CHAPTER

8

Multilevel Security

Most high assurance work has been done in the area of kinetic devices and infernal machines that are controlled by stupid robots. As information processing technology becomes more important to society, these concerns spread to areas previously thought inherently harmless, like operating systems.

— Earl Boebert

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

— British Civil Service Verb

They constantly try to escape
From the darkness outside and within
By dreaming of systems so perfect that no one will need to be good

— TS Eliot

8.1 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 only be read by a principal whose level is at least as high as the data's classification. The policies they implement are known as *multilevel secure* or alternatively as *mandatory access control* or MAC.

Multilevel secure systems are important because:

1. a huge amount of research has been done on them, thanks to military funding for computer science in the USA. 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;
2. although multilevel concepts were originally developed to support confidentiality in military systems, many commercial systems now use multilevel integrity policies. For example, telecomms operators want their billing system to be able to see what's happening in their switching system, but not affect it;
3. recently, products such as Microsoft Vista and Red Hat Linux have started to incorporate mandatory access control mechanisms, and they have also appeared in disguise in digital rights management systems. For example, Red Hat uses SELinux mechanisms developed by the NSA to isolate different servers running on a machine — so that even if your web server is hacked, it doesn't necessarily follow that your DNS server gets taken over too. Vista has a multilevel integrity policy under which Internet Explorer runs by default at 'Low' — which means that even if it gets taken over, the attacker should not be able to change system files, or anything else with a higher integrity level. These mechanisms are still largely invisible to the domestic computer user, but their professional use is increasing;
4. multilevel confidentiality ideas are often applied in environments where they're ineffective or even harmful, because of the huge vested interests and momentum behind them. This can be a contributory factor in the failure of large system projects, especially in the public sector.

Sir Isiah 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.

8.2 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, we 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, that 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 8.1 gives a simple example:

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 8.1: A typical corporate information security policy

This sort of waffle is very common but is useless to the security engineer.

Its first failing is 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 should logically 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 honour'? Fourth, how are breaches to be detected and who has a specific duty to report them?

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 which have come into use to describe the specification of protection requirements.

A *security policy model* is a succinct statement of the protection properties which 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 with 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* [935]. (I discuss the Common

Criteria in Part III; 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 either a security policy model or a security target. I will never use it 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, there already exists a model; 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 of the book 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) which a security target can use to meet them.

Finally, you may come across a third usage of the phrase 'security policy' — as a list of specific configuration settings for some protection product. We will refer to this as *configuration management*, or occasionally as *trusted configuration management*, in what follows.

8.3 The Bell-LaPadula Security Policy Model

The classic example of a security policy model was proposed by Bell and LaPadula in 1973, in response to US Air Force concerns over the security of time-sharing mainframe systems[1]. By the early 1970's, people had realised 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 26.2.4.) 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. (Viruses were not invented until the following decade; the 70's concern was about Trojans.) 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 I discuss below.

A study by James Anderson led the US government to conclude that a secure system should do one or two things well; and that these protection

[1]This built on the work of a number of other researchers: see section 9.2.1 below for a sketch of the technical history.

properties should be enforced by mechanisms which were simple enough to verify and that would change only rarely [29]. It introduced the concept of a *reference monitor* — a component of the operating system which 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, . . .) 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?

8.3.1 Classifications and Clearances

The Second World War, and the Cold War which followed, led NATO governments to move to a common protective marking scheme for labelling the sensitivity of documents. *Classifications* are labels, which run upwards from *Unclassified* through *Confidential*, *Secret* and *Top Secret* (see Figure 8.2.). 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 USA, for example, a 'Secret' clearance involves checking FBI fingerprint files, while 'Top Secret' also involves background checks for the previous five to fifteen years' employment [379].

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 upwards, from confidential to secret to top secret, but it may never flow downwards 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,

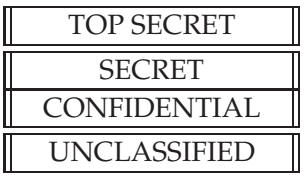

Figure 8.2: Multilevel security

while higher levels may require safes of an approved type, guarded rooms with control over photocopiers, and so on. (The NSA security manual [952] gives a summary of the procedures used with 'top secret' intelligence data.)

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. The UK has an extra level, 'Restricted', between 'Unclassified' and 'Confidential'; the USA used to have this too but abolished it after the Freedom of Information Act was introduced. America now has two more specific markings: 'For Official Use only' (FOUO) refers to unclassified data that can't be released under FOIA, while 'Unclassified but Sensitive' includes FOUO plus material which might be released in response to a FOIA request. In the UK, '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 US document which is sent across the Atlantic automatically becomes 'Restricted' in the UK and then 'Confidential' when shipped back to the USA. American military system builders complain that the UK policy breaks the US classification scheme!)

There is also a system of codewords whereby information, especially at Secret and above, can be further restricted. For example, information which might reveal 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 chapter on multilateral security.

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 'UK Eyes Only', or the US equivalent, 'NOFORN'; there are also *International Defence Organisation* markings such as *NATO*. The lack of obvious differences between codewords, descriptors, caveats and IDO marking is one of the things that can make the system confusing. A more detailed explanation can be found in [1051].

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 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 at all, let alone what data.

8.3.2 Information Flow Control

It was in this context of the classification of government data that the *Bell-LaPadula* or *BLP* model of computer security was formulated in 1973 [146]. It is also known as *multilevel security* and systems which implement it are often called *multilevel secure* or *MLS* systems. Their basic property is that information cannot flow downwards.

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 which would look for secret documents to copy. If it could then write 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 also assumed that most staff are careless, and some are dishonest; extensive operational security measures have long been used, especially in defence environments, to prevent people leaking paper documents. (When I worked in defense avionics as a youngster, all copies of circuit diagrams, blueprints etc were numbered and had to be accounted for.) So there was a pre-existing culture that security policy was enforced independently of user actions; the move to computers didn't change this. It had to be clarified, which is what Bell-LaPadula does: the security policy must be enforced not just independently of

users' direct actions, but of their indirect actions (such as the actions taken by programs they run).

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 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 like Unix where users can take their own access decisions about their files.

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 which 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, we'll ignore this; we'll also ignore the possibility of incompatible security levels for the time being, and return to them in the next chapter; and finally, in order to simplify matters still further, we 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 original idea was to implement a reference monitor by beefing up the part of an operating system which supervises all operating system calls and checks access permissions to decide whether the call can be serviced or not. However in practice things get much more complex as it's often 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 has got cheaper and faster is to replicate systems. This replication was often physical in the 1990s, and since about 2005 it may use virtual machines; some promising recent work builds on virtualization products such as VMware and Xen to provide multiple systems at different security levels on the same PC. One might, for example, have one database running at Low and another at High, on separate instances of Windows XP, with a *pump* that constantly copies information from Low up to High, all running on VMware on top of SELinux. I'll discuss pumps in more detail later.

8.3.3 The Standard Criticisms of Bell-LaPadula

The introduction of BLP caused a lot of excitement: here was a straightforward security policy which 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 change 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 off 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, her 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 sub-system and file handles, rather than processes, this means that state changes when access rights change, rather than when data actually moves.

The practical implication is that a process acquires the security labels of all the files it reads, and these become the default label set of every file that it writes. So a process which 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' then 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. Read-time changes in security level introduce the problem that access to resources can be revoked at any time, including in the middle of a transaction. Now the revocation problem is generally unsolvable in modern operating systems, at least in any complete form, which means that the applications have to cope somehow. Unless you invest some care and effort, you can easily find that everything ends up in the highest compartment — or that the system fragments into thousands of tiny compartments that don't communicate at all with each other. I'll discuss this in more detail in the next chapter.

Another problem with BLP, and indeed with all mandatory access control systems, is that separating users and processes is relatively straightforward; the hard part is when some controlled interaction is needed. Most real applications need some kind of 'trusted subject' that can break the security policy; an example is a trusted word processor that helps an intelligence analyst scrub a Top Secret document when she's editing it down to Secret [861]. BLP is silent on how the system should protect such an application. Does it become part of the Trusted Computing Base? I'll discuss this in more detail below.

Finally it's worth noting that even with the high-water-mark 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).

8.3.4 Alternative Formulations

Multilevel security properties have been expressed in several 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 [1334]. 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 [632].

Shortly after that, a number of teams produced primitive versions of the lattice model, which I'll discuss in more detail in the next chapter. These also made a significant contribution to the Bell-LaPadula work, as did engineers working on Multics. Multics had started as an MIT project in 1965 and developed into a Honeywell product; it became the template for the 'trusted systems' specified in the Orange Book, being the inspirational example of the B2 level operating system. The evaluation that was carried out on it by Paul Karger and Roger Schell was hugely influential and was the first appearance of the idea that malware could be hidden in the compiler [693] — which led to Ken Thompson's famous paper 'On Trusting Trust' ten years later. Multics itself developed into a system called SCOMP that I'll discuss in section 8.4.1 below.

Noninterference was introduced by Joseph Goguen and Jose Meseguer in 1982 [532]. 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 David Sutherland in 1986 [1233]. 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 compatible 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. (Quite a lot else 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 [1096] for an early example of such a system.)

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

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 [864] 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 which causes an access right to leak to somewhere it was initially not present [584]. This is more expressive than BLP, but is 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 [1357, 552]. 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 Dick Kain [198], assigns subjects to *domains* and objects to *types*, with matrices defining permitted domain-domain and domain-type interactions. This is used in a popular and important mandatory access control system, SELinux, which simplifies it by putting both subjects and objects in types and having a matrix of allowed type pairs [813]. In effect this is a second access-control matrix; in addition to having a user ID and group ID, each process has a security ID. The Linux Security Modules framework provides pluggable security modules with rules operating on SIDs.

Type enforcement was later extended by Badger and others to *Domain and Type Enforcement* [106]. They introduced their own language for configuration (DTEL), and implicit typing of files based on pathname; for example, all objects in a given subdirectory may be declared to be in a given domain. TE

and DTE are more general than simple MLS policies such as BLP, as they start to deal with integrity as well as confidentiality concerns. One of their early uses, starting in the LOCK system, was to enforce trusted pipelines: the idea is to confine a set of trusted processes in a pipeline so that each can only talk to previous stage and the next stage. This can be used to assemble guards and firewalls which cannot be bypassed unless at least two stages are compromised [963]. Type-enforcement mechanisms are used, for example, in the Sidewinder firewall. A further advantage of type enforcement mechanisms is that they can be aware of code versus data, and privileges can be bound to code; in consequence the tranquility problem can be dealt with at execute time rather than as data are read. This can make things much more tractable.

The downside of the greater flexibility and expressiveness of TE/DTE is that it is not always straightforward to implement BLP, because of the state explosion; when writing a security policy you have to consider all the possible interactions between different types. (For this reason, SELinux also implements a simple MLS policy. I'll discuss SELinux in more detail below.)

Finally, a policy model getting much attention from researchers in recent years is *role-based access control* (RBAC), introduced by David Ferraiolo and Richard Kuhn [466, 467]. This provides a more general framework for mandatory access control than BLP in which access decisions don't depend on users' names but on the functions which they are currently performing within the organization. Transactions which 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 be used to give finer-grained control, for example by granting different access rights to 'Ross as Professor', 'Ross as member of the Planning and Resources Committee' and 'Ross reading private email'. Implementations vary; the banking systems of twenty years ago kept the controls in middleware, and some modern RBAC products do control at the application layer where it's easy to bypass. SELinux builds it on top of TE, so that users are mapped to roles at login time, roles are authorized for domains and domains are given permissions to types. On such a platform, RBAC can usefully deal with integrity issues as well as confidentiality, by allowing role membership 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.

8.3.5 The Biba Model and Vista

The incorporation into Windows Vista of a multilevel integrity model has revived interest in a security model devised in 1975 by Ken Biba [168], which

textbooks often refer to as 'Bell-LaPadula upside down'. The Biba model deals with integrity alone and ignores confidentiality. 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 can write or alter it.

As a concrete application, an electronic medical device such as an ECG may have two separate modes: calibration and use. Calibration data must be protected from corruption 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 use a multilevel integrity policy with the rules that we can read data at higher levels (i.e., a user process can read the calibration data) and write to lower levels (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 discussed above: 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 signalling 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.

Vista marks file objects with an integrity level, which can be Low, Medium, High or System, and implements a default policy of NoWriteUp. Critical Vista files are at System and other objects are at Medium by default — except for Internet Explorer which is at Low. The effect is that things downloaded using IE can read most files in a Vista system, but cannot write them. The idea is to limit the damage that can be done by viruses and other malware. I'll describe Vista's mechanisms in more detail below.

An interesting precursor to Vista was LOMAC, a Linux extension that implemented a low water mark policy [494]. It provided two levels — high and low integrity — with system files at High and the network at Low. As soon as a program (such as a daemon) received traffic from the network, it was automatically downgraded to Low. Thus even if the traffic contains an attack that forks a root shell, this shell could not write to the password file as a normal root shell would. As one might expect, a number of system tasks (such as logging) became tricky and required trusted code.

As you might expect, Biba has the same fundamental problems as Bell-LaPadula. It cannot accommodate real-world operation very well without

numerous exceptions. For example, a real system will usually require 'trusted' subjects that can override the security model, but Biba on its own fails to provide effective mechanisms to protect and confine them; and in general it doesn't work so well with modern software environments. In the end, Vista dropped the NoReadDown restriction and did not end up using its integrity model to protect the base system from users.

Biba also cannot express many real integrity goals, like assured pipelines. In fact, the Type Enforcement model was introduced by Boebert and Kain as an alternative to Biba. It is unfortunate that Vista didn't incorporate TE.

I will consider more complex models when I discuss banking and bookkeeping systems in Chapter 10; these are more complex in that they retain security state in the form of dual control mechanisms, audit trails and so on.

8.4 Historical Examples of MLS Systems

Following some research products in the late 1970's (such as KSOS [166], a kernelised secure version of Unix), products that implemented multilevel security policies started arriving in dribs and drabs in the early 1980's. By about 1988, a number of companies started implementing MLS versions of their operating systems. MLS concepts were extended to all sorts of products.

8.4.1 SCOMP

One of the most important products was the *secure communications processor* (SCOMP), a derivative of Multics launched in 1983 [491]. This was a no-expense-spared implementation of what the US Department of Defense believed it wanted for handling messaging at multiple levels of classification. It 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*. These are specialised firewalls which typically allow mail to pass from Low to High but not vice versa [369]. (In general, a device which does this is known as a *data diode*.) SCOMP's successor, XTS-300, supported C2G, the Command and Control Guard. This was used in the time phased force deployment data (TPFDD) system whose function was to plan US troop movements and associated logistics. Military plans are developed as TPFDDs at a high classification level, and then distributed at the appropriate times as commands to lower levels for implementation. (The issue of how high information is deliberately downgraded raises a number of issues, some of which I'll deal

with below. 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* [375] — the US Trusted Computer Systems Evaluation Criteria. This was the first systematic set of standards for secure computer systems, being introduced in 1985 and finally retired in December 2000. The Orange Book was enormously influential not just in the USA but among allied powers; countries such as the UK, Germany, and Canada based their own national standards on it, until these national standards were finally subsumed into the Common Criteria [935].

The Orange Book allowed systems to be evaluated at a number of levels with A1 being the highest, and moving downwards 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 was only evaluated to B3 (the formal proofs of correctness required for an A1 evaluation were dropped).

8.4.2 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 then 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 [1335]. As late as 1994, it was the only communications security device with an A1 evaluation [161]. So it too had an effect on later systems. It was not widely used though, and its successor (the Motorola Network Encryption System), had only a B2 evaluation.

8.4.3 MLS Unix and Compartmented Mode Workstations

MLS versions of Unix started to appear in the late 1980's, such as AT&T's System V/MLS [27]. 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

SecureWare (and its derivatives, such as SCO and HP VirtualVault), and Addamax. By the time of writing (2007), Sun's Solaris has emerged as the clear market leader, being the platform of choice for high-assurance server systems and for many clients as well. Trusted Solaris 8 gave way to Solaris trusted Extensions 10, which now been folded into Solaris, so that every copy of Solaris contains MLS mechanisms, for those knowledgeable enough to use them.

Comparted Mode Workstations (CMWs) are an example of MLS clients. They 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 which 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 [631]. For the engineering issues involved in doing mandatory access control in windowing systems, see [437, 438] 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 which 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 [451].

8.4.4 The NRL Pump

It was soon realised 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.

The US Naval Research Laboratory (NRL) therefore developed the *Pump* — a one-way data transfer device (a data diode) to allow secure one-way information flow (Figure 8.3). The main problem is that while sending data from Low to High is easy, the need for assured transmission reliability means that acknowledgement messages must be sent back from High to Low. The Pump limits the bandwidth of possible backward leakage using a number of mechanisms such as using buffering and randomizing the timing of acknowledgements [685, 687, 688]. The attraction of this approach is that one can build MLS systems 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

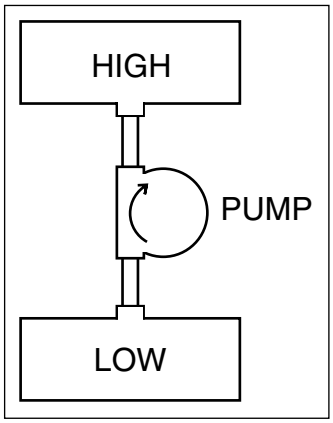

Figure 8.3: The NRL pump

built from cheap commercial-off-the-shelf (COTS) components [689]. As the cost of hardware falls, this is often the preferred option where it's possible. The pump's story is told in [691].

The Australian government developed a product called *Starlight* that uses pump-type technology married with a keyboard switch to provide a nice MLS-type windowing system (albeit without any visible labels) using a bit of trusted hardware which connects the keyboard and mouse with High and Low systems [30]. There is no trusted software. It's been integrated with the NRL Pump [689]. A number of semi-commercial data diode products have also been introduced.

8.4.5 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 USA and UK have ended up as expensive disasters. In the UK, the Royal Air Force's Logistics Information Technology System (LITS) was a 10 year (1989–99), £500m project to provide a single stores management system for the RAF's 80 bases [932]. 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 with 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). In order 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 [910].

8.4.6 Sybard Suite

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 was the 'Purple Penelope' software, from Qinetiq in the UK, now sold as Sybard Suite. This puts an MLS wrapper round a Windows workstation, implementing the high water mark version of BLP. It displays in the background the current security level of the device and upgrades 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, the user must confirm the release of the data using a trusted path interface, thus ensuring no Trojan Horse or virus can release anything completely unnoticed. Of course, a really clever malicious program can piggy-back 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 malware) can be traced after the fact [1032]. The security policy was described to me by one of its authors as 'we accept that we can't stop people leaking the order of battle to the Guardian newspaper if they really want to; we just want to make sure we arrest the right person for it.'

8.4.7 Wiretap Systems

One of the large applications of MLS is in wiretapping systems. Communications intelligence is generally fragile; once a target knows his traffic is being read he can usually do something to frustrate it. Traditional wiretap kit, based on 'loop extenders' spliced into the line, could often be detected by competent targets; modern digital systems try to avoid these problems, and

provide a multilevel model in which multiple agencies at different levels can monitor a target, and each other; the police might be tapping a drug dealer, and an anti-corruption unit watching the police, and so on. Wiretaps are commonly implemented as conference calls with a silent third party, and the main protection goal is to eliminate any covert channels that might disclose the existence of surveillance. This is not always met. For a survey, see [1161], which also points out that the pure MLS security policy is insufficient: suspects can confuse wiretapping equipment by introducing bogus signalling tones. The policy should thus have included resistance against online tampering.

Another secondary protection goal should have been to protect against software tampering. In a recent notorious case, a wiretap was discovered on the mobile phones of the Greek Prime Minister and his senior colleagues; this involved unauthorised software in the mobile phone company's switchgear that abused the lawful intercept facility. It was detected when the buggers' modifications caused some text messages not to be delivered [1042]. The phone company was fined 76 million Euros (almost $100m). Perhaps phone companies will be less willing to report unauthorized wiretaps in future.

8.5 Future MLS Systems

In the first edition of this book, I wrote that the MLS industry's attempts to market its products as platforms for firewalls, web servers and other exposed systems were failing because 'the BLP controls do not provide enough of a protection benefit in many commercial environments to justify their large development costs, and widely fielded products are often better because of the evolution that results from large-scale user feedback'. I also noted research on using mandatory access controls to accommodate both confidentiality and integrity in environments such as smartcards [692], and to provide real-time performance guarantees to prevent service denial attacks [889]. I ventured that 'perhaps the real future of multilevel systems is not in confidentiality, but integrity'.

The last seven years appear to have proved this right.

8.5.1 Vista

Multilevel integrity is coming to the mass market in Vista. As I already mentioned, Vista essentially uses the Biba model. All processes do, and all securable objects (including directories, files and registry keys) may, have an integrity-level label. File objects are labelled at 'Medium' by default, while Internet Explorer (and everything downloaded using it) is labelled 'Low'. User action is therefore needed to upgrade downloaded content before it can modify

existing files. This may not be a panacea: it may become so routine a demand from all installed software that users will be trained to meekly upgrade viruses too on request. And it must be borne in mind that much of the spyware infesting the average home PC was installed there deliberately (albeit carelessly and with incomplete knowledge of the consequences) after visiting some commercial website. This overlap between desired and undesired software sets a limit on how much can be achieved against downloaded malware. We will have to wait and see.

It is also possible to implement a crude BLP policy using Vista, as you can also set 'NoReadUp' and 'NoExecuteUp' policies. These are not installed as default; the reason appears to be that Microsoft was principally concerned about malware installing itself in the system and then hiding. Keeping the browser 'Low' makes installation harder, and allowing all processes (even Low ones) to inspect the rest of the system makes hiding harder. But it does mean that malware running at Low can steal all your data; so some users might care to set 'NoReadUp' for sensitive directories. No doubt this will break a number of applications, so a cautious user might care to have separate accounts for web browsing, email and sensitive projects. This is all discussed by Joanna Rutkowska in [1099]; she also describes some interesting potential attacks based on virtualization. A further problem is that Vista, in protected mode, does still write to high-integrity parts of the registry, even though Microsoft says it shouldn't [555].

In passing, it's also worth mentioning rights management, whether of the classical DRM kind or the more recent IRM variety, as a case of mandatory access control. Vista, for example, tries to ensure that no high definition video content is ever available to an untrusted process. I will discuss it in more detail later, but for now I'll just remark that many of the things that go wrong with multilevel systems might also become vulnerabilities in, or impediments to the use of, rights-management systems. Conversely, the efforts expended by opponents of rights management in trying to hack the Vista DRM mechanisms may also open up holes in its integrity protection.

8.5.2 Linux

The case of SELinux and Red Hat is somewhat similar to Vista in that the immediate goal of the new mandatory access control mechanisms is also to limit the effects of a compromise. SELinux [813] is based on the Flask security architecture [1209], which separates the policy from the enforcement mechanism; a security context contains all of the security attributes associated with a subject or object in Flask, where one of those attributes includes the Type Enforcement type attribute. A security identifier is a handle to a security context, mapped by the security server. It has a security server where policy decisions are made, this resides in-kernel since Linux has a monolithic kernel

and the designers did not want to require a kernel-userspace call for security decisions (especially as some occur on critical paths where the kernel is holding locks) [557]). The server which provides a general security API to the rest of the kernel, with the security model hidden behind that API. The server internally implements RBAC, TE, and MLS (or to be precise, a general constraints engine that can express MLS or any other model you like). SELinux is included in a number of Linux distributions, and Red Hat's use is typical. There its function is to separate various services. Thus an attacker who takes over your web server does not thereby acquire your DNS server as well.

Suse Linux has taken a different path to the same goal. It uses AppArmor, a monitoring mechanism maintained by Novell, which keeps a list of all the paths each protected application uses and prevents it accessing any new ones. It is claimed to be easier to use than the SELinux model; but operating-system experts distrust it as it relies on pathnames as the basis for its decisions. In consequence, it has ambiguous and mutable identifiers; no system view of subjects and objects; no uniform abstraction for handling non-file objects; and no useful information for runtime files (such as /tmp). By forcing policy to be written in terms of individual objects and filesystem layout rather than security equivalence classes, it makes policy harder to analyze. However, in practice, with either AppArmor or SELinux, you instrument the code you plan to protect, watch for some months what it does, and work out a policy that allows it to do just what it needs. Even so, after you have fielded it, you will still have to observe and act on bug reports for a year or so. Modern software components tend to be so complex that figuring out what access they need is an empirical and iterative process[2].

It's also worth bearing in mind that simple integrity controls merely stop malware taking over the machine — they don't stop it infecting a Low compartment and using that as a springboard from which to spread elsewhere.

Integrity protection is not the only use of SELinux. At present there is considerable excitement about it in some sections of government, who are excited at the prospect of 'cross department access to data . . . and a common trust infrastructure for shared services' (UK Cabinet Office) and allowing 'users to access multiple independent sessions at varying classification levels' (US Coast Guard) [505]. Replacing multiple terminals with single ones, and moving from proprietary systems to open ones, is attractive for many reasons — and providing simple separation between multiple terminal emulators or browsers running on the same PC is straightforward. However, traditional MAC might not be the only way to do it.

[2]Indeed, some of the mandatory access control mechanisms promised in Vista — such as remote attestation — did not ship in the first version, and there have been many papers from folks at Microsoft Research on ways of managing access control and security policy in complex middleware. Draw your own conclusions.

8.5.3 Virtualization

Another technological approach is virtualization. Products such as VMware and Xen are being used to provide multiple virtual machines at different levels. Indeed, the NSA has produced a hardened version of VMware, called NetTop, which is optimised for running several Windows virtual machines on top of an SELinux platform. This holds out the prospect of giving the users what they want — computers that have the look and feel of ordinary windows boxes — while simultaneously giving the security folks what they want, namely high-assurance separation between material at different levels of classification. So far, there is little information available on NetTop, but it appears to do separation rather than sharing.

A current limit is the sheer technical complexity of modern PCs; it's very difficult to find out what things like graphics cards actually do, and thus to get high assurance that they don't contain huge covert channels. It can also be quite difficult to ensure that a device such as a microphone or camera is really connected to the Secret virtual machine rather than the Unclassified one. However, given the effort being put by Microsoft into assurance for high-definition video content, there's at least the prospect that some COTS machines might eventually offer reasonable assurance on I/O eventually.

The next question must be whether mandatory access control for confidentiality, as opposed to integrity, will make its way out of the government sector and into the corporate world. The simplest application might be for a company to provide its employees with separate virtual laptops for corporate and home use (whether with virtualisation or with mandatory access controls). From the engineering viewpoint, virtualization might preferable, as it's not clear that corporate security managers will want much information flow between the two virtual laptops: flows from 'home' to 'work' could introduce malware while flow from 'work' to 'home' could leak corporate secrets. From the business viewpoint, it's less clear that virtualization will take off. Many corporates would rather pretend that company laptops don't get used for anything else, and as the software industry generally charges per virtual machine rather than per machine, there could be nontrivial costs involved. I expect most companies will continue to ignore the problem and just fire people whose machines cause visible trouble.

The hardest problem is often managing the interfaces between levels, as people usually end up having to get material from one level to another in order to get their work done. If the information flows are limited and easy to model, as with the Pump and the CMW, well and good; the way forward may well be Pump or CMW functionality also hosted on virtual machines. (Virtualisation per se doesn't give you CMW — you need a trusted client for that, or an app running on a trusted server — but it's possible to envisage a trusted thin client plus VMs at two different levels all running on the same box.

So virtualization should probably be seen as complementary to mandatory access control, rather than a competitor.

But many things can go wrong, as I will discuss in the next session.

8.5.4 Embedded Systems

There are more and more fielded systems which implement some variant of the Biba model. As well as the medical-device and railroad signalling applications already mentioned, there are utilities. In an electricity utility, for example, operational systems such as power dispatching should not be affected by any others. The metering systems can be observed by, but not influenced by, the billing system. Both billing and power dispatching feed information into fraud detection, and at the end of the chain the executive information systems can observe everything while having no direct effect on operations. These one-way information flows can be implemented using mandatory access controls and there are signs that, given growing concerns about the vulnerability of critical infrastructure, some utilities are starting to look at SELinux.

There are many military embedded systems too. The primitive mail guards of 20 years ago have by now been supplanted by guards that pass not just email but chat, web services and streaming media, often based on SELinux; an example is described in [478]. There are many more esoteric applications: for example, some US radars won't display the velocity of a US aircraft whose performance is classified, unless the operator has the appropriate clearance. (This has always struck me as overkill, as he can just use a stopwatch.)

Anyway, it's now clear that many of the lessons learned in the early multilevel systems go across to a number of applications of much wider interest. So do a number of the failure modes, which I'll now discuss.

8.6 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.

8.6.1 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

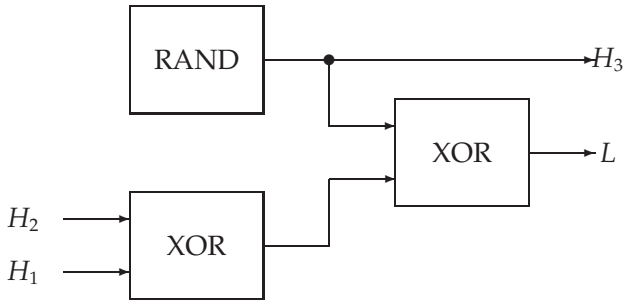

Figure 8.4: Insecure composition of secure systems with feedback

ciphertext, which being encrypted with a cipher system giving perfect secrecy, is considered to be low (output L), as in Figure 8.4.

In isolation, this device is provably secure. However, 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 [854]). 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 low-level problems arise when some sort of feedback is introduced into the system; without it, composition can be achieved under a number of formal models [865]. However, in real life, feedback is pervasive, and composition of security properties can be made even harder by detailed interface issues, feature interactions and so on. For example, one system might produce data at such a rate as to perform a service-denial attack on another. (I'll discuss some of the interface problems with reference monitors in detail in Chapter 18, 'API Attacks'.)

Finally, the composition of secure components or systems is very often frustrated by higher-level incompatibilities. Components might have been designed in accordance with two different security policies, or designed according to requirements that are inconsistent or even incompatible. 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. Composability is a long-standing and very serious problem with trustworthy systems; a good recent survey is the final report of the CHATS project [963].

8.6.2 The Cascade Problem

An example of the difficulty of composing multilevel secure systems is given by the cascade problem (Figure 8.5). After the Orange book introduced a series

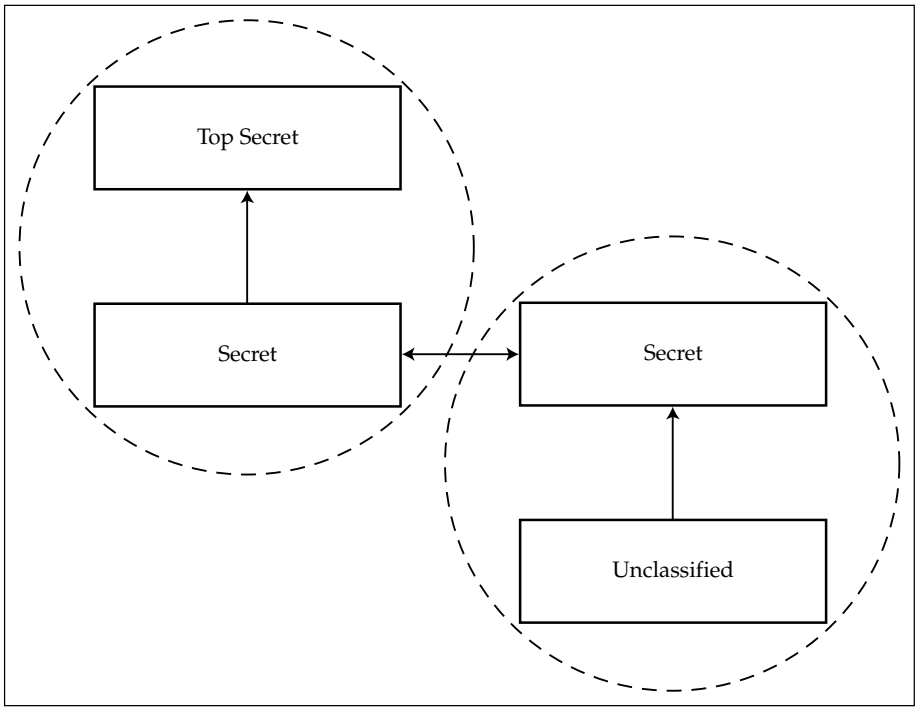

Figure 8.5: The cascade problem

of graduated evaluation levels, this led to led to rules about the number of levels which a system can span [379]. For example, a system evaluated to B3 was in general allowed to process information at Unclassified, Confidential and Secret, or at Confidential, Secret and Top Secret; there was no system permitted to process Unclassified and Top Secret data simultaneously [379].

As the diagram shows, it is straightforward to connect together two B3 systems in such a way that this security policy is broken. The first system connects together Unclassified, Confidential and Secret, and its Confidential and Secret levels communicate with the second system which also processes Top Secret information. (The problem's discussed in more detail in [622].) This illustrates another kind of danger which formal models of security (and practical implementations) must take into account.

8.6.3 Covert Channels

One of the reasons why these span limits are imposed on multilevel systems emerges from a famous — and extensively studied — problem: the *covert channel*. First pointed out by Lampson in 1973 [768], a covert channel is a mechanism that was not designed for communication but which 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 one 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 Top Secret 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 downwards; and 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 enormously 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 [686].)

It is also possible to limit the covert channel capacity by introducing noise. Some machines have had randomised 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 [554].)

Many covert channels occur at the application layer, and are a real concern to security engineers (especially as they are often overlooked). An example from social care is a UK proposal to create a national database of all children, for child-protection and welfare purposes, containing a list of all professionals with which each child has contact. Now it may be innocuous that child X is registered with family doctor Y, but the fact of a child's registration with a social work department is not innocuous at all — it's well known to be stigmatizing. For example, teachers will have lower expectations of children whom they know to have been in contact with social workers. So it is quite reasonable for parents (and children) to want to keep any record of such contact private [66].

A more subtle example is that in general personal health information derived from visits to genitourinary medicine clinics is High in the sense that it can't be shared with the patient's normal doctor and thus 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 [886]. The insurer knew that a smear test had been done already by the clinic, and didn't want to pay twice.

Another case of general interest arises in multilevel integrity systems such as banking and utility billing, where a programmer who has inserted Trojan code in a 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 in the specific case of bookkeeping.

The highest-bandwidth covert channel of which I'm aware 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 best that developers have been able to do consistently with BLP confidentiality protection in regular time-sharing operating systems is to limit it to 1 bit per second or so. (That is a DoD target [376], and techniques for doing a systematic analysis may be found in Kemmerer [706].) 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. It 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.

8.6.4 The Threat from Viruses

The vast majority of viruses are found on mass-market products such as PCs. However, the defense computer community was shocked when Cohen used viruses to penetrate multilevel secure systems easily in 1983. In his first experiment, a file virus which took only eight hours to write managed to penetrate a system previously believed to be multilevel secure [311].

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, then a virus could deliver the entire system to the attacker, for example by issuing him with an unauthorised clearance. For this reason, slightly looser rules apply to so-called *closed security environments* which are defined to be those where 'system applications are adequately protected against the insertion of malicious logic' [379]. But even if the TCB remains intact, the virus could still use any available covert channel to signal information down.

So 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 reasonably possible to stop them signalling back down.

8.6.5 Polyinstantiation

Another problem that has much exercised the research community is *polyinstantiation*. Suppose that our High user has created a file named `agents`, and our Low user now tries to do the same. If the MLS operating system prohibits him, it will have leaked information — namely that there is a file called `agents` at High. But if it lets him, it will now have two files with the same name.

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 [1112]. 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 favoured in the USA 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 Figure 8.6.

In the UK, 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 in Figure 8.7.

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

Figure 8.6: How the USA deals with classified data

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

Figure 8.7: How the UK deals with classified data

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 in order to get their work done. (In practice, of course, cover stories still get used in order not to advertise the existence of a covert mission any more than need be.)

There may be an interesting new application to the world of online gaming. Different countries have different rules about online content; for example, the USA limits online gambling, while Germany has strict prohibitions on the display of swastikas and other insignia of the Third Reich. Now suppose a second-world-war reenactment society wants to operate in Second Life. If a German resident sees flags with swastikas, an offence is committed there. Linden Labs, the operator of Second Life, has suggested authenticating users' jurisdictions; but it's not enough just to exclude Germans, as one of them might look over the fence. An alternative proposal is to tag alternative objects for visibility, so that a German looking at the Battle of Kursk would see only inoffensive symbols. Similarly, an American looking at an online casino might just see a church instead. Here too the lie has its limits; when the American tries to visit that church he'll find that he can't get through the door.

8.6.6 Other Practical Problems

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

1. 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.
2. 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 [1044].
3. Many applications need to be rewritten or at least greatly modified to run under MLS operating systems [1092]. For example, compartmented mode workstations 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 which tries to manipulate the colour map. Access to files might be quite different, as well as the format of things like access control lists. Another source of conflict with commercial software is the licence server; if a High user invokes an application, which goes to a licence server for permission to execute, then an MLS operating system will promptly reclassify the server High and deny access to Low users. So in practice, you usually end up (a) running two separate

license servers, thus violating the license terms, or (b) you have an MLS license server which tracks licenses at all levels (this restricts your choice of platforms), or (c) you only access the licensed software at one of the levels.

4. 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 of all this is a chronic tendency for things to be overclassified.
5. 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 acknowledgement 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.
6. The classification of data can get complex:
   - 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 at '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 once the file is downgraded. So downgrading procedures may involve all sorts of special filters, such as lossy compression of images and word processors which scrub and reformat text, in the hope that the only information remaining is that which lies in plain sight. (I will discuss information hiding in more detail in the context of copyright marking.)
   - 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 may 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 will look at this issue — the 'aggregation problem' — in more detail in section 9.3.2.)

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

7. There are always system components — such as memory management — that must be able to read and write at all levels. This objection is dealt with by abstracting it away, and assuming that memory management is part of the trusted computing base which enforces our mandatory access control policy. The practical outcome is that often a quite uncomfortably large part of the operating system (plus utilities, plus windowing system software, plus middleware such as database software) ends up part of the trusted computing base. 'TCB bloat' constantly pushes up the cost of evaluation and reduces assurance.
8. 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 they 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 UK law to extend wiretaps to Internet Service Providers (ISPs). (I'll discuss wiretapping in Part III). 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 Internet Service Providers to support an MLS security policy — which would have been totally impractical. So the UK government had to declare that it wouldn't apply the laid down standards in this case because of cost.

8.7 Broader Implications of MLS

The reader's reaction by this point may well be that mandatory access control is too hard to do properly; there are just too many complications. This may be true, and we are about to see the technology seriously tested as it's deployed in hundreds of millions of Vista PCs and Linux boxes. We will see to what extent mandatory access control really helps contain the malware threat, whether to commodity PCs or to servers in hosting centres. We'll also see whether variants of the problems described here cause serious or even fatal problems for the DRM vision.

However it's also true that Bell-LaPadula and Biba are the simplest security policy models we know of, and 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 in which a secure system could 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's talking to a genuine copy of the operating system?), *Trusted Distribution* (how does a user know he's 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 led to many things being highlighted which previously were glossed over. The resulting lessons can be applied to systems with quite different policies.

These lessons were set out in the 'Rainbow Series' of books on computer security, 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 the different coloured covers by which they're known. The series did a lot to raise consciousness of operational and evaluation issues that 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. I will have much more to say on this topic in Part III, and in the context of a number of case studies throughout this book.

Apart from the official 'lessons learned' from MLS, there have been other effects noticed over the years. In particular, the MLS program has had negative effects on many of the government institutions that used it. 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 and then regurgitate the description of this class as the security specification of the overall system [1044].

One should never lose sight of the human motivations which drive a system design, and the costs which it imposes. Moynihan's book [907] provides a critical study of the real purposes and huge costs of obsessive secrecy in US 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 his book puts it: '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.' He reports, for example, that in 1996 the number 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.

I wrote in the first edition in 2001: '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 inter-agency feuding and refusal to share information, and by the lack of effective external critique[3]. So a strong case can be made that MLS systems, by making the classification process easier and controlled data sharing harder, actually impair operational effectiveness'. A few months after the book was published, the attacks of 9/11 drove home the lesson that the US intelligence community, with its resources fragmented into more than twenty agencies and over a million compartments, was failing to join up the dots into an overall picture. Since then, massive efforts have been made to get the agencies to share data. It's not clear that this is working; some barriers are torn down, others are erected, and bureaucratic empire building games continue as always. There have, however, been leaks of information that the old rules should have prevented. For example, a Bin Laden video obtained prior to its official release by Al-Qaida in September 2007 spread rapidly through U.S. intelligence agencies and was leaked by officials to TV news, compromising the source [1322].

In the UK, the system of classification is pretty much the same as the U.S. system described in this chapter, but the system itself is secret, with the full manual being available only to senior officials. This was a contributory factor in a public scandal in which a junior official at the tax office wrote a file containing the personal information of all the nation's children and their families to two CDs, which proceeded to get lost in the post. He simply was not aware that data this sensitive should have been handled with more care [591]. I'll describe this scandal and discuss its implications in more detail in the next chapter.

So multilevel security can be a double-edged sword. It has become entrenched in government, and in the security-industrial complex generally, and is often used in inappropriate ways. Even long-time intelligence insiders have documented this [671]. There are many problems which we need to be a 'fox' rather than a 'hedgehog' to solve. 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 next.

[3] Although senior people follow the official line when speaking on the record, once 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!'

8.8 Summary

Mandatory access control was developed for military applications, most notably specialized kinds of firewalls (guards and pumps). They are being incorporated into commodity platforms such as Vista and Linux. They have even broader importance in that they have been the main subject of computer security research since the mid-1970's, and their assumptions underlie many of the schemes used for security evaluation. It is important for the practitioner to understand both their strengths and limitations, so that you 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 comprehensively done to death by generations of DARPA-funded research students. The opportunity now is to explore what can be done with the second-generation mandatory access control systems shipped with Vista and SELinux, and with virtualization products such as VMware and Xen; what can be done to make it easier to devise policies for these systems that enable them to do useful work; in better mechanisms for controlling information flow between compartments; the interaction which multilevel systems have with other security policies; and in ways to make mandatory access control systems usable.

An ever broader challenge, sketched out by Earl Boebert after the NSA launched SELinux, is to adapt mandatory access control mechanisms to safety-critical systems (see the quote at the head of this chapter, and [197]). As a tool for building high-assurance, special-purpose devices where the consequences of errors and failures can be limited, mechanisms such as type enforcement and role-based access control look like they will be useful outside the world of security. By locking down intended information flows, designers can reduce the likelihood of unanticipated interactions.

Further Reading

The report on the Walker spy ring is essential reading for anyone interested in the system of classifications and clearances [587]: this describes in great detail the system's most spectacular known failure. It brings home the sheer complexity of running a system in which maybe three million people have a current SECRET or TOP SECRET clearance at any one time, with a million applications being processed each year — especially when the system was

designed on the basis of how people should behave, rather than on how they actually do behave. And the classic on the abuse of the classification process to cover up waste, fraud and mismanagement in the public sector was written by Chapman [282].

On the technical side, one of the better introductions to MLS systems, and especially the problems of databases, is Gollmann's *Computer Security* [537]. Amoroso's *'Fundamentals of Computer Security Technology'* [27] 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 MLS 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 were published by NIST until the conference ended in 1999, 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'* [311]. Many of the classic early papers in the field can be found at the NIST archive [934].