PART II

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

There are three broad themes. Chapters 9–12 look at conventional computer security issues, and by discussing what we're trying to do and how it's done in different environments – the military, healthcare, the census and banking – we introduce security policy models which set out the protection concepts that real systems try to implement. These range from multilevel security through compartments to anonymisation and internal control. We introduce our first detailed case studies, from government networks through medical records to payment systems.

Chapters 13–20 look at the hardware and system engineering aspects of information security. This ranges from biometrics, through the design of hardware security mechanisms from physical locks, security printing and seals, to chip-level tamper-resistance and emission security. We study applications that illustrate these technologies, ranging from burglar alarms through curfew tags, utility meters and payment cards to the control of nuclear weapons. We end up with a chapter on advanced cryptographic engineering, where hardware and software security meet: there we discuss topics from secure messaging and anonymous communications through hardware security modules and enclaves to blockchains.

Our third theme is attacks on networks and on highly-networked systems. We start off in Chapter 21 with attacks on computer networks and defensive technologies ranging from firewalls to PKI. We then study the phone ecosystems in Chapter 22, from bad apps down through switching exploits and out to the policy tussles over 5G. Chapter 23 tackles electronic and information warfare, showing how far techniques of denial, deception and exploitation can be taken by a serious combatants, and helping us hone our appreciation of anonymity and traffic analysis. Chapter 24 shows how some of these techniques are adapted in systems for digital rights management.

Finally, in Chapter 25 I present four areas of bleeding-edge security research in 2020. First are autonomous vehicles, including the 'autopilot' systems starting to appear in family cars. Second, we look at the machine-learning systems on which such vehicles increasingly rely, and which are starting to be used in many other applications. Third, we work through the realistic options for people to protect ourselves using privacy technology and operational security measures in a world where surveillance is being turbocharged by machine-learning techniques. Finally, we look at elections, which are becoming ever more fraught with claims (whether true or false) of interference and cheating.

This ordering tries to give the chapters a logical progression. Thus, for example, I discuss frauds against magnetic stripe bank cards before going on to describe the smartcards which replaced them and the phone payment systems which rely on smartcards for the SIMs that authenticate devices to the network, but which have so many more vulnerabilities thanks to the rich environment that has evolved since the launch of the iPhone.

Often a technology has evolved through a number of iterations over several applications. In such cases I try to distill what I know into a history. It can be confusing and even scary when you first dive into a 5,000-page manual for something that's been evolving for thirty years like the card payment system, or an Intel or Arm CPU; the story of how it evolved and why is often what you need to make sense of it.

CHAPTER

9

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

The password on the government phone always seemed to drop, and I couldn't get into it.

– US diplomat and former CIA officer KURT VOLKER, explaining why he texted from his personal phone

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

– BRITISH CIVIL SERVICE VERB

9.1 Introduction

In the next few chapters I'm going to explore the concept of a security policy using case studies. A security policy is a succinct description of what we're trying to achieve; it's driven by an understanding of the bad outcomes we wish to avoid and in turn drives the engineering. After I've fleshed out these ideas a little, I'll spend the rest of this chapter exploring the *multilevel security* (MLS) policy model used in many military and intelligence systems, which hold information at different levels of classification (Confidential, Secret, Top Secret, …), and have to ensure that data can be read only by a principal whose clearance level is at least as high. Such policies are increasingly also known as *information flow control* (IFC).

They are important for a number of reasons, even if you're never planning to work for a government contractor:

1. from about 1980 to about 2005, the US Department of Defense spent several billion dollars funding research into multilevel security. So the

model was worked out in great detail, and we got to understand the second-order effects of pursuing a single policy goal with great zeal;

2. the *mandatory access control* (MAC) systems used to implement it have now appeared in all major operating systems such as Android, iOS and Windows to protect core components against tampering by malware, as I described in Chapter 6;
3. although multilevel security concepts were originally developed to support confidentiality in military systems, many commercial systems now use multilevel integrity policies. For example, safety-critical systems use a number of safety integrity levels[1].

The poet Archilochus famously noted that a fox knows many little things, while a hedgehog knows one big thing. Security engineering is usually in fox territory, but multilevel security is an example of the hedgehog approach.

9.2 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 functionality and the safety case for safety. Like the specification, its primary function is to communicate.

Many organizations use the phrase 'security policy' to mean a collection of vapid statements, as in Figure 9.1:

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 9.1: typical corporate policy language

[1]Beware though that terminology varies between different safety-engineering disciplines. The safety integrity levels in electricity generation are similar to Biba, while automotive safety integrity levels are set in ISO 26262 as a hazard/risk metric that depends on the likelihood that a fault will cause an accident, together with the expected severity and controllability.

This sort of language is common, but useless – at least to the security engineer. It dodges the central issue, namely 'Who determines "need-to-know" and how?' Second, it mixes statements at 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?

When you think about it, this is political language. A politician's job is to resolve the tensions in society, and this often requires vague language on which different factions can project their own wishes; corporate executives are often operating politically, to balance different factions within a company[2].

Because the term 'security policy' is often abused to mean using security for politics, more precise terms have come into use by security engineers.

A *security policy model* is a succinct statement of the protection properties that a 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 a manner that is independent of the implementation, so as 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* [1398]. (I discuss the Common Criteria in section 28.2.7; they are 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're 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

[2]Big projects often fail in companies when the specification becomes political, and they fail even more often when run by governments – issues I'll discuss further in Part 3.

a security target. Neither of these steps is easy. In this section of the book, I 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.

9.3 Multilevel security policy

On March 22, 1940, President Roosevelt signed Executive Order 8381, enabling certain types of information to be classified Restricted, Confidential or Secret [980]. President Truman later added a higher level of Top Secret. This developed into a common protective marking scheme for the sensitivity of documents, and was adopted by NATO governments too in the Cold War. *Classifications* are labels, which run upwards from *Unclassified* through *Confidential*, *Secret* and *Top Secret* (see Figure 9.2). 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 and contractors 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 plus an interview and often a polygraph test [548]. Candidates have to disclose all their sexual partners in recent years and all material that might be used to blackmail them, such as teenage drug use or gay affairs[3].

The access control policy was simple: you can read a document only if your clearance is 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. So information may only flow upwards, from confidential to secret to top secret, but never downwards – unless an authorized person takes a deliberate decision to declassify it.

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. Information that is neither classified nor public is known as 'Controlled Unclassified Information' (CUI) in the USA while Britain uses 'Official'[4].

[3]In June 2015, the clearance review data of about 20m Americans was stolen from the Office of Personnel Management by the Chinese intelligence services. By then, about a million Americans had a Top Secret clearance; the OPM data also covered former employees and job applicants, as well as their relatives and sexual partners. With hindsight, collecting all the dirt on all the citizens with a sensitive job may not have been a great idea.

[4]Prior to adopting the CUI system, the United States had more than 50 different markings for data that was controlled but not classified, including For Official Use Only (FOUO), Law Enforcement Sensitive (LES), Proprietary (PROPIN), Federal Tax Information (FTI), Sensitive but Unclassified (SBU), and many, many others. Some agencies made up their own labels, without any

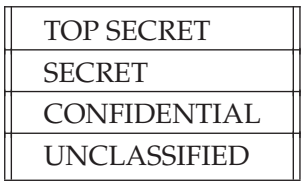

Figure 9.2: multilevel security

There is also a system of codewords whereby information, especially at Secret and above, can be restricted further. For example, information that might reveal intelligence sources or methods – such as the identities of agents or decryption capabilities – 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 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. Compartmentation is typically implemented nowadays using discretionary access control mechanisms; I'll discuss it in the next chapter.

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"; they do create restrictions. There are also *International Defence Organisation* markings such as *NATO*[5]. The lack of obvious differences between codewords, descriptors, caveats and IDO marking helps make the system confusing. A more detailed explanation can be found in [1565].

9.3.1 The Anderson report

In the 1960s, when computers started being widely used, the classification system caused serious friction. Paul Karger, who worked for the USAF then, described having to log off from a Confidential system, walk across the yard

coordination. Further problems arose when civilian documents marked Confidential ended up at the National Archives and Records Administration, where CONFIDENTIAL was a national security classification. Moving from this menagerie of markings to a single centrally-managed government-wide system has taken more than a decade and is still ongoing. The UK has its own post-Cold-War simplification story.

[5]Curiously, in the UK 'NATO Secret' is less secret than 'Secret', so it's a kind of anti-codeword that moves the content down the lattice rather than up.

to a different hut, show a pass to an armed guard, then go in and log on to a Secret system – over a dozen times a day. People soon realised they needed a way to deal with information at different levels at the same desk, but how could this be done without secrets leaking? As soon as one operating system bug was fixed, some other vulnerability would be discovered. The NSA hired an eminent computer scientist, Willis Ware, to its scientific advisory board, and in 1967 he brought the extent of the computer security problem to official and public attention [1989]. 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 1980s; the 70's concern was Trojans.) There was then a serious scare when it was discovered that the Pentagon's World Wide Military Command and Control System (WWMCCS) was vulnerable to Trojan Horse attacks; this had the effect of restricting its use to people with a 'Top Secret' clearance, which was inconvenient.

The next step was a 1972 study by James Anderson for the US government which concluded that a secure system should do one or two things well; and that these protection properties should be enforced by mechanisms that were simple enough to verify and that would change only rarely [52]. It introduced the concept of a *reference monitor* – a component of the operating system that would mediate access control decisions and be small enough to be subject to analysis and tests, the completeness of which could be assured. In modern parlance, such components – together with their associated operating procedures – make up the *Trusted Computing Base* (TCB). More formally, the TCB is defined as the set of components (hardware, software, human, …) 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.

9.3.2 The Bell-LaPadula model

The multilevel security policy model that gained wide acceptance was proposed by Dave Bell and Len LaPadula in 1973 [211]. Its basic property is that information cannot flow downwards. More formally, the *Bell-LaPadula* (BLP) 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 WWMCCS debacle and the more general fear of Trojan-horse attacks. An uncleared user might write a Trojan and leave it around where a system administrator cleared to 'Secret' might execute it; it could then copy itself into the 'Secret' part of the system, read the data there and try to signal it down somehow. It's also quite possible that an enemy agent could get a job at a commercial software house and embed some code in a product that would look for secret documents to copy. If it could then write them down to where its creator could read them, 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. So the pre-existing culture assumed that security policy was enforced independently of user actions; Bell-LaPadula sets out to enforce it 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'. More generally we must prevent any process at High from signalling to any object at Low. Systems that enforce a security policy independently of user actions 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 enabled designers to prove theorems. Given both the simple security property (no read up), and the star property (no write down), various results can be proved: in particular, if your starting state is secure, then your system will remain so. To keep things simple, we will generally assume from now on that the system has only two levels, High and Low.

9.3.3 The standard criticisms of Bell-LaPadula

The introduction of BLP caused a lot of excitement: here was a security policy that did what the defence establishment thought it wanted, was intuitively clear, yet still allowed people to prove theorems. Researchers started to beat up on it and refine it.

The first big controversy was about John McLean's *System Z*, which he 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. Dave Bell countered that System Z cheats by doing something his model

doesn't allow (changing labels isn't a valid operation on the state), and John McLean's retort was that it didn't explicitly tell him so: so the BLP rules were not in themselves enough. The issue is dealt with by introducing a *tranquility property*. Strong tranquility says that security labels never change during system operation, while weak tranquility says that labels never change in such a way as to violate a defined security policy.

Why weak tranquility? 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 they then access a confidential email, their session is automatically upgraded to 'Confidential'; in general, a process is upgraded each time it accesses data at a higher level (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 'Secret Crypto'. This will include temporary copies made of other files. If it then reads a file at 'Secret Nuclear' then all files it creates after that will be labelled 'Secret Crypto Nuclear', 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. Real-time changes in security level mean that access to resources can be revoked at any time, including in the middle of a transaction. And as the revocation problem is generally unsolvable in modern operating systems, at least in any complete form, 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. In order to prevent this, labels are now generally taken outside the MLS machinery and dealt with using discretionary access control mechanisms (I'll discuss this in the next chapter).

Another problem with BLP, and indeed with all mandatory access control systems, is that separating users and processes is the easy part; 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; the classic example was a trusted word processor that helps an intelligence analyst scrub a Top Secret document when she's editing it down to Secret [1272]. BLP is silent on how the system should protect such an application. So it becomes part of the Trusted Computing Base, but a part that can't be verified using models based solely on BLP.

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).

9.3.4 The evolution of MLS policies

Multilevel security policies have evolved in parallel in both the practical and research worlds.

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 [2010]. 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. Nonetheless, it laid the foundation for BLP, and also led to the current IBM S/390 mainframe hardware security architecture [942].

The next big step was Multics. This had started as an MIT project in 1965 and developed into a Honeywell product; it became the template and inspirational example for 'trusted systems'. 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 [1022] – and led to Ken Thompson's famous paper 'Reflections on Trusting Trust' ten years later [1887]. Multics had a derivative system called SCOMP that I'll discuss in section 9.4.1.

The torrent of research money that poured into multilevel security from the 1980s led to a number of alternative formulations. *Noninterference* was introduced by Joseph Goguen and Jose Meseguer in 1982 [774]. 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 [1851] to model applications such as a LAN on which there are machines at both Low and High, with the High machines encrypting their LAN traffic[6]. Nondeducibility turned out to be too weak, as there's nothing to stop Low making deductions about High input with 99% certainty. Other theoretical models include *Generalized Noninterference* and *restrictiveness* [1278]; the *Harrison-Ruzzo-Ullman* model tackles the problem of how to deal with the creation and deletion of files, on which BLP is silent [869]; and the *Compartmented Mode Workstation* (CMW) policy attempted to model the classification of information using floating labels, as in the high water mark policy [808,2042].

Out of this wave of innovation, the model with the greatest impact on modern systems is probably the *type enforcement* (TE) model, due to Earl Boebert and Dick Kain [272], later extended by Lee Badger and others to *Domain and Type*

[6]Quite a lot else is needed to do this right, such as padding the High traffic with nulls so that Low users can't do traffic analysis – see [1635] for an early example of such a system. You may also need to think about Low traffic over a High network, such as facilities for soldiers to phone home.

Enforcement (DTE) [154]. This assigns subjects to *domains* and objects to *types*, with matrices defining permitted domain-domain and domain-type interactions. This is used in SELinux, now a component of Android, which simplifies it by putting both subjects and objects in types and having a matrix of allowed type pairs [1189]. 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 (SID). The Linux Security Modules framework provides pluggable security where you can set rules that operate on SIDs.

DTE introduced a language for configuration (DTEL), and implicit typing of files based on pathname; so all objects in a given subdirectory may be declared to be in a given domain. DTE is more general than BLP, as it starts to deal with integrity as well as confidentiality concerns. One of the early uses was to enforce trusted pipelines: the idea is to confine a set of processes in a pipeline so that each can only talk to the previous stage and the next stage. This can be used to assemble guards and firewalls that cannot be bypassed unless at least two stages are compromised [1432]. Type-enforcement mechanisms 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. They are used, for example, in the Sidewinder firewall.

The downside of the greater flexibility and expressiveness of TE/DTE is that it is not always straightforward to implement policies like BLP, because of state explosion; when writing a security policy you have to consider all the possible interactions between different types. Other mechanisms may be used to manage policy complexity, such as running a prototype for a while to observe what counts as normal behaviour; you can then turn on DTE and block all the information flows not seen to date. But this doesn't give much assurance that the policy you've derived is the right one.

In 1992, *role-based access control* (RBAC) was introduced by David Ferraiolo and Richard Kuhn to manage policy complexity. It formalises rules that attach primarily to roles rather than to individual users or machines [678, 679]. Transactions that may be performed by holders of a given role are specified, then mechanisms for granting membership of a role (including delegation). Roles, or groups, had for years been the mechanism used in practice in organizations such as banks to manage access control; the RBAC model started 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 Admissions Committee' and 'Ross reading private email'. A variant of it, aspect-based access control (ABAC), adds context, so you can distinguish 'Ross at his workstation in the lab' from 'Ross on his phone somewhere on Earth'. Both have been supported by Windows since Windows 8.

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. I discuss SELinux in more detail at 9.5.2.

9.3.5 The Biba model

The incorporation into Windows 7 of a multilevel integrity model revived interest in a security model devised in 1975 by Ken Biba [238], which deals with integrity alone and ignores confidentiality. Biba's observation was 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. So you can recycle BLP into an integrity policy by turning it upside down.

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, so normal users should 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 must remain unchanged. Only an authorised technician should be able to redo the calibration.

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 – i.e., 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 shouldn't be able to affect it; and an electricity utility's power dispatching system will be able to see the safety systems' state but not interfere with them. The safety-critical systems community talks in terms of *safety integrity levels*, which relate to the probability that a safety mechanism will fail and to the level of risk reduction it is designed to give.

Windows, since version 6 (Vista), marks file objects with an integrity level, which can be Low, Medium, High or System, and implements a default policy of NoWriteUp. Critical files are at System and other objects are at Medium by default – except for the browser which is at Low. So things downloaded using

IE can read most files in a Windows system, but cannot write to them. The goal is to limit the damage that can be done by malware.

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 cannot protect and confine them, any more than BLP can. For example, a car's airbag is on a less critical bus than the engine, but when it deploys you assume there's a risk of a fuel fire and switch the engine off. There are other real integrity goals that Biba also cannot express, such as assured pipelines. In the case of Windows, Microsoft even dropped the NoReadDown restriction and did not end up using its integrity model to protect the base system from users, as this would have required even more frequent user confirmation. In fact, the Type Enforcement model was introduced by Boebert and Kain as an alternative to Biba. It is unfortunate that Windows didn't incorporate TE.

9.4 Historical examples of MLS systems

The second edition of this book had a much fuller history of MLS systems; since these have largely gone out of fashion, and the MLS research programme has been wound down, I give a shorter version here.

9.4.1 SCOMP

A key product was the *secure communications processor* (SCOMP), a derivative of Multics launched in 1983 [710]. 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 to keep things simple. Its operating system, STOP, used Multics' system of 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 firewalls that allow mail to pass from Low to High but not vice versa [538]. (In general, a device which supports one-way flow 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. SCOMP's most significant contribution was to serve as a model for the *Orange Book* [544] – 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 [1398].

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 a target for the next generation of military systems.

MLS versions of Unix started to appear in the late 1980s, such as AT&T's System V/MLS [48]. This added security levels and labels, showing that MLS properties could be introduced to a commercial operating system with minimal changes to the system kernel. By this book's second edition (2007), Sun's Solaris had emerged as the platform of choice for high-assurance server systems and for many clients as well. *Comparted Mode Workstations* (CMWs) were an example of the latter, allowing data at different levels to be viewed and modified at the same time, so an intelligence analyst could read 'Top Secret' data in one window and write reports at 'Secret' in another, without being able to accidentally copy and paste text downwards [934]. For the engineering, see [635, 636].

9.4.2 Data diodes

It was soon realised that simple mail guards and crypto boxes were too restrictive, as more complex networked services were developed besides mail. First-generation MLS mechanisms were 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 9.3. The main problem is that while sending data from Low to

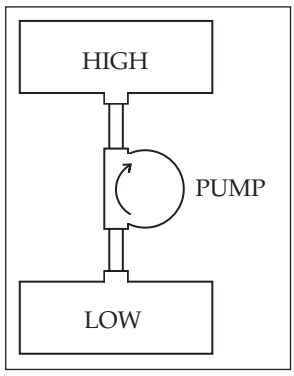

Figure 9.3: The NRL pump

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 buffering and random timing of acknowledgements [1014,1016, 1017]. The attraction of this approach is that one can build MLS systems by using data diodes to connect separate systems at different security levels. As these systems don't process data at more than one level – an architecture called *system high* – they can be built from cheap *commercial-off-the-shelf* (COTS) components. You don't need to worry about applying MLS internally, merely protecting them from external attack, whether physical or network-based. As the cost of hardware has fallen, this has become the preferred option, and the world's military bases are now full of KVM switches (which let people switch their keyboard, video display and mouse between Low and High systems) and data diodes (to link Low and High networks). The pump's story is told in [1018].

An early application was logistics. 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. The systems needed to manage all this can be hard to build; 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 [1388]. 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 changes in requirements. 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 that are now kept in strategic stockpiles rather than at airbases). To save money, the 'Secret' information is now kept on a CD and locked up in a safe.

Another major application of MLS is in wiretapping. The target of investigation should not know they are being wiretapped, so the third party must be silent – and when phone companies started implementing wiretaps as silent conference calls, the charge for the conference call had to go to the wiretapper, not to the target. The modern requirement is a multilevel one: multiple agencies at different levels may want to monitor a target, and each other, with the police tapping a drug dealer, an anti-corruption unit watching the police, and so on. Eliminating covert channels is harder than it looks; for a survey from the mid-2000s, see [1710]; a pure MLS security policy is insufficient, as suspects can try to hack or confuse wiretapping equipment, which therefore needs to resist online tampering. In one notorious case, a wiretap was discovered on the

mobile phones of the Greek Prime Minister and his senior colleagues during the Athens olympics; the lawful intercept facility in the mobile phone company's switchgear was abused by unauthorised software, and was detected when the buggers' modifications caused some text messages not to be delivered [1553]. The phone company was fined 76 million Euros (almost $100m). The clean way to manage wiretaps nowadays with modern VOIP systems may just be to write everything to disk and extract what you need later.

There are many military embedded systems too. In submarines, speed, reactor output and RPM are all Top Secret, as a history of these three measurements would reveal the vessel's performance – and that's among the few pieces of information that even the USA and the UK don't share. The engineering is made more complex by the need for the instruments not to be Top Secret when the vessel is in port, as that would complicate maintenance. And as for air combat, some US radars won't display the velocity of a US aircraft whose performance is classified, unless the operator has the appropriate clearance. When you read stories about F-16 pilots seeing an insanely fast UFO whose speed on their radar didn't make any sense, you can put two and two together. It will be interesting to see what sort of other side-effects follow when powerful actors try to bake MAC policies into IoT infrastructure, and what sort of superstitious beliefs they give rise to.

9.5 MAC: from MLS to IFC and integrity

In the first edition of this book, I noted a trend to use mandatory access controls to prevent tampering and provide real-time performance guarantees [1021, 1315], and ventured that "perhaps the real future of multilevel systems is not in confidentiality, but integrity." Government agencies had learned that MAC was what it took to stop malware. By the second edition, multilevel integrity had hit the mass market in Windows, which essentially uses the Biba model.

9.5.1 Windows

In Windows, all processes do, and all securable objects (including directories, files and registry keys) may, have an integrity-level label. File objects are labelled '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. It's also possible to implement a crude BLP policy using Windows, as you can also set 'NoReadUp' and 'NoExecuteUp' policies. These are not installed as default; Microsoft was 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 this integrity-only approach to MAC does mean that malware running at Low can steal all your data; so some users might care to set 'NoReadUp' for sensitive directories. This is all discussed by Joanna Rutkowska in [1637]; she also describes some interesting potential attacks based on virtualization.

9.5.2 SELinux

The case of SELinux is somewhat similar to Windows in that the immediate goal of mandatory access control mechanisms was also to limit the effects of a compromise. SELinux [1189] was implemented by the NSA, based on the Flask security architecture [1815], 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. This is where policy decisions are made and resides in the kernel for performance [820]. It has been mainstream since Linux 2.6. The server provides a security API to the rest of the kernel, behind which the security model is hidden. The server internally implements a general constraints engine that can express RBAC, TE, and MLS. In typical Linux distributions from the mid-2000s, it was used to separate various services, so an attacker who takes over your web server does not thereby acquire your DNS server as well. Its adoption by Android has made it part of the world's most popular operating system, as described in Chapter 6.

9.5.3 Embedded systems

There are many fielded systems that implement some variant of the Biba model. As well as the medical-device and railroad signalling applications I already mentioned, there are utilities. In an electricity utility, for example, there is typically a hierarchy of safety systems, which operate completely independently at the highest safety integrity level; these are visible to, but cannot be influenced by, operational systems such as power dispatching; retail-level metering systems can be observed by, but not influenced by, the billing system. Both retail meters and the substation-level meters in the power-dispatching system feed information into fraud detection, and finally there are the executive information systems, which can observe everything while having no direct effect on operations. In cars, most makes have separate CAN buses for the powertrain and for the cabin, as you don't want a malicious app on your radio to be able to operate your brakes (though

in 2010, security researchers found that the separation was completely inadequate [1087]).

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, or to issue instructions to other machines.

To sum up, many of the lessons learned in the early multilevel systems go across to a number of applications of wider interest. So do a number of the failure modes, which I'll now discuss.

9.6 What goes wrong

Engineers learn more from the systems that fail than from those that succeed, and here MLS systems have been an effective teacher. The billions of dollars spent on building systems to follow a simple policy with a high level of assurance have clarified many second-order and third-order consequences of information flow controls. I'll start with the more theoretical and work through to the business and engineering end.

9.6.1 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 ciphertext, which being encrypted with a cipher system giving perfect secrecy, is considered to be low (output L), as in Figure 9.4.

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 break the composition of two secure systems (noted by Daryl McCullough [1262]).

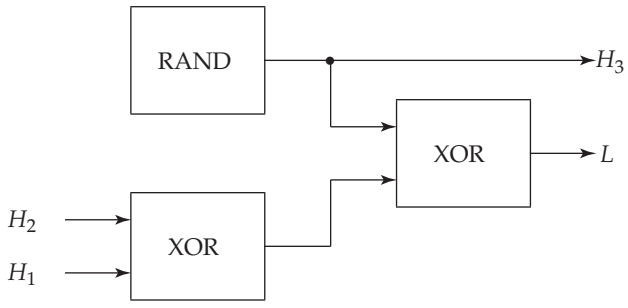

Figure 9.4: Insecure composition of secure systems with feedback

In general, the *composition problem* – 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 [1432]. (Simple information flow doesn't compose; neither does noninterference or nondeducibility.) Most of the low-level problems arise when some sort of feedback is introduced; without it, composition can be achieved under a number of formal models [1279]. However, in real life, feedback is pervasive, and composition of security properties can be made even harder by 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. And the composition of secure components is often frustrated by higher-level incompatibilities. Components might have been designed in accordance with two different security policies, or designed according to inconsistent requirements.

9.6.2 The cascade problem

An example of the composition problem is given by the *cascade problem* (Figure 9.5). After the Orange book introduced a series of evaluation levels, this led to span-limit rules about the number of levels at which a system can operate [548]. 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 [548].

As the diagram shows, it is straightforward to connect together two B3 systems in such a way that this policy is broken. The first system connects together Unclassified and Secret, and its Secret level communicates with the second system – which also processes Top Secret information [925]. This defeats the span limit.

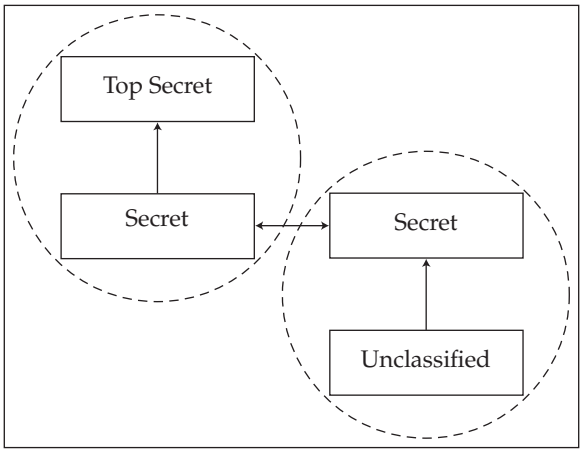

Figure 9.5: The cascade problem

9.6.3 Covert channels

One of the reasons why 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 [1127], a covert channel is a mechanism that was not designed for communication but that 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. In a modern multicore CPU, it could increase the clock frequency of the CPU core it's using at time t_i to signal that the i-th bit in a Top Secret file was a 1, and let it scale back to signal that the bit was a 0. This gives a covert channel capacity of several tens of bits per second [36]. Since 2018, CPU designers have been struggling with a series of covert channels that exploit the CPU microarchitecture; with names like Meltdown, Spectre, and Foreshadow, they have provided not just ways for High to signal to Low but for Low to circumvent access control and read memory at High. I will discuss these in detail in the chapter on side channels.

The best that developers have been able to do consistently with confidentiality protection in regular operating systems is to limit it to 1 bit per second or so. (That is a DoD target [545], and techniques for doing a systematic analysis may be found in Kemmerer [1038].) 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. However, it's potentially a lethal threat to high-value cryptographic keys. This is one of the reasons for the military and banking doctrine of doing crypto in special purpose hardware.

The highest-bandwidth covert channel of which I'm aware 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, which 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.

9.6.4 The threat from malware

The defense computer community was shocked when Fred Cohen wrote the first thesis on computer viruses, and used a virus to penetrate multilevel secure systems easily in 1983. In his first experiment, a file virus that took only eight hours to write managed to penetrate a system previously believed to be multilevel secure [452]. People had been thinking about malware since the 1960s and had done various things to mitigate it, but their focus had been on Trojans.

There are many ways in which malicious code can be used to break access controls. If the reference monitor (or other TCB components) can be corrupted,

then malware can deliver the entire system to the attacker, for example by issuing 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' [548], and this in turn created an incentive for vendors to tamper-proof the TCB, using techniques such as TPMs. But even if the TCB remains intact, malware could still copy itself up from Low to High (which BLP doesn't prevent) and use a covert channel to signal information down.

9.6.5 Polyinstantiation

Another problem that exercised the research community is *polyinstantiation*. Suppose 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, such as giving Low and High users different directories. But the problem remains a hard one for databases [1652]. 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.

Here the US and UK practices diverge. The solution favoured in the USA 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 9.6.

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 9.7.

This makes the system engineering simpler. It also prevents the mistakes and covert channels that 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, cover stories still get used in order not to advertise the existence of a covert mission any more than need be.)

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

Figure 9.6: how the USA deals with classified data

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

Figure 9.7: how the UK deals with classified data

9.6.6 Practical problems with MLS

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

1. They 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; so many MLS systems are installed without their features being used.
3. Many applications need to be rewritten or at least greatly modified to run under MLS operating systems [1632].
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. There's a particular problem when system components accumulate all the labels they've seen, leading to *label explosion* where they acquire such a collection that no single principal can access them any more. So they get put in the trusted computing base, which ends up containing a quite uncomfortably large part of the operating system (plus utilities, plus windowing system software, plus middleware such as database software). This 'TCB bloat' constantly pushes up the cost of evaluation and reduces assurance.
5. The classification of data can get complex:
   - in the run-up to a conflict, the location of 'innocuous' stores such as food could reveal tactical intentions, and so may be suddenly upgraded;
   - classifications are not always monotone. Equipment classified at 'confidential' may easily contain components classified 'secret', and on the flip side it's hard to grant access at 'secret' to secret information in a 'top secret' database;

- 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'. In case information was covertly hidden in the image by a virus, this may involve special filters, lossy compression of images and so on. One option is a 'print-and-fax' mechanism that turns a document into a bitmap, and logs it for traceability.
- 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. (I will look at this *aggregation problem* in more detail in section 11.2.)
- Similarly, the output of an unclassified program acting on unclassified data may be classified, for example if standard data mining techniques applied to an online forum throw up a list of terror suspects.

6. Although MLS systems can prevent undesired things (such as information leakage), they also prevent desired things too (such as building a search engine to operate across all an agency's Top Secret compartmented data). So even in military environments, the benefits can be questionable. After 9/11, many of the rules were relaxed, and access controls above Top Secret are typically discretionary, to allow information sharing. The cost of that, of course, was the Snowden disclosures.
7. Finally, obsessive government secrecy is a chronic burden. The late Senator Daniel Moynihan wrote a critical study of its real purposes, and its huge costs in US foreign and military affairs [1348]. For example, President Truman was never told of the Venona decrypts because the material was considered 'Army Property'. As he put 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."

   More recent examples of MLS doctrine impairing operational effectiveness include the use of unencrypted communications to drones in the Afghan war (as the armed forces feared that if they got the NSA bureaucracy involved, the drones would be unusable), and the use of the notoriously insecure Zoom videoconferencing system for British government cabinet meetings during the coronavirus crisis (the government's encrypted videoconferencing terminals are classified, so ministers aren't allowed to take them home). This brings to mind a quip 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!"

There has been no shortage of internal strategic critique. A 2004 report by Mitre's JASON programme of the US system of classification concluded that it was no longer fit for purpose [980]. There are many interesting reasons, including the widely different risk/benefit calculations of the producer and consumer communities; classification comes to be dominated by distribution channels rather than by actual risk. The relative ease of attack has led government systems to be too conservative and risk-averse. It noted many perverse outcomes; for example, Predator imagery in Iraq is Unclassified, and was for some time transmitted in clear, as the Army feared that crypto would involve the NSA bureaucracy in key management and inhibit warfighting.

Mitre proposed instead that flexible compartments be set up for specific purposes, particularly when getting perishable information to tactical compartments; that intelligent use be made of technologies such as rights management and virtualisation; and that lifetime trust in cleared individuals be replaced with a system focused on transaction risk.

Anyway, one of the big changes since the second edition of this book is that the huge DoD research programme on MLS has disappeared, MLS equipment is no longer very actively promoted on the government-systems market, and systems have remained fairly static for a decade. Most government systems now operate system high – that is, entirely at Official, or at Secret, or at Top Secret. The difficulties discussed in the above section, plus the falling cost of hardware and the arrival of virtualisation, have undermined the incentive to have different levels on the same machine. The deployed MLS systems thus tend to be firewalls or mail guards between the different levels, and are often referred to by a new acronym, MILS (for multiple independent levels of security). The real separation is at the network level, between unclassified networks, the Secret Internet Protocol Router Network (SIPRNet) which handles secret data using essentially standard equipment behind crypto, and the Joint Worldwide Intelligence Communications System (JWICS) which handles Top Secret material and whose systems are kept in Secure Compartmentalized Information Facilities (SCIFs) – rooms shielded to prevent electronic eavesdropping, which I'll discuss later in the chapter on side channels.

There are occasional horrible workarounds such as 'browse-down' systems that will let someone at High view a website at Low; they're allowed to click on buttons and links to navigate, just not to enter any text. Such ugly hacks have clear potential for abuse; at best they can help keep honest people from careless mistakes.

9.7 Summary

Mandatory access control was initially developed for military applications, where it is still used in specialized firewalls (guards and data diodes). The main

use of MAC mechanisms nowadays, however, is in platforms such as Android, iOS and Windows, where they protect the operating systems themselves from malware. MAC mechanisms have been a major subject of computer security research since the mid-1970's, and the lessons learned in trying to use them for military multilevel security 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 research literature when it's appropriate, and avoid being dragged into overdesign when it's not.

There are many problems which we need to be a 'fox' rather than a 'hedgehog' to solve. By trying to cast all security problems as hedgehog problems, MLS often leads to inappropriate security goals, policies and mechanisms.

Research problems

A standing challenge, sketched out by Earl Boebert in 2001 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 [271]). 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 should be useful outside the world of security. Will we see them widely used in the Internet of Things? We've mentioned Biba-type mechanisms in applications such as cars and electricity distribution; will the MAC mechanisms in products such as SELinux, Windows and Android enable designers to lock down information flows and reduce the likelihood of unanticipated interactions?

The NSA continues to fund research on MLS, now under the label of IFC, albeit at a lower level than in the past. Doing it properly in a modern smartphone is hard; for an example of such work, see the Weir system by Adwait Nadkarni and colleagues [1374]. In addition to the greater intrinsic complexity of modern operating systems, phones have a plethora of side-channels and their apps are often useful only in communication with cloud services, where the real heavy lifting has to be done. The commercial offering for separate 'low' and 'high' phones consists of products such as Samsung's Knox.

A separate set of research issues surround actual military opsec, where reality falls far short of policy. All armed forces involved in recent conflicts, including US and UK forces in Iraq and Afghanistan, have had security issues around their personal mobile phones, with insurgents in some cases tracing their families back home and harassing them with threats. The Royal Navy tried to ban phones in 2009, but too many sailors left. Tracking ships via Instagram is easy; a warship consists of a few hundred young men and women, aged 18-24, with nothing much else to do but put snaps on social media. Discipline tends to focus on immediate operational threats, such as when a sailor is seen

snapchatting on mine disposal: there the issue is the risk of using a radio near a mine! Different navies have tried different things: the Norwegians have their own special network for sailors and the USA is trying phones with MLS features. But NATO exercises have shown that for one navy to hack another's navigation is shockingly easy. And even the Israelis have had issues with their soldiers using mobiles on the West Bank and the Golan Heights.

Further reading

The unclassified manuals for the UK government's system of information classification, and the physical, logical and other protection mechanisms required at the different levels, have been available publicly since 2013, with the latest documents (at the time of writing) having been released in November 2018 on the Government Security web page [803]. The report on the Walker spy ring is a detailed account of a spectacular failure, and brings home the sheer complexity of running a system in which maybe three million people have a clearance at any one time, with a million applications being processed each year [878]. And the classic on the abuse of the classification process to cover up waste, fraud and mismanagement in the public sector is by Chapman [409].

On the technical side, textbooks such as Dieter Gollmann's *Computer Security* [780] give an introduction to MLS systems, while many of the published papers on actual MLS systems can be found in the proceedings of two conferences: academics' conference is the *IEEE Symposium on Security & Privacy* (known in the trade as 'Oakland' as that's where it used to be held), while the NSA supplier community's unclassified bash is the *Computer Security Applications Conference* (ACSAC) 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 [452]. Many of the classic early papers in the field can be found at the NIST archive [1397]; NIST ran a conference series on multilevel security up till 1999. Finally, a history of the Orange Book was written by Steve Lipner [1172]; this also tells the story of the USAF's early involvement and what was learned from systems like WWMCCS.