Mathematics, Technology, and Trust: Formal Verification, Computer Security, and the U.S. Military

DONALD MACKENZIE AND GARREL POTTINGER

A distinctive concern in the U.S. military for computer security dates from the emergence of time-sharing systems in the 1960s. This paper traces the subsequent development of the idea of a "security kernel" and of the mathematical modeling of security, focusing in particular on the paradigmatic Bell–LaPadula model. The paper examines the connections between computer security and formal, deductive verification of the properties of computer systems. It goes on to discuss differences between the cultures of communications security and computer security, the bureaucratic turf war over security, and the emergence and impact of the Department of Defense's Trusted Computer System Evaluation Criteria (the so-called Orange Book), which effectively took its final form in 1983. The paper ends by outlining the fragmentation of computer security since the Orange Book was written.

Introduction

How can computer systems be made secure? In the modern world, such systems are crucial components of economic and military power. From the 1940s onward, the U.S. military has been the world's single most important customer for, and commissioner of, computer systems. By the mid-1990s, the Department of Defense employed over two million computers, 10,000 local networks, and 100 long-distance networks. This information infrastructure has become essential to almost all U.S. military activities. Its possible vulnerability to intrusion, espionage, and sabotage—a concern of insiders for at least 30 years—has recently been the subject of public comment.[1]

Since the late 1960s, computer security has been at the heart of one of the most important research and development efforts in computer science. Reducing or removing computer systems' vulnerabilities is not simply an important practical matter. It has interacted closely with one of the central questions of computer science: How can we know how computer systems will behave? Their complexity makes it extraordinarily difficult—in practice, usually impossible—to test them exhaustively. The history of computer security is intertwined with the effort to gain deductive, proof-based (in addition to inductive, test-based) knowledge of the properties of computer systems, and thus is intertwined with a significant part of the last three decades' research in computer science. This intertwining has had surprising results: For example, it led to some automated mathematical theorem provers—an apparently utterly esoteric technology—becoming classed as auxiliary military equipment for the purposes of export control.

Computer security is an area of conflict. Most obviously, it raises important questions of the balance among privacy, crime prevention, and government surveillance, an issue that has been sharpest in respect to encryption technology. Less obviously, computer security is a field in which the desires for national security and for corporate profit have often been implicitly in tension, and where the civilian and military agencies of government have fought turf wars. Even within organizations such as the National Security Agency (NSA), there has been a degree of conflict between those steeped in the older, secretive arts of cryptoanalysis and communications security and those in the more open, more academic discipline of computer security. The latter have even asked the heretical (albeit naive) question: If a computer system is secure and has been proven mathematically to be secure, is there any reason to keep its details secret?

In its account of the history of computer security, this article seeks to be neither definitive—too much of the detail of this area remains classified, even in the United States, for that to be possible—nor comprehensive. Communications security and encryption remain in the background of our story, as do developments in Europe and the history of computer security concerns in banking and other areas of civil industry and commerce.

We begin by outlining the origins of a distinctive concern for computer security in the emergence of time-sharing systems in the 1960s, concern that had come into focus by 1967. We next turn to the first systematic investigations of computer security within the U.S. military: the panels led by Willis H. Ware (which reported in 1970) and James P. Anderson (which reported in 1972). We trace the emergence of the idea of a "security kernel" and of the mathematical modeling of security, focusing in particular on what was to become the paradigmatic definition of what security

1058-6180/97/$10.00 © 1997 IEEE

means: the Bell–LaPadula model.

We then turn to the connections between computer security and formal, deductive verification of the properties of computer systems and describe two of the phenomena that make the formal analysis of security more complicated than it first appears: the need for "trusted subjects" and the practical infeasibility of eliminating "covert channels." We trace the sometimes stormy histories of the Department of Defense Computer Security Evaluation Center, of its famous criteria for the evaluation of secure systems (the so-called Orange Book), and of the attempts to develop systems to meet the Orange Book's highest security class, A1.

Finally, we outline the fragmentation of computer security since the creation of the Orange Book. The boundary between computer security and communications security has become blurred; the operation of the secure computing market has undercut efforts to produce high-security, high-assurance systems; despite efforts at synthesis, differences remain between commercial and military approaches to computer security; and the Bell–LaPadula model has lost its dominant role. Against the background of this fragmentation, however, we note that there is tentative evidence of some convergence between the previously distinct spheres of safety and security.

Time-Sharing and Computer Security

Specific concerns about computer security began with the advent, in the 1960s, of time-sharing computer systems. These made it possible for several people to interact, seemingly simultaneously, with the same computer system via terminals that could be in separate rooms or separate buildings. In earlier batch-operated computer systems, different users' programs were executed one after the other.[2] Users could not interact with the computer system while their programs were being run. Having sent or carried their coding forms, punched cards, or reels of paper tape to their organization's computer center, users had to wait to pick up their programs' output.

Batch systems had potential security problems; for example, the output of a previous program would normally still be in peripheral storage, such as magnetic tape or disks, when a new one was being run, but these issues elicited little comment or concern. Time-sharing, however, was different, both technically and socially. Programs or data "belonging" to different users would be present simultaneously in the computer's main memory, not just in peripheral storage. Users could interact with their programs as they were being run and could do so while sitting at their own terminals, unseen by each other and by a system's operators. The activities, and even the identities, of users were potentially problematic.

Time-sharing greatly increased the efficiency of computer installations. Most importantly, users could debug programs interactively, instead of having to wait for several hours to see if a program had run successfully. However, time-sharing also raised the issue of how to prevent different users and different programs from interfering with each other. Most obviously, the computer's main memory had to be divided between different users' programs so as to prevent one program from overwriting a memory location being used by another program. Ideally, though, these memory bounds had to be flexible; otherwise, portions of memory might remain unused by programs with modest memory demands, while other users were unnecessarily constrained. The twin demand for efficient use of resources *and* for keeping different programs from interfering with each other rendered the design and development of system software for time-sharing systems a difficult and crucial task.

Even in the late 1980s, entry to the Edinburgh Multiple-Access System was controlled by each user's four-letter password, which could be a meaningful English word, and which users were never under any compulsion to change. (Current computer security specialists would regard each of these features of the password with scorn.)

In the late 1950s and early 1960s, much of the early development of time-sharing took place at the Massachusetts Institute of Technology (MIT). By 1963, MIT's Multiple Access Computer could serve up to 24 users at once, via teletypewriter terminals connected to the central computer through MIT's telephone system.[3] In a university environment, security was not a dominant issue. Different programs certainly had to be prevented from *writing* in portions of memory being used by other programs, but stopping different users from *reading* others' data was not, in practice, a major concern.[4] Furthermore, freedom to grant permission to read or to modify files was entirely at users' discretion (unlike in the military, where the basic rules of security are *mandatory*), and controls over access to the overall system were typically relaxed. One of us (MacKenzie) was for many years a user of one pioneering university time-sharing system, the Edinburgh Multiple-Access System. Even in the late 1980s, entry to the Edinburgh Multiple-Access System was controlled by each user's four-letter password, which could be a meaningful English word, and which users were never under any compulsion to change. (Current computer security specialists would regard each of these features of the password with scorn. For example, passwords that are meaningful words—especially meaningful words of a short, set length—are vulnerable to "dictionary attack," in which a machine-readable dictionary is used to generate and try possible passwords.)

The quite different priorities of national defense were, however, present from the very beginning of time-sharing. Much of the interest at MIT in time-sharing grew out of earlier experience developing the interactive air defenses that eventually became the continent-wide Semiautomatic Ground Environment system. The Department of Defense's Advanced Research Projects Agency (ARPA) funded Project MAC (Multiple Access Computer) and other early time-sharing work, notably at the Rand Corporation's spin-off company, the System Development Corporation (which was responsible for programming the Semiautomatic Ground Environment system).[5]

The armed services could not be expected to take the relaxed attitude to security that was possible at universities. By the second half of the 1960s, important actors in the U.S. defense sector had realized that time-sharing computer systems posed security issues

that went beyond the traditional concerns for secure communications, physical protection against intrusion, and the vetting of key personnel. These issues first came into clear focus in 1967, with an authoritative statement of them by Bernard Peters of the NSA at the Spring Joint Computer Conference. Based in Fort Meade, Md., NSA had, and has, responsibility not just for decoding the communications of actual or potential foes (its more famous role) but also for protecting the security of classified U.S. government communications. Peters's speech was unusual in that his affiliation with NSA was openly stated at a time when that agency's existence was not widely advertised: Computer-industry insiders used to joke that the initials stood for "No Such Agency."

Ware (Fig. 1), one of the senior figures of U.S. computer science, opened the special session. Ware had taken part in the celebrated digital computer project, inspired by John von Neumann, at the Institute for Advanced Study[6] and had gone on to become a member of the Rand Corporation's computer science department and, later, its head. Ware, Peters, and some of Ware's Rand colleagues had been discussing computer security for some time, and Rand had done some penetration studies (experiments in circumventing computer security controls) of early time-sharing systems on behalf of the government.[7] In his talk, Ware tied the new security problem firmly to time-sharing:

> with the advent of computer systems which share the resources of the configuration among several users or several problems, there is the risk that information from one user (or computer program) will be coupled to another user (or program)

and he identified possible threats using what was to become a famous, often copied illustration (Fig. 2).[8]

Fig. 1. Willis H. Ware.

Courtesy Willis H. Ware, Rand Corporation.

Ware then turned the floor over to the NSA's Peters. Peters spelled out the question that was to dominate research and development in computer security for much of the following 20 years: how to embody security in the system software of a "large multiprogrammed system with remote terminals."[9] NSA's judgment was that no adequate solution to this problem was available, either in the university time-sharing systems or in the commercial products that were beginning to appear (in October 1967, for example, IBM released its Time Sharing System, for the System/360 Model 67).[10] Peters emphasized that from a military security point of view, it was necessary to do more than prevent one user's program from inadvertently overwriting a memory location being used by another program. Programs—and, by extension, human users—"must be considered to be hostile," said Peters. So:

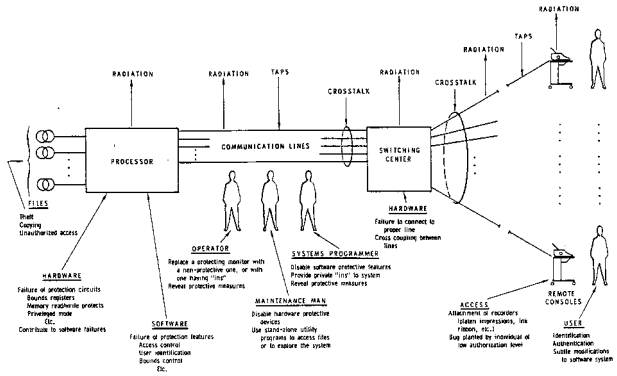


Fig. 2. A 1967 view of the security vulnerabilities of a resource-sharing computer system. (A monitor is what would now be called an operating system, and a private in would now be called a trap door. Radiation refers to the possibility of breaching security by analysis of electromagnetic emissions from a system.)

From Willis H. Ware, "Security and Privacy in Computer Systems," *AFIPS Conf. Proc., vol. 30, Spring Joint Computer Conf.* Washington, D.C.: Thompson Books, 1967, p. 280.

> Memory protect must be sufficient so that any reference, read or write, outside of the area assigned to a given user program must be detected and stopped. There are several forms of memory protect on the market which guard against out-of-bounds write, thus protecting program integrity, but they do not guard against illegal read. Read protect is as important as write protect, from a security standpoint, if classified material is involved.[11]

Peters did little more than sketch how it might be possible to design system software to prevent both illegal reads and illegal writes. Nevertheless, his talk did outline three issues that were to become of great importance as the field of computer security developed. The first was the key role in security played by the operating system or monitor:

> the monitor acts as the overall guard to the system. It provides protection against the operators and the users at the remote terminals.[12]

Second, Peters raised the issue of certification, emphasizing that a monitor must be "approved by the appropriate authority." In the armed services, this would be a security officer; in business, it would be corporate management. "Who can tell who is in charge in a university?" he added dryly. The need for authoritative approval implied that it was necessary "to adequately demonstrate the security capability to the governing authority." To facilitate this, suggested Peters (raising a third issue that was a harbinger of later developments), the monitor "must be carefully designed to limit the amount of critical coding [program writing]." Critical security functions (in particular, the software handling the "interrupts" that transferred control from user programs to the monitor) should be embedded in relatively small amounts of code:

> When an interrupt occurs ... [t]he monitor must, as soon as reasonable, adjust the memory bounds to provide limits on even the monitor's own coding. This requires that the coding which receives interrupts be certified as error free for *all* possible inputs and timing. ... By keeping the amount of coding that can reference any part of core without restriction to a few well-tested units, confidence in the monitor can be established.

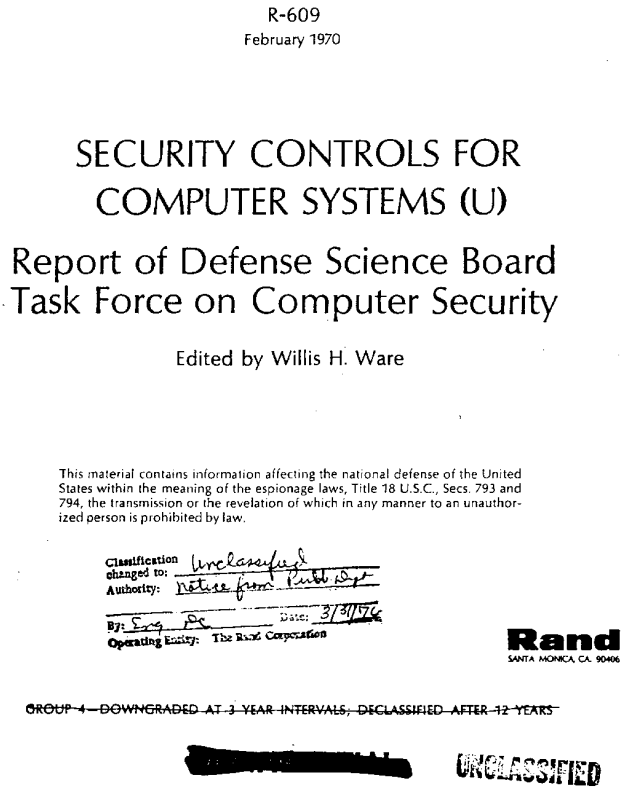
On the economics of computer security, Peters was optimistic, saying that "the cost of a properly designed monitor is probably not more than 10 percent greater than that of a monitor which is minimally acceptable for multiprogramming."[13]

During 1967, increasing numbers of the new time-sharing systems were procured for U.S. government installations. Concerns about their security properties began to grow, for example, after a large defense contractor proposed selling, to commercial users, time on an IBM mainframe computer employed in a classified aircraft project, and the Department of Defense realized it had no policy to cover such a situation.[14] In response to these concerns, the Defense Science Board set up a task force in October 1967 to address the security problems of "multi-access, resource-sharing computer systems." Rand's Ware chaired the task force, with representation from the NSA, Central Intelligence Agency, Department of Defense, defense contractors, and academia. It produced its classified report (originally drafted by Ware, but extensively rewritten by Thomas Chittenden of NSA) in February 1970 (see Fig. 3). The report's proposals covered a wide range of organizational and technical matters, with, as in Peters's paper, a particular focus on how to design a secure operating system, or "Supervisor."[15]

The task force proposed that the system maintain a "catalog" of "flags" to indicate the security status of users, of terminals, and of the files within which information was stored in computer memory. The system of flags would be modeled on the existing system for classifying written documents. Users, terminals, files, particular jobs (computing tasks), input, and output would be classed into a vertical hierarchy of top secret, secret, confidential, and unclassified, according to their clearances and security statuses. There was also provision for horizontal compartmentalization, such as the special category of clearance (Q-clearance) required for access to information about nuclear weapons.

The Ware task force's proposed access-control rules were common-sense extensions of those that applied to documents. Fundamental was the rule that can be summarized as "no read up": For example, a user with a secret clearance, using an appropriate terminal, would (unless barred by horizontal compartmentalization) be permitted to run jobs requiring read access to files bearing the flags secret, confidential, and unclassified, but not a job that required access to files bearing the top-secret flag.

A no-read-up rule (see Fig. 4) seemed extremely simple. But by 1970, it was already clear to the task force that implementing it securely was a difficult task. Complications arose from the fact that computing was a dynamic process: A figure for the range or effectiveness of a weapon might be more sensitive than the data from which it was calculated. More generally, "as a job unfolds, [its] security flag may have to be modified automatically by the system to reflect the security flags of files of information or files of other programs that are used." The task force also noted that "operating systems are very large, complex structures, and thus it is impossible to exhaustively test for every conceivable set of conditions that might arise," so it was necessary to separate the design of the supervisor into distinct modules, each of which "must be fully described with flowcharts to assist in its security analysis."[16]



R-609
February 1970

SECURITY CONTROLS FOR COMPUTER SYSTEMS (U)

Report of Defense Science Board Task Force on Computer Security

Edited by Willis H. Ware

This material contains information affecting the national defense of the United States within the meaning of the espionage laws, Title 18 U.S.C., Secs. 793 and 794, the transmission or the revelation of which in any manner to an unauthorized person is prohibited by law.

Classification changed to: Unclassified
Authority: Notice from Def Supply Agency
By: Evy Date: 3/31/76
Operating Entity: The Rand Corporation

Rand
SANTA MONICA, CA. 90406

~~GROUP 4 — DOWNGRADED AT 3 YEAR INTERVALS; DECLASSIFIED AFTER 12 YEARS~~

[redacted] UNCLASSIFIED

Fig. 3. The Ware Report.

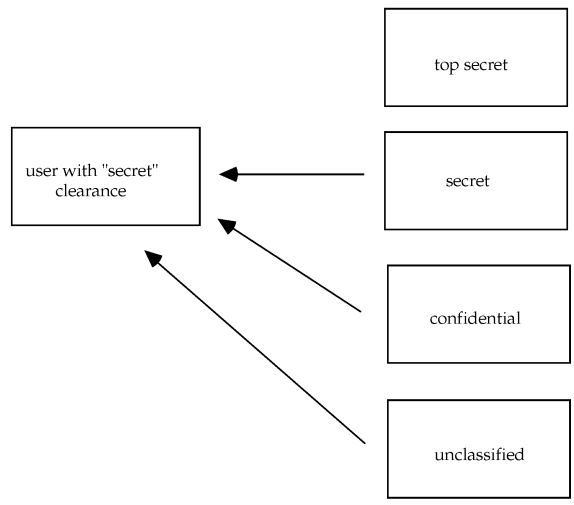


Fig. 4. An example of no read up. A user with secret clearance can read secret, confidential, and unclassified files, but not top-secret files.

Despite the attention that Ware and his colleagues devoted to their task, their proposals appear not to have produced any immediate response. Two years later, in February 1972, a further study was commissioned by Major Roger Schell of the Electronic Systems Division of the U.S. Air Force. In Schell's view, "the Ware panel put together an assessment that says, 'You've got all these problems.' They offer almost nothing by way of solutions. And what the Air Force was interested in was 'How do we provide solutions?'"[17]

Heading the study Schell commissioned was Anderson, a computer consultant who headed his own company based in Fort Washington, Pa. The tone of the Anderson panel's report, completed in October 1972, was more urgent and more alarmist than that of its predecessor. Its argument was that no existing system could be operated securely in a multilevel mode (that is, containing information for which some users were not cleared), and the Air Force was losing $100 million a year through the resulting inefficiencies. Whenever "tiger teams" had attempted to circumvent the security controls of existing systems, they had succeeded. (As Ware now puts it, the operating systems of the 1960s were "Swiss cheese in terms of security loopholes."[18]) It was difficult, expensive, and probably futile to try to patch the vulnerabilities that made this possible. Nor, in the view of the Anderson panel, was computer science or the computer industry producing solutions. "There is virtually nothing now being done that is applicable to the problem of secure computing in the USAF," and if the Air Force itself did nothing:

> The situation will become even more acute in the future as potential enemies recognize the attractiveness of Air Force data systems as intelligence targets, and perceive how little effort is needed to subvert them.[19]

The Anderson panel proposed an $8 million research and development program to address these problems. It incorporated into its report two notions that Schell had formulated: the "reference monitor" and "security kernel." The former was the requirement "that all references by any program to any program, data or device are validated against a list of authorized types of reference based on user and/or program functions." The Anderson panel defined the latter as the "software portion of the reference monitor and access control mechanisms."[20]

The notion of a security kernel involved a shift in design philosophy. Instead of adding security controls to an existing operating system, security functions were to be isolated into a kind of primitive operating system, directly interacting with the system hardware. The analogy that naturally comes to mind is of concentric shells (see Fig. 5). A kernel "represents a distinct internal security perimeter. In particular, that portion of the system responsible for maintaining internal security is reduced from essentially the entire computer to the kernel."[21] If security functions are properly implemented in the kernel (which was sometimes taken to include security-relevant hardware as well as software),[22] then the design of the rest of the operating system is not critical from a security point of view. All security-relevant requests by nonkernel programs (for example, requests for access to data files) had to invoke the kernel's subroutines, and the kernel would accept only those requests that did not compromise security. Its functions were many fewer than those of a full operating system, so the hope was that it could be kept "simple" and "formal."[23] Although later kernels were to become significantly more complex, a demonstration security kernel commissioned by the Air Force Electronic Systems Division consisted of only about 20 subroutines, totaling around 1,000 instructions. The kernel was developed for the Digital Equipment Corporation (DEC) PDP-11/45 by the Mitre Corporation, an offshoot of MIT's Lincoln Laboratory that was originally set up to take over the latter's responsibilities for the Semiautomatic Ground Environment system.[24]

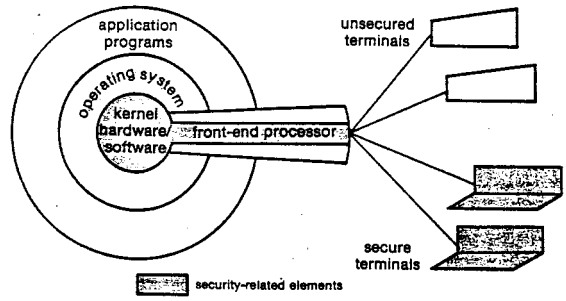


Fig. 5. Security kernel (schematic).
From Roger R. Schell, "Computer Security: The Achilles' Heel of the Electronic Air Force?" *Air Univ. Rev.*, vol. 30, p. 29, Jan.-Feb. 1979.

Embodying security in a small kernel had social as well as technical advantages. A persistent worry in the development of secure systems was the risk that they might be compromised from the very start. A hostile agent who was part of the development team might, for example, deliberately build in a trap door (a surreptitious entry point) to circumvent security controls. It was far easier to guard against this in the development of a kernel than in that of a whole system:

> protecting the kernel ... involves far fewer people and a much more controlled environment ... thus, in contrast to contemporary systems, the kernel makes it tractable to protect against subversion.[25]

The VAX security kernel, discussed below, exemplified what "controlled environment" meant:

> The CPU and console of the development machine were kept inside a lab that only members of the VAX Security Kernel development group could enter. Within that lab, the development machine was protected by a *cage*, which consists of another room with a locked door. Physical access to both the lab and to the cage within the lab was controlled by a key-card security system. ... Our development machine was not connected to Digital's internal computer network, so as to minimize the external threat to our development environment and our project.[26]

Modeling Security

In order to design a security kernel successfully, however, one also had to have a clear notion of what security was. To the Electronics Systems Division's Schell, ad hoc "penetrate and patch" approaches were grossly inadequate. The fact that a system survived tiger team efforts at penetration might mean only that the team had not been skilled or imaginative enough. Indeed, in prac-

tice, the situation was worse than that: Serious tiger team efforts seemed always to succeed, even after expensive "patching" of the vulnerabilities discovered by earlier penetration experiments. This implied:

> the impossibility of arriving at anything that you would classify as a secure system by informal means. I saw ... instances of systems where people had put millions of dollars into making them secure, and to no real avail.[27]

For Schell, what was needed was a mathematical model of security that would:

> raise ... the kernel design and evaluation process above a mere game of wits with an attacker. ... A dramatic effect is that the kernel facilitates objective evaluation of internal security. The evaluator need not examine the nearly endless number of possible penetration attempts; he need only verify that the mathematical model is correctly implemented by the kernel.[28]

The Ware panel had provided a "formal specification of the decision rules ... determining whether an individual with a particular clearance and need-to-know can have access to a quantum of classified information in a given physical environment."[29] Its model (largely restricted to formalizing the no-read-up rule) was never applied practically. The first practical attempt to apply a mathematical model of multilevel military security was the Adept-50 operating system, developed in the late 1960s by Clark Weissman and colleagues at the Rand offshoot, the System Development Corporation, with support from ARPA. Adept-50 was designed to improve the security of a standard commercial computer (the IBM 360 model 50). It embodied an algorithm that dynamically reclassified the security status of files, analyzing the security profile of a job and classifying the files the job created according to the job's security "high-water mark." The term was analogous to "the bath tub ring that marks the highest water level attained."[30]

Schell and the Air Force Electronic Systems Division were not satisfied with either the high-water-mark model or the Ware panel's approach. Indeed, the Anderson panel explicitly criticized the latter for possibly having "a negative effect due to its specification of necessary, but not sufficient, criteria" of security.[31] The tighter definition they sought emerged from research funded by the Electronic Systems Division between 1972 and 1976 at Case Western Reserve University (where K.G. Walter and colleagues developed a security model similar to, if less elaborate than, that developed by David Elliott Bell and Leonard J. LaPadula)[32] and at the Mitre Corporation, where Bell and LaPadula developed the paradigmatic mathematical definition of security that came to bear their names.

Bell and LaPadula's fundamental approach was to model a system as "a relation on abstract sets." A particular influence, especially on LaPadula (see Fig. 6), was the General Systems Theory then current, notably a mathematical version of it put forward by M.D. Mesarovic, D. Macko, and Y. Takahara in 1970.[33] Bell and LaPadula's model of a system was dauntingly abstract in its most general formulation, but as far as security properties are concerned, the key issue is access by "subjects" (not only human users but also their "surrogates," i.e., processes and programs in execution) to "objects" such as data files. Bell and LaPadula realized that the rules of access had to go beyond the no-read-up rule (or simple security property, as they called it) governing access by human readers to documents. In particular, it was vital to have an explicit mechanism to prevent "high classification material [being] added to a low classification file without appropriate changes being made to the security classifications list."[34]

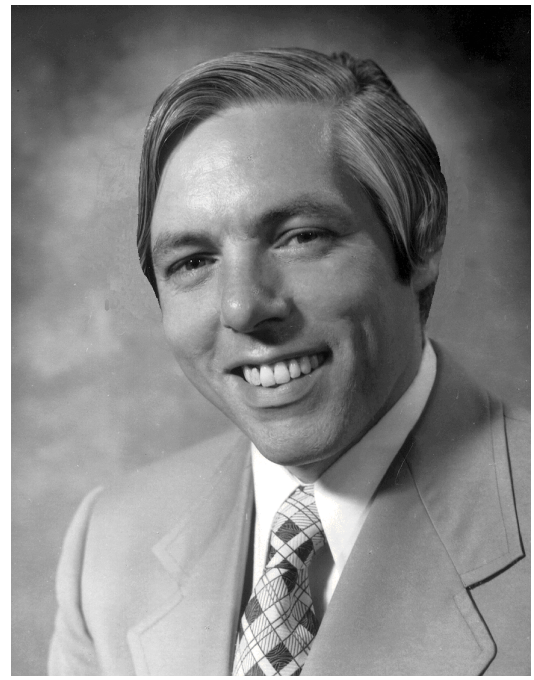


Fig. 6. Leonard J. LaPadula.

Courtesy Leonard J. LaPadula.

In document-based systems, human users were implicitly trusted not to "write down" in this way. However, the broadening of the notion of "subject" to include computer processes and programs raised a new issue: the risk that a hostile agent might insinuate a Trojan horse into a trusted system. Introduced by Daniel Edwards, an NSA representative on the Anderson panel,[35] the term *Trojan horse* referred to a program that, in addition to performing its overt function, surreptitiously violated security controls, for example, by writing classified data to an unclassified file (see Fig. 7).

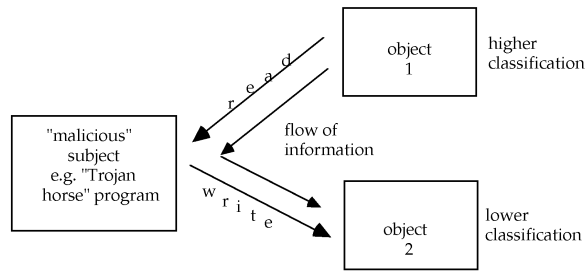


Fig. 7. Why the *-property is needed. (Based on a figure in D.E. Bell and L.J. LaPadula, *Secure Computer System: Unified Exposition and Multics Interpretation.* Bedford, Mass.: Air Force Electronic Systems Division, Mar. 1976, ESD-TR-75-306, p. 17.)

Continually checking each and every program to ensure it was not a Trojan horse would be a daunting task, but Trojan horses could be defeated, Bell and LaPadula realized, if systems were designed in such a way as to ensure subjects *could* not write down. A secure system must satisfy not just the simple security property but also what Bell and LaPadula called the "*-property" (pronounced star property):

> We require that if [a subject] S has *write* or *append* access to some objects and *read* or *write* access to some objects, then the classifications of the objects to which S has *write* or *append* access must exceed or equal the classifications of the objects to which S has *read* or *write* access.[36]

Put more simply, the *-property requires that a subject can have simultaneous "observe" access to one object and "alter" access to another only if the classification level of the second object was greater than or equal to that of the first.[37] If the simple security property was no-read-up, the *-property was no-write-down (see Fig. 8).

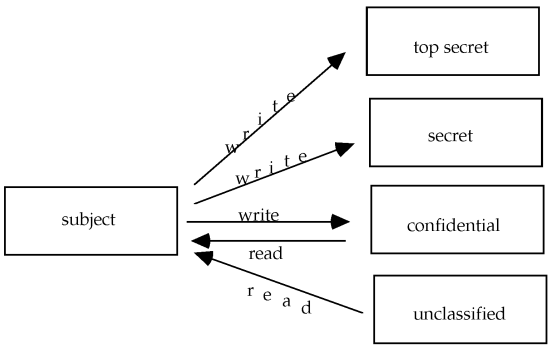


Fig. 8. An example of no-write-down. A subject with read access to confidential objects has write access to confidential, secret, and top-secret objects, but does not have write access to unclassified objects.

As well as formulating and showing the need for *-property, Bell and LaPadula also formulated a theorem (the Basic Security Theorem) that, in their view, greatly simplified the mathematics of security. Security, they concluded, had a mathematically "inductive nature." If changes to the state of a system satisfy the simple security property and *-property, together with a matrix representing discretionary security (where individuals can extend access to a document to anyone permitted by the mandatory security rules to view it), then the system would remain secure. More formally, the Basic Security Theorem is:

> $\Sigma(R, D, W, z_0)$ is a secure system iff [if and only if] z_0 is a secure state and W satisfies the conditions of theorems A1, A2, and A3 for each action.

Σ represents a system; R represents requests (e.g., for access); D represents decisions in response to requests; W represents the rules governing changes of state; z_0 represents an initial state of the system; and A1, A2, and A3 are theorems concerning the characteristics of W that are necessary and sufficient to maintain the simple security property, *-property, and discretionary security property. Bell and LaPadula wrote:

> The importance of this result [the Basic Security Theorem] should not be underestimated. Other problems of seemingly comparable difficulty are not of an inductive nature. ... The result therefore that security (as defined in the model) is inductive establishes the relative simplicity of maintaining security: the minimum check that the proposed new state is secure is both necessary and sufficient for full maintenance of security.[38]

That was an optimistic conclusion—albeit immediately qualified by some important provisos concerning, inter alia, "trusted subjects" and "covert channels" (see below)[39]—and an influential one. In 1979, for example, Schell (by then promoted to lieutenant colonel) spelled out for an Air Force audience what he believed to be the significance of the "foundation of mathematical completeness" provided by the Bell–LaPadula analysis and similar modeling efforts:

> Security theorems have been proved showing that (since the kernel precisely follows the model) the kernel will not permit a compromise, regardless of what program uses it or how it is used. That is, the kernel design is penetration-proof—in particular to all those clever attacks that the kernel designers never contemplated.[40]

Security and Proof

How, though, was an evaluator to verify that a kernel or system was a correct implementation of a mathematical model of security? Here, the history of computer security became intertwined with the more general history of software engineering. During the 1960s, there was a growing sense of dissatisfaction with existing ways of developing software. By 1968, a software crisis had famously been diagnosed.[41] Among a variety of responses to this crisis[42] was a growing interest in showing that programs were correct implementations of their specifications, not just by testing them empirically (as the NSA's Peters had implied in 1967)[43] but also by applying mathematical proof. By 1972, "program proof" or "formal verification" was a focus of much research effort in academic computer science, and it was particularly attractive to those concerned with military security, because it, together with a formal model of what security was, appeared to promise *certainty* that systems were without security flaws.

Without proof, even the most determined efforts to make systems secure could produce ambiguous results. For example, "the ease of defeating" the Adept-50's security controls was "a matter of some debate."[44] The Anderson panel wrote in 1972:

> Because the reference validation mechanism is *the* security mechanism in the system, it must be possible to ascertain that it works correctly in all cases and is always invoked. If this cannot be achieved, then there is no way to know that the reference validation takes place correctly in all cases, and therefore there is no basis for certifying a system as secure. ... Structured programming and program-proving techniques can be used to assure that the [security] model design and implementation correspond.[45]

In the United States, the group most prominent in the early application of "formal verification" to security was SRI International in Menlo Park, Calif. A series of grants in the 1970s from government agencies with responsibility for secure com-

puting supported SRI's work on the development and application of formal techniques. Among members of the SRI group were the following:

- Peter Neumann, who had previously worked at Bell Labs with MIT on the development of Multics, the first important operating system developed with a strong emphasis on security, and who was later to become well-known in computer science generally for his work as convenor of the famous RISKS Bulletin Board;
- Robert S. Boyer, whose joint work with J. Strother Moore (then at the Xerox Palo Alto Research Center) created the Boyer–Moore theorem prover, the key tool used in the automation of proofs in the early SRI work;
- Richard J. Feiertag, who developed an influential tool for analyzing information flow between the variables in a system specification (and for detecting the "covert channels" discussed below);[46] and
- Karl N. Levitt and Lawrence Robinson, who played a crucial role in the development of SRI's overall methodology, called Hierarchical Development Methodology.[47]

Perhaps SRI's most significant individual project was Provably Secure Operating System (PSOS), which began in 1973 and in which the techniques the SRI group developed were applied to the design of an operating system "intended to meet advanced security requirements" and to the verification of its security properties.[48] PSOS was an entire operating system, not a security kernel. Although Feiertag and Neumann also worked on a Kernelized Secure Operating System, the SRI team felt that the advantages of kernels had to be weighed against the disadvantages, such as the inflexibility arising from the fact that a kernel was specific to a particular security model.[49] The SRI team began their work on PSOS with a loosely defined system and gradually moved to a complete specification of the design, which was decomposed into a hierarchy of different modules at different levels. The full formal specification of PSOS was a 400-page volume.[50] Considerable effort was devoted to showing that the most detailed specification of PSOS was a correct implementation of the system's security model (which was basically the Bell–LaPadula model, together with a loosely analogous model covering the integrity of data, rather than data confidentiality).

However, although much work was done on proving that the specifications for PSOS implemented the security model, and some work was done to prove that the implementation of this specification was correct, the 1980 report on the PSOS project, by then seven years old, was careful not to exaggerate what had been done:

> "PSOS" might be considered an acronym for a "potentially secure operating system," in that PSOS has been carefully designed in such a way that it might some day have both its design and its implementation subjected to rigorous proof; considerable effort has gone in to trying to enhance that possibility.[51]

Slow progress with PSOS verification was not an experience unique to SRI and was encountered even when a kernel, rather than an operating system, was the target. At the University of California at Los Angeles (UCLA), considerable effort was expended in the late 1970s to verify the correctness of a kernel designed to allow the Unix operating system to be run securely on a DEC PDP-11 computer. The approach taken was different from that of SRI, with less attention to work on specifications and more on verification "of the lowest level assembly code." Again, though, considerable difficulties were encountered. The work was never completed, although "verification of 35 to 40 percent of the kernel led developers to claim that, given sufficient resources, verification of the entire kernel was feasible."[52]

It was a demanding task to verify formally that a real operating system (or even just a real kernel) conformed to a mathematical model of security.

With the SRI and UCLA research projects meeting with problems, it is not surprising that the most important practical development effort of the 1970s involving formal verification ran into severe difficulties. Automatic Digital Network (Autodin) II was an ambitious plan to provide a multilevel-secure packet-switching system for the Department of Defense: The earlier Autodin I was a much simpler record message system. Two industry teams competed: one, involving the Arpanet prime contractor firm of Bolt, Beranek and Newman, essentially proposed simply adding encryption to a network akin to the existing Arpanet; the other team, led by Western Union, and involving Ford Aerospace and the System Development Corporation, proposed a system based around the emerging computer security notions of a secure kernel and formal verification. The latter team was awarded the contract in late 1976. The request for proposals was, however, drawn up "without adequately defining 'kernel' or the requirements for formal specification and verification. There were many problems in its development, including a court fight over the definition of 'formal specification.'"[53] These problems contributed to difficulties in achieving security certification, and, although this was eventually gained, the system was, by then, two-and-a-half years behind schedule, and there were worries about its cost and survivability in the face of attack. In 1982, it was canceled in favor of a revived version of the original, more standard, cryptographic alternative.[54]

Trusted Subjects and Convert Channels

The difficulties of PSOS, UCLA "data secure Unix," and Autodin II indicated that it was a demanding task to verify formally that a real operating system (or even just a real kernel) conformed to a mathematical model of security, such as the Bell–LaPadula model. One reason for the problems was, simply, that formal verification was a new field, where many of those involved were still feeling their way and where the automated tools to assist them were still immature. Another reason was that practically applying security models, such as the Bell–LaPadula model, to real systems was more difficult than it first appeared. As Bell put it, "simple rules ... need to be refined to accommodate reality better." The Mitre effort, referred to above—to apply the Bell–LaPadula model to a security kernel for the DEC PDP-11/45—led to the realization that the *-property was "overly simple."[55] There was, in practice, a need for "trusted subjects," which could be relied on never to "mix information of different security levels," and which could therefore be allowed to operate without the *-property be-

ing imposed on them. Permitting trusted subjects, then, would "free a design as much as possible from excessive preventive measures."[56]

System designers found trusted subjects to be a practical necessity in multilevel time-sharing systems. Consider, for example, the subject (print spooler or print driver) that controls printing in a multilevel secure system. Unless completely separate print control paths are provided for all security classes, the print spooler must be permitted to violate the *-property. If it is to read, say, top-secret information and write it to the queue for an appropriate printer, the spooler must be a subject with top-secret status. The *-property would then prevent it writing secret, confidential, or unclassified data to queues with only these classification levels. So, it must be trusted to perform writes in violation of the *-property without mixing information of different security levels.[57]

The need for trusted subjects increased the formal verification task. It was not enough to verify that the simple security property and *-property had been implemented correctly in a security kernel. It was also necessary to prove that the nonkernel trusted subjects in a system—such as file system backup and retrieval programs, network interfaces, and input–output spoolers[58]—were indeed trustworthy: that they would not violate security, even if "attacked" by a hostile agent. The "trusted computing base"[59] that had to be subject to formal verification had, therefore, to include the trusted subjects as well as the mechanisms enforcing the simple security property and *-property.

The second practical complication in applying security models was "covert channels," a term introduced into the open literature in 1973 by Butler Lampson of the Xerox Palo Alto Research Center. Lampson was not directly considering military security: His 1973 paper on the "confinement problem" talked about a customer using a computer service who wished to ensure that data could not be read or modified without permission.[60] However, the phenomena Lampson was referring to were already known to defense-sector computer security practitioners,[61] and those theorizing about the latter field, such as Bell and LaPadula,[62] quickly realized that here was an issue they had to tackle.

"Covert channel" is a slippery term, with no entirely stable meaning. In a sense, the notion is parasitic on the enterprise of modeling security: Covert channels are all those means of transferring information that are *not* encompassed by a formal security model, especially the Bell–LaPadula model. In any real, physical computer system, there are, potentially, a variety of mechanisms by which a subject with a high security clearance (a subject that could be a Trojan horse program) could signal information to an uncleared, or less highly cleared, subject without violating the simple security property and *-property.

For example, suppose a subject with read access to top-secret files and an uncleared subject both have access to the same unclassified file. Because of the *-property, the top-secret subject cannot modify the file's contents, but programs can read the file; the uncleared subject can both read it and write to it. Any practical multiuser system will contain an interlock mechanism to prevent a file being modified by one user while being read by another, so the top-secret subject can "signal" to the uncleared subject simply by reading the file, because the uncleared subject's requests to write to the file will be rejected while the top-secret subject is reading it. The interlock is not intended as a repository of information (so it would not normally be treated as a Bell–LaPadula "object"), but it can thus be used covertly for that purpose, with the secret user signaling information, one binary digit at a time, by either reading the file or refraining from doing so.

The above example would be referred to by computer security specialists as a "storage channel," because the interlock is being used covertly to store information.[63] The other type of covert channel is a "timing channel," in which the high-security subject signals to the low-security subject by affecting the amount of time it takes the latter to detect a change in some system attribute or to receive access to some system resource. Nearly all computer systems, for example, contain a clock, which all user programs can read. Suppose (for reasons of simplicity) that only two user programs are present. The higher security program could be designed in such a way that it signaled to the lower security program by occupying the system central processor unit up until a particular clock time. The lower security program could then infer that clock time by recording when it was able to begin execution, and so clock time could be used as a signaling medium.[64]

In any real, physical computer system, there are, potentially, a variety of mechanisms by which a subject with a high security clearance ... could signal information to an uncleared, or less highly cleared, subject without violating the simple security property and *-property.

Of course, any actual multiuser system will normally support more than two users, so a timing channel is likely to be "noisy," and sophisticated information-theory techniques may be needed to exploit it in practice. Furthermore, both storage channels and timing channels will typically have small bandwidths; that is, they transmit information very slowly. However, transmitting information cannot, in practice, be eliminated entirely. A 1976 theoretical analysis suggested that the confinement problem, as Lampson defined it, was unsolvable in the most general case: It was equivalent to the Turing machine halting problem. So:

> there is no hope of finding an algorithm which can certify the safety [i.e., security] of an arbitrary configuration of an arbitrary protection system, or of all configurations for a given system.[65]

That did not rule out finding more-restricted cases where the confinement problem was tractable, but the practical secure systems development efforts of the 1970s found that it was infeasible to remove completely all possible covert channels. In a multilevel time-sharing system, resources *had* to be shared, and efforts to block the resultant covert channels (which typically involved "virtualizing" these resources) often had serious penalties in degradation of system performance for legitimate users. The best that could be done in practice, concluded the authors of one influential project (to develop a security kernel for IBM's widely used System/370 computers), was to reduce covert channels to "acceptable bandwidths."[66]

The Computer Security Evaluation Center

The difficulties of formal verification, the need for trusted subjects, and the practical infeasibility of entirely eliminating covert channels all indicated, by the end of the 1970s, the technical complexity of computer security. Some of its social complexities were also beginning to emerge by then. The most pressing issue in the United States was who should have organizational responsibility for ensuring the security of the design of military computer systems. The obvious answer was the NSA. However, though the NSA was powerful and well-funded, there were tensions between its entrenched culture and the perceived requirements of the new field.

The key relevant facet of the NSA's culture was its approach to what NSA insiders called COMSEC, communications security. The goal of COMSEC was to preserve the communications security of the United States, which was the obverse of NSA's more celebrated role of breaking other nations' codes. Practitioners of COMSEC were steeped in a culture of secrecy and of government dominance of industry. In effect, NSA's experts decided what technologies were needed to protect U.S. government communications, provided industry with detailed instructions as to what it was to produce, and insisted that firms comply with a stringent security classification. The emphasis on secrecy was understandable. Cryptography was central to COMSEC, and traditional codes were immediately vulnerable if their keys were known (this is no longer straightforwardly the case with "public key encryption," but that was a later development). Even knowledge of general design features of encryption devices could be of great use to an enemy.

The COMSEC tradition in the United States stretched back to World War I.[67] The success of Allied COMSEC in World War II, and the successful cracking of the Axis codes, gave the activity considerable status within the inner circles of government and the intelligence community. COMPUSEC, as computer security is known within NSA, was much more recent, much less prestigious, and much less well-entrenched in NSA's hierarchy.

COMPUSEC's attitude to secrecy was different from that of COMSEC. Academics—with their need to publish—were far more important in the emergence of COMPUSEC than they had been in that of COMSEC. The technical bases of the two fields were different. COMSEC typically sought only probabilistic security: codes with an extremely low probability of being broken. At least until the practical infeasibility of entirely eliminating covert channels was accepted, COMPUSEC's goal was deterministic security (systems that *could* not be penetrated) and deductive, proof-based certainty of that security. If these goals were achievable, it was natural to ask whether the secrecy that had to surround COMSEC was necessary for COMPUSEC. In the words of NSA officer George F. Jelen: "If a system could be *proven* to be impenetrable, then there would appear to be nothing gained from secrecy." Jelen also suggests that the different circumstances of the two fields' births left their mark:

> Most of the significant early development of COMSEC devices took place during an era when government was generally trusted. That of COMPUSEC did not. When COMSEC device technology was maturing, the environment was that of a country unified by war against a common, foreign enemy. The current [1985] political environment surrounding the government's efforts in computer security is set against the backdrop of Watergate. ... The "high politics" factor, then, is that in the minds of many people today, the enemy most to be feared is government itself.[68]

More immediately troublesome for computer security practitioners, however, was what Jelen called "low politics": "bureaucratic battles over turf."[69] Despite early interventions such as the Peters speech quoted above, NSA did not move decisively and openly into the new field of computer security, preferring to operate behind the scenes and anonymously. Computer security specialists receiving research grants from NSA, for example, could not openly refer to the organization as their source of funding, and NSA staff attending technical meetings were not normally permitted to disclose their affiliations.

ARPA was suspected by many in the military of insufficient focus on real defense needs.

NSA's secrecy and hesitancy left computer security in something of an organizational vacuum. An obvious alternative to NSA was ARPA, with its considerable experience in supporting state-of-the-art computer research and development in both industry and academia.[70] However, ARPA was suspected by many in the military of insufficient focus on real defense needs. The Anderson panel noted pointedly that ARPA-funded projects "appear to be focusing on one or more interesting (to the principal investigator) research problems, but do not evidence a comprehensive or cohesive approach to solve the ... Computer Security Problem."[71]

As we have seen, the vacuum was filled in the early 1970s by Major Schell and his colleagues at the Air Force's Electronic Systems Division. They gave currency to the notions of reference monitor and security kernel; created the Anderson panel; supported the modeling work of Bell, LaPadula, and others; and, by the mid-1970s, had made significant progress toward practical implementation of many of these ideas. They were, however, less successful in keeping the support of their Air Force superiors (who were perhaps unconvinced that their service should be seeking to solve what was really a generic Department of Defense problem). Without wholehearted top-level Air Force backing, congressional support, in turn, became problematic. The upshot was a sudden cutoff of funding in 1976, which created a "hiatus from which the [Department of Defense] has had difficulty recovering."[72]

Stephen T. Walker was central to the efforts to recover momentum. ARPA hired Walker in 1974 to head its computer security work, and, in early 1978, he moved on to the Office of the Secretary of Defense, where he became the single most influential public figure in U.S. computer security: His formal title was Director, Information Systems, Office of the Assistant Secretary of Defense for Communications, Command, Control, and Intelligence. Although Walker went from NSA to ARPA,[73] he had been profoundly influenced by ARPA's more open style. In particular, he concluded that computer security could not successfully follow the traditional COMSEC approach of government direction and tight security.

Walker favored using government research and development funding (such as the Department of Defense's Computer Security Initiative, that Walker headed and launched in 1978) to encourage

academic and industrial activity in computer security, but he did not believe in directing that activity closely or in shrouding it in a high security classification. Walker was beginning to see computer security as a government-wide problem, not just a problem specific to the Department of Defense. He also wanted "to get industry involved in figuring out how to build trusted systems"[74] and then to have a government agency evaluate how successful firms had been in this effort.

The natural candidate for the evaluative role was, of course, NSA, but its COMSEC culture did not fit the vision Walker was developing. He briefly and unsuccessfully floated a plan for a Program Management Office for computer security, located within NSA but organizationally autonomous.[75] However, he soon broadened his horizons to propose a federal (*not* just Department of Defense) Computer Security Center. Such a center:

> needed to be able to sit down and talk with industry and convince industry to do things, as opposed to the typical NSA communications security model, which was, "I'll write you a contract, I will clear all your people, I will tell you exactly what to do, and you'll do it only my way." I had had extensive discussions in the '79–'80 timeframe with folks at NSA, trying to argue that this needed to be done as an open activity, where you tried to convince people, as opposed to telling them. ... We wanted to get industry to do this as part of their normal product, which they'd make available to anyone, not as a special purpose product just for the Defense Department.[76]

Walker saw funding for such a center coming from the Department of Commerce, as well as Department of Defense, and envisioned a potential home for it at the National Bureau of Standards, a civilian organization (part of the Department of Commerce) that had, inter alia, responsibility for advising federal agencies on computer procurement.[77]

Early in 1980, Walker began circulating this proposal around Washington, D.C. The threat that computer security might move outside its ambit galvanized NSA. Walker realized that NSA was unhappy, and he sought a meeting with its director, Vice Admiral Bobby R. Inman, to explain his ideas. He finally got to meet Inman in August 1980. Thirteen years later, that meeting, in Admiral Inman's impressively huge office, remained vivid in Walker's mind. Inman listened quietly as Walker spoke, until he came to his proposal to involve the Department of Commerce and to site the center in the National Bureau of Standards. Then Inman erupted, rolling forward in his chair, pounding his fist on his desk: "I will never let that happen. I will go to the President to keep that from happening!"[78]

To Walker's surprise ("I was sitting there thinking, *How can I get out that door?*"),[79] Inman's tone then became conciliatory. He asked Walker what had become of his earlier proposal for a computer security center situated within NSA and indicated that he would be willing to support that idea. He even agreed with Walker when the latter insisted "that this should not be done in the same way COMSEC was done."[80] Inman asked Walker to set the bureaucratic procedures in motion by writing to him about a computer security center.

Unknown to Walker, his dramatic meeting with Inman was following a partially prearranged script. Hearing of Walker's campaign several months earlier, Inman had asked Walker's boss, Assistant Secretary of Defense Gerald P. Dineen, to meet with him privately, and the two men had, between them, hammered out the agreement that "emerged" from Walker's meeting with Inman.[81] It was Walker, though, who was responsible for the crucial—albeit, as we shall see, temporary—separation of the computer security center from the NSA COMSEC organization.[82] In the final, lame-duck days of President Carter's administration, Walker worked furiously to flesh out, and gather further support for, what came to be called, first, the Department of Defense Computer Security Evaluation Center and, then (from 1985), the National Computer Security Center. In the words of the 1982 official directive that, finally, officially established the center, it was to be "a separate and unique entity within the NSA," whose prime tasks were to "establish and maintain technical standards and criteria for the evaluation of trusted computer systems," to evaluate actual systems against these criteria, and to conduct and sponsor computer security research and development.[83]

The Orange Book

From the late 1970s onward, attention began to focus on the criteria that the proposed new center would use to evaluate trusted computer systems. By then, there was little doubt that some form of mathematical proof would be necessary for systems in the highest assurance category. However, what form of proof was still an open question. Thus, in 1979, when G.H. Nibaldi of the Mitre Corporation sketched out a hierarchy of security classes for operating systems, the three highest levels were distinguished largely by different requirements for proof. Level 4 required mathematical proof that a detailed specification of the system was a correct implementation of an approved model of security. Level 5 extended this formal verification to the source code of the implemented system. Level 6 extended the analysis (though not necessarily the formal proof) to the object code generated by the compiler. Formal proof ought eventually to extend even to the hardware itself, wrote Nibaldi:

> Axiomatization of the underlying hardware base, and formal verification of the security-relevant hardware mechanisms, are also required. It is recognized, however, that these requirements are beyond the anticipated state-of-the-art of verification in the 1980s.[84]

During the early 1980s, doubts began to appear not just about hardware verification but also about the verification requirements of Nibaldi's Levels 5 and 6. The essential problem was the practical feasibility of formal verification of programs of the size required for a security kernel. As Walker put it in 1980:

> Our success in achieving the widespread availability of trusted systems is more dependent upon progress in the verification field than in any other single activity. It is for this reason that I have made support of verification technology the single primary technology development activity in the Computer Security Initiative.[85]

Despite this support for program verification, progress remained slow. The automated program verification systems that had been developed in the 1970s and early 1980s, often with computer security funding, were still very far from automatic. They needed highly skilled human beings to operate them. For example, the Gypsy verification system, developed by Donald

Good at the University of Texas at Austin, was used in code verification for one early security project, the Encrypted Packet Interface. Using that project as a benchmark:

> yields programmer-verifier productivity levels of perhaps 2-6 verified lines of code per work day, by *highly trained verification specialists*. The entire community of such individuals probably numbers less than 200 individuals.[86]

The Encrypted Packet Interface was unusual among the early verification projects in that it was brought to a successful conclusion: As we have seen, others were simply never finished. Futhermore, though the Encrypted Packet Interface was a substantial program (over 4,000 lines long), it was still considerably smaller than the security kernels being considered in the early 1980s.

So, when the definitive version of the Orange Book, the Department of Defense's *Trusted Computer System Evaluation Criteria*, was issued in 1985 (a final draft was circulated in 1983), code verification was not required, even for the highest evaluation category. The Orange Book provided for four ascending divisions, some with subcategories. Division D consisted of systems that provided minimal or no protection. Division C was systems that contained audit capabilities and could support discretionary security, but were not suited for use in a multilevel environment with users with different levels of security clearance. Division B systems were judged suitable for multilevel use. The trusted computer base of a Division B system preserves "the integrity of sensitivity [i.e., classification] labels and uses them to enforce a set of mandatory access control rules." B2 systems had to be "based on a clearly defined and documented formal security policy model," and in B3 systems there also had to be "a convincing argument" that the specification of the trusted computer base was consistent with the security model. A1 certification (the highest) required use of an automated verification system endorsed by the National Computer Security Center to show, using both "formal and informal techniques," that the system specification was consistent with the security model.[87]

Even for A1 Orange Book systems, therefore, proof meant "design proof" (demonstration that the specification of a system was "correct" in terms of a formal security model) rather than "code proof" (demonstration that an actual program was a correct implementation of the specification). In the early 1980s, it had been expected that a higher, A2, subdivision, incorporating code proof, would eventually be added, but the addition was never made. To some, the restriction of the meaning of "proof" to "design proof" was unjustifiable. Automated theorem proof specialists Boyer and Moore (who had by then left SRI) wrote in 1985:

> Of course, a program whose design has been verified is unworthy of trust until the running program has been shown to implement the design. Especially to be distrusted are those software products constructed by two unrelated teams: those who write the code, and those who simultaneously and independently write the formal specifications which are then checked for security. Alas, several such projects are currently funded by the U.S. Government. This travesty of mathematical proof has been defended with the claim that it at least gives the government better documentation. The Department of Defense has published official standards authorizing this nonsense.[88]

To others, such as SRI's Neumann, "design proof" was a perfectly sensible strategy:

> The attitude of having to prove everything from the hardware, or from the ions, all the way up into the application, is untenable ... by the time you have finished it, the system is no longer what it was when you proved it, because the system tends to be a moving target. ... By the time you get a system evaluated against those criteria [such as the Orange Book], the vendor has already moved on to three or four versions later. And so the idea that one can thoroughly prove a system from stem to start, and from bottom to top, is unreal. So the question is, where does the biggest pay-off come? ... We took the attitude that the code proofs were absolutely irrelevant if the specifications were wrong, and that the immediate pay-off would come from showing that the design was no good. Rather than trying to prove things are correct, you are really trying to find the flaws. So the interesting challenge becomes to model the properties of the system that you are trying to achieve, at whatever layer of abstraction you are dealing with, and to try to prove ... that the specifications are consistent with those properties.[89]

The Encrypted Packet Interface was unusual among the early verification projects in that it was brought to a successful conclusion.

Even with "proof" restricted to design proof, meeting the demands of the Orange Book's A1 category was expensive and difficult. The specialized skills required for formal proof, and the sheer time it took, meant that the tendency to dissociation between design verification and system construction, identified by Boyer and Moore, was certainly present in early efforts at A1.[90] Furthermore, the practical limitations of the available verification systems on the Computer Security Center's Endorsed Tools List were troublesome. The first system to achieve A1 rating was Honeywell's Secure Communications Processor (SCOMP):

> The amount of effort required to verify the SCOMP system was very large. The tools were often very slow, difficult to use, and unable to completely process a complex specification. There were many areas where tedious hand analysis had to be used.[91]

Three sets of verification tools were endorsed: Good's Gypsy, SRI's Hierarchical Development Methodology, and the System Development Corporation's Formal Development Methodology.[92] While support from agencies with computer security interests was certainly important in the development of these tools, not all of those involved agree that its effects were entirely beneficial. After a verification system became an endorsed tool, it had to be converted to be run on the Multics system used by the Computer Security Evaluation Center, a very considerable effort. Furthermore, endorsed tools became subject to export controls. Walker comments:

> They made everybody port everything to Multics at the same time that the world was going to stand-alone machines. ... They burned up everybody's energy on the conversions to Multics, and then they restricted who could see them or use them. So without actually advancing the technology at all, they basically submerged it. ... I remember complaining rather bitterly to management at NSA that, "don't let your people get a hold of some of these technologies, because they'll kill them." I have this image of giant oak trees ... and then these vines start growing up round them. ... Some of these technologies, that are very important, are being killed by the very guys that ought to be trying to promote them.[93]

Nevertheless, progress *was* made. The System Development Corporation, for example, found considerable difficulty in its early work using formal verification on security-critical systems such as Autodin II:

> The field of formal notations and mathematical formalism was new and most programmers unaware of its properties. ... The formal specs were treated like code, not math, and impossible for them to comprehend, let alone prove. [The] "throwing it (formal specs) over the wall" method failed.
>
> Next we tried doing it ourselves with skilled mathematicians ... that was not a success ... because of ... "dissociation" of teams. The [specification] and [code] were in a race to finish, and code won. As is typical in large programming jobs, the code deviated from the DTLS [detailed top-level specification] and the DTLS was not updated. In the end, the FTLS [formal top-level specification] was being developed from the code, a terrible form of "reverse engineering."[94]

With time and experience, along with development of the Formal Development Methodology tools, the state of the art at the System Development Corporation eventually improved considerably by comparison with these earlier problems. This was evidenced by the corporation's Blacker program, an integrated set of devices designed to make the Defense Data Network secure. Development of Blacker began in 1984, and it received A1 rating in 1991, "after 5 years of very detailed ... evaluation" by the National Computer Security Center. Classes were held to educate the programmers involved in the formal notation being used, while

> the formal team was trained on the programming tools and methods. ... To integrate the teams further, the formal team was required to do quality control on all the specs and design documentation, and later the unit code. Many hours of "burned eyeballs" were spent by the formal team reading DTLS [detailed top-level specification] and code and uncovering problems early.[95]

This integration ensured that the design proof was not epiphenomenal to the actual coding, although the former still required considerable skill:

> There is a delicate balance between the level of detail in the spec, the number of invariants, and the difficulty in achieving proofs. Achieving that balance is still an art. We have on average written/proven specs about three times before the balance is achieved. There is a considerable learning curve with experience.[96]

The Fragmentation of Computer Security

Blacker, however, underlined an emerging problem. It was not a classical computer-security system—that is, a kernelized operating system. The problem was in its network role, a system that bridged the fields of COMPUSEC and COMSEC. Its security model, however, was the traditional COMPUSEC Bell–LaPadula model, and design proof against a formal security model was not carried out for its COMSEC functions:

> Since Blacker is to be COMSEC and COMPUSEC secure, we initially considered a security policy that captured both requirements. We had no examples of formal specification of COMSEC design reduced to practice, and Blacker is a product development program, not an R&D vehicle for advancing the state of the art. It was concluded that COMSEC and COMPUSEC was more than A1 certification required, and beyond the state of the art.[97]

In a network, however, the Bell–LaPadula model is not always straightforward to apply: It is, for example, not always clear which entities should be considered "subjects" and which "objects." In the development of Blacker, "there were many unusual situations that arose ... that required going back to first security principles to arrive at a solution."[98] The different cultures of COMSEC and COMPUSEC caused problems:

> It was decided from inception to keep the bulk of the staff and the formal specifications [cleared] at the lowest possible security level, to encourage peer review. That is an underpinning of COMPUSEC, but not, however, of COMSEC. ... Most of the staff are clear to the Secret level, with some special clearances. All the system formal specs were kept Unclassified, Official Use Only. Device formal specs are classified at least Secret.[99]

The growing interconnection of computers into networks blurred the boundary between COMSEC and COMPUSEC. The National Computer Security Center's hard-won autonomy was gradually lost as NSA brought COMSEC and COMPUSEC together. A 1985 reorganization merged NSA's COMSEC and COMPUSEC functions, and, in 1990, the National Computer Security Center's "research and evaluation functions were integrated with the NSA's communication security functions."[100]

Yet another problem was indicated by the six-year gap between SCOMP's A1 rating, achieved in 1985, and Blacker's, achieved in 1991. Other projects aiming at A1 either progressed slowly or were canceled. A dramatic case in point is the security kernel DEC developed for the VAX architecture (see Fig. 9). Despite some dissociation between the formal proof work and code writing ("we never really achieved what I would call ideal coupling between the formal specification/verification process and the system development process,"[101] and the formal top-level specification was never completed), those involved were confident that the VAX security kernel was "capable of receiving an A1 rating," and it "underwent a highly successful external field test,"[102] yet DEC never brought it to market. While some of those involved felt that there were enough potential users,[103] Steven Lipner, then of DEC, believed otherwise:

> The project was abandoned because there was not a sufficient market and Digital (I in particular as the responsible group manager) determined that canceling the project was a better business decision than shipping the product and living with an unprofitable offering.[104]

Export restrictions were part of the reason why DEC decided that the market would be insufficient:

> The current U.S. State Department export controls on operating systems at the B3 and A1 levels are extremely onerous and would likely have interfered with many potential sales, even to close NATO allies.[105]

The project was conceived in 1981, with a feasibility study concluding favorably in 1982. A research prototype of the security kernel was operating by 1984, but it was late 1989 before full external field testing of the VAX security kernel began. By then, its intended hardware base "was late in its life cycle and thus slow and costly compared to the newest VAX offerings."[106] To shorten development time, Ethernet support was not included, an absence that attracted critical comment from potential users.[107] More fundamentally, the world of computing was seen as having moved on: "we were prepared to offer an A1 timesharing system in a world that had largely moved to work stations." DEC had plans to address all these issues, "but doing the requisite development would have cost money that we could not convince ourselves the market would ever repay us."[108]

These were symptoms of a general problem, not just specific to this one particular development. The original version of SCOMP, for example, achieved few sales: "under 30. It never broke even."[109] When SCOMP's developer, Honeywell, decided to rework the system for a newer, more sophisticated hardware platform, the company's marketing staff investigated how much "additional market is opened up by having an A1 system versus a B3 system." The marketing staff came back with the answer: "at best, you are maybe looking at 5 percent." So the decision was taken not "to do any more formal work" and to aim only for the lower rating.[110]

By the early 1990s, there was a widespread perception that the computer security market "has not worked well."[111] There was a vicious circle in which the requirements of developing a demonstrably secure system, and having it evaluated by the Computer Security Center (a process that could take years), led to products that were expensive and, by the time they were on the market, outdated by comparison with analogous technology that had been developed without high security in mind. The temptation for system procurers, then, was to go for the cheaper, more up-to-date, less secure alternative:

> The government was saying all along that they needed this type of system, yet the market was never really as large as people expected. ... You would get ... Requests for Proposals that would imply strong security requirements, and then they would be waived, watered down. ... Things never really materialized like people expected.[112]

A small market increased unit costs, and so the vicious circle continued. Export controls further intensified the problem, however understandable they are from the point of view of national security interests. Besides the desire not to make systems available for detailed scrutiny, there was also the consideration that "adversaries' uses of computer security technologies can hamper U.S. intelligence gathering for national security purposes."[113]

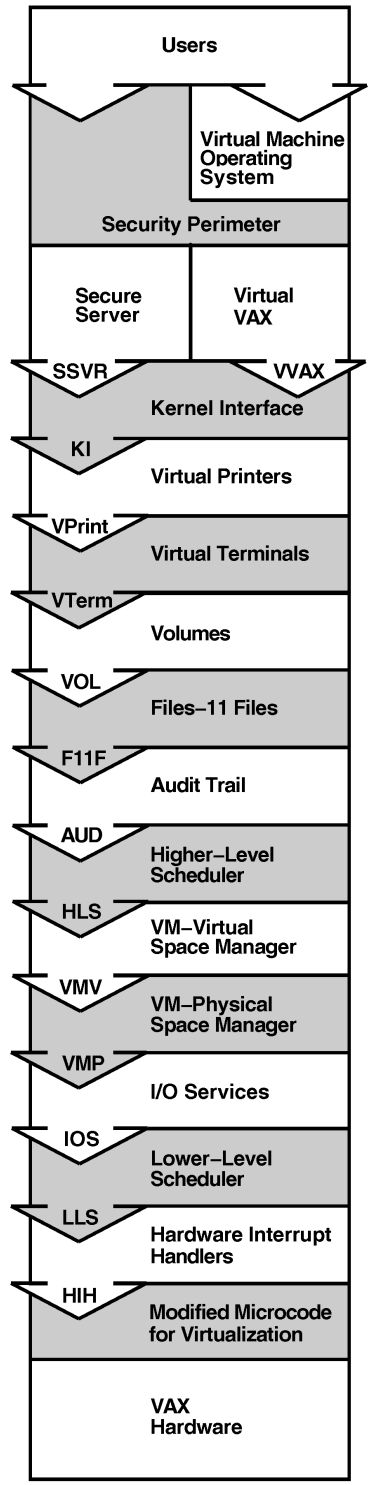


Fig. 9. VAX security kernel layers.
From Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn, "Retrospective on the VAX VMM Security Kernel," *IEEE Transactions on Software Engineering,* vol. 17, p. 1,154, 1991.

Therefore, many saw it was imperative to increase the size of the security-critical market. There were two obvious routes to reach that goal. One was to internationalize the market. The Orange Book had considerable influence on the United States' close allies, with Canada, Germany, the Netherlands, France, and Britain all setting up their own organizations akin to the National Computer Security Center and writing their own analogues to the Orange Book. But overseas, especially in Britain, the Orange Book was perceived as embodying a serious flaw. Its classes were hierarchical in two senses: security functionality and assurance. A Division B system, for example, was intended to have more-sophisticated security functions than a Division C system *and* greater assurance of the correct implementation of those functions. This bundling of security and assurance in the Orange Book was a quite deliberate decision. The goal was to provide a simple system of categories that "the average marketing guy, the average program manager" could understand:

> You are giving him a shorthand for something, that if it meets this level, I can use it here. If I need something better than that, I need something higher. ... People understand. They don't understand what goes into a B2, but they know what to do with it. And that's very, very valuable.[114]

The problem with bundling functionality and assurance was that it ruled out systems that had simple functions but high assurance of the correctness of those functions. Despite the emphasis on simplicity in the notion of a security kernel, systems aiming at A1 were typically quite large. The SCOMP security kernel, for example, involved approximately 10,000 lines of Pascal code, and the trusted software outside the kernel consisted of approximately 10,000 lines of C.[115] The VAX security kernel consisted of almost 50,000 executable statements; even its formal top-level specification would, if it had been completed, have been over 12,000 lines long.[116] Overseas observers concluded that part of the reason for the difficulty of conducting even design proof was the sheer complexity of the systems to which it was being applied. So, they sought to unbundle security and functionality, to provide for the possibility of applying formal verification to simple systems.

The development of international standards forced a decision between the Orange Book bundled approach and the European unbundled one. The international Common Criteria that emerged from this effort largely reflect the European approach, although there is a provision for bundled protection profiles.[117] Even draft U.S. Federal Criteria, issued in December 1992 by the NSA and the National Bureau of Standards (now known as the National Institute of Standards and Technology), went too far for one of the key figures in the background of the Orange Book:

> We have gone way too far [with unbundling]. The *Federal Criteria* makes it even worse, because there is even more detail and more levels, and varying levels of levels. ... It really bothers me that we have just sort of blown this thing wide open, you can pick and choose whatever you want. And people will do that, and they will get hurt real bad.[118]

The other way of extending the computer security market was to include in it sectors that were nonmilitary but had computer security concerns: other departments of government and the commercial sector, especially banking. This, for example, was part of the rationale for elevating the Department of Defense Computer Security Center to the status of National Computer Security Center. However, the attempt to integrate military, nonmilitary governmental, and commercial security was only partially successful. Civil libertarians opposed the extension of NSA's role, and congressional reaction forced the abandonment of the more controversial aspects of the move.[119] Furthermore, civil government and industry did not see themselves as facing the same high-level threat as the defense sector faced, and so did not perceive themselves as requiring the same high-assurance, formally verified systems to meet it.

Its response to any request was to downgrade the security level of every subject and object in the system to the lowest possible level.

In addition, the meaning of "security" for the banking and financial services sector, which obviously did have strong security concerns, was subtly different from its meaning in the defense sector. The primary defense meaning of "security" was confidentiality: prevention of unauthorized disclosure of data. Banks and financial institutions, on the other hand, were more interested in "security" in the sense of integrity: prevention of unauthorized alteration of data.[120] While integrity was clearly of importance in military systems, and had been modeled in the early 1970s' Mitre work,[121] it was never as prominent a concern as confidentiality. Furthermore, other sectors did not have the military's elaborate system of multilevel clearances and security classifications. The A and B divisions of the Orange Book, designed to satisfy the needs of a military multilevel environment, were largely irrelevant to the requirements of other sectors. What nonmilitary users felt they needed were systems with security functions of the type of the Orange Book's Division C. The relatively low levels of assurance that went with Division C were not a major concern.

Even within military security, the dominance of the Bell–LaPadula model had been severely shaken by the late 1980s. It had been sharply criticized, especially by John McLean, one of a group of computer security specialists at the U.S. Naval Research Laboratory.[122] In 1985, McLean claimed that the Bell–LaPadula Basic Security Theorem (described above) could be proven for a system—"System Z"—that was patently insecure, in that its response to any request was to downgrade the security level of every subject and object in the system to the lowest possible level.[123] Whether McLean's System Z genuinely refutes the Bell–LaPadula model is a controversial matter that space prohibits us from discussing here.

Influential alternatives to the Bell–LaPadula model emerged during the 1980s. The first important one was the model SRI's Joseph Goguen and José Meseguer developed. Its basic concept was noninterference:

> One group of users, using a certain set of commands, is non-interfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see.[124]

Fig. 10. A group, mainly of formal methods specialists, attending a lecture by Professor C.A.R. Hoare of Oxford University at the National Security Agency, circa 1987.

Courtesy Carl Landwehr, U.S. Naval Research Laboratory.

The noninterference notion was more general than the Bell–LaPadula model. For example, covert channel analysis (essentially a separate activity in the Bell–LaPadula approach) was subsumed under noninterference. The noninterference model was first used on a real system in the development of Honeywell's LOCK (LOgical Coprocessing Kernel). LOCK involves a hardware-based reference monitor, SIDEARM, designed to be integrated into a variety of host systems to enforce security.[125] However, critics have seen the price of the new model's generality as an even greater constraint:

> Unfortunately, by providing a means to define even more rigid security controls this approach [noninterference] exacerbated the conflict between security and functionality.[126]

Conclusion

In the late 1970s and early 1980s, the path to achieving computer security appeared clear. A dominant problem had been identified: designing a multilevel secure operating system. The route to a solution—implementing a reference monitor in a security kernel—was widely agreed. There was a single dominant model of what security meant: the Bell–LaPadula model. There was widespread acceptance that formal proof should be applied to demonstrate correspondence to that model, and it was anticipated that while proof might initially mean design proof, code proof would follow. Formal methods—on the face of it, an academic and abstract approach to computer science—had aroused the interest of powerful, practically minded organizations such as the NSA (see, for example, Fig. 10) and even secured their endorsements. The United States played the unquestioned leading role in matters of computer security, and, within the United States, there was at least outline agreement as to the appropriate nature and role of an organization to certify the security of actual systems.

A decade later, this partial consensus had fragmented. As computing moved beyond the traditional multiuser mainframe, the classical computer security problem had largely been superseded by a host of more diverse problems, many having to do with the integration of computer and communications security in networks; and there was no clear unitary route to the solution of network security. The Bell–LaPadula model was no longer dominant. Instead of formal verification of trusted systems progressing downward from formal specifications deeper into systems, as had been anticipated at the start of the 1980s, it remained frozen as "design verification," and even the latter was no more common in practice in the mid-1990s than it had been in the mid-1980s. The United States was no longer as dominant as it had been, and the workings of its computer security evaluation system had been criticized sharply.[127]

In the interim, of course, the Cold War had ended, and the Soviet Union had collapsed. The consequent cuts in defense budgets threw a harsh light on the costs of high-security, high-assurance systems designed specifically for defense needs, and the search was on for cheaper commercial off-the-shelf solutions. Yet, the fragmentation of computer security was not the result of the end of the Cold War alone. Computer security was torn by internal conflicts, in particular, the vicious circle that could be seen, even in the 1980s, undermining the high-security end of the computer security marketplace. These conflicts mean that, even though there is sharply increased interest in computer security evident in the latter part of the 1990s (with reports claiming large numbers of unauthorized intrusions into defense computer systems and high-level conferences on "information warfare"),[128] it is unlikely that the fragmentation of the classical computer security approach will be reversed.

Commercial off-the-shelf products represent one form of synthesis in the face of this fragmentation, with the same products meeting both commercial and defense security needs. (Microsoft's Windows NT, for example, is being used increasingly in the defense sector, although it is worth noting that, as of summer 1996, its C2 rating is dependent on it being run on stand-alone machines with their floppy drives disabled).[129] As noted above, this synthesis is likely to be primarily at assurance levels corresponding to the Orange Book's Division C, not higher. In particular, there is no immediate likelihood of the commercial sector demanding formal verification of computer security products.

Another possible form of synthesis is as yet far more tentative.

The need to know the properties of computer systems with great certainty is at least as strong in safety-critical systems as in security-critical ones. The two spheres have largely been separate socially, with different constituencies of sponsoring organizations and only partial overlap in suppliers. Yet, there are important potential commonalities of interest. As computer systems develop, safety and security concerns have begun to merge. In some versions of the Boeing 777, for example, there is a physical interconnection between in-flight entertainment units on each seat and flight-critical computer systems. So, the developers of the 777's computer systems had to attend to the computer security problem of ensuring that passengers cannot interfere with flight control computers. They also used a traditional security technique—tiger team attempts at penetration—to check the partitioning protections between different processes in the flight control software.[130]

The concerns we discussed in this paper meant that security-critical systems were the leading early field of application of formal verification, especially in the United States. As computer security has fragmented, that leadership is passing to safety-critical systems, with Europe playing a far more important role there than in security. Perhaps, then, the skills so painstakingly built up to provide high assurance of security will blossom in a worldwide effort to provide high assurance of safety.

Acknowledgments

This research was supported by the U.K. Economic and Social Research Council (grant R000290008 and, earlier, WA35250006) and by the U.K. Science and Engineering Research Council/Safety Critical Systems Research Programme (GR/J58619). Additional support was provided by the U.S. Navy through the Naval Research Laboratory and by the U.S. Army Research Office through the Mathematical Sciences Institute of Cornell University. We are grateful to our interviewees, without whom this paper could not have been written, and would like particularly to thank Carl Landwehr for his help and advice. Of course, the views expressed here, and any errors we have committed, are entirely our responsibility.

References

[1] See, e.g., General Accounting Office, *Information Security: Computer Attacks at Department of Defense Pose Increasing Risks*. Washington, D.C.: General Accounting Office, 1986, GAO/AIMD-96-84.

[2] For a good description of batch operation and of the shift to time-sharing, see A.L. Norberg and J.E. O'Neill, *A History of the Information Processing Techniques Office of the Defense Advanced Research Projects Agency*. Minneapolis, Minn.: Charles Babbage Institute, Oct. 1992, chapter 1 and appendix 1; an amended version of this report has been published as *Transforming Computer Technology: Information Processing for the Pentagon, 1962-1986*. Baltimore, Md.: Johns Hopkins Univ. Press, 1996.

[3] K.L. Wildes and N.A. Lindgren, *A Century of Electrical Engineering and Computer Science at MIT, 1882-1982*. Cambridge, Mass.: MIT Press, 1985, p. 348. Project MAC had wider aims, captured in the alternative version of the acronym, Machine-Aided Cognition, for which see, e.g., Norberg and O'Neill, *Transforming Computer Technology*, and P.N. Edwards, *The Closed World*: *Computers and the Politics of Discourse in Cold War America*. Cambridge, Mass.: MIT Press, 1996.

[4] See, for example, J.B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," *J. ACM*, vol. 12, pp. 589-602, esp. 599, 1965.

[5] Norberg and O'Neill, *A History of the Information Processing Techniques Office of the Defense Advanced Research Projects Agency*; K.C. Redmond and T.M. Smith, *Project Whirlwind: The History of a Pioneer Computer*. Bedford, Mass.: Digital Press, 1980; C. Baum, *The System Builders: The Story of SDC*. Santa Monica, Calif.: System Development Corporation, 1981; Edwards, *The Closed World*.

[6] H.H. Goldstine, *The Computer from Pascal to von Neumann*. Princeton, N.J.: Princeton Univ. Press, pp. 253, 306-307, 314, 1972.

[7] W.H. Ware, personal communication to D. MacKenzie, 17 Oct. 1996.

[8] W.H. Ware, "Security and Privacy in Computer Systems," *AFIPS Conf. Proc., Spring Joint Computer Conf.* Washington, D.C.: Thompson Books, vol. 30, pp. 279-282, at p. 279, 1967.

[9] B. Peters, "Security Considerations in a Multi-Programmed Computer System," *AFIPS Conf. Proc., Spring Joint Computer Conf.* Washington, D.C.: Thompson Books, 1967, vol. 30, pp. 283-286, at p. 283, 1967.

[10] E.W. Pugh, L.R. Johnson, and J.H. Palmer, *IBM's 360 and Early 370 Systems*. Cambridge, Mass.: MIT Press, pp. 362-363, 1991.

[11] Peters, "Security Considerations," p. 283.

[12] Peters, "Security Considerations," p. 283.

[13] Peters, "Security Considerations," pp. 283 and 285, emphasis in original.

[14] W.H. Ware, personal communication to D. MacKenzie, 17 Oct. 1996.

[15] W.H. Ware, ed., *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security*. Santa Monica, Calif.: Rand Corporation, Feb. 1970, R-609. Originally classified "confidential," the report was declassified in Oct. 1975. The quotations are from Ware's foreword to the version reissued by Rand in Oct. 1979 and from p. 18 of the latter.

[16] Ibid., pp. 36, 8, and 29, respectively.

[17] R. Schell, telephone interview by G. Pottinger, 10 Oct. 1993.

[18] W.H. Ware, personal communication to D. MacKenzie, 21 Oct. 1996.

[19] J.P. Anderson, *Computer Security Technology Planning Study*. Bedford, Mass.: HQ Electronic Systems Division, U.S. Air Force, 1972, ESDF-TR-73-51, vol. 1, pp. 3 and 33.

[20] Ibid, footnote 9, footnote 14.

[21] R.R. Schell, "Computer Security: The Achilles' Heel of the Electronic Air Force?" *Air Univ. Rev.*, vol. 30, pp. 16-33, at pp. 28-29, Jan.-Feb. 1979.

[22] G. Pottinger, *Proof Requirements in the Orange Book: Origins, Implementation, and Implications*, typescript, p. 39, Feb. 1994.

[23] Anderson panel, op. cit., p. 15.

[24] W.L. Schiller, *Design of a Security Kernel for the PDP-11/45*. Bedford Mass.: Electronic Systems Division, Dec. 1973, ESD-TR-73-294; Schell, "Computer Security," op. cit., p. 28; Wildes and Lindgren, *A Century*, op. cit., p. 300.

[25] R.R. Schell, "Computer Security: The Achilles Heel of the Electronic Air Force?" *Air Univ. Rev.*, vol. 30, pp. 16-33, at p. 31, Jan.-Feb. 1979.

[26] P.A. Karger, M.E. Zurko, D.W. Bonin, A.H. Mason, and C.E. Kahn, "Retrospective on the VAX VMM Security Kernel," IEEE *Transactions on Software Engineering*, vol. 17, pp. 1,147-1,165, at p. 1,159, 1991, emphasis in original.

[27] Schell interview, op. cit.

[28] Schell, "Computer Security," op. cit., p. 29.

[29] Ware report, op. cit., p. 48.

[30] C. Weissman, "Security Controls in the ADEPT-50 Time-Sharing System," *Proc. FJCC*, vol. 5, pp. 119-131, quotation at p. 122, 1969.

[31] Anderson panel, op. cit., vol. 1, p. 4.

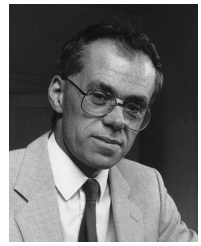
[32] K.G. Walter, W.F. Ogden, W.C. Rounds, F.T. Bradshaw, S.R. Ames, and D.G. Shumway, *Primitive Models for Computer Security*. Bedford, Mass.: Air Force Electronic Systems Division, 25 Jan. 1974, AD-778 467.

[33] D.E. Bell and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations*. Bedford, Mass: Air Force Electronic Systems Division, Nov. 1973, ESD-TR-73-278, vol. 1; L.J. LaPadula, personal communication to D. MacKenzie, 29 Oct. 1996; M.D. Mesarovic, D. Macko, and Y. Takahara, *Theory of Hierarchical, Multilevel, Systems*. New York: Academic Press, 1970.

[34] D.E. Bell and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*. Bedford, Mass: Air Force Electronic Systems Divi-

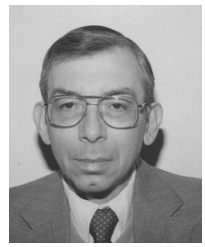
sion, Nov. 1973, ESD-TR-73-278, vol. 2, p. 15.
[35] Anderson panel, op. cit., vol. 2, p. 62.
[36] D.E. Bell and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*, op. cit., vol. 2, p. 17.
[37] D.E. Bell and L.J. LaPadula, *Secure Computer System: Unified Exposition and Multics Interpretation.* Bedford, Mass: Air Force Electronic Systems Division, Mar. 1976, ESD-TR-75-306, p. 17.
[38] Ibid., 94, pp. 19-21.
[39] Ibid., pp. 64-73.
[40] Schell, "Computer Security," op. cit., p. 29.
[41] P. Naur and B. Randell, eds., *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee.* Brussels: NATO Scientific Affairs Division, p. 120, 1969.
[42] E. Peláez, *A Gift From Pandora's Box*: T*he Software Crisis.* PhD thesis, Univ. of Edinburgh, 1988.
[43] Peters, "Security Considerations," op. cit., p. 285.
[44] C.E. Landwehr, "The Best Available Technologies for Computer Security," *Computer*, vol. 16, pp. 86-100, at p. 96, 1983.
[45] Anderson panel, op. cit., 10.
[46] R.J. Feiertag, *A Technique for Proving Specifications Are Multi-Level Secure.* Menlo Park, Calif.: SRI Computer Science Laboratory, Jan. 1980, CSL-109.
[47] L. Robinson and K.N. Levitt, "Proof Techniques for Hierarchically Structured Programs," *Comm. ACM*, vol. 20, pp. 271-283, 1977.
[48] P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, and L. Robinson, *A Provably Secure Operating System: The System, Its Applications, and Proofs.* Menlo Park, Calif.: SRI Computer Science Laboratory, May 1980, CSL 116.
[49] R.J. Feiertag and P.G. Neumann, "The Foundations of a Provably Secure Operating System (PSOS)," *National Computer Conference.* New York: AFIPS, 1979, pp. 329-343.
[50] P. Neumann interviewed by A.J. Dale, Menlo Park, Calif., 25 Mar. 1994.
[51] Neumann et al., op. cit., p. 2.
[52] Landwehr, "Technologies for Computer Security," op. cit., p. 96.
[53] Landwehr, "Technologies for Computer Security," op. cit., p. 97.
[54] Jelen, "Information Security," op. cit., pp. III-83 to III-91.
[55] D.E. Bell, "Concerning 'Modelling' of Computer Security," *IEEE Computer Society Symp. Security and Privacy*, 1988, pp. 8-13, at p. 12.
[56] D.E. Bell, *Secure Computer Systems: A Refinement of the Mathematical Model.* Bedford, Mass.: Air Force Electronic Systems Division, Apr. 1974, ESD-TR-73-278, vol. 3, pp. 29 and 31.
[57] This example is taken from G. Pottinger, *Proof Requirements in the Orange Book*, op. cit., p. 46.
[58] We draw these examples from Feiertag and Neumann, "Foundations of PSOS," p. 333.
[59] A term used, e.g., by G.H. Nibaldi, *Proposed Technical Evaluation Criteria for Trusted Computer Systems.* Bedford, Mass.: Mitre Corporation, Oct. 1979, M79-225, p. 9.
[60] B.W. Lampson, "A Note on the Confinement Problem," *Comm. ACM*, vol. 16, pp. 613-615, 1973.
[61] Schiller et al., "Design of a Security Kernel," op. cit., p. 7; C.E. Landwehr, "Formal Models for Computer Security," *Computing Surveys*, vol. 13, pp. 247-278, at p. 252, Sept. 1981.
[62] Bell and LaPadula, *Unified Exposition and Multics Interpretation*, pp. 67ff.
[63] See, e.g., M. Schaefer, B. Gold, R. Linde, and J. Scheid, "Program Confinement in KVM/370," *ACM 77: Proceedings of the Annual Conference.* New York: Association for Computing Machinery 1977, pp. 404-410; J.T. Haigh, R.A. Kemmerer, J. McHugh, and W.D. Young, "An Experience Using Two Covert Channel Analysis Techniques on a Real System Design," *IEEE Transactions on Software Engineering*, vol. 13, pp. 157-168, 1987.
[64] See, e.g., J.K. Millen, "Security Kernel Validation in Practice," *Comm. ACM*, vol. 19, pp. 244-250, 1976.
[65] M.A. Harrison, W.L. Ruzzo, and J.C. Ullman, "Protection in Operating Systems," *Comm. ACM*, vol. 19, pp. 461-471, at p. 470, 1976.
[66] Schaefer et al., "Program Confinement in KVM/370," op. cit., p. 409.
[67] See D. Kahn, *The Codebreakers.* London: Sphere, 1973; and J. Bamford, *The Puzzle Palace: A Report on America's Most Secret Agency.* Boston: Houghton Mifflin, 1982.
[68] G.F. Jelen, *Information Security: An Elusive Goal.* Cambridge, Mass.: Harvard Univ., Center for Information Policy Research, June 1985, P-85-8, pp. III-43, I-II.
[69] Ibid., I-11.
[70] Norberg and O'Neill, *Transforming Computer Technology*.
[71] Anderson panel, op. cit., vol. 1, p. 4, parenthetical remark in original.
[72] Jelen, *Information Security*, op. cit., pp. i and II-74 to II-75.
[73] Jelen, *Information Security*, op. cit., p. II-75.
[74] S.T. Walker, interviewed by G. Pottinger, Glenwood, Md., 24 Mar. 1993.
[75] Jelen, *Information Security*, op. cit., p. II-79.
[76] Walker interview, op. cit.
[77] "A Plan for the Evaluation of Trusted Computer Systems," typescript, 22 Feb. 1980, reprinted in Jelen, *Information Security*, op. cit., pp. V-2 to V-9.
[78] Walker interview.
[79] Ibid.
[80] Walker, quoted by Jelen, *Information Security*, op. cit., p. II-81.
[81] Admiral B.R. Inman, as interviewed by G. Jelen, as quoted in Jelen, *Information Security*, op. cit., p. II-81.
[82] S.T. Walker, personal communication to D. MacKenzie, 22 Aug. 1996.
[83] F.C. Carlucci, Department of Defense Directive 5215.1, "Computer Security Evaluation Center," 25 Oct. 1982, reproduced in Jelen, *Information Security*, op. cit., pp. V-11 to V-17.
[84] G.H. Nibaldi, *Proposed Technical Evaluation Criteria for Trusted Computer Systems.* Bedford, Mass.: Mitre Corporation, Oct. 1979, M79-225, p. 37.
[85] S.T. Walker, "Thoughts on the Impact of Verification Technology on Trusted Computer Systems (and Vice Versa)," *Software Engineering Notes*, vol. 5, p. 8, July 1980.
[86] W.D. Young and J. McHugh, "Coding for a Believable Specification to Implementation Mapping," *IEEE Computer Society Symp. Security and Privacy*, pp. 140-148, at p. 140, Washington, D.C., 1987, emphasis in original.
[87] Department of Defense, *Trusted Computer System Evaluation Criteria*. Washington, D.C.: Department of Defense, Dec. 1985, DOD 5200.28-STD, pp. 19, 26, 40, and 50.
[88] R.S. Boyer and J.S. Moore, "Program Verification," *J. Automated Reasoning*, vol. 1, pp. 17-23, at p. 22, 1985.
[89] Neumann interview, op. cit.
[90] Pottinger, *Proof Requirements in the Orange Book*, op. cit.
[91] T.C.V. Benzel, "Verification Technology and the A1 Criteria," *Software Engineering Notes*, vol. 10, pp. 108-109, at p. 109, Aug. 1985.
[92] For descriptions of them, see M.H. Cheheyl, M. Gasser, G.A. Huff, and J.K. Millen, "Verifying Security," *Computing Surveys*, vol. 13, pp. 279-339, Sept. 1981.
[93] Walker interview, op. cit.
[94] C. Weissman, personal communication to G. Pottinger, 18 Dec. 1993.
[95] Ibid.
[96] C. Weissman, "Blacker: Security for the DDN. Examples of A1 Security Engineering Trades," *IEEE Computer Society Symp. Research in Security and Privacy*, pp. 286-292, at p. 289, 1992. This paper was presented by Weissman to the 1988 *IEEE Symp. Research and Privacy*, but "not published at that time because of a four year rescission of publication release," ibid., p. 286.
[97] Ibid., p. 289.
[98] Weissman electronic mail, op. cit.
[99] Weissman, "Blacker," op. cit., p. 291.
[100] National Research Council, System Security Study Committee, *Computers at Risk: Safe Computing in the Information Age.* Washington, D.C.: National Academy Press, 1991, p. 195.
[101] S. Lipner, personal communication to G. Pottinger, 22 Oct. 1993.
[102] P.A. Karger, M.E. Zurko, D.W. Bonin, A.H. Mason, and C.E. Kahn, "Retrospective on the VAX VMM Security Kernel," *IEEE Transactions on Software Engineering*, vol. 17, pp. 1,147-1,165 at p. 1,147, 1991.
[103] Ibid., p. 1,163.
[104] Lipner, personal communication, op. cit.
[105] Karger et al., "A Retrospective," op. cit., p. 1,163.
[106] Lipner, personal communication, op. cit.

[107] Karger et. al., "A Retrospective," op. cit., p. 1,163.
[108] Lipner, personal communication, op. cit.
[109] C. Bonneau, telephone interview by G. Pottinger, 20 Nov. 1993.
[110] Ibid.
[111] National Research Council, *Computers at Risk*, op. cit., p. 143.
[112] Bonneau interview, op. cit.
[113] National Research Council, *Computers at Risk*, op. cit., p. 154.
[114] Walker interview, op. cit.
[115] Benzel, "Verification Technology," op. cit., p. 108.
[116] Karger et al., "A Retrospective," p. 1,156.
[117] *Common Criteria for Information Technology Security Evaluation.* Cheltenham, Gloucesteshire, U.K., IT Security and Certification Scheme, 1996.
[118] Walker interview, op. cit.
[119] Pottinger, *Proof Requirements in the Orange Book*, op.cit., pp. 16-17.
[120] D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *IEEE Computer Society Symp. Security and Privacy*, pp. 184-194, Apr. 1987.
[121] K.J. Biba, *Integrity Considerations for Secure Computer Systems.* Bedford, Mass.: Air Force Electronic Systems Division, 1977, ESD-TR-76-372.
[122] C.E. Landwehr, C.L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," *ACM Trans. Computer Systems*, vol. 2, pp. 198-222, 1984.
[123] J. McLean, "A Comment on the 'Basic Security Theorem' of Bell and LaPadula," *Information Processing Letters*, vol. 20, pp. 67-70, 1985.
[124] J.A. Goguen and J. Meseguer, "Security Policies and Security Models," *Proc. Berkeley Conf. Computer Security*. Los Alamitos, Calif.: IEEE CS Press, 1982, pp. 11-22.
[125] T. Fine, J.T. Haigh, R.C. O'Brien, and D.L. Toups, "Noninterference and Unwinding for LOCK," *Computer Security Foundation Workshop*, Franconia, N.H., 11 June 1989. For a description of LOCK, see National Research Council, *Computers at Risk*, op. cit., pp. 251-252. LOCK is now known as the Secure Network Server, and the Honeywell division responsible for it is now the Secure Computing Corporation.
[126] C.T. Sennett, *Formal Methods for Computer Security.* Malvern, Worcestershire, U.K.: Defence Research Agency, 1995, typescript, p. 2.
[127] National Research Council, *Computers at Risk*, op cit.
[128] See, e.g., General Accounting Office, *Information Security*, op. cit.
[129] Anon, "Microsoft Admits Limited NT Security," *Computing*, 4 July 1996, p. 3.
[130] P. Mellor, *Boeing 777 Avionics Architecture: A Brief Overview.* London: City Univ. Centre for Software Reliability, 1995, typescript.



Donald MacKenzie holds a personal chair in sociology at Edinburgh University, where he has taught since 1975. He is the author of *Statistics in Britain, 1865-1930: The Social Construction of Scientific Knowledge* (Edinburgh: Edinburgh Univ. Press, 1981), *Inventing Accuracy: A Historical Sociology of Nuclear Missile Guidance* (Cambridge, Mass.: MIT Press, 1990), and *Knowing Machines: Essays on Technical Change* (Cambridge, Mass.: MIT Press, 1996). In 1992, he was awarded the IEEE Life Members' Prize in Electrical History for his article "The Influence of the Los Alamos and Livermore National Laboratories on the Development of Supercomputing," *Annals of the History of Computing*, vol. 13, pp. 179-201, 1991. He has written numerous other articles in the sociology and social history of science and technology and edited, with Judy Wajcman, *The Social Shaping of Technology* (Milton Keynes, Buckinghamshire, U.K.: Open Univ. Press, 1985).

The author can be contacted at
Department of Sociology
University of Edinburgh
18 Buccleuch Place
Edinburgh EH8 9LN, Scotland
e-mail: D.MacKenzie@ed.ac.uk



Garrel Pottinger, a philosophically trained logician, has been working with computers full time since 1983. He holds a BA degree from Michigan State University and MA and PhD degrees from the University of Pittsburgh. He has been a member of the philosophy faculties of Hayward State University and Carnegie Mellon University. He joined the computer and information science faculty at SUNY Potsdam in 1983. In 1986, Dr. Pottinger moved from academe to industry, joining the technical staff of Odyssey Research Associates in Ithaca, New York. At Odyssey, he was involved in developing two formal verification systems, one of which was designed for use in verifying security properties of computer systems. Pottinger left Odyssey in 1991. He was a Visiting Fellow at the Mathematical Sciences Institute, Cornell University, from 1992 to 1995. His tenure there overlapped a period of employment at Maines and Associates that began in 1994 and ended in 1996, when he accepted his present position as a member of the technical staff of Digicomp Research Corp.

The author can be contacted at
Digicomp Research Corp.
930 Danby Road
Ithaca, NY 14850, U.S.A.
e-mail: 75054.2156@compuserve.com