm's way.

—DANIEL PATRICK MOYNIHAN

I brief;
you leak;
he/she commits a criminal offence
by divulging classified information

—BRITISH CIVIL SERVICE PROVERB

7.1 Introduction

I mentioned in the introduction that military database systems, which can hold information at a number of different levels of classification (confidential, secret, top secret, . . .) have to ensure that data can be read only by a principal whose level is at least as high as the data's classification. These systems are important because:

- A huge amount of research has been done on them, thanks to military funding for computer science in the United States. So the military model of protection has been worked out in much more detail than any other, and it gives us a lot of examples of the second-order and even third-order effects of implementing a security policy rigorously.
- Some of the products developed to support military multilevel security may find a new lease on life as platforms for firewalls and Web servers. They give some assurance that even although a firewall or server software might be hacked, the underlying operating system is not likely to be.

- Although multilevel concepts were originally developed to support confidentiality in military systems, there are now many commercial systems that use multilevel integrity policies. For example, phone companies want their billing system to be able to see what's happening in their switching system, but not affect it.
- Multilevel confidentiality ideas are often applied in environments where they're ineffective or even harmful, because of the major vested interests and momentum behind them.

Sir Isaiah Berlin famously described thinkers as either foxes or hedgehogs: a fox knows many little things, while a hedgehog knows one big thing. The multilevel philosophy is the hedgehog approach to security engineering.

7.2 What Is a Security Policy Model?

Where a top-down approach to security engineering is possible, it will typically take the form of *threat model—security policy—security mechanisms*. The critical, and often neglected, part of this process is the security policy.

By a security policy, I mean a document that expresses clearly and concisely what the protection mechanisms are to achieve. It is driven by our understanding of threats, and in turn drives our system design. It will often take the form of statements about which users may access which data. It plays the same role in specifying the system's protection requirements, and evaluating whether they have been met, as the system specification does for general functionality. Indeed, a security policy may be part of a system specification, and like the specification, its primary function is to communicate.

Many organizations use the phrase 'security policy' to mean a collection of vapid statements. Figure 7.1 gives a simple example. This sort of waffle is very common, but is useless to the security engineer.

Its first failing is that it dodges the central issue, namely 'Who determines "need-to-know" and how?' Second, it mixes statements at a number of different levels (organizational approval of a policy logically should not be part of the policy itself). Third, there is a mechanism, but it's implied rather than explicit: "staff shall obey"—but what does this mean they actually have to do? Must the obedience be enforced by the system, or are users "on their honor?" Fourth, how are breaches to be detected and who has a specific duty to report them?

Megacorp Inc security policy

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a "need-to-know".
4. All breaches of this policy shall be reported at once to Security.

Figure 7.1 A typical corporate information security policy.

We must do better than this. In fact, because the term 'security policy' is widely abused to mean a collection of managerialist platitudes, there are three more precise terms that have come into use to describe the specification of protection requirements:

A *security policy model* is a succinct statement of the protection properties that a system, or generic type of system, must have. Its key points can typically be written down in a page or less. It is the document in which the protection goals of the system are agreed to by an entire community, or with the top management of a customer. It may also be the basis of formal mathematical analysis.

A *security target* is a more detailed description of the protection mechanisms that a specific implementation provides, and how they relate to a list of control objectives (some but not all of which are typically derived from the policy model). The security target forms the basis for testing and evaluation of a product.

A *protection profile* is like a security target but expressed in an implementation-independent way to enable comparable evaluations across products and versions. This can involve the use of a semi-formal language or at least of suitable security jargon. A protection profile is a requirement for products that are to be evaluated under the *Common Criteria* [574] (I discuss the Common Criteria in Part 3; they are associated with a scheme used by many governments for mutual recognition of security evaluations of defense information systems).

When I don't have to be so precise, I may use the phrase 'security policy' to refer to any or all of the above. I will never use the term to refer to a collection of platitudes.

Sometimes, we are confronted with a completely new application, and have to design a security policy model from scratch. More commonly, a model already exists; we just have to choose the right one, and develop it into a security target. Neither of these steps is easy. Indeed, one of the purposes of this section is to provide a number of security policy models, describe them in the context of real systems, and examine the engineering mechanisms (and associated constraints) that a security target can use to meet them.

Finally, there is a third usage of the phrase 'security policy,' to mean a list of specific configuration settings for some protection product. I will refer to this as *configuration management* or, occasionally, as *trusted configuration management*, in what follows.

7.3 The Bell-LaPadula Security Policy Model

The best-known example of a security policy model was proposed by David Bell and Len LaPadula in 1973, in response to U.S. Air Force concerns over the security of time-sharing mainframe systems. By the early 1970s, people had realized that the protection offered by many commercial operating systems was poor, and was not getting any better. As soon as one operating system bug was fixed, some other vulnerability would be discovered. (Modern reliability growth models can quantify this and confirm that the pessimism was justified; I discuss them further in Section 23.2.4). There was the constant worry that even unskilled users would discover loopholes, and use them opportunistically; there was also a keen and growing awareness of the threat from malicious code. There was a serious scare when it was discovered that the Pentagon's World Wide Military Command and Control System was vulnerable to Trojan Horse attacks; this had the effect of restricting its use to people with a 'Top Secret' clearance,

which was inconvenient. Finally, academic and industrial researchers were coming up with some interesting new ideas on protection, which we'll discuss below.

A study by James Anderson led the U.S. government to conclude that a secure system should do one or two things well; and that these protection properties should be enforced by mechanisms that were simple enough to verify and that would change only rarely [16]. It introduced the concept of a *reference monitor*, a component of the operating system that would mediate access control decisions and be small enough to be subject to analysis and tests, the completeness of which could be assured. In modern parlance, such components—together with their associated operating procedures—make up the *Trusted Computing Base* (TCB). More formally, the TCB is defined as the set of components (hardware, software, human, etc.) whose correct functioning is sufficient to ensure that the security policy is enforced, or, more vividly, whose failure could cause a breach of the security policy. The Anderson report's goal was to make the security policy simple enough for the TCB to be amenable to careful verification.

But what are these core security properties that should be enforced above all others?

7.3.1 Classifications and Clearances

World War II, and the Cold War that followed, led NATO governments to move to a common protective marking scheme for labelling the sensitivity of documents. *Classifications* are labels, which run upward from *Unclassified* through *Confidential, Secret*, and *Top Secret.* The details change from time to time. The original idea was that information whose compromise could cost lives was marked 'Secret' while information whose compromise could cost many lives was 'Top Secret'. Government employees have *clearances* depending on the care with which they've been vetted; in the United States, for example, a 'Secret' clearance involves checking FBI fingerprint files, while 'Top Secret' also involves background checks for the previous 5 to 15 years' employment [244].

The access control policy was simple: an official could read a document only if his clearance was at least as high as the document's classification. So an official cleared to 'Top Secret' could read a 'Secret' document, but not vice versa. The effect is that information may only flow upward, from Confidential to Secret to Top Secret (see Figure 7.2), but it may never flow downward unless an authorized person takes a deliberate decision to declassify it.

There are also document-handling rules; thus, a 'Confidential' document might be kept in a locked filing cabinet in an ordinary government office, while higher levels may require safes of an approved type, guarded rooms with control over photocopiers, and so on. (The NSA security manual [582] gives a summary of the procedures used with 'Top Secret' intelligence data.)

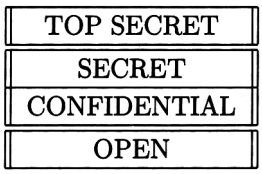

Figure 7.2 Multilevel security.

The system rapidly became more complicated. The damage criteria for classifying documents were expanded from possible military consequences to economic harm and even political embarrassment. Britain has an extra level, 'Restricted', between 'Unclassified' and 'Confidential'; the United States had this, too, but abolished it after the Freedom of Information Act was passed. America now has two more specific markings: 'For Official Use only' (FOUO) refers to unclassified data that can't be released under the Freedom of Information Act (FOIA), while 'Unclassified but Sensitive' includes FOUO plus material that might be released in response to a FOIA request. In Britain, restricted information is in practice shared freely, but marking everything 'Restricted' allows journalists and others involved in leaks to be prosecuted under Official Secrets law. (Its other main practical effect is that an unclassified U.S. document sent across the Atlantic automatically becomes 'Restricted' in Britain, and then 'Confidential' when shipped back to the United States. American military system builders complain that the U.K. policy breaks the U.S. classification scheme!)

There is also a system of codewords whereby information, especially at Secret and above, can be further restricted. For example, information that might contain intelligence sources or methods—such as the identities of agents or decrypts of foreign government traffic—is typically classified 'Top Secret Special Compartmented Intelligence,' or TS/SCI, which means that so-called *need-to-know* restrictions are imposed as well, with one or more codewords attached to a file. Some of the codewords relate to a particular military operation or intelligence source, and are available only to a group of named users. To read a document, a user must have all the codewords that are attached to it. A classification label, plus a set of codewords, makes up a *security category* or (if there's at least one codeword) a *compartment*, which is a set of records with the same access control policy. I discuss compartmentation in more detail in the next Chapter 8.

There are also *descriptors, caveats*, and *IDO markings*. Descriptors are words such as 'Management', 'Budget', and 'Appointments': they do not invoke any special handling requirements, so we can deal with a file marked 'Confidential—Management' as if it were simply marked 'Confidential'. Caveats are warnings, such as "U.K. Eyes Only," or the U.S. equivalent, 'NOFORN'; there are also *International Defense Organization* (IDO) markings such as 'NATO'. The lack of obvious differences between codewords, descriptors, caveats, and IDO marking is one of the factors that can make the system confusing. (A more detailed explanation can be found in [630].)

The final generic comment about access control doctrine is that allowing upward-only flow of information also models what happens in wiretapping. In the old days, tapping someone's telephone meant adding a physical wire at the exchange; nowadays, it's all done in the telephone exchange software, and the effect is somewhat like making the target calls into conference calls with an extra participant. The usual security requirement is that the target of investigation should not know he is being wiretapped, so the third party should be silent—and its very presence must remain unknown to the target. For example, now that wiretaps are usually implemented as silent conference calls, care has to be taken to ensure that the charge for the conference call facility goes to the wiretapper, not to the target. Wiretapping requires an information flow policy in which the 'High' principal can see 'Low' data, but a 'Low' principal can't tell whether 'High' is reading any data, and if so what.

7.3.2 Information Flow Control

It was in the context of the classification of military and intelligence data that the *Bell-LaPadula* (*BLP*) model of computer security was formulated in 1973 by David Bell and Len LaPadula [86]. It is also known as *multilevel security;* systems that implement it are often called *multilevel secure*, or *MLS*, systems. Their basic property is that information cannot flow downward.

More formally, the Bell-LaPadula model enforces two properties:

- The *simple security property:* no process may read data at a higher level. This is also known as *no read up (NRU);*
- The *-*property:* no process may write data to a lower level. This is also known as *no write down (NWD).*

The *-property was Bell and LaPadula's critical innovation. It was driven by the fear of attacks using malicious code. An uncleared user might write a Trojan and leave it around where a system administrator cleared to 'Secret' might execute it; it could then copy itself into the 'Secret' part of the system, read the data there and try to signal it down somehow. It's also quite possible that an enemy agent could get a job at a commercial software house and embed some code in a product that would look for secret documents to copy. If it could then copy them down to where its creator could read it, the security policy would have been violated. Information might also be leaked as a result of a bug, if applications could write down.

Vulnerabilities such as malicious and buggy code are assumed to be given. It is therefore necessary for the system to enforce the security policy independently of user actions (and, by extension, of the actions taken by programs run by users). So we must prevent programs running at 'Secret' from writing to files at 'Unclassified'; or, more generally, prevent any process at High from signalling to any object (or subject) at Low. In general, when systems are built to enforce a security policy independently of user actions, they are described as having *mandatory access control*, as opposed to the *discretionary access control* in systems such as Unix where users can take their own access decisions about their files. (I won't use these phrases much, as they traditionally refer only to BLP-type policies and don't include many other policies whose rules are just as mandatory).

The Bell-LaPadula model makes it relatively straightforward to verify claims about the protection provided by a design. Given both the simple security property (no read up), and the star property (no write down), various results can be proved about the machine states that can be reached from a given starting state, and this simplifies formal analysis.

There are some elaborations, such as a *trusted subject*, a principal who is allowed to declassify files. To keep things simple, I'll ignore this; I'll also ignore the possibility of incompatible security levels for the time being, and return to them in the next chapter. Finally, in order to simplify matters still further, I will assume from now on that the system has only two levels, High and Low (unless there is some particular reason to name individual compartments).

Multilevel security can be implemented in a number of ways. The textbook mechanism is to implement a reference monitor by beefing up the part of an operating system that supervises all operating system calls and checks access permissions to decide whether the call can be serviced or not. In practice, things get much more complex as

it's hard to build systems whose trusted computing base is substantially less than the whole operating system kernel (plus quite a number of its utilities).

Another approach that has been gaining ground as hardware costs have fallen is to replicate systems. One might, for example, have one database at Low and another at High, with a *pump* that constantly copies information from Low up to High. I'll discuss pumps in more detail later.

7.3.3 Standard Criticisms of Bell-LaPadula

The introduction of BLP caused some excitement: here was a straightforward security policy that was clear to the intuitive understanding, yet still allowed people to prove theorems. But John McLean showed that the BLP rules were not in themselves enough. He introduced *System Z*, defined as a BLP system with the added feature that a user can ask the system administrator to temporarily declassify any file from High to Low. In this way, Low users can read any High file without breaking the BLP assumptions.

Bell's argument was that System Z cheats by doing something the model doesn't allow (changing labels isn't a valid operation on the state), and McLean's argument was that it didn't explicitly tell him so. The issue is dealt with by introducing a *tranquility property*. The strong tranquility property says that security labels never cnange during system operation, while the weak tranquility property says that labels never change in such a way as to violate a defined security policy.

The motivation for the weak property is that in a real system we often want to observe the principle of least privilege, and start a process at the uncleared level, even if the owner of the process were cleared to 'Top Secret'. If she then accesses a confidential email, that session is automatically upgraded to 'Confidential'; and in general, her process is upgraded each time it accesses data at a higher level (this is known as the *high water mark* principle). As subjects are usually an abstraction of the memory management subsystem and file handles, rather than processes, this means that state changes when access rights change, rather than when data actually moves.

The practical implication of this is that a process accumulates the security label or labels of every file that it reads, and these become the default label set of every file that it writes. So a process that has read files at 'Secret' and 'Crypto' will thereafter create files marked (at least) 'Secret Crypto'. This will include temporary copies made of other files. If it then reads a file at 'Top Secret Daffodil', all files it creates after that will be labelled 'Top Secret Crypto Daffodil', and it will not be able to write to any temporary files at 'Secret Crypto.' The effect this has on applications is one of the serious complexities of multilevel security; most application software needs to be rewritten (or at least modified) to run on MLS platforms.

Finally it's worth noting that even with this refinement, BLP still doesn't deal with the creation or destruction of subjects or objects (which is one of the hard problems of building a real MLS system).

7.3.4 Alternative Formulations

Multilevel security properties have been expressed in many other ways. The first multilevel security policy was a version of high water mark written in 1967–8 for the ADEPT-50, a mandatory access control system developed for the IBM S/360 mainframe [798]. This used triples of level, compartment and group, with the groups being files, users, terminals, and jobs. As programs (rather than processes) were subjects, it

was vulnerable to Trojan horse compromises, and it was more complex than need be. Nonetheless, it laid the foundation for BLP, and also led to the current IBM S/390 mainframe hardware security architecture [394].

Shortly thereafter, a number of teams produced primitive versions of the lattice model, which I'll discuss in more detail in Chapter 8, Section 8.2.1. These also made a significant contribution to the Bell-LaPadula work, as did Honeywell engineers working on Multics—which led to a system called SCOMP, which I'll discuss in Section 7 below.

Noninterference was introduced by Joseph Goguen and Jose Meseguer in 1982 [339]. In a system with this property, High's actions have no effect on what Low can see. *Nondeducibility* is less restrictive and was introduced by Sutherland in 1986 [743]. Here the idea is to try and prove that Low cannot deduce anything with 100 percent certainty about High's input. Low users can see High actions, just not understand them; a more formal definition is that any legal string of high-level inputs is compatable with every string of low-level events. So for every trace Low can see, there's a similar trace that didn't involve High input. But different low-level event streams may require changes to high-level outputs or reordering of high-level/low-level event sequences.

The motive for nondeducibility is to find a model that can deal with applications such as a LAN on which there are machines at both Low and High, with the High machines encrypting their LAN traffic. (A lot more is needed to do this right, from padding the High traffic with nulls so that Low users can't do traffic analysis, and even ensuring that the packets are the same size—see [659] for an early example of such a system.)

Nondeducibility has historical importance, as it was the first nondeterministic version of Goguen and Meseguer's ideas. But it is hopelessly weak. There's nothing to stop Low making deductions about High input with 99 percent certainty. There are also a whole lot of problems when we are trying to prove results about databases; we have to take into account any information that can be inferred from data structures (such as from partial views of data with redundancy), as well as consider the traces of executing programs. I'll discuss these problems further in Chapter 8, Section 8.3.

Improved models include *generalized noninterference* and *restrictiveness.* The former is the requirement that if one alters a high-level input event in a legal sequence of system events, the resulting sequence can be made legal by, at most, altering one or more subsequent high-level output events. The latter adds a further restriction on the part of the trace, where the alteration of the high-level outputs can take place. This is needed for technical reasons, to ensure that two systems satisfying the restrictiveness property can be composed into a third, which also does. (See [540] which explains these issues.)

The *Harrison-Ruzzo-Ullman* model tackles the problem of how to deal with the creation and deletion of files, an issue on which BLP is silent. It operates on access matrices and verifies whether there is a sequence of instructions that causes an access right to leak to somewhere it was initially not present [373]. This is more expressive than BLP, but more complex and thus less tractable as an aid to verification.

John Woodward proposed a *compartmented mode workstation* (CMW) policy, which attempted to model the classification of information using floating labels, as opposed to the fixed labels associated with BLP [809, 351]. It was ultimately unsuccessful, because labels tend to either float up too far too fast (if done correctly), or they float up more slowly (but don't block all the opportunities for malicious information flow).

However, CMW ideas have led to real products—albeit products that provide separation more than information sharing.

The *type enforcement* model, due to Earl Boebert and Richard Kain [122] and later extended by Lee Badger and others [66], assigns each subject to a *domain*, and each object to a *type*. There is a *domain definition table* (DDT), which acts as an access control matrix between domains and types. This is a natural model in the Unix setting, as types can often be mapped to directory structures. It is more general than policies such as BLP, as it starts to deal with integrity as well as confidentiality concerns.

Finally, the policy model getting the most attention at present from researchers is *role-based access control* (RBAC), introduced by David Ferraiolo and Richard Kuhn [291]. This sets out to provide a more general framework for mandatory access control than BLP in which access decisions don't depend on users' names but on the functions they are currently performing within the organization. Transactions that may be performed by holders of a given role are specified, then mechanisms for granting membership of a role (including delegation). Roles, or groups, had for years been the mechanism used in practice in organizations such as banks to manage access control; the RBAC model starts to formalize this. It can deal with integrity issues as well as confidentiality, by allowing role membership (and thus access rights) to be revised when certain programs are invoked. Thus, for example, a process calling untrusted software that had been downloaded from the Net might lose the role membership required to write to sensitive system files.

7.3.5 The Biba Model

Many textbooks mention in passing a model due to Ken Biba [100], which is often referred to as "Bell-LaPadula upside down." It deals with integrity alone and ignores confidentiality entirely. The key observation is that confidentiality and integrity are in some sense dual concepts: confidentiality is a constraint on who can read a message, while integrity is a constraint on who may have written or altered it.

As a concrete application, an electronic medical device such as an ECG may have two separate modes: calibration and use. The calibration data must be protected from being corrupted by normal users, who will therefore be able to read it but not write to it; when a normal user resets the device, it will lose its current user state (i.e., any patient data in memory) but the calibration will remain unchanged.

To model such a system, we can build a multilevel integrity policy with the rules that we must only read up (i.e., a user process can read the calibration data) and write down (i.e., a calibration process can write to a buffer in a user process); but we must never read down or write up, as either could allow High-integrity objects to become contaminated with Low—that is, potentially unreliable—data. The Biba model is often formulated in terms of the *low water mark* principle, which is the dual of the high water mark principle already discussed: the integrity of an object is the lowest level of all the objects that contributed to its creation.

This was the first formal model of integrity. A surprisingly large number of real systems work along Biba lines. For example, the passenger information system in a railroad may get information from the signaling system, but certainly shouldn't be able to affect it (other than through a trusted interface, such as one of the control staff). However, few of the people who build such systems are aware of the Biba model or what it might teach them.

One interesting exception is LOMAC, an extension to Linux that implements a low water mark policy [313]. It is designed to deal with the problem of malicious code arriving somehow over the Net. The system provides two levels—high and low integrity—with system files at High and the network at Low. As soon as a program (such as a demon) receives traffic from the network, it is automatically downgraded to Low. Thus, even if the traffic contains an attack that succeeds in forking a root shell, this shell won't have the capability to write to the password file, for example, as a normal root shell would. As one might expect, a number of system tasks (such as logging) become tricky and require trusted code. Note, though, that this approach merely stops the malware getting root access; it doesn't stop it infecting the Low compartment and using it as a springboard from which to spread elsewhere.

As mentioned above, integrity concerns can also be dealt with by the type enforcement and RBAC models. However, in their usual forms, they revise a principal's privilege when an object, is invoked, while low watermark revises it when an object is read. The latter policy is more prudent where we are concerned with attacks exploiting code that is not formally invoked but simply read (as with buffer overflow attacks conducted by "data" read from the Internet).

I will introduce more complex models when I discuss banking and bookkeeping systems in Chapter 9; these are more complex in that they retain security state in the form of dual control mechanisms, audit trails, and so on.

7.4 Examples of Multilevel Secure Systems

Following some research products in the late 1970s (such as KSOS [99], a kernelized secure version of Unix), products that implemented multilevel security policies started arriving in dribs and drabs in the early 1980s. By about 1988, a number of companies had started implementing MLS versions of their operating systems. MLS concepts were extended to all sorts of products.

7.4.1 SCOMP

One of the most important products was the *Secure Communications Processor* (SCOMP), a Honeywell derivative of Multics, launched in 1983 [311]. This was a no-expense-spared implementation of what the U.S. Department of Defense believed it wanted for handling messaging at multiple levels of classification. SCOMP had formally verified hardware and software, with a minimal kernel and four rings of protection (rather than Multics' seven) to keep things simple. Its operating system, STOP, used these rings to maintain up to 32 separate compartments, and to allow appropriate one-way information flows between them.

SCOMP was used in applications such as military *mail guards*, specialized firewalls that typically allow mail to pass from Low to High, but not vice versa [234]. (In general, a device that makes information flow one way only is known as a *data diode*.) SCOMP's successor, XTS-300, supports C2G, the Command and Control Guard. This is used in the *time-phased force deployment data* (TPFDD) system whose function is to plan U.S. troop movements and associated logistics. Overall, military plans are developed as TPFDDs, at a high classification level, then distributed at the appropriate times as commands to lower levels for implementation. The deliberate downgrading of high

information raises a number of issues, some of which I'll deal with later. (In the case of TPFDD, the guard examines the content of each record before deciding whether to release it.)

SCOMP's most significant contribution was to serve as a model for the *Orange Book* [240], also known as the *Trusted Computer Systems Evaluation Criteria* (TCSEC). This was the first systematic set of standards for secure computer systems, being introduced in 1985 and finally retired in December 2000. Although it has since been replaced by the Common Criteria, the Orange Book was enormously influential, not just in the United States but among allied powers; countries such as Britain, Germany, and Canada based their own national standards on it, and these national standards were finally subsumed into the Common Criteria [574].

The *Orange Book* allowed systems to be evaluated at a number of levels, with A1 being the highest, and moving down through B3, B2, B1, and C2 to C1. SCOMP was the first system to be rated A1. It was also extensively documented in the open literature. Being first, and being fairly public, it set the standard for the next generation of military systems. This standard has rarely been met since; in fact, the XTS-300 has been evaluated only to B3 (the formal proofs of correctness required for an A1 evaluation were dropped).

7.4.2 Blacker

Blacker was a series of encryption devices designed to incorporate MLS technology. Previously, encryption devices were built with separate processors for the ciphertext, or *Black*, end, and the cleartext, or *Red*, end. Various possible failures can be prevented if one can coordinate the Red and Black processing. One can also make the device simpler, and provide greater operational flexibility: the device isn't limited to separating two logical networks, but can provide encryption and integrity assurance selectively, and interact in useful ways with routers. But a high level of assurance is required that the 'Red' data won't leak out via the 'Black'.

Blacker entered service in 1989, and the main lesson learned from it was the extreme difficulty of accommodating administrative traffic within a model of classification levels [799]. As late as 1994, it was the only communications security device with an A1 evaluation [97]. So it too had an effect on later systems. It was not widely used though, and its successor (the Motorola Network Encryption System), which is still in use, has only a B2 evaluation.

7.4.3 MLS Unix, CMWs, and Trusted Windowing

Most of the available MLS systems are modified versions of Unix, and an example is AT&T's System V/MLS [15]. This added security levels and labels, initially by using some of the bits in the group ID record, and later by using this to point to a more elaborate structure. This enabled MLS properties to be introduced with minimal changes to the system kernel. Other products of this kind included Secure Ware (and its derivatives, such as SCO and HP VirtualVault) and Addamax.

Compartmented mode workstations (CMWs) allow data at different levels to be viewed and modified at the same time by a human operator, and ensure that labels attached to the information are updated appropriately. The initial demand came from the intelligence community, whose analysts may have access to 'Top Secret' data, such as decrypts and agent reports, and produce reports at the 'Secret' level for users such as political leaders and officers in the field. As these reports are vulnerable to capture, they must not contain any information that would compromise intelligence sources and methods.

CMWs allow an analyst to view the 'Top Secret' data in one window, compose a report in another, and have mechanisms to prevent the accidental copying of the former into the latter (i.e., cut-and-paste works from 'Secret' to 'Top Secret', but not vice versa). CMWs have proved useful in operations, logistics, and drug enforcement as well [396].

For the engineering issues involved in doing mandatory access control in windowing systems, see [273,274], which describe a prototype for Trusted X, a system implementing MLS but not information labelling. It runs one instance of X Windows per sensitivity level, and has a small amount of trusted code that allows users to cut and paste from a lower level to a higher one. For the specific architectural issues with Sun's CMW product, see [281].

7.4.4 The NRL Pump

It was soon realized that simple mail guards and crypto boxes were too restrictive, as many more networked services were developed besides mail. Traditional MLS mechanisms (such as blind write-ups and periodic read-downs) are inefficient for real-time services.

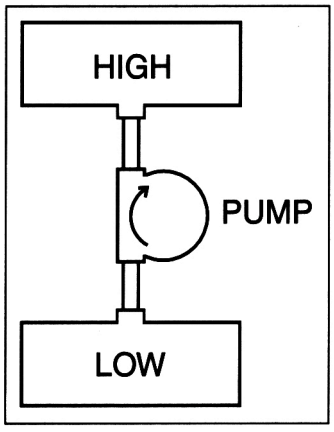

Figure 7.3 The NRL pump.

The US Naval Research Laboratory (NRL) therefore developed the *Pump* (see Figure 7.3), a one-way data transfer device (a data diode) using buffering to allow one-way information flow while limiting the bandwidth of possible backward leakage by a number of mechanisms such as timing randomization of acknowledgment messages [434,436,437]. The attraction of this approach is that it is possible to build MLS sys-

tems by using pumps to connect separate systems at different security levels. As these systems don't process data at more than one level, they can be built from cheap commercial-off-the-shelf (COTS) components [438]. As the cost of hardware falls, this becomes the preferred option where it's possible.

The Australian government has developed a product called *Starlight* that uses pump-type technology married to a keyboard switch to provide a nice MLS-type windowing system (albeit without any visible labels), using a bit of trusted hardware that connects the keyboard and mouse with High and Low systems [17]. There is no trusted software. It's been integrated with the NRL Pump. A number of semi-commercial data diode products have also been introduced.

7.4.5 Logistics Systems

Military stores, like government documents, can have different classification levels. Some signals intelligence equipment is 'Top Secret', while things like jet fuel and bootlaces are not; but even such simple commodities may become 'Secret' when their quantities or movements might leak information about tactical intentions. There are also some peculiarities. For example, an inertial navigation system classified 'Confidential' in the peacetime inventory might contain a laser gyro platform classified 'Secret' (thus security levels are *nonmonotonic*).

The systems needed to manage all this seem to be hard to build, as MLS logistics projects in both the United States and Britain have ended up as expensive disasters. The Royal Air Force's Logistics Information Technology System (LITS) was a 10-year (1989–1999), $500 million project to provide a single stores management system for the RAF's 80 bases [571]. It was designed to operate on two levels: 'Restricted' for the jet fuel and boot polish, and 'Secret' for special stores such as nuclear bombs. It was initially implemented as two separate database systems connected by a pump to enforce the MLS property. The project became a classic tale of escalating costs driven by creeping requirements changes. One of these changes was the easing of classification rules at the end of the Cold War. As a result, it was found that almost all the 'Secret' information was now static (e.g., operating manuals for air-drop nuclear bombs, which are now kept in strategic stockpiles rather than at airbases). To save money, the 'Secret' information is now kept on a CD and locked up in a safe.

Logistics systems often have application security features too. The classic example is that ordnance control systems alert users who are about to breach safety rules by putting explosives and detonators in the same truck or magazine [563].

7.4.6 Purple Penelope

In recent years, most governments' information security agencies have been unable to resist user demands to run standard applications (such as MS Office), which are not available for multilevel secure platforms. One response is 'Purple Penelope'. This software, from Britain's Defence Evaluation and Research Agency, puts an MLS wrapper round a Windows NT workstation. It implements the high water mark version of BLP, displaying in the background the current security level of the device, and upgrading it when necessary as more sensitive resources are read. It ensures that the resulting work product is labelled correctly.

Rather than preventing users from downgrading, as a classical BLP system might do, it allows them to assign any security label they like to their output. However, if this

involves a downgrade, it requires the user to confirm the release of the data using a trusted path interface, thus ensuring no Trojan or virus can release anything completely unnoticed. Of course, a really clever malicious program can piggyback classified material on stuff that the user does wish to release, so there are other tricks to make that harder. There is also an audit trail to provide a record of all downgrades, so that errors and attacks (whether by users or by malicious code) can be traced after the fact [620].

7.4.7 Future MLS Systems

The MLS industry sees an opportunity in using its products as platforms for firewalls, Web servers, and other systems that are likely to come under attack. Thanks to the considerable effort that has often gone into finding and removing security vulnerabilities, MLS platforms can give more assurance than commodity operating systems that, even if the firewall or Web server software is hacked, the underlying operating system is unlikely to be. The usual idea is to use the MLS platform to separate trusted from untrusted networks, then introduce simple code to bypass the separation in a controlled way. In fact, one of the leading firewall vendors (TIS) was until recently focused on developing MLS operating systems, while Secure Computing Corporation, Cyberguard, and Hewlett-Packard have all offered MLS-based firewall products. The long tradition of using MLS systems as pumps and mail guards means that firewall issues are relatively well understood in the MLS community. (A typical design is described in [162].)

However, the BLP controls do not provide enough of a protection benefit in many commercial environments to justify their high development costs, and widely fielded products are often better because of the evolution that results from large-scale user feedback. We find, for example, two firewall products from the same corporation, doing much the same thing, with one of them MLS (the Standard Mail Guard) and the other based on open source code and aimed at commercial markets (Sidewinder). According to users, the former has "never been able to hold a candle to the latter."

Perhaps the real future of multilevel systems is not in confidentiality, but integrity. Many fielded systems implement some variant of the Biba model (even though their designers may never have heard the word "Biba"). In an electricity utility, for example, the critical operational systems such as power dispatching should not be affected by any others; they can be observed by, but not influenced by, the billing system. Similarly, the billing system and the power dispatching system both feed information into the fraud detection system, and so on, until at the end of the chain we find the executive information systems, which can observe everything (or at least, summaries of everything) while having no direct effect on operations.

Researchers are now starting to build models that accommodate both confidentiality and integrity to observe their interaction and workout how they might apply in environments such as smartcards [440]. Another topic is how mandatory access control models can provide real-time performance guarantees to help prevent service denial attacks [552]. It's already clear that many of the lessons learned in multilevel confidentiality systems also go across. So do a number of the failure modes, which I discuss in the next section.

7.5 What Goes Wrong

As I've frequently pointed out, engineers learn more from the systems that fail than from those that succeed, and MLS systems have certainly been an effective teacher. The large effort expended in building systems to follow a simple policy with a high level of assurance has led to the elucidation of many second- and third-order consequences of information flow controls. I'll start with the more theoretical and work through to the business and engineering end.

7.5.1 Composability

Consider a simple device that accepts two High inputs H_1 and H_2, multiplexes them, encrypts them by xor'ing them with a one-time pad (i.e., a random generator), outputs the other copy of the pad on H_3, and outputs the ciphertext, which being encrypted with a cipher system giving perfect secrecy, is considered to be Low (output L). This is shown in Figure 7.4.

In isolation, this device is provably secure. But if feedback is permitted, then the output from H_3 can be fed back into H_2, with the result that the high input H_1 now appears at the low output L.

Timing inconsistencies can also lead to the composition of two secure systems being insecure (see for example McCullough [534]). Simple information flow doesn't compose; neither does noninterference or nondeducibility. In general, the problem of how to compose two or more secure components into a secure system is hard, even at the relatively uncluttered level of proving results about ideal components. Most of the problems arise when some sort of feedback is introduced into the system; without it, composition can be achieved under a number of formal models [541]. However, in real life, feedback is pervasive, and composition of security properties can be complicated by detailed interface issues, feature interactions, and so on.

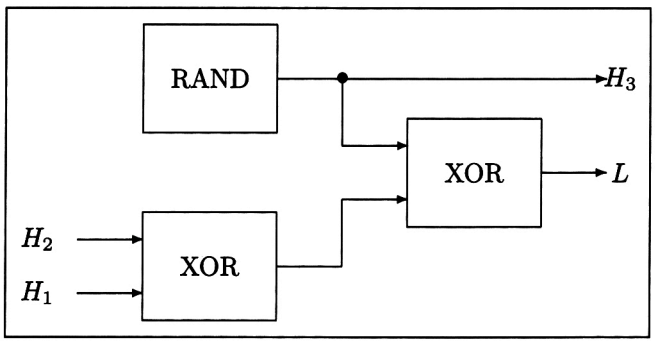

Figure 7.4 Insecure composition of secure systems with feedback.

Finally, the composition of components that have been designed in accordance with two different security policies is harder still. This is bad enough for different variants on the BLP theme but even worse when one of the policies is of a non-BLP type, as we will encounter in the following two chapters.

7.5.2 The Cascade Problem

An example of the difficulty of composing multilevel secure systems is given by the *cascade problem* (Figure 7.5). After the Orange Book introduced a series of graduated evaluation levels, this led to rules about the number of levels a system can span. For example, a system evaluated to B3 is in general allowed to process information for users with a clearance level of Unclassified through Secret, or of Confidential through Top Secret, but not to process Top Secret data with some users restricted to Unclassified only [244].

As the diagram shows, it is straightforward to connect two A1 systems in such a way that this security policy is broken. The first system connects Unclassified and Secret; and its Secret level communicates with the second system, which also processes Top Secret information. (The problem is discussed in more detail in [391].) It illustrates another kind of danger that formal models of security (and practical implementations) must take into account.

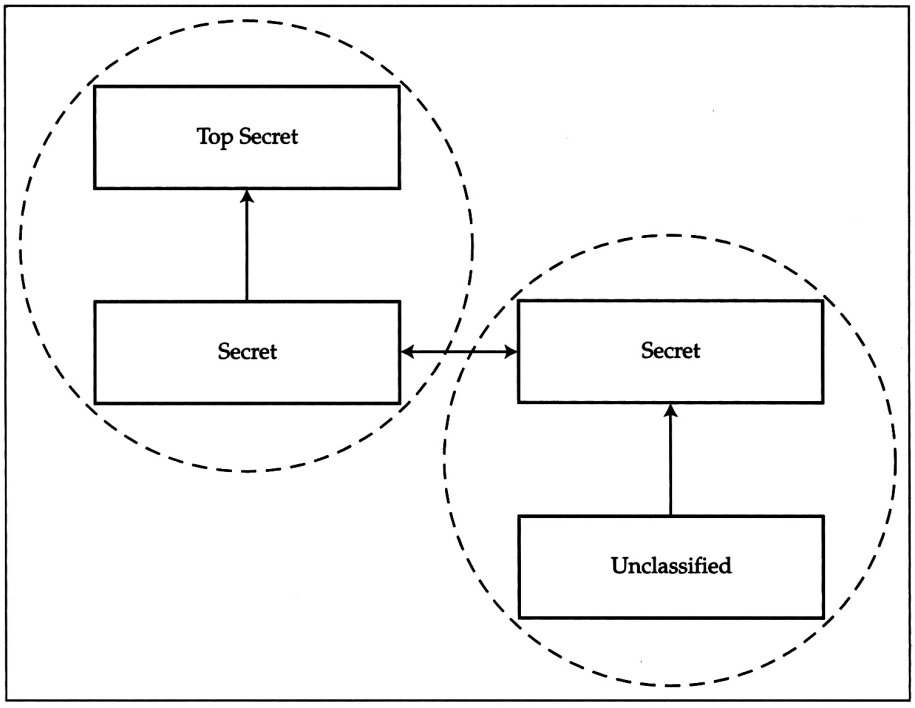

Figure 7.5 The cascade problem.

7.5.3 Covert Channels

One of the reasons these span limits are imposed on multilevel systems emerges from a famous—and extensively studied—problem: the *covert channel*. First pointed out by Butler Lampson in 1973 [488], a covert channel is a mechanism that, though not designed for communication, can nonetheless be abused to allow information to be communicated down from High to Low.

A typical covert channel arises when a High process can signal to a Low process by affecting some shared resource. For example, it could position the disk head at the outside of the drive at time t_i to signal that the i-th bit in a High file was a 1, and position it at the inside to signal that the bit was a 0.

All systems with shared resources must find a balance between covert channel capacity, resource utilization, and fairness. If a machine is shared between High and Low, and resources are not allocated in fixed slices, then the High process can signal by filling up the disk drive, or by using a lot of CPU or bus cycles (some people call the former case a *storage channel* and the latter a *timing channel*, though in practice they can often be converted into each other). There are many others, such as sequential-process IDs, shared file locks, and last access times on files; reimplementing all of these in a multilevel secure way is an enormous task. Various strategies have been adopted to minimize their bandwidth. For example, we can arrange that the scheduler assigns a fixed disk quota to each level, and reads the boot sector each time control is passed downward; we might also allocate a fixed proportion of the available time slices to processes at each level, and change these proportions infrequently. Each change might allow one or more bits to be signalled, but such strategies can significantly reduce the available bandwidth. (A more complex multilevel design, which uses local schedulers at each level, plus a global scheduler to maintain overall consistency, is described in [435].)

It is also possible to limit the covert channel capacity by introducing noise. Some machines have had randomized system clocks for this purpose. But some covert channel capacity almost always remains. (Techniques to analyze the trade-offs between covert channel capacity and system performance are discussed in [353].)

Covert channels also occur at the application layer. A medical example is that, in Britain, personal health information derived from visits to genitourinary medicine (GUM) clinics is High in the sense that it can't be shared with the patient's normal doctor and thus won't appear in their normal medical record (Low) unless the patient consents. In one case, a woman's visit to a GUM clinic "leaked" when the insurer failed to recall her for a smear test, which her normal doctor knew was due [551]. The insurer knew that a smear test had been done already by the clinic, and didn't want to pay twice. (Some people might say this was a failure of polyinstantiation, which I discuss in 7, or an inference attack, which I'll come to in Chapter 8, Section 8.3.)

The worst case known to me as far as bandwidth is concerned is also a feature of a specific application. It occurs in large early-warning radar systems, where High—the radar processor—controls hundreds of antenna elements that illuminate Low—the target—with high-speed pulse trains that are modulated with pseudorandom noise to make jamming harder. In this case, the radar code must be trusted, as the covert channel bandwidth is many megabits per second.

The case of greatest general interest is multilevel integrity systems, such as banking and utility billing, where a programmer who has inserted Trojan code in a high-integrity bookkeeping system can turn off the billing to an account by a certain pattern of behavior (in a phone system he might call three numbers in succession, for example). Code review is the only real way to block such attacks, though balancing controls can also help (I discuss this in Chapter 9).

The best that developers have been able to do consistently with covert channel bandwidth in time-sharing multilevel operating systems is to limit it to one bit per second or so. (That is now the DoD target [241]; techniques for doing a systematic analysis may be found in [448].) One bit per second may be tolerable in an environment where we wish to prevent large TS/SCI files—such as satellite photographs—leaking down from TS/SCI users to 'Secret' users, and is much less than the rate at which malicious code might hide data in outgoing traffic that would be approved by a guard. However, it is inadequate if we want to prevent the leakage of a cryptographic key. This is one of the reasons for the military doctrine of doing crypto in special-purpose hardware rather than in software. It also explains why span limits are relaxed for *closed security environments*—systems in which application code can be introduced only by suitably cleared personnel (and where "system applications are adequately protected against the insertion of malicious logic"); in such a case, an A1 system is allowed to process both Top Secret and Unclassified data simultaneously [244].

7.5.4 The Threat from Viruses

The vast majority of viruses are found in mass-market products such as PCs and Macs. However, the defense computer community was shocked when Fred Cohen used viruses to penetrate multilevel secure systems easily in 1983. In his first experiment, a file virus which took only eight hours to penetrate a system previously believed to be multilevel secure [192].

There are a number of ways in which viruses and other malicious code can be used to perform such attacks. If the reference monitor (or other TCB components) can be corrupted, a virus could deliver the entire system to the attacker, for example by issuing him with an unauthorized clearance. This is why slightly looser rules apply to closed security environments. But even if the TCB remains intact, the virus could still use any available covert channel to signal information down.

In many cases, a TCB will provide some protection against viral attacks, as well as against careless disclosure by users or application software—which is often more important than malicious disclosure. However, the main effect of viruses on military doctrine has been to strengthen the perceived case for multilevel security. The argument goes that, even if personnel can be trusted, one cannot rely on technical measures short of total isolation to prevent viruses moving up the system, so one must do whatever is reasonably possible to stop them signalling back down.

7.5.5 Polyinstantiation

Another problem that has much exercised the research community is *polyinstantiation.* Suppose that our High user has created a file named `agents`, and that our Low user now tries to do the same. If the MLS operating system prohibits this, it will have leaked information—namely, that there is a file called `agents` at High. But if it doesn't, it will now have two files with the same name.

Level	Cargo	Destination
Secret	Missiles	Iran
Restricted	–	–
Unclassified	Engine spares	Cyprus

Figure 7.6 How the United States deals with classified data.

Level	Cargo	Destination
Secret	Missiles	Iran
Restricted	Classified	Classified
Unclassified	–	–

Figure 7.7 How Britain deals with classified data.

Often, we can solve the problem by a naming convention, which could be as simple as giving Low and High users different directories. But the problem remains a hard one for databases [669]. Suppose that a High user allocates a classified cargo to a ship. The system will not divulge this information to a Low user, who might think the ship is empty, and try to allocate it another cargo or even to change its destination.

The solution favored in the United States for such systems is that the High user allocates a Low cover story at the same time as the real High cargo. Thus, the underlying data will look something like that shown in Figure 7.6.

In the Britain, which does not have a Freedom of Information Act, the theory is simpler: the system will automatically reply 'Classified' to a Low user who tries to see or alter a High record. The two available views would be as shown in Figure 7.7.

This makes the system engineering simpler. It also prevents the mistakes and covert channels which can still arise with cover stories (e.g., a Low user tries to add a container of ammunition for Cyprus). The drawback is that everyone tends to need the highest available clearance to get their work done. (In practice, of course, cover stories may still get used so as not to advertise the existence of a covert mission any more than need be.)

7.5.6 Other Practical Problems

Multilevel secure systems are surprisingly expensive and difficult to build and deploy. There are many sources of cost and confusion.

- MLS systems are built in small volumes, and often to high standards of physical robustness, using elaborate documentation, testing, and other quality control measures driven by military purchasing bureaucracies.
- MLS systems have idiosyncratic administration tools and procedures. A trained Unix administrator can't just take on an MLS installation without significant further training. A USAF survey showed that many MLS systems were installed without their features being used [624].

- Many applications need to be rewritten or at least greatly modified to run under MLS operating systems [655]. For example, CMWs that display information at different levels in different windows, and prevent the user from doing cut-and-paste operations from High to Low, often have problems with code that tries to manipulate the color map. Access to files might be quite different, as might the format of things like access control lists. Another source of conflict with commercial software is the license server; if a High user invokes an application, which goes to a license server for permission to execute, an MLS operating system will promptly reclassify the server High and deny access to Low users. So, in practice, we usually end up (a) running two separate license servers, thus violating the license terms; or (b) having an MLS license server that tracks licenses at all levels and hence must be part of the TCB (this restricts your choice of platforms); or (c) using the licensed software at only one of the levels.
- Because processes are automatically upgraded as they see new labels, the files they use have to be, too. New files default to the highest label belonging to any possible input. The result is a chronic tendency for things to be overclassified.
- It is often inconvenient to deal with "blind write-up"; when a low-level application sends data to a higher-level one, BLP prevents any acknowledgment being sent. The effect is that information vanishes into a "black hole." The answer to this is varied. Some organizations accept the problem as a fact of life; in the words of a former NSA chief scientist, "When you pray to God, you do not expect an individual acknowledgment of each prayer before saying the next one." Others use pumps rather than prayer, and accept a residual covert bandwidth as a fact of life.
- The classification of data is not entirely straightforward:
  - In the run-up to a military operation, the location of "innocuous" stores, such as food, could reveal tactical intentions, and so may be suddenly upgraded. It follows that the tranquility property cannot simply be assumed.
  - Classifications are not necessarily monotone. Equipment classified as 'Confidential' in the peacetime inventory may easily contain components classified 'Secret'.
  - Information may need to be downgraded. An intelligence analyst might need to take a satellite photo classified at TS/SCI, and paste it into an assessment for field commanders at 'Secret'. However, information could have been covertly hidden in the image by a virus, and retrieved later when the file is downgraded. So, downgrading procedures may involve all sorts of special filters, such as lossy compression of images and word processors that scrub and reformat text, in the hope that the only information remaining is that which lies in plain sight. (I discuss information hiding in more detail in the context of copyright marking in Chapter 20.)
  - We may need to worry about the volume of information available to an attacker. For example, we might be happy to declassify any single satellite photo, but declassifying the whole collection would reveal our surveillance capability and the history of our intelligence priorities. Similarly, the government payroll might not be very sensitive per se, but it is well known that journalists can often identify intelligence personnel working under civilian cover from studying the evolution of departmental staff lists over a

period of a few years. (I delve into this issue—the "aggregation problem"—in more detail in Chapter 8, Section 8.3.2.)

  - A related problem is that the output of an unclassified program acting on unclassified data may be classified. This is related to the aggregation problem just noted.

- There are always system components—such as memory management—that must be able to read and write at all levels. This problem is dealt with by "abstracting it away," and assuming that memory management is part of the trusted computing base that enforces BLP. The practical outcome is that an uncomfortably large part of the operating system (plus utilities, plus windowing system software, plus middleware such as database software) often ends up in the trusted computing base. "TCB bloat" constantly pushes up the cost of evaluation, and reduces assurance.
- Finally, although MLS systems can prevent undesired things (such as information leakage) from happening, they also prevent desired things from happening, too (such as efficient ways of enabling data to be downgraded from High to Low, which are essential if many systems are to be useful). So even in military environments, the benefits MLS systems provide can be very questionable. The associated doctrine also sets all sorts of traps for government systems builders. A recent example comes from the debate over a U.K. law to extend wiretaps to Internet service providers (ISPs). (I discuss this law further in Chapter 21, "E-Policy"). Opponents of the bill forced the government to declare that information on the existence of an interception operation against an identified target would be classified 'Secret'. This would have made wiretaps on Internet traffic impossible without redeveloping all the systems used by ISPs to support an MLS security policy—which would be impractical regardless of the time or budget available. The U.K. government had to declare that it wouldn't apply the laid-down standards in this case because of cost.

7.6 Broader Implications of MLS

The nonmilitary reader's reaction by this point may well be that MLS is clearly impossible to do properly; there are just too many complications. This may be true, but it's also true that Bell-LaPadula is the simplest security policy model we know of; everything else is even harder. We'll look at other models in the next few chapters.

Anyway, although the MLS program has not delivered what was expected, it has spun off a lot of useful ideas and know-how. Worrying about not just the direct ways a secure system can be defeated, but also about the second- and third-order consequences of the protection mechanisms, has been important in developing the underlying science. Practical work on building MLS systems also led people to work through many other aspects of computer security, such as *Trusted Path* (how does a user know he or she is talking to a genuine copy of the operating system?), *Trusted Distribution* (how does a user know he or she is installing a genuine copy of the operating system?) and *Trusted Facility Management* (how can we be sure it's all administered correctly?). In effect, tackling one simplified example of protection in great detail cast light on many things that previously were glossed over. The resulting lessons can be applied to sys-

tems with quite different policies. An excellent recent example comes from Cipress, a prototype system built by the Fraunhofer Institute to provide strict copy and usage control of digital media [149]. The security policy amounted to a kind of high water mark; an application which combined a number of protected media streams would produce an output that could only be accessed by a principal with access to all the keys that controlled the input streams. This gave rise to many of the problems we have discussed above, and more: for example, if a media owner revoked access rights to some content, then this could propagate a lock to large numbers of derivative works.

These lessons are set out in the Rainbow Series of books on computer security, which were produced by the NSA following the development of SCOMP and the publication of the Orange Book which it inspired. (These books are so called because of their different-colored covers.) Though the series is notably silent on some topics, such as crypto and emission security, it has done a lot to raise consciousness of operational and evaluation issues which are otherwise easy to ignore (or to dismiss as boring matters best left to the end purchasers). In fact, the integration of technical protection mechanisms with operational and procedural controls is one of the most critical, and neglected, aspects of security engineering. The secure operation of MLS systems is usually much the weakest link in the chain. The main vulnerability of the STU-III secure telephone, for example, is that generals forget to press the 'go secure' button before discussing classified matters. A particularly instructive case history is that of Former CIA Director John Deutch. Deutch was supposed to have separate machines at home for classified and unclassified material, Top Secret communications intelligence files were found on his unclassified machine, which had been used to access high-risk web sites in his absence. Deutch said he was unwilling to use the classified CIA network for some purposes because of the risk that CIA colleagues might get access. A domestic servant, who was an alien, had access to his private machine. Nonetheless, the risk of compromise was held to be less than that of an intruder sneaking into his home to take an image of the disk. The report into this mess by the CIA Office of Inspector General makes instructive reading for anyone concerned with security usability [761]. I will have more to say on this topic in Part 3, and in the context of a number of case studies throughout this book.

All that said, the contribution of the MLS model is not all positive. There is a tactical problem, and a strategic one.

The tactical problem is that the existence of trusted system components, plus a large set of bureaucratic guidelines, has a strong tendency to displace critical thought. Instead of working out a system's security requirements in a methodical way, designers just choose what they think is the appropriate security class of component, then regurgitate the description of this class as the security specification of the overall system [624].

One should never lose sight of the human motivations that drive a system design, and the costs it imposes. Daniel Moynihan [562] provides a critical study of the real purposes and huge costs of obsessive secrecy in U.S. foreign and military affairs. Following a Senate enquiry, he discovered that President Truman was never told of the Venona decrypts because the material was considered 'Army Property'—despite its being the main motivation for the prosecution of Alger Hiss. As he writes in his book: "Departments and agencies hoard information, and the government becomes a kind of market. Secrets become organizational assets, never to be shared save in exchange for another organization's assets." Moynihan reports, for example, that in 1996, the num-

ber of original classification authorities decreased by 959 to 4,420 (following post-Cold-War budget cuts), but that the total of all classification actions reported for fiscal year 1996 increased by 62 percent to 5,789,625.

Yet despite the huge increase in secrecy, the quality of intelligence made available to the political leadership appears to have declined over time. Effectiveness is undermined by interagency feuding and refusal to share information, and by the lack of effective external critique.[1] A strong case can be made that MLS systems, by making the classification process easier but controlled data sharing harder, actually impair operational effectiveness.

So the strategic problem is that multilevel security has become so entrenched in government, and in some parts of industry, that it is often used in highly inappropriate ways. Even long-time intelligence insiders have documented this [425]. To solve many problems, we need to be a "fox" rather than a "hedgehog." Even where a simple, mandatory, access control system could be appropriate, we often need to control information flows across, rather than information flows down. Medical systems are a good example of this; and we will look at them in the next chapter.

7.7 Summary

Multilevel secure systems are used in a number of military applications, most notably specialized kinds of firewalls (mail guards and pumps). They may turn out to be excellent platforms for more general kinds of firewall or Web server. Their broader importance comes from two facts: they have been the main subject of computer security research since the mid-1970s, and their assumptions underlie many of the schemes used for security evaluation. It is important for practitioners to understand both the strengths and limitations of MLS systems, so that they can draw on the considerable research literature when it's appropriate, and avoid being dragged into error when it's not.

Research Problems

Multilevel confidentiality appears to have been "done to death" by generations of research students. The opportunities that remain may concern multilevel integrity, and the interaction that multilevel systems have with other security policies: how, for example, should a military hospital combine BLP with the bookkeeping and patient privacy policies discussed in the next two chapters?

[1] Although senior people follow the official line when speaking on the record, in private they rail at the penalties imposed by the bureaucracy. My favorite quip is from an exasperated British general: "What's the difference between Jurassic Park and the Ministry of Defence? One's a theme park full of dinosaurs, and the other's a movie!"

Further Reading

One of the better introductions to MLS systems, and especially the problems of databases, is Gollmann's *Computer Security* [344]. Amoroso's *Fundamentals of Computer Security Technology* [15] is the best introduction to the formal mathematics underlying the Bell-LaPadula, noninterference and nondeducibility security models.

The bulk of the published papers on engineering actual multilevel systems can be found in the annual proceedings of three conferences: the IEEE Symposium on Security & Privacy (known as "Oakland," as that's where it's held), the National Computer Security Conference (renamed the National Information Systems Security Conference in 1995), whose proceedings are published by the National Institute of Standards and Technology, and the Computer Security Applications Conference, whose proceedings are (like Oakland's) published by the IEEE. Fred Cohen's experiments on breaking MLS systems using viruses are described in his book, *A Short Course on Computer Viruses* [192]. Many of the classic early papers in the field can be found at the NIST archive [573]. Finally, the classic on the abuse of the classification process to cover up waste, fraud, and mismanagement in the public sector was written by Leslie Chapman [176].