Looking Back at the Bell-La Padula Model

David Elliott Bell
Reston VA, 20191
December 7, 2005

Abstract

The Bell-La Padula security model produced conceptual tools for the analysis and design of secure computer systems. Together with its sibling engineering initiatives, it identified and elucidated security principles that endure today. This paper reviews those security principles, first in their own time, and then in the context of today's computer and network environment.

1. Looking Back

I look back at the Bell-La Padula Model over a career in security engineering that began with a concentrated burst of security modeling between 1972 to 1975. It is difficult, therefore, to limit myself to modeling and to exclude security topics without which real systems would never reach the field. I choose, then, to look back on both the modeling work and its engineering siblings so as to highlight their contributions to the DNA of network and computer security. What follows is not a synthesized chronicle of everything that happened but my own experiences and knowledge since the publication of the Bell-La Padula model.

2. Before the Bell-La Padula Model

In the late 1960's, developments in commercial operating systems suggested the possibility of tremendous cost savings. Time-sharing was starting to provide commercial customers the ability to share the leasing costs of IBM and other big-iron computers through simultaneous or sequential use of the expensive mainframe computers. For those in classified government circles, this new capability promised even more savings. Before time-sharing, separate computers had to be used for each different security level which was processed on computers, or careful "color changes" had to be made so that the same equipment could be used sequentially to process information at different security levels (referred to as "periods processing"). There was therefore the possibility of sharing those computer systems across security levels, with an important proviso. It was crucial that that processing artifacts of each security level (files, registers, data) be kept rigorously separate with a high degree of confidence.

An initial effort in this direction was commissioning computer experts to test the security robustness of computer systems that were developed in response to market forces. The experts were called "tiger teams." The success of the tiger teams was spectacular. "It is a commentary on contemporary systems that none of the known tiger team efforts has failed to date" [1]. The situation was in reality even worse than it first sounds. Tiger Teams, flush with success in attacking and taking over system A, would try their successful system-A attacks on system B. Alarmingly, many previous attacks worked immediately. Even more worrying were the possibilities opened by a successful attack. After capturing the system and inserting a back-door entrance, penetrators could report the initial flaw and gain a reputation for good citizenship. This planting of back-doors, particularly back-doors that would persist through system and compiler recompilations, was documented in an Air Force report [2] and was the direct stimulus for the back door Ken Thompson described in his Turing lecture [3] [4].

The conclusions drawn from Tiger Teams included the ultimate futility of "penetrate and patch" and the necessity of designing and building computer systems using a sound notion of "security" in computer systems. The U.S. Air Force initiated a set of engineering tasks, paired with several parallel modeling efforts to produce "...a formal statement of what is meant by a secure system — that is a model or ideal design" [5].

3. Bell-La Padula Model, 1972–1975

3.1. Problem Statement

In the summer of 1972, The MITRE Corporation initiated its task to produce a report entitled "Secure Computer Systems." The report was to describe a "mathematical model of security in computer systems." This task was one of several in an overall security project, mostly engineering tasks.

The modeling task fell to Len La Padula and me. Our initial review of available literature showed that some of existing papers were theoretical (emphasizing abstract computation), while others were intimately tied to specific computer operating systems. Our plan was to stay free of system-specific details, first identifying and investigating general principles, and afterwards addressing specific solutions.

3.2. Mathematical Foundations

Definition. Our initial work focused on a definition of "security" within a mathematical (conceptual) framework. At the start, we viewed access monolithically, rather like possession of a book from a lending library. Having just completed abstract investigations into data sharing and appreciative of the complexities of "deadly embrace," we began to think about the complexities that would result when an "object" (the generalization of all things that could hold information) had its security level changed. Would the system immediately change the security level? Would "subjects" currently accessing the object have their access terminated? Wouldn't sudden termination of access imply that the classification had been increased? A mole in the organization could cache copies of a wide array of documents, determine when something was upgraded, and pass copies along to her evil spy master. On the other hand, if one took the tack of delaying the upgrade of the classification until all users had voluntarily given up access, there was the possibility of indefinite delay: user A has access; user B requests and gets access; user A gives up access; user C requests and gets access; user B gives up access; user A requests and gets access. In the midst of these considerations we attended a project status meeting and provided a brief summary of the complexities of changing an object's security classification dynamically. We were directed to exclude dynamic changes of classification from our investigations. The direction to exclude dynamic changes of classification seemed to remove all interest from the problem. We set about to demonstrate that fact and to document our results.

Précis. Our report [6] provided a conceptual framework for talking about computer security, a vocabulary, and an initial result that showed that security (as defined) is inductive. We termed this result the "Basic Security Theorem," in contradistinction to the "Complex and Sophisticated Security Theorem" that we had hoped to produce[1]:

Basic Security Theorem. *Let* $W \subseteq R \times D \times V \times V$ *be any relation such that* $(R_i,\ D_j,\ (b^\star, M^\star, f^\star),\ (b, M, f)) \in W$ *implies*

(i) $f = f^\star$

(ii) every $(S, O) \in b^\star - b$ *satisfies* SC *rel* $f^\star$.

$\Sigma(R, D, W, z)$ *is a secure system for any secure state* z.

[1]This theorem and those that follow are included to show general form and changes to "the model" over time. The casual reader is not expected to grapple with the theorems' details. If the details are of interest, refer to the original reports.

3.3. A Mathematical Model

Our initial report left much to be explored. Finer-grained access control reflecting modes of access was missing. Consideration of access modes led to the unexpected identification of a hard-to-name information flow property, the $\star$-property. The relation W that conceptualized allowable changes of state was not constructive and was therefore insufficient for the analysis and formulation of core system calls that change the security state.

Access Modes. There is a dilemma in dealing with access modes in a general, conceptual way. One advantage of speaking in terms of α and ρ, as was done in the Bell-La Padula model, is that the mind focuses on the logic of the argument, preventing connotations from words (like "read" or "execute") from biasing the analysis. This primes the pump of readers' intuitions. A disadvantage lies in the same lack of connotation. In this volume, connotation-light mathematical symbology was used throughout, with the notable exception of the naming of access modes. The access modes within the model were $\underline{r}$, $\underline{w}$, $\underline{a}$, $\underline{e}$, and $\underline{c}$, the first four obvious cognates for operating-system access modes *read*, *write*, *append*, and *execute* and the last representing "control" of extending or rescinding access to an object. (It was clear that the variety of control in setting access controls justified deferring a more detailed treatment until later.) This priming of the pump proved useful when the report was first published. In later use however, the connotations of access modes in actual computer systems caused some difficulties. Engineers insisted that $\underline{\text{execute}}$ access required "reading" the code before executing it. As a result, Multics's $\underline{\text{execute}}$ mode was later listed as corresponding to $\{\underline{r}, \underline{e}\}$. In later modeling work, the original modes of $\underline{r}$, $\underline{w}$, $\underline{a}$, and $\underline{e}$ were replaced by the four combinations of $\underline{\text{view}}$ and $\underline{\text{alter}}$ access modes, $\underline{\text{view}}$ indicating the ability to see an object's contents, and $\underline{\text{alter}}$ indicating the ability to modify its contents.

Origin of $\star$-property. Careful consideration of generalized access modes led us to the realization that the lending-library analogy had outlived its usefulness. In the first volume, access was binary: either the file was accessed (checked out from the library) or not accessed (on the shelves). With fine-grained access modes, many different patrons could have simultaneous access in different modes. The important insight was that with files, unlike physical books, a user could add marginalia without other users realizing. Information could also be altered or re-

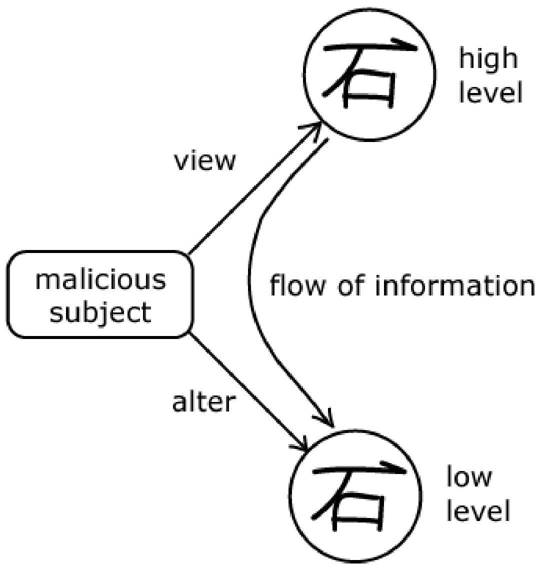

Figure 1. Information Flow

moved with no trace. Since unrestricted users could alter the contents of files undetected, there was the danger of information being copied from one file to another. If a paragraph from the intelligence summary could be copied into the bowling scores, then the system would have lost accurate control of the situation. Our analysis paralleled that in [7], where it was observed that one cannot protect resources with finer granularity than the protection mechanisms themselves support: different diamonds in a safety-deposit box receive exactly the same level of protection. Preventing this unwanted transfer of information admitted of two immediate solutions. One was to monitor every transfer of information (every `load` instruction, every `store`, and every `copy`). The other was to prevent simultaneous access to two objects if flow of information between them could be objectionable. The latter course was chosen, not least because of the overhead of making policy decisions for each instruction. The approach taken to preventing an unwanted transfer of information was to deny the second access request. That is, if the high-level object is successfully opened for read, a subsequent request to open the low-level object for write would be denied. If the first request is to write the low-level object, the later request to read the high-level object would be denied.

A condition to prevent deleterious flows was easily formulated, but a descriptive name was elusive. When I first raised the idea, I scribbled the heading "⋆-property" on the blackboard over a figure much like figure 2. After a burst of energetic discussion, I pointed out that if we didn't change

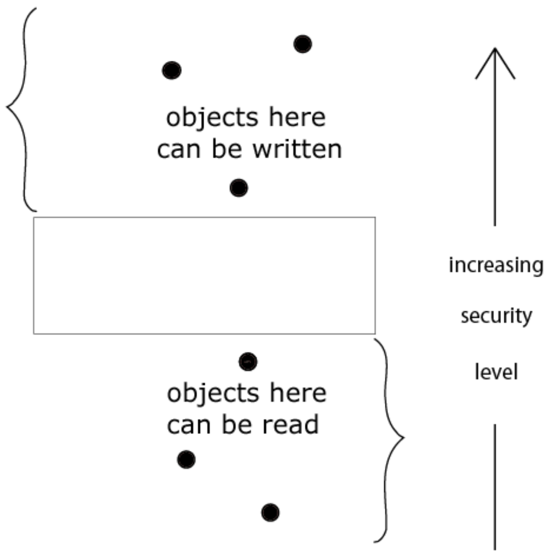

Figure 2. Original ⋆-property

the name right then, we'd be stuck with it forever. Nothing came to us and we continued our discussion. "⋆-property" it remained.

"Rule" Structure. W in the first volume was an undefined relation that conceptualized allowable changes of state. In the first volume, a relation with structure was not required in order to establish that volume-I security is inductive. If the model were to provide direct assistance in the formulation of core system calls, a more constructive form was desirable. The approach adopted was to specify W as a function of a set of "rules," each addressing a particular change of state. The conditions limiting the assembly of rules ω to define a complete relation $W(\omega)$ were minimal: each had to address a well-formed class of request and no two rules could have responsibility for the same request for change of state. Within that structure, a set of rules governing every possible change of state was devised: getting access to objects (in the four access modes, read, append, execute, and write); releasing access; giving access to another subject; rescinding other subjects' accesses; changing security levels; creating objects; and deleting them. Each of ten rules was then proved to preserve simple-security (that is, the subject's classification is greater than the object's classification); discretionary-security (that is, the subject is listed as having access to the object); and ⋆-property. Any combination of non-overlapping rules resulted in a conceptual system that stayed in secure states if it began in one.

Précis. The second volume [8] was a step away from the totally conceptual towards the pragmatic engineering needs of building a secure computer system. Aspects of this direction are found in the inclusion of access modes and the use of rules that addressed all conceivable actions between subjects and objects rather than an undifferentiated relation W. The original definition of security was refined into simple-security, while nuances of different access modes, discretionary-secrity and $\star$-property were added. With these underlying changes, a corresponding Basic Security Theorem was stated and proved:

Theorem 4-1. *Let $\omega = \{\rho_1, \rho_2, \ldots, \rho_{10}\}$, the ρ_i as defined in the section entitled* The Rules, *and z_0 be a secure state which satisfies $\star$-property. Then $\Sigma(R, D, W(\omega), z_0)$ is a secure system which satisfies $\star$-property.*

3.4. Refinement of a Mathematical Model

At the outset of the second year's modeling work, there did not seem to be many pending topics and only one staff member (me) was assigned modeling duties. However, the engineering tasks attempting to use the modeling results in building secure prototypes were having difficulties. Their simplifications and frustrations led to a number of euphemistically termed "refinements" to make the model more usable and more attuned to the realities of computer systems.

Object Hierarchy. The first refinement addressed control of access privileges, based on the object hierarchy within the operating system. For Multics (and by inheritance, for Unix), control of objects (segments in Multics, directories and files in Unix) is limited by access to the object's parent directory. As a result, applying the general notion of control in volumes II and III to Multics required an enormous amount of interpretation. The engineers building secure Unix prototypes and participating in securing Multics wanted and needed some conceptual guidance on the security implications of this variety of control. That is, they preferred me to make the conceptual leap rather than themselves. To deal with this diffuse, implicit control, the model had to be extended to include an object hierarchy H that indicates the structure of objects in any state. The collection of rules for building and analyzing systems was augmented by the addition of four alternative control rules: rules to give and rescind access in a system with implicit hierarchical control; and rules to create and delete objects, with and without "compatibility." The term "compatibility" was used for systems where the security levels are monotonically non-decreasing from the root directory down each pathname. That is, each object has a security level the same as or greater than that of its parent.

The development of compatibility is worth a brief discussion. Since metadata about a file resides in its parent directory in a hierarchical file system, the accuracy and protection of the parent is crucial to the proper mediation of access to the file. In many cases, files and directories are even different instances of a single underlying operation system construct, segment for Multics and file for Unix. I

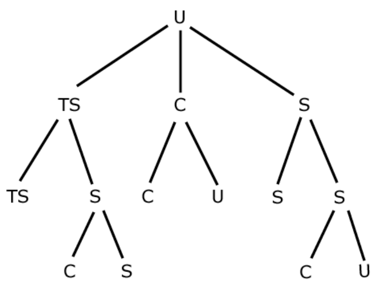

Figure 3. Anti-Compatibility

believed that a parent should be more classified than its children because it contains crucial information about them. As a result, my position was that a pathname should be monotonically non-increasing (staying the same or decreasing) away from the root, except for the first step. Since the root is already an exception (having no parent), an additional exception did not seem of much concern. This way, no subject could introduce errors into the metadata of any object without having been granted a higher level of clearance that the object itself. Researchers at Case Western Reserve University argued the reverse [9]. Since pathname resolu-

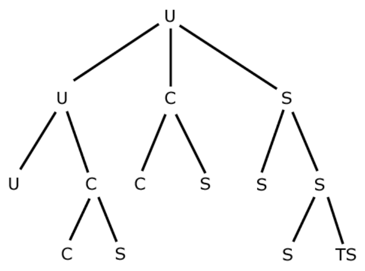

Figure 4. Compatibility

tion requires reading all intermediate nodes, they asserted that the path should be monotonically non-decreasing away from the root (staying the same or increasing). This disagreement was resolved by program direction, not by debate. I incorporated the Case position into the model as directed, terming it "compatibility" and giving Case credit, subtly disavowing responsibility.

There are two items of note. One was the dynamic nature of conceptual developments. The second was my error of conflation: "if this is important, it must be highly classified." With the hindsight of thirty years, I believe the issue was not the sensitivity of the object's metadata contents but its integrity (accuracy and modification limited to authorized agents). While some metadata can be read as half of a covert channel, there is no conceptual reason that the classification of an object and that of its metadata have any particular relation to each other.[2]

Simplification of $\star$-property. The second refinement

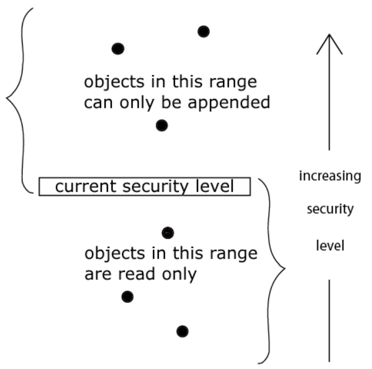
objects in this range can only be appended
current security level
objects in this range are read only
increasing security level

Figure 5. Revised $\star$-property

simplified the $\star$-property based on an engineering short cut. Lee Schiller noted that the original formulation of $\star$-property required comparing the security level of a newly requested object to that of every object to which the subject currently has access. His simplification was to record each subject's "current security level" in a system variable and to compare the new object's security level to the current-security-level, replacing many comparisons with a single one. For viewing accesses, the current security level had to be greater or equal to (or to "dominate") that of the new object. For altering accesses, the current security level had to be less than or equal to (or be dominated by) that of the new object. For access that included both view and alter, the two security levels had to be the same. This engineering short cut not only simplified the statement of the $\star$-property but also narrowed the gap between practical implementations and modeling versions. This refinement required the addition of the current security level to the modeling definitions, then recasting rules ρ_1, ρ_2, and ρ_4 (get-read, get-append, and get-write) in terms of current security level.

Untrustworthy and Trusted Subjects. The third refinement was a revision of the $\star$-property that introduced the concept of the "trusted subject." The stimulus for this revision came from the engineering task focused on building a secure operating system prototype. The intention was to apply the $\star$-property to every process within the prototype. Consider the scheduler. Its function is to swap jobs when conditions require. If the current process were running at TOP SECRET, then the scheduler would have to read all the current state information and write it into swap space. If the next process were UNCLASSIFIED, the scheduler would similarly have to read all swapped out information, move it into place for execution, then kick off the new process. In summary, the scheduler would have to read and write TOP SECRET information and then read and write UNCLASSIFIED information. "What security level," the engineers asked me, "shall we assign to the scheduler?" On reflection, the premise of the original $\star$-property was that *no* subject could be trusted not to copy part of the intelligence summary into the bowling scores. That assumption clearly does not apply to the core of a secure operating system where one can and would insist that the entirety of such core processes be carefully and thoroughly reviewed. The model was altered to identify a subset S' of the full set S of all subjects. These subjects were the subjects that "are untrustworthy and may mix information as described" [10]. The concept of "$\star$-property" was replaced by "$\star$-property relative to S'" and the subjects outside S (mathematically, the set $S - S'$) were not subjected to the restrictions under the original $\star$-property. Subjects as a whole were divided into two parts, "untrusted subjects" and "trusted subjects." Those "trusted subjects" were precisely those who "will never mix information of different security levels" [11].

Précis. The three refinements all derived from engineering experience in trying to apply the security model to actual system implementations. Raising these issues to the project modeler benefited both the modeling and the engineers: the model was refined to represent actual computer systems more faithfully and the modeling results provided justification and more explicit guidance to system prototyping and implementation.

Revised BST [12]. *If one desires a secure computer system which exhibits only compatible states satisfying the $\star$-property, one can use the set of rules*

$$\omega'_{iii} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_{12}, \rho_{13}, \rho_{15}, \rho_{16}, \rho_{17}\}$$

together with a compatible, secure initial state z_0 which satisfies the $\star$-property [relative to S'].

Note that omitted rules ρ_6 through ρ_{11} are the non-hierarchy versions of giving and rescinding access, creating and deleting objects, and changing the security level function f. Those non-hierarchical rules were neither voided nor superseded and are available wherever they are applicable.

3.5. Unified Exposition & Multics Interpretation

Background. The first three volumes of the Bell-La Padula model were produced during two fiscal years. In the following year, there were other priorities. The year after that, an additional modeling effort was mounted. The

[2]The cross-product of a data sensitivity lattice with an integrity lattice happens to be a sublattice of a boolean lattice. Metadata integrity *could* be handled with modern lattice policies, but that subtlety of thought came later.

reasons were several. One was that the changes in the model over three volumes made it difficult for readers and engineers to keep the definitions and state-changes rules straight. The other was the large conceptual distance between the model as it stood and the Multics operating system that was in the process of being hardened and secured.

The "unified exposition" part of the tasking was easy for us to understand. The "Multics interpretation" part required an understanding of the Multics operating system. A request for information and assistance from the staff assigned to securing Multics resulted in us receiving a copy of Elliott Organick's *The Multics System: An Examination of Its Structure* [13] with an injunction to learn the system from the book. Learning Multics from Organick was highly educational. The exercise revealed operating system theory and practice in a concentrated, focused way. Independent study provided cross-training for systems engineering and analysis, in addition to security modeling and "security engineering."

Précis. The fourth volume gathered the changing definitions and results into a single place [14]. It provided new "rules" tailored to the Multics system. It provided proofs for the new Multics-specific rules,

It also included a time-sensitive statement that sounded like eternal truth: "All significant material to date on the mathematical model has been collected in one place in the Appendix of this report" [15]. This statement was true at the time with the known audience (computer security specialists in 1974). It was not true for later readers that had not worked in computer security during the 1970's.

This volume completed the transition of Bell-La Padula modeling from the purely conceptual to the system-specific. The general conceptual tools, tempered by engineering use and frustration, had been brought to bear on a specific system problem, the securing of Multics.

Theorem A10 [16]. *Let ρ be a rule and $\rho(R_k, v) = (D_m, v^\star)$, where $v = (b, M, f, H)$ and $v^\star = (b^\star, M^\star, f^\star, H^\star)$.*

1. *if $b^\star \subseteq b$ and $f^\star = f$, then ρ is ss-property-preserving.*
2. *if $b^\star \subseteq b$ and $f^\star = f$, then ρ is $\star$-property-preserving.*
3. *if $b^\star \subseteq b$ and $M^\star_{ij} \supseteq M_{ij}$,, then ρ is ds-property-preserving.*
4. *if $b^\star \subseteq b$, $f^\star = f$, and $M^\star_{ij} \supseteq M_{ij}$, then ρ is secure-state-preserving.*

4. Observations on Modeling Results

4.1. Clear Definition

The Bell-La Padula Model demonstrated the importance of a clear definition of the "security" being addressed. Without a clear definition, one faces unending complaints about "essential" aspects of security being omitted. "How can you call a system secure if it doesn't prohibit (or require) $\aleph$?"

4.2. Grounded in Engineering Reality

The modeling also implicitly demonstrated the importance of being grounded in immediate engineering problems. A purely academic exercise in computer security would not have unearthed and addressed the practical issues in building or retrofitting an operating system for sound security.

4.3. Toolbox Produced

Consider the Bell-La Padula modeling reports as a toolbox for the design, analysis and implementation of secure computer systems. After the first four reports had been published, the toolbox was full and complete. It included a descriptive capability: a precise definition of "security" that incorporated information compromise, fine-grained control of individuals to information objects, and preventive measures to prohibit information from flowing from high classification levels to lower classification levels. There were general results guaranteeing that security would be present provided the system began in a secure state and never introduced a security violation during a change of state. There were "rules" for handling a wide array of routine state changes (getting and releasing access; giving and rescinding access privileges; creating and deleting objects) as well as a method of combining arbitrary sets of such rules to describe a particular instance of a secure system. The complete set of rules included both generic rules as well as rules closely tailored to the Multics operating system, the first instance of a specific modeling solution. I believed then as now that no radical security modeling would be required later, with two possible exceptions. One would be the need to analyze a form of "security" beyond those specified: compromise, fine-grained discretionary access control, and $\star$-property-like information flow. Another would be the need to analyze technology artifacts that did not patently correspond to the descriptive machinery already available. Between 1975 and the present, both exceptions were experienced, but only infrequently.

4.4. Avenues of Futility

At the time, no one felt it necessary to state that pinning one's security hopes on having smarter geeks than the opposition was a failed concept. Neither did most experts believe that market forces would produce secure systems: they hadn't produced systems that stopped the 1970's tiger teams. Moreover, prominent industry representatives were arguing that the only security required was personnel security, while technical security was bunk.

4.5. Strong, Built-in Security

Experts agreed that deep, significant security required security being built in and being built for high confidence. Today, we would say such systems require "high assurance."[3] Commercial systems, even commercial systems with some security labels, were not sufficient to keep information of adjacent collateral classification levels separate: UNCLASSIFIED from CONFIDENTIAL, for example. Too much was known about the weakness of such systems through tiger team results to believe that feature hardening or security appliques (security features added on top, or inserted in the middle) could really protect one's resources from a dedicated, resourceful, and intelligent foe. Imagine the devastation if your foe had the skills and resources of the wily hacker [17], much less those of highly skilled security specialists [18].

4.6. Self-Protection and Subversion

Both the engineering and modeling tasks had addressed flanking attacks in the form of subversion. The last modeling report discussed "sabotage" ("undesired alteration or destruction of information by purposeful action of an agent") and integrity, noting that a system's ability to protect itself is essential and that a lack of strong self-protection makes conceptual analysis worthless [19]. The engineering tasks went further in demonstrating how penetration and the insertion of backdoors and other maleficent logic could completely undermine the intended security characteristics of a system. Just as important, the demonstrated harm that subversion could do necessitated anti-subversion requirements, especially formal verification and trusted distribution.

It is worth including a description of one of the most ominous subversion attacks. It is called the "two-card loader," through analogy to mainframe loaders. A general-purpose loader could be packed into two punched cards. The hard-wired loader would read those two cards into memory then pass execution to the two-card loader. It would then load any program for execution. A two-card loader subversion would add a general-purpose program loader to a system. The loader begins execution after receipt of an obscure signal, similar to the signals used to unlock "Easter eggs" in commercial software. This kind of addition to an allegedly secure system would allow an attacker to craft a pointed attack much later and launch that attack on command. The possibility of commercial software having covert two-card-loader logic makes tight controls over a secure system's configuration vitally important.

[3]By high assurance I mean confidence in proper design and implementation represented by systems at B3 and A1. Marketing claims that B1-like systems are "high assurance" are purposefully misleading or a reflection of ignorance.

5. General Developments, 1975–2005

Engineering and conceptual efforts continued from 1975 through 1983, but I was not involved. When I returned to security work in 1983, there were three parallel and interrelated strands of security development: consolidating the successes of computer security, a technology shift to smaller computers and more networking, and initial glimmerings of new security policies.

5.1. Consolidation

At the beginning of the 1980's, there was a concentrated effort to solidify and build on the successes of the 1970's work in computer security and to learn from the failures. The various engineering tasks, conceptual tasks, and research prototypes had identified security principles, methods and techniques and proved those techniques in exemplar systems such as the Kernelized Secure Operating System (KSOS) [20], the Provably Secure Operating System (PSOS) [21], the Kernelized Virtual Machine (KVM) [22], and Multics [23]. Steve Walker at the Office of the Secretary of Defense initiated the Computer Security Initiative with three goals: to formulate a metric for measuring the security of systems, to establish a center whose mission was to measure commercial systems against that metric, and to initiate a technical conference dedicated to computer security. The metric was the *Trusted Computer System Evaluation Criteria* [24], published in 1985. The center was the Department of Defense Computer Security Evaluation Center (later the National Computer Security Center). The conference was the Department of Defense/National Bureau of Standards Computer Security Conference (later the National Computer Security Conference).

The construction of the *TCSEC* was an exercise in giving partial credit. The best security known became the highest security class, A1. The lower classes were developed as logical subsets of security requirements down to "no security at all," or D. Solid commercial systems with good discretionary access control were expected to meet the requirements of the C2 class, "Controlled Access Protection." Adding security labels to some of the objects in the system resulted in the B1 class, "Labeled Security Protection." C2 and B1 systems were viewed as no stronger than that required to keep order between cooperative colleagues. At B2, "Structured Protection," the first approximation of a really secure system could be seen. At B3, "Security Domains," the system has its core (its "trusted computing base") minimized and it is the lowest class that is "highly resistant to penetration." A1, "Verified Protection," added life-cycle protection against changes to the hardware and software (anti-subversion), trusted distribution of the system and updates, and the use of "formal verification."

I had doubts whether "formal verification" was necessary, based on my reading of the 1970's proofs-of-correctness literature and more particularly on my co-authorship of a MITRE report [25] on the topic of software validation for certification. In the mid-1970's, there seemed to be a disconnect between most of the literature on proof-of-correctness and the situation of validating a secure system. Proofs-of-correctness had focused on simple comparisons of an algorithm to a program embodying that algorithm: a principal class of papers addressed bridge-bidding programs, for example. The challenge for verifying a secure system was avoiding insecurity and maintaining a state of security. As the *TCSEC* was being finalized, I was surprised to learn that past intentions to verify automatic verification systems had been quietly abandoned. While at the Computer Security Center, I became more familiar with both the theory and practice of automatic verification systems and my reservations were not reversed. Work-arounds to assure that the verification tools could process the large specification files included manual stubs, drivers, and artful commenting-out of sections of the specification to assure that the tools could run to completion. In many ways, human brain-power provided a back-stop for the limitations of the tools. Nevertheless, I came to feel that formal specifications held some value. My feeling was captured best by Marv Schaefer, observing that the greatest benefit of formal verification is gained in writing down the specification. Normally loquacious verificationists were struck dumb.

Part of the original motivation for the *TCSEC* and the Trusted Product Evaluation Program was to streamline and regularize the formal process of approving Department of Defense systems for operation – certification and accreditation (C&A). While there were policies and procedures for assessing secure computer systems, there were problems with having each acquisition's engineering staff assess the underlying operating system or systems for security compliance. Not only was there no reuse of the analytical results, but also there was no guarantee of consistency across acquisitions. If all certification engineers facing, say, a Trusted Solaris component could refer to a published summary of its security properties in the context of its *TCSEC* rating of B1, then redundant analyses would be avoided and consistency across different acquisitions would be increased. Moreover, since computer security was a specialty field, the acquisition engineering staff were not always trained and current in the relevant security topics. The intention was to centralize and reuse the security evaluation results to obviate redundant analyses, to assure consistent results, and thus to provide an acquisition task with a solid, published product evaluation report.

The Trusted Product Evaluation Program was begun immediately after the Center was formed in 1981. The first candidate systems included the commercial "archetype" systems, the systems whose successes were the basis for the requirements in the *TCSEC*. The initial set of candidates included SCOMP at A1 (verified security) and Multics at B2 (structured protection). At the lower levels were Unisys's MCP/AS, IBM's RACF, Hewlett Packard's MPE V/E, CA's ACF2/VM, CA's TOP SECRET/NVS, and CDC's Network Operating System at C2 (best commercial practice); and Univac's OS 1100 at B1 (labeled security protection). Over time, many more C2-candidate systems entered the program and completed evaluation. The most significant was Windows NT 3.5 in 1995. The most significant almost-B1 system in today's context is Trusted Solaris V2.5.1 which received an EAL4/B1 rating in 1998. Both of these systems, of course, are inadequate to counter any focused attack; their proper use is among cooperating friends. At the higher levels, the successful systems were VSLAN, Multics, and Trusted Xenix at B2; XTS-300/400 at B3; and SCOMP, GEMSOS, Boeing's MLS LAN at A1. Honorable mention also goes to DEC's Security-Enhanced VMS (SEVMS), an A1 candidate nearing final evaluation when the product was withdrawn from the market [26]. The success of these systems was far less than might have been hoped, but it was a substantial achievement.

5.2. Technological Changes

At the same time that the computer security successes of the 1970's were being consolidated, Moore's Law and general networking were changing the face of computing. Computers began to shrink and traditionally clear categories of computers (computers, minicomputers, microcomputers) began to blur and overlap. The technological changes began to accelerate away from the assumptions and verities of computer security just as they were being codified.

5.3. Policies

From the beginnings of the Computer Security Initiative, there were objections that the Department of Defense and the intelligence community security policies based on classifications did not apply in the civilian and commercial world. Work in security policies remained largely the bailiwick of the classified world and pure academics until 1989. Up until that time, new conceptual policies were primarily minor alterations to existing policies, variations on discretionary security, for example. When David Clark and David Wilson published "A Comparison of Commercial and Military Security Policy" [27] in 1987, it was widely hailed and stimulated further consideration of novel security policies. Its primary contribution to security policies was to introduce access triples (user, program, file), where previous work had used duples (user, file) or (subject, object). The following year witnessed an explosion of Clark-Wilson response papers (see for example Lee [28] and Shockley [29]). In 1989, Brewer and Nash published "The Chinese Wall Security Policy" [30], abstracting British financial regulations. Its contribution was a user's free-will choice limiting future actions in a non-discretionary way. The multiplicity of these policies, however, was more apparent than real, as will be described in paragraph 6.3.

5.4. Cross-Currents

The technology movement away from single logical platforms well understood in the computer security world posed a dilemma for the Computer Security Center with regard to the *TCSEC* and the product evaluation program. A series of initiatives focused on the special security issues, first in networks and later in database management systems. The results were the *Trusted Network Interpretation of the Trusted Computer System Evaluation Critieria* (*TNI*) [31] and the *Trusted Database Interpretation of the Trusted Computer System Evaluation Critieria* (*TDI*) [32]. Absent clear promulgated guidance on networking issues, product evaluation faced a hard choice: exclude networking (unrealistic) or include networking without clear guidance (risky). Neither choice was good. Caution led to the exclusion of networking features in the early days.

The growth of networking brought the "composition problem" to the fore: what are the security properties of a collection of interconnected components, each with known security properties? Efforts were made to derive methods and principles to deal with the unconstrained composition problem, with limited success. In the sense of product evaluation and eventual system certification and accreditation, one needed to leverage the conclusions about components to reach conclusions about the whole without having to reanalyze from first principles. Unfortunately, closed-form, engineering-free solutions were not discovered and may not be possible.

6. Conceptual Design, 1975–2005

6.1. Consolidation

Part of consolidation was the inclusion of a requirement for security policy modeling for higher security products (B2 and above) in the *TCSEC*. In addition, the *TCSEC* listed the Bell-La Padula model as a good representative of a model with the characteristics required. As a result, I undertook modeling interpretation several times in early product evaluations (Multics at B2 and SCOMP at A1). Later, I produced a modeling interpretation for Trusted Xenix as part of a product evaluation [33].

The formulation of interpretations of the *TCSEC* for networks (*TNI*) and later for databases (*TDI*) necessarily included the topic of mathematical modeling. Conceptualizing networks, in particular, pivoted on the scope of the network components being considered. A homogeneous network (one viewable as a single virtual networked system) admits of a conceptual design. Not so with an *ad hoc* heterogeneous network (one without a single worldview). The problem was precisely the unconstrained composition problem and the need to consider engineering interfaces in a general way, as described below.

6.2. Technology Changes

In the early 1980's the A1 security modeling requirement was levied on Blacker Phase 1 (BP1), an inherently networked system [34]. The conceptual difficulties were several. BP1 was composed of three different components, often geographically remote. The three componens had different design specifications and only working in concert did they achieve the requirements of the overall system. In addition, the networking nature of the BP1 components raised puzzling questions about all the A1 requirements. Specifically, each of the components was a trusted computer system in itself, while participating with its siblings to produce a "trusted network system." The dual faces of the A1 requirements were documented for the project and were later provided as input to the invitational workshop in New Orleans, convened to begin work on guidance for trusted networks. Application of traditional modeling results to the individual components was straightforward, but addressing the larger BP1 network was not at all obvious. Eventually, it became clear that the system itself was concerned with an entirely different set of subjects and objects than were the operating systems of the components. The system administered a policy concerning the labeled "connections" that subscriber hosts could establish. Thus, at the network level, the active entities were "hosts" and the resources were "connections." This realization led to a modeling interpretation of the Bell-La Padula model that conceptualized hosts and connections (called *liaison* in the model) [35]. Note that this interpretation did not build on the Multics-specific rules, but on the general rules in volume 3, augmented by rules constructed to address the specific topic of access-controlled *liaison*. This was an example of modeling being required because available resources did not patently apply.

In the *TNI*, composition is addressed at length, but composition there is tightly constrained. In broad outline, it is similar to the approach taken in BP1. To evaluate an network composed of independently evaluated components, one must have network policies for identification and authentication, audit, discretionary access policy and mandatory access policy; a network architecture; protocols for communicating user identification, authentication, audit, labeling; and defined interfaces for reporting events to audit components. This constitutes a network-view together with engineering limits and requirements that made the *TNI* usable and successful [36].

In conceptualizing heterogenous networks, the general composition problem is unavoidable and one faces enormous difficulties in the absence of a network-view like that of the *TNI*. With the increasing use of networking and systems construction from smaller quasi-independent parts, understanding the security properties of a collection of connected secure systems is extremely important. I found none of the conceptual and modeling results fully satisfying (primarily because they ignored the engineering issues included in the *TNI*) and therefore was unconvinced. From a purely modeling point of view, there is an explicit presumption of a global awareness of all aspects of the conceptual system. Few security models (if any) even include a way of talking about an interface to another system. It is therefore hard to see how two different models, even two instances of the same model, could relate to each other and produce a single security model. In a broader context, the *TCSEC* requirements outside the modeling requirements are even more difficult to satisfy. In fact, I conjectured that there is no closed-form solution to the composition of several secure systems that does not include a network-view and engineering interfaces and protocols.

Consider two identical A1 systems. Those systems *might* be combined so that they mesh perfectly, both operationally and theoretically, in an A1 composite. They could also be also be combined so badly that the composite has no security at all (and would be rated D). One could probably artfully adjust the requirements failed in such as way as to make the composite any *TCSEC* level between D and A1. It is the engineering details of the interface that determine whether systems complement and enhance each other or undermine each other's original security qualities.

The unavoidability of heterogeneous networks and the lack of a purely conceptual way to analyze their security properties was a hard dilemma. One could not stand in the way of the network tide coming in. Nor could one insist on homogeneous networks throughout. In the absence of multilevel networking equipment that shielded unilevel hosts and networks, there were very few practicable courses of action.

What was generally chosen was to deploy isolated networks. The rationale was that if all users and all hosts on a particular network were cleared for the information on that network and were trusted with the network's operational mission, then one could manage without multilevel secure networks. Unfortunately, the isolation between networks was incomplete. There were and are operational needs to move information between networks of different classifications and sensitivity levels. Those connections were accomplished with guards and other dedicated equipment whose function was to enforce the rules about communication between the networks. Over time, the multilevel nature of these components came to be ignored, abetted by the usual way of viewing networks with their details hidden. A prime example is the network cloud. It is important to remember that the cloud is made up of processing platforms and communication lines. Hence, a thin line between two separate network clouds is, on closer inspection, a combination of communications lines and computing platforms. That being the case, the following "Computer Security Intermediate Value Theorem" (CS-IVT) becomes important:

CS-IVT. *If computer A at security level α is connected to computer B at security level β through a network cloud and $\alpha \neq \beta$, then some processing platform in the cloud is multilevel.*

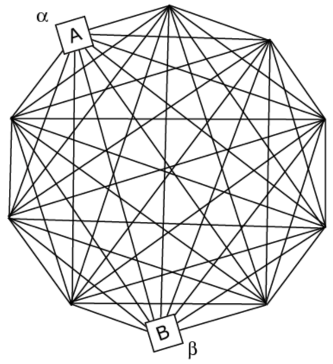

Figure 6. Intermediate Value Theorem

***Proof Sketch*:** If it were not so, then each node on any path through the cloud to the distant end would be single-level at the same security level as its immediate predecessor. As a result, the first and last nodes would be operating at the same level, a contradiction.

The irreducible complexity of computer security cannot be eliminated by assigning a multilevel function to a component that is represented by a line in a network cloud or a wiring diagram. Claims that highly secure systems can be constructed from low-security or untrusted system components and lines on a blackboard have been roundly criticized every time they have been propounded, and with good reason: they have to contend with the CS-IVT. Connecting two isolated networks with a slim line creates a single network with a single cut-point. The CS-IVT then says that some platform is multilevel. If neither cloud is multilevel, then the slim line is. This is an illustration of the conservation of complexity: you can relocate the complexity of multilevel security, but you cannot eliminate it. Unfortunately, some have forgotten this lesson or are trying to ignore it.

6.3. Policies

In 1987 before the explosion of new policies stimulated by Clark-Wilson, I began to ponder a remark made by Marv Shaefer. I had just made a presentation on lattices for commercial isolation. As I was returning to my seat, Marv observed that we could not then support a policy expressed as an *OR*. "Why not?" I wondered. I began a personal investigation which quickly turned into self study in lattice theory using Birkhoff's classic text [37]. I came to realize that naive use of lattice theory in computer security's infancy had led to an identification of the fundamental elements in a policy (which I termed "policy alphabet") with the lattice's "atoms" (the elements directly above the $\emptyset$ element. This realization justified the working solution I had earlier

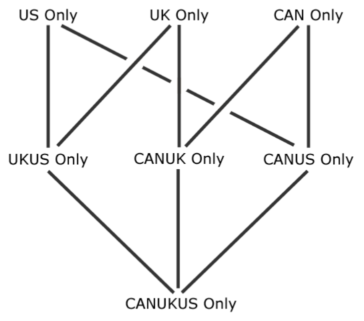

Figure 7. CANUKUS

crafted for multilateral sharing, such as CANUKUS information (information shared in various combination between the governments of Canada, the United Kingdom, and the United States). Further, a fundamental boolean lattice result had implications for security policies and for implementations. That result is as follows [38]:

Thm 11. *The meet-irreducibles of the boolean lattice on the alphabet A are*

$$\bigwedge_{a \in A} s(a),$$

where $s(a)$ is either a or $\neg a$ for a in the alphabet A.

Since there are many different-seeming formulations of boolean lattices, one can choose the formulation that is most familiar. All representations lead to the same boolean lattice (up to isomorphism). Most telling, one of the formulations combines uninterpreted symbols with *AND*, *OR*, and *NOT*. Surely *most* security policies can be so defined. For example, the usual boolean lattice *can* deal with *OR*s of the policy elements

$$\{\textit{FINANCIAL}, \textit{ENGINEERING}, \text{STRATEGIC}\}$$

by using the meet-irreducibles (atoms)

$$\{F \wedge E \wedge S, F \wedge E \wedge \neg S,\\ F \wedge \neg E \wedge S, F \wedge \neg E \wedge \neg S,\\ \neg F \wedge E \wedge S, \neg F \wedge E \wedge \neg S,\\ \neg F \wedge \neg E \wedge S, \neg F \wedge \neg E \wedge \neg S\}.$$

Any security policy that can be defined with *AND*/*OR*/*NOT* is isomorphic to any other one of the same width (number of atoms). If they have different widths, the smaller one is isomorphic to a sublattice of the larger one. An implementation built to support any boolean lattice could support any other boolean policy, given enough width and the provision of a personality module. This result was published in 1990 and demonstrated that statements that the military security lattices are not applicable to other situations were in error [39].

After presenting my paper, I asked Jim Anderson whether he'd heard my talk. He had not, but observed that if I could use my results to support ORCON, he knew people who could use it immediately. Even 1-step ORCON would be a big help. (ORCON stands for "Originator Control." It refers to access granted only for reading. Any use or citation must be explicitly requested and granted by the originating organization before it can be used.) My first thoughts were that ORCON could not be done. If you provide an electronic copy of a report, no more control can be exerted. My second thoughts considered the publication process for classified documents more closely. Classified publications are first drafted and then reviewed before publication and dissemination. From that perspective, a solution was constructed that marked each object with the originating organization and distinguished between DRAFT and PUBLISHED materials. Changes of state were governed by approval officials for each organization. An analyst preparing a DRAFT could use any information to which she had access in her own organization and any PUBLISHED information that other organizations had made available to her organization. Publication (producing a PUBLISHED clone of a DRAFT document) required approval by the local organization and any organization whose information was included. With joint approval, roll-back, and limited changes of state, one-step ORCON had a conceptual design. With only a few extensions, n-step ORCON did too.[4]

With multilateral sharing, one-step ORCON, and n-step ORCON solved, I attempted to collect all the claims of

[4]Note that this conceptual design requires a homogeneous environment. All the computers and components must be cooperating to enforce the policy. The design does not solve ORCON in a heterogeneous environment.

"different" security policies in the literature, noting in particular whether the authors claimed their policy was outside the DoD's classification policy. I structured my analysis in terms of an abstract computer that included a lattice policy built with *AND*/*OR*/*NOT* and access control lists. For this abstract machine, I produced constructive solutions for every security policy in the literature. In fact, two solutions were provided, a strong solution building on non-discretionary security policy and a weak solution building on discretionary security policy. The weak solution was included for those situations where strong, labeled security policy is not available. The results were reported in "Putting Policy Commonalities to Work" [40]. This was an example of conceptual work addressing policies outside confidentiality, discretionary security, and $\star$-property. This paper showed tangentially that the profusion of new security policies did not pose the problem it might have. Had all the new policies been fundamentally different, vendors would have had to choose which policies to support and which to neglect. With the ability of any boolean policy implementation to support any other boolean policy, the plethora of policies merely provided comfortable terminology for all communities.

7. Observations for the 21st Century

7.1. Overview

Our computer security legacy at the beginning of the 21st Century is not extensive, but neither is it inconsiderable. It is the result of dedicated work by master technologists and government champions. Lasting principles were hammered out conceptually, then refined and honed on engineering workbenches. The lessons were codified and an immense effort to prime the pump with secure, trusted systems was undertaken, compensating for the ineffectiveness of market forces. In the opening years of this century, our legacy includes two very high security products and the community knowledge to deploy them wisely. One could argue that we are in a much better situation now than in the early 1980's. The general feeling, however, is that computer and network security is in decline. Why is that? I believe it is the confluence of three currents of change: technology, the processes for fielding secure systems, and Government's role in nurturing and expanding our secure system resources.

7.2. Technology

Technology's increasing pace of development and shortening product cycles have made most computer users full-time beta testers. Something can be imagined, demonstrated, and become mission-critical before a stable version is available. In Internet time, "immediate gratification takes too long" [41] is an understatement. Our networks are assemblies of beta products, hooked together heterogeneously with more attention to features than to fundamental security. Added to that is the normal marketing of commercial products. One's own feature set, or dominance of the market, or previous deployment — *any* impression of a market advantage can and will be used to make a sale. There is nothing wrong with that in any particular case. It is an unbroken sequence of such decisions and interactions with other currents that causes problems.

7.3. Fielding Systems

At its inception, the Computer Security Initiative's intent to improve security certification suffered from a dearth of secure products. Doubtful or hostile program managers (government and contractor) could cite competitiveness regulations and system requirements not met (sometimes carefully crafted) by available secure products to seek waivers. This reluctance to embrace change was neither unexpected nor blameworthy. That initial reluctance, however, combined with an increasing pace of technological change and slow increases in the availability of secure products, resulted in stultification and no meaningful progress towards the routine use of secure products.

The formal process of evaluation suffered under shrinking time-to-market schedules because analysis and understanding take time and thought. The structural changes to product evaluation initiated in the early 1990's had the effect of terminating community evaluation of highly secure systems. The stock of secure-system resources was therefore frozen and attrition further thinned the ranks. Without sales of their secure products, the vendors rightly viewed them as under-performers.

On the certification and accreditation front, quick was the enemy of strong security. System acquisitions and deployments increasingly began to face hard choices between sound security and speedy approval. Interested parties in the form of vendors lobbied in favor of speedy approval, and incidentally, sales of their products. I know from direct experience that successful system acquisition requires balancing many different demands and imperatives. One must make compromises to deliver at all. The best one can hope is to minimize compromises on essential issues. Such compromises, however, are local optimizations. A sequence of local optimizations need not produce global optimization. Were I currently a program manager who needed a highly secure component, I would far rather leverage someone else's selfless act than act selflessly myself. The running joke is that everyone wants to be second through the gate.

With few secure resources to build upon, with little C&A to leverage, and with active lobbying by the vendors of low-security systems, it is easy to see why individual program managers and certifiers do not lead ambitious initiatives to better the security in deployed systems. Surely someone somewhere should be doing so.

7.4. Nurturing Secure System Resources

It is hard to argue persuasively that anyone is currently nurturing our commercial security resources. From the 1960's through the 1980's, the military services and ARPA funded research and development in computer security. In the early 1980's, the Department of Defense assigned formal evaluation of commercial products to the National Security Agency, paired with the mission of encouraging secure products and mandating their use. None of these missions flourish today.

At the beginning of the 21^{st} Century, product evaluation for high-security systems is moribund. DARPA correctly views computer and network security as past the research stage. Individual program offices rightly have primary focus on their program, not on the greater good. No one has the mission of "selfless acts of security."

7.5. What Can We Do?

The first thing we must do is tell ourselves the truth about security. Mostly isolated networks have not solved our security problems. The Computer Security Intermediate-Value Theorem still holds: slim lines between single-level networks *are* multilevel devices, secure or not. Because the slim lines are multilevel, it is unconscionable to use overly weak components. Such connections require high security, meaning A1, although in some cases one could accept a B3 solution. Systems of C2 or B1 heritage (Windows NT/XP/Vista or Trusted Solaris, respectively) are totally inadequate. Their forebears could not thwart the Tiger Teams of the 1970's; they cannot stop a skilled and dedicated attacker today. A Cuckoo's Egg attack on an operational system would be bad enough. A two-card loader subversion requiring only a six-line "toehold" would be catastrophic [42] [43].

As a corollary to telling ourselves the truth, we must look askance at those who insist that semi-isolated networks *have* solved our problems. Do they not know? or do they ignore the risks they expose us to?

Not only can we use our scarce high-security resources for the critical network connections, the current prevalence of networking actually works to our advantage in this case. There was once the vision of a world of computing where every computing platform was multilevel and secure. Contrary to the assertions of critics, no one I knew ever believed multilevel security had to be pervasive to succeed at all. Every multilevel project I ever worked on explicitly acknowledged and dealt with a mixed environment: multilevel secure platforms and single-level platforms, interacting. (At a conceptual level, a single-level system is a degenerate multilevel system: its highest and lowest levels are the same.) Now, by contrast, our networks are heterogeneous and the connections between networks of different security levels are relatively few. There is the opportunity, therefore, to place high-security products at the weakest points, the connections between the not-quite-isolated networks.

Consider the kinds of connections one might want between networks. One might want a common file server. One might want a web server. The connection might be a multilevel database of a simple kind. Current web practice builds pages from information assembled from many sources. This makes utilization of several single-level databases or a simple SeaViews-style secure database eminently reasonable [44]. One might want thin clients attached to a multilevel server, itself attached to resources of different security levels. All of these could easily be built on current high-security platforms. None of them require functions available exclusively on weak platforms.

Suppose one built these applications on a *Yahoo!*-like base without a high level of security. The problems for a configurable *Yahoo!* portal are primarily functional: database access, web display, and the marshaling of searches via multiple search engines. Only protection against defacement and masquerade have a security flavor. Extending a *Yahoo!* situation to the connection of networks of differing security levels is a massive and fundamental change. The functional issues are the same, but the security requirements cannot be satisfied by tighter configuration or by constantly vigilant geeks. Without a solid, secure foundation, such functionality is built on such a shaky foundation that it cannot be trusted with multiple security classifications.

The next thing we must do is use the resources we have. The hard-won triumphs of the Computer Security Initiative have been neglected so long that it is a miracle anything has survived. We must constantly be on the lookout for opportunities to use them to good advantage. Knowing the conflicting pressures on active acquisitions, we must generate opportunities to perform selfless acts of security in the form of crafting and sharing reference implementations of widely needed components. A good list to start with is the following:

- high-security file servers,
- high-security static web servers,
- simple high-security multilevel databases,
- high-security dynamic web sites, and
- high-security thin-client architectures.

None of these breaks new ground in either security or in general functionality, but each of them would be an extremely valuable building block. (In fact, the first three build to the fourth.) We as a community need a champion to lead these and similar reference implementations for the good of the entire community. We cannot wait for market forces to reverse course and surprise us with truly secure building blocks.

8. Conclusion

Our computer security legacy needs to be used and it needs to be nurtured. If we do not, we risk losing what little we have and having our enduring security principles fade from community consciousness. In addition, we must remember that

- Tight configuration and smart geeks are not a substitute for good security.
- Multilevel secure components are unavoidable.
- Multilevel links between security levels must be as strong as we can make them.
- Government must perform its crucial role in nurturing our commercial security resources.

We have known for nearly a third of a century how to build and deploy strong and secure systems. Our shared knowledge applies in today's networked world with its emphasis on fast, secure information sharing. In the 21st Century, we should utilize our existing resources in the slim lines between networks. It is time we got started.

Acknowledgment

Thanks to all of my readers, with special thanks to Roger Schell who reminded me of the *TNI*'s treatment of composition.

References

[1] J. P. Anderson, "Computer Security Technology Planning Study," ESD–TR–73–51, Vol. I, AD–758 206, ESD/AFSC, Hanscom AFB, MA, October 1972, p. 4.

[2] Paul A. Karger and Roger R. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD–TR–74–193, Vol. II, ESD/AFSC, Hanscom AFB, MA, June 1974.

[3] Ken Thompson, "Reflections on Trusting Trust," *Comm ACM*, August 1984, **27**(8), 761–763.

[4] Ken Thompson, "On Trusting Trust," **Unix Review**, November 1989, **7**(11), 70–74.

[5] J. P. Anderson, "Computer Security Technology Planning Study," ESD–TR–73–51, Vol. II, ESD/AFSC, Hanscom AFB, MA, October 1972, p. 16.

[6] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations," MTR–2547, Vol. I, The MITRE Corporation, Bedford, MA, 1 March 1973. (ESD–TR–73–278–I)

[7] Jerome H. Saltzer, Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, **63**(9) (September 1975), pp. 1278–1308.

[8] Leonard J. La Padula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model," MTR–2547, Vol. II, The MITRE Corporation, Bedford, MA, 31 May 1973. (ESD–TR–73–278–II)

[9] K. G. Walter *et al.,* "Primitive Models for Computer Security," ESD–TR–74–117, Electronic Systems Division, Hanscom AFB, MA, January, 1974.

[10] D. Elliott Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR–2547, Vol. III, The MITRE Corporation, Bedford, MA, December 1973. (ESD–TR–73–278–III), p. 25.

[11] *Ibid.*

[12] *Ibid.,* p. 34.

[13] Elliott I. Organick. *The Multics System: An Examination of its Structure*. The MIT Press: Cambridge, MA, 1972.

[14] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR–2997, The MITRE Corporation, Bedford, MA, July 1975. (ESD–TR–75–306)

[15] *Ibid.,* p. 5.

[16] *Ibid.,* p. 98.

[17] Clifford Stoll. **The Cuckoo's Egg**. Doubleday: New York, NY, 1989.

[18] Paul Karger and Roger R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation," *Proceedings*, 18th ACSAC, Las Vegas, NV, December 2002, 119–126.

[19] Bell and La Padula, *op. cit.,* 1975, p. 70–71.

[20] E. J. McCauley and P. J. Drongowski, "KSOS—The design of a secure operating system," *Proceedings*, AFIPS 1979 NCC, v48, 345–353.

[21] 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," Technical Report CSL–116, SRI International, 1980.

[22] Marvin Schaefer, R. R. Linde, *et al.*, "Program Confinement in KVM/370," *Proceedings*, ACM National Conference, Seattle, October, 1977.

[23] Jerome Saltzer, "Protection and the Control of Information in Multics," *Comm. ACM* **17**(7), July 1974, 388–402.

[24] *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28–STD, December 1985.

[25] D. Elliott Bell and Edmund L. Burke, "A Software Validation Technique for Certification: The Methodology," MTR–2932, Vol. I, The MITRE Corporation, Bedford, MA, April 1975. (ESD–TR–75–54)

[26] Paul Karger *et al.*, "A VMM Security Kernel for the VAX Architecture," *Proceedings*, 1990 IEEE Symposium on Security and Privacy, Oakland, CA, 7–9 May 1990, 2–19.

[27] David Clark and David Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings*, 1987 IEEE Symposium on Security and Privacy, Oakland, CA, 27–29 April 1987, 184–194.

[28] T. M. P. Lee, "Using Mandatory Integrity to Enforce 'Commercial' Security," *Proceedings*, 1988 IEEE Symposium on Security and Privacy, Oakland, CA, April 1988, 140–146.

[29] William R. Shockley, "Implementing the Clark/Wilson Integrity Policy Using Current Technology," *Proceedings*, 11th National Computer Security Conference, Baltimore, Maryland, October 1988, 29–37.

[30] David Brewer and Michael Nash, "The Chinese Wall Security Policy," *Proceedings*, 1989 IEEE Symposium on Security and Privacy, Oakland, CA, May 1989, 206–214.

[31] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, DoD 5200.28–STD, 31 July 1987, NCSC–TG–005.

[32] *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, DoD 5200.28–STD, April 1991, NCSC–TG–021.

[33] D. Elliott Bell, "Trusted Xenix Interpretation: Phase I," *Proceedings*, 13^{th} National Computer Security Conference, Washington, DC, 1–4 October 1990, 333–339.

[34] Clark Weissman, "BLACKER: Security for the DDN, Examples of A1 Security Engineering Trades," *Proceedings*, 1992 IEEE Computer Security Symposium on Research in Security and Privacy, 4–6 May 1992, Oakland, CA, 286–292.

[35] D. Elliott Bell, "Secure Computer Systems: A Network Interpretation", *Proceedings*, Second Aerospace Computer Conference, McLean, VA, 2–4 December 1986, 32–39.

[36] *TNI*, *op. cit.,* Appendix A.

[37] Garrett Birkhoff, *Lattice Theory* (1^{st} ed.), American Mathematical Society: Ann Arbor, Michigan, 1948.

[38] *Ibid.*, p. 163.

[39] D. Elliott Bell, "Lattices, Policies, and Implementations," *Proceedings*, 13^{th} National Computer Security Conference, Washington, DC, 1–4 October 1990, 165–171.

[40] D. Elliott Bell, "Putting Policy Commonalities to Work," *Proceedings*, 14^{th} National Computer Security Conference, Washington, DC, 1–4 October 1991, 456–471.

[41] Carrie Fisher. **Postcards from the Edge.** Simon & Schuster: New York, NY, 1987.

[42] Cynthia E. Irvine, "Considering Lifecycle Subversion," Invited presentation, MLS Workshop, Alexandria, VA, 24 September, 2003.

[43] Emory A. Anderson, Cynthia E. Irvine, and Roger R. Schell, "Subversion as a Threat in Information Warfare," *Journal of Information Warfare*, **3**(2), June 2004, pp. 52–65.

[44] Dorothy Denning, *et al.,* "A Multilevel Relational Data Model," *Proceedings*, 1987 IEEE Computer Security Symposium on Research in Security and Privacy, 27–29 April 1987, Oakland, CA, 220–242.