If A1 is the Answer, What was the Question? An Edgy Naïf's Retrospective on Promulgating the *Trusted Computer Systems Evaluation Criteria*

Marvin Schaefer
Books With a Past, LLC
bwapast@erols.com

Abstract

This paper provides an introspective retrospective on the history and development of the United States Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). Known to many as the Orange Book, the TCSEC contained a distillation of what many researchers considered to be the soundest proven principles and practices for achieving graded degrees of sensitive information protection on multiuser computing systems. While its seven stated evaluation classes were explicitly directed to standalone computer systems, many of its authors contended that its principles would stand as adequate guidance for the design, implementation, assurance, evaluation and certification of other classes of computing applications including database management systems and networks. The account is a personal reminiscence of the author, and concludes with a subjective assessment of the TCSEC's validity in the face of its successor evaluation criteria.

1. Introductory: From the primordial ooze

In the beginning, there was no computer security problem.[1] There was no external threat. There was no intrusion problem.

You could ask almost anyone who used or operated computers in those days of yesteryear. Computers were expensive, so they were kept and operated in physically protected rooms. Only authorized, trained personnel were allowed physical access to mainframes or peripherals. Users submitted jobs on punched card decks or on tape, jobs were run successively, and every job had a stated duration in which to run or be "kicked off" the machine. If one was lucky, an aborted or failed job would produce a dump before being unceremoniously dumped. Common belief was that physical protection and personnel background checks were adequate to protect data in the government, at banks, and in industry.

This paper is a personal account of my involvement in the events leading to the development, writing, trial use, promulgation, official use, and misuse of the United States *Department of Defense Trusted Computer System Evaluation Criteria* (TCSEC). Even after it became a Department of Defense Standard, many came to know it by its paper cover as the *Orange Book.* Orange was the final color of an evolving series of published drafts that began on 24 May of 1982 with powder blue, and progressed through white and a sickly shade of olive green, until it reached its distinctive final draft shade of orange on 15 August 1983.

1.1. Early education in computer security

I[2] first left academia in 1965 for an experimental summer research and technology training program in Santa Monica at the System Development Corporation (SDC), a non-profit spin-off of the RAND Corporation. The atmosphere provided to our group of "special trainees" at SDC was a radical departure from that of

[1] Earl Boebert would dispute this, having exploited a flaw in the early 1960s at Stanford University to read and modify memory [now called storage] to plant a Trojan horse. Boebert spoke of this as his *locked room mystery*.

[2] Because this is a personal account, I use both the first person singular and plural pronouns. The latter are used for most contexts, as important results often came not from individuals but as a result of close collaboration with many colleagues in several research and development institutions in academia, industry and government.

the UCLA mathematics department. SDC had a staff of academic mathematicians and researchers from the social and hard sciences in addition to its computing staff. SDC received the majority of its funding from the Department of Defense and other government agencies. The company teemed with modern vacuum tube and semitransistorized computers, consuming fully half of the electric power generated for the city of Santa Monica. Our three-month training class and the opportunities to which we were exposed were so exciting that most of us cancelled our future plans and stayed on afterward in the Research and Technology Directorate.

The young people in our training class held freshly-minted degrees in mathematics, physics, music, literature, and philosophy. We were assigned to use the new experimental IBM A/N-FSQ-32(V) Time Sharing System[3]. In our three months of training, we received lectures from pioneering researchers in hardware and operating system design, assemblers, programming language design, compilers, interpreters, metacompilers, natural language processing, database management, list processing (LISP 1.5), and time sharing system design. But the most exciting was our chance to use the Q-32 for our classwork. The Q-32 would support up to 24 interactive users at a time. Our class got to share this computer with SDC's researchers, and we were given individual login IDs so that our projects could be billed for the time we used. These IDs were not used for identification or authorization.

Occasionally we were asked to get off the machine to allow a remote demonstration of the system to run smoothly and rapidly. Many of these demonstrations were scheduled and conducted from overseas by telephone dataphone and teletypewriter. Other than for these demonstrations, there were no public or employee dialup services on the Q-32.

There were no access controls on data, and files were generally meant to be shared with colleagues. Indeed, the concept of protection was soon revealed to be nonexistent as a few of us inadvertently discovered how to subvert careful operating system policies and mechanisms on the single protection state Q-32 architecture.

1.1.1. Modifying an operating system. It was here that my first experiences penetrating computer system took place. While we could code in the full systems programming language JOVIAL, this required overnight batch-mode compilation before we could interact with our programs under time-sharing. This time delay could be avoided by programming in the fully-interactive **T**ime **S**hared **Int**erpreter (TINT) for rapid prototyping, a subset time-shared compiler **J**OVIAL **T**ime Sharing **S**ubset (JTS), and LISP 1.5. However, Q-32 TSS required that adequate space be available in a *contiguous* block on one of the swap drums in order to load the entire compiler or LISP or to do any work with a user program. This was because dynamic paging had not yet been invented. So I and a couple of colleagues managed to write a very small program (appropriately named CANCER) that would usurp the operating system, repack the drums that contained other user programs, modify the internal systems tables, and make room for our own programs to load. All of this had to be completed inside of a single quantum. Sometimes it didn't. The resulting system crash got other users angry. It was also less than amusing to the operating systems staff. Our actions were dismissed as those of college kids having fun, and not those of malicious users. Besides, until our program (CANCER) was developed and fully debugged, everyone had to waste time waiting for adequate space to become available.

1.1.2. Cat and MOUSE. Q-32 TSS scheduling was initially a "democratic" system. Every program[4] was given a 300 ms quantum in a strictly round robin scheme. This proved to interfere with the performance of highly interactive programs, and it resulted in very long compilation time. So Clark Weissman decided to implement queues for different kinds of jobs: initially, there were two queues an "interactive" queue and a "production" queue. Membership in the interactive queue depended on the program performing an input or output during every few quanta, and a program that failed to do this was moved into the production queue. Here, a program would execute less frequently, but once it reached the head of the queue it would alternate through ten quanta interleaved with members of the interactive queue prior to being sent to the back of the queue. In practice, users needed to get their work done, and they found ways to avoid being placed in the production queue. Soon most user-written programs were soon laced with code that would perform useless single-character output operations to the terminal to avoid being moved out of the interactive queue. This did nothing to shorten compilation time for people

[3] Developed under contract to ARPA as one of two "competing" projects. The other contemporaneous time sharing system was the Compatible Time Sharing System (CTSS) at MIT's Project MAC and Bell Labs designed in 1959 and operational from 1961-71.

[4] Process was not yet a developed concept and 'program' was synonymous with the executing context as well as with the code image. Privilege state was not yet a well-recognized concept either.

using compilers or LISP, so additional dodges were found to avoid the production queue.

On learning of these, Weissman and his staff added intermediate queues and introduced foils to the user's avoidance techniques. Ultimately, a few users collaborated on a set of means to modify the operating system's scheduling algorithms, during execution of course, in their favor. As there was no protected memory, and there were no privileged instructions, there was little other than procedural controls that could be applied by the operating system staff to control usage. Soon, passwords were associated with user IDs, and audit logs were generated out to tape along with the billing information. But, since there was no protection on the machine, these proved to be illusory at best—nothing in the hardware could prevent any program from accessing system audit files. The systems programming staff introduced monitoring programs that would check on the behavior of specific users and programs, and we found means to use programs to abort or replace the monitoring programs. And so, for each protective or regulatory move made by the defenders, the researchers soon found themselves forced to launch offensive countermoves in order to feel that they could get their work done on the system.

In effect, the system's design was still a prototype, and if future system versions were to evolve, test users were needed to provide useful data and feedback. Q-32 TSS was not designed with security or protection in mind, and because of hardware inadequacies, no form of strong security could have been provided in any case. To the best of my knowledge, the only penetrations or subversions of the system were performed in order to get work done more rapidly, and no user data was maliciously corrupted or spied-upon. But such acts of user anarchy resulted in a semideclared "state of war" between users and the operating system staff.

1.1.3. Concepts *in absentia* but not forgotten. The primitive understandings of protection mechanisms made early time sharing systems look like the Wide Open Old West. The following important concepts were soon learnt to be absent in contemporaneous computers and systems:

- Protection policy
- Multiple privilege states
- Segmented memory
- Privileged instructions
- The process as subject concept
- Access controls on objects
- Individual accountability
- Protected audit trails

It soon became obvious that if systems were to control users, such concepts would need to be implemented. But, of course, we didn't know that then…. Many of these lacking controls remained AWOL through the 1970s and 1980s on a majority of ARPANet sites, and thence well into the 1990s and early 21st century on the Internet. But, it must be observed, there was "no known security problem that wasn't caused by improper management and that couldn't be corrected by proper procedural controls."

1.2. The Ware Report

In our research lab, as elsewhere, there was no perceived computer security problem. All SDC employees had a Defense Department clearance because there were some classified projects in SDC's buildings. Guests had to sign in with a guard and wear a visitor badge while being escorted.

However, a series of events in the spring and summer of 1967 focused the Department of Defense's attention to the question of security control in resource-sharing systems. In June of that year, Bob Taylor, director of the Office of Information Processing Techniques at ARPA was tasked to "form a Task Force to study and recommend hardware and software safeguards that would satisfactorily protect classified information in multi-access, resource-sharing computer systems." The RAND Corporation's Willis Ware was asked to chair the Task Force under the authority of the Defense Science Board, and began meeting in October. The Task Force and its panels included a number of future security visionaries and colleagues, including E.L. (Ted) Glaser, Pat Haverty, Art Bushkin, Bob Mosier, Jerry Moskowitz, Larry Roberts, Bob von Buelow, Roy McCabe, Barry Wessler, James P. Anderson, Ed Bensley, Dan Edwards, Jerry Saltzer, Hilda Faust [later Mathieu], Bob Balzer, and Clark Weissman.[5]

Their landmark report, *Security Controls for Computer Systems* [27], was published as a classified CONFIDENTIAL RAND report in February 1970. Its findings are still of interest today, but unfortunately, were not widely disseminated at the time because of the classification.[6]

[5] SDC was well represented on the Task Force by Mosier, von Buelow, McCabe and Weissman. This eventually proved to benefit my education in computer security.

[6] In the preface to the publicly reissued version of the report, Ware states: "At that time it was felt that because representatives

The Ware Report was prescient in its presentation of:

1. Security risks and the nature of security vulnerabilities. In addition, computer network vulnerabilities were enumerated to include: hardware and software failures; emanations; wiretaps and crosstalk; sabotage from operators, maintenance staff, systems programmers, and subtle software modifications by users, and attachment of bugs to terminals. Identified difficulties of securing a computer system included:
2. consideration of the difficulty and complexity of security controls on a range of ranging from the most easily controlled processing environments: local-access batch, remote-access batch, local-access multiprogramming, remote-access multiprogramming, to the most difficult of all, remote-access time-shared systems
3. consideration of the challenges presented by increasing user capability and the complexity of security controls based on the simplest risks posed by template-based file query systems, through a scale of increasing challenges of programming via interpretation, programming via limited languages and checked-out compilers,[7] up to users having full-programming capabilities as the most risky environment
4. The specifics of access control policy to specific files were to be-based on system-confirmed authentication of the user's identity and clearance, the clearance of the facility from which the user would access the file, and the clearances of designate output devices
5. Defense Department needs were identified for adaptive access control policies in the face of degraded operation or national emergency. The Report specifically addressed the costs, financial to mission-specific, of implementing access controls.
6. Hardware and software, acting together, were to provide isolation from user programs of all operating system security controls, including the audit trail. The emerging design of Multics was specifically given salutary comments in an explication of how this could be reliably achieved.
7. Specific hardware features were identified as necessary for access control, including a minimum of a supervisor state and a user state, program isolation mechanisms such as base and bounds mechanisms, machine fault detection, etc.
8. Operating system complexity was recognized as an impediment to actually proving that the design and implementation were correct. To compensate for this, the Ware Report called for detailed documentation in text and flowcharts of the *modules* on which access control was based.
9. Penetration testing was required on a periodic basis, and it was highly recommended that some of these tests would necessarily be conducted by outside personnel not connected with the design or implementation of the system.

The system would need to periodically perform dynamic operating system integrity tests and to validate the efficacy of specific security tests by running attempts to subvert the access control policy.

Although incomplete by contemporary standards, many of the Ware Report's recommendations and findings are still valid. The Report contained an appendix in which system access controls were formally specified as a set of transformation rules in Backus-Naur Form (BNF).

Ultimately, the Report's technical recommendations came from the research community. Many ideas, including segmented virtual memory and privilege states are reflected in the Task Force's conclusions and recommendations. The significant outgrowth of the Task Force's work was the funded development of two multiprocessing systems that were designed incorporating computer security principles from the start. These were the refined security design of Multics at MIT (first specified in 1965), and the multilevel time-sharing system adept-50 in 1967-8 at SDC.

1.2.1. Multics. The Multics project introduced a number of security principles to the technology. The system was designed as a collaboration between MIT, General Electric and Bell Labs under arpa Funding to run a modified GE 635 computer, renamed the GE 645.. Early management and technical contribution spanned the three organizations, and included principal contributions from Fernando "Corby" Corbató, Ed

from government agencies participated in the work of the Task Force, the information in the report would appear to be of an official nature, suggestive of the policies and guidelines that would eventually be established. Consequently, it was felt prudent to classify the report CONFIDENTIAL overall…."

[7] There was a long-standing belief that allowing assembly-language programming would pose the greatest threats to computer security. Programming languages like FORTRAN did not explicitly support assembly-language programming, although the systems programming language JOVIAL did. Hence, FORTRAN was consider "safer" at the time of the Ware Report. A few years later, this fallacy was corrected through a demonstration that EQUIVALENCE and DIMENSION declarations could be used in a way that would enable the execution of assembled code posing as binary data.

Vance, Vic Vyssotsky, Peter Neumann, Jerry Saltzer, Ted Glaser, Butler Lampson, Tom Van Vleck and Charlie Clingen.

Many of the concepts I mentioned above were ultimately incorporated in Multics, including: Segmented virtual memory, protection and privilege rings, the operating system was coded in PL/I instead of assembler, and it incorporated modular design and least privilege.[8]

Multics design, implementation, and use ran over the period 1963-2000. Multics history is documented at http://www.multicians.org/.

Because of the lack of an identified computer security problem, Bell Labs' participation in the Multics project ended after a few years. A laboratory version of a stripped-down Multics-like system soon came about that had no file access controls because they weren't perceived as necessary in the lab environment. Peter Neumann suggested the name for the system should be Unics (making the pun of an emasculated single-user Multics) because of the many Multics features that had been eliminated. It was Brian Kernighan who claims credit for naming it UNIX.[9]

1.2.2. ADEPT-50. ARPA awarded a contract in 1966 to Sdc to implement a Time Shared Database Management System (TDMS) on an IBM S/360 model 50 computer. TDMS was specified to support rapid retrieval on ad-hoc queries over any field combination as a recursively-defined hierarchical fully-inverted database. The underlying operating system reflected a multilevel security design created by Clark Weissman, who directed a staff that included Dick Linde, Sal Aranda, Martha Bleier, Barry Gold, Steve Sherman, and Clay Fox.

In 1967, Weissman asked me to prepare a formal mathematical formulation of the ADEPT-50 [28] interpretation of the military security policy-based model described in the Ware Report. The model addressed clearance, compartments/categories, franchise (specific permissions), and need-to-know for authenticated users. It implemented a dynamic user-specific High Water-Mark (HWM) access policy that determined the level at which objects and files created by the user could be read or stored to media.[10]

I ported META6, the compiler-compiler I had been developing, to ADEPT-50 which served as a base for producing tools for converting flat-structure record-based databases automatically into the TDMS data model. In 1967-8, I ended up commuting on a weekly basis from Santa Monica to the Pentagon,[11] which had a growing need to convert classified databases over to TDMS. Keypunch errors and semantic confusions among the military users were the causes of my many trips to the National Military Command System Support Center (NMCSSC) during the Vietnam War.

ADEPT-50 was being used to support multilevel operation, and its dynamic HWM policy, in anticipation of the *-Property, proved to be confusing and frustrating to nmcssc users. The most plaguing of problems was that a user would create a new file or modify an old one such that it could not be accessed once closed. This was because the new file's security level would be the least upper bound of the <classification, category, franchise> attributes of all files opened during the user's session. The new file's access control list became the intersection of the access control lists (and rights) of all open files that had been accessed during the session. This rapidly became a singleton access control list, thereby isolating the new or modified file from all other users.

ADEPT-50 was in use through the Vietnam War and well into the 1970s.

1.3. The Anderson Report

The U.S. Air Force awarded a contract in 1972 to the James P. Anderson Company to produce a computer security planning study. A panel of experts, chaired by E.L. "Ted" Glaser, met between February and September 1972 to produce an UNCLASSIFIED two-volume report [2]. Additional participants included Eldred Nelson, Bruce Peters, Dan Edwards, Hilda Faust, Steve Lipner, Roger Schell, Clark Weissman and Chuck Rose. The Anderson Report outlined a sound approach to the development of multiuser multilevel computer systems.

The most important contribution of the Anderson Panel was its strong direction toward the use of formalisms in addressing controlled sharing in terms of an access control policy model, the design of the security mechanisms, and the production of security assurances. The concept of an access control model, or security policy model, soon led to formalisms. G. Scott Graham and Peter Denning had published a model in which the accesses by *subjects* to *objects* were represented in a *matrix M* wherein: rows represented subjects *S*, columns represented objects *O*, and

[8] A military version, AFDSC Multics, was derived from Multics that provided for hierarchical and compartmented security clearances/classifications. This later version implemented a version of the Bell-LaPadula security policy models.

[9] Peter Neumann, private communication.

[10] ADEPT-50 was certified and accredited for use by part of the U.S. Intelligence Community and in the Pentagon.

[11] This was a difficult period for me. I was working full-time both at SDC *and* at UCLA on a doctorate in mathematics.

elements $M_{s,o}$ represented the specific *modes of access* or *permissions* subjects had to objects. It was possible in this model to identify all subjects who had any form of access to any specified object as well as to identify all objects to which each subject had any specified form of access. In this model, interprocess communication implied that subjects also be viewed as objects, a deduction that ultimately resulted in the recognition of a subject as a <process, domain>-pair, wherein *domain* included all of the attributes of the executing process (including security, integrity, and privilege-state attributes). In a large system, it was tacitly assumed that *M* would be a large sparse matrix. Thus, implementation concerns suggested that *M* either have its attributes stored in lists associated with subjects (*capability lists*) or with objects (*access control lists*). Harrison, Ruzzo and Ullman [10] also showed that *discretionary access control* (*DAC*), the form of access modeled in *M*, had a degree of uncertainty equivalent to the Turing Halting Problem.

The Anderson Study also resulted in publication of a series of formal models that addressed abstractions of military classifications and clearances, so-called *nondiscretionary access control* (later to be known as *mandatory access control* or *MAC*). The most popular of these models were those of Case Western Reserve University [26] and of D. Elliott Bell and Leonard LaPadula [3]. At the time of their publication and refinement, these models were reputed to have formally provable security properties. This was later questioned by John McLean [17] and others.

Central to the Anderson Study's framework is its elaboration of the reference monitor concept. Its implementation is a reference validation mechanism (RVM), often called a "security kernel," a term introduced by Major Roger R. Schell at an early Stockton Gaines workshop. Specifically, the Study prescribed three requirements characterizing the RMC:

a. The reference validation mechanism must be tamper proof.
b. The reference validation mechanism must *always* be invoked.
c. The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

It elaborated:

> Each of these requirements is significant, for without them the mechanism cannot be considered secure. The first is obvious, since if the reference validation mechanism can be tampered with, its validity is destroyed, as is any hope of achieving security through it. The second requirement of *always* invoking the reference validation mechanism simply states that if the reference validation is (or must be) suspended for some group of programs, then those programs must be considered part of the security apparatus, and be subject to the first and last requirements. The last requirement is equally important. It states that *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. [2, vol. I, p. 10]

The Anderson Report recognized that operating systems were larger and more complex than most programs, and that exhaustive testing was out of the question. So the report called for modularization that would support analysis and credible testing. There is an interesting consideration of the possibility of subjecting operating systems to mathematical proofs of correctness, but this was understood to be well-beyond the capability of human individuals. There is a discussion of the potential for research that would support generation of computer-aided formal verification assurances sometime in the future.

Like the Ware Report, the Anderson Report recognized the need to consider system use environment and functional characteristics as part of the overall risk and vulnerability assessment. It considered secure transaction systems to be the least threatening and most achievable of the "open use" multiprogrammed system contexts. The next most risky category to be considered was the High-Order-Language (HOL)—the only system described in the Ware Report which raises the possibility of breaking out of the confines of FORTRAN. No explicit details on how this might be achieved are presented.

The Report outlines several development plans for research and development needed to achieve secure open-use systems. There is specific reference to planning research programs on secure networks, security models, security software engineering, security surveillance, certification techniques, architecture research, data integrity and reliability, automatic classification, magnetic media, and computer aided integrated design. Of interest is an allusion to

> The possibility of internal encryption of computer programs and data was first advanced in 1966 prior to the Defense Science Board Task Force on Computer Security. Since that time it has received sporadic attention. It appears that it is possible to apply this technique either as an appliqué or as an integral part of the design of computer systems [2, vol. II, p. 44].

It is interesting to note that the Anderson Panel had the foresight to predict that:

> Perhaps the most interesting potential of modern technology will be the radical reduction of cost of computer main frames. We can, for all intents and purposes assume that computer main frames will be effectively "free" in the not too distant future. As a consequence, if it is really necessary to separate various users, each can be given his own computer. However, more often than not, they are dealing with common data bases and must hand off certain common data to one another and, on occasion share programs. As a result, we are still in need of secure computing systems. With very low cost computer logic however, we have the possibility of a distributed system. By this is meant a system in which the various system functions may be distributed among different machines which are "netted" together. Netting does not imply a number of machines doing identical tasks, nor does it imply a number of necessarily identical machines… [2, vol. II, p. 102].

2. Principles, perceptions, and worked examples

Following publication of the Ware and Anderson studies, interest in "secure" systems and products appeared to have increased in number and depth. Donn Parker began to publish books on computer crime, but most of what appeared in the popular press described variants on insiders automating white collar crime against financial institutions. Among institutions that had potential computer security vulnerabilities, their identified concerns focused in on reliability and protection from disruption of service. Very few organizations expressed concern over protecting confidential information from unauthorized modification or display.[12] At the beginning of the 1970s, however, there was still no consensus that *technical* measures were required to counter any identified computer security threats. From Parker's books on through banking, commercial and military users, the expressed belief was that trained system managers and the use of guards, badges, personnel background examinations, and encrypted external communications would suffice to meet their protection requirements.

During this period, several security working groups were formed that brought security researchers together. One, hosted at the RAND Corporation, was chaired by R. Stockton Gaines. In addition to members of the Ware Panel, the group periodically included additional researchers from UCLA, USC Information Sciences Institute (USC-ISI), MIT, RAND, SDC, MITRE, NSA, ESD, and other institutes. A second group, organized later by Steve Walker while he was at DARPA, added principals from Carnegie-Melon, Bell Labs and various DARPA contractors This latter working group evolved into a kernel of the Department of Defense's Computer Security Initiative. Participants who met in these two groups included: Jim Anderson, Roger Schell, Dan Edwards, Ann Marmor-Squires, Anita Jones, Butler Lampson, Jerry Saltzer, Gerry Popek, Clark Weissman, Dick Bisby, Dennis Hollingsworth, Jim Gillogly, David Bonyun, Mike Schroeder, Peter Neumann, Ed Burke, Steve Lipner, and me.

From the Anderson Report and refined by these groups also came the notion of the ***security perimeter***, which consisted exclusively of security-relevant code and the minimal subset of the operating system required to support it. The code within the security perimeter became known as the ***security kernel***, the *minimal* body of code required to enforce the system security policy. Almost as a mantra, many in the research community claimed that if "the good guys write the code inside the security perimeter, then the system will be secure, even if the adversary writes the remainder of the operating system and its utilities."

2.1. Access control policy considerations

Up through the early 1970s, I worked closely with clients in both military and commercial application sectors. I became aware of the wide range of discretionary access controls that were asked for by different user groups. Their requirements included:

- *Password protected files*: where knowledge of a password suffices to gain complete file access
- *Time-based access controls:* where specific actions could only be performed on specified days and/or during identified time periods. One such defined mode of access may be *no access.*
- *Group-based access controls:* where access modes are uniformly applied to users assigned membership in named groups, *e.g.*, only members of the salary administrator group may modify salaries. One such defined mode of access may be *no access.*
- *Rôle-based access controls:* where, *e.g.*, persons acting in the rôle of salary administrator may only perform specified accesses and no others (*e.g.*, no general programming while acting in the rôle of salary administrator. One such defined mode of access may be *no access.*

[12] Banks were far more concerned over unauthorized modification of transactions than breeches of confidentiality.

- *User-specific access controls*: associated with each user and each protected object is a set of specific modes of access that the user is permitted to have for the object; this is the form of access control typified by access control lists. As in the Graham-Denning model, one mode of access may be *control* access, a mode that allows the user to grant or deny access for other users. One such defined mode of access may be *no access.*
- *Prohibited-access controls*: sets of users for whom specific forms of access to objects are specifically prohibited. These prohibitions may be broader than the *no access* form described *supra*.
- *Combinations of the above*: *e.g.*, a specified user in a particular rôle at a given time from an administrator's terminal may access a controlled object in only a specified set of modes.
- *Formularies*: a term coined by Lance Hoffman wherein access to a specific object is computed by a specified, possibly *ad hoc*, function. More robust than the foregoing access controls, a formulary may, *e.g.*, restrict a salary administrator to modifying only the salaries of those reporting to managers in a specific department providing that the new salary does not exceed a stated percentage of the employee's manager's salary.

In full generality, discretionary access control was recognised as being much more complex than a simple label-based policy. Further, the undecidability issues raised by Harrison, Ruzzo and Ullman and additional implementation complexity issues suggested strongly that high-assurance systems could not be uniquely based on DAC mechanisms.

2.2. Implementation considerations

Early attempts at implementing robust, production-quality secure operating systems proved to be surprisingly difficult, despite the application of sound principles. In part, this was because of the lack of experience in dealing with the problems of reducing theoretical objectives of layered architectures, least privilege and least common mechanism to engineering practice within the framework of information flow confinement.

2.2.1. Multics. It has become common contemporary practice for scarcely tested and inadequately debugged software products to be sold commercially. Various vendors, notably Microsoft, have been accused of launching products that have never been subjected to beta testing. Disgruntled customers have complained and lampooned such companies and their programming staffs' capabilities, not entirely without justification.

Knowledge gained from penetration studies had made it clear that complexity is a strong contributing cause of system vulnerabilities. Complex designs are harder to implement than simple ones. Complex programming languages are more difficult to use than simpler ones, but complex coding sequences—in any language—are more difficult to understand weeks or months later than simpler ones. Any error in a security-relevant code sequence provides a potential foothold for a penetration attack.

This and related observations led the programming methodology and software engineering communities to espouse the use of high-order type-safe programming languages for the development of all programs and, to the extent possible, of systems. Modularity and data hiding were considered essential to a divide-and-conquer strategy for breaking down systems into manageable and easily programmed bodies of correct code. Beginning with the seminal early 1970s' work of Edsger Dijkstra and C.A.R. Hoare, system developers had begun to decompose their designs into strictly ordered hierarchies of modules in which there were no "upward" or circular functional dependencies.[13]

Dave Parnas subsequently produced worked examples of system decompositions of independent modules wherein the internal implementation and all internal variables were "hidden" and where there were no global variables – thus requiring that intermodule communication would require explicit parameter passing as values rather than as references. All modules were to "advertise" their external interface, and each was required to validate its inputs prior to accepting them. Niklaus Wirth moved forward and designed Modula, a systematically refined family of system programming languages that supported "toy" system implementation from such modules. Similar programming languages were proposed for production work, including Euclid, a Defense Department-sponsored effort to produce a fully-verifiable systems programming language that supported the modularity and data-hiding methodology. Eventually, this led to the development of the programming language Ada.[14]

Corporate misunderstandings of the philosophy of "structured programming" and its call for the abolition of the GO-statement and FOR-loops resulted in added code complexity. Some companies naïvely misunderstood the modularity concept and restricted

[13] Coincidentally, penetration teams had identified and exploited such functional dependencies in systems.

[14] In 1972, I participated in Jean Ichbiah's design of the system programming language LIS, the direct predecessor of Ada.

their programmers to writing modules no longer than a certain number of statements or lines of code. While this achieved modules that appeared to be small, in reality, modules were often prematurely terminated with a call on another module that continued from where the first left off. This resulted in obscure, and sometimes complex, calling sequences. Many also insisted that in a strict hierarchy of modules, a function call could only be between adjacent levels (or identical levels if within a tree of the same module) in the hierarchy.

The Anderson Report's prescriptions called for a RVM that was "small enough to be subject to analysis and tests, the completeness of which can be assured." This requirement represented a misconception that smallness implied conceptual simplicity. As will be seen, additional understanding gleaned from experiential missteps led to a restatement of this attribute. The rationale introduced in the first (powder-blue) draft of the TCSEC [8] restated it as "[the RVM] must be of sufficiently simple organization and complexity to be subjected to analysis and tests, the completeness of which can be assured." This revision persisted into the published DoD standard. [7, p. 67]

2.2.2. Efficiency considerations. Early attempts to implement systems along the lines of a Parnas decomposition uncovered problems in efficiency and size. Parameter passing and the lack of global variables proved to be cumbersome and inefficient. Attempts to work around the inefficiencies led to larger modules and to greater complexity. In some cases, the work-arounds involved linkages to machine language structures.

In addition, calls to different modules or different privilege states required a context change, and even in rapid, specially-designed hardware, several "unnecessary" instructions needed to be executed for both the calling and return sequence, thereby adding even more overhead to the program. This became a fundamental clash between the principles of least privilege and least common mechanism on the one hand, and performance efficiency, which called for placing as much as possible in the same domain of execution.

2.3. False senses of security

Most operating system vendors subjected their products to extensive internal testing prior to their release. In those days, most systems were sold (or provided gratis along with the hardware) "bundled" with assemblers, compilers, and utilities. For various reasons, testing was performed far more aggressively prior to product release than it is today, and even after product release, patches to identified errors were regularly distributed to users as part of vendors' product maintenance programs.

However, even then, new features tended to be included in the maintenance releases. Experience showed that the patches and new features introduced new bugs into the systems. System reliability was always a problem area, and denials of service had always been a focus area for testing. Grafted-on features proved to be most vulnerable to runaway programs. A form of testing, known as stress testing or security penetration testing, became more common.

Vendors started to make isolated product security claims. Some vendors even asked tiger teams to try to penetrate their products. When security testing failed to find a problem, vendors advertised the fact as a system strength.

The failure of security testers to find flaws did not suffice to prove their absence. It showed, instead, the limitations of resources or imagination on the part of the penetration team. Indeed, a new team was generally able to penetrate such systems in a matter of a couple of weeks. Even on systems whose identified flaws had all been "corrected", testing by a new team usually found exploitable security vulnerabilities, often of a completely different kind than found by their predecessors. And, tiger teams soon focused on penetrating patches rather than the original code.

2.4. Tiger team efforts

Security research and development were seriously in need of funding in the 1970s. Without a *validated requirement statement*, military funding was limited and shaky. Many a study or seed project was cancelled in order to fund military acquisitions of war matériel.

So, the researchers mobilized to create legitimate demand for security research and development programs.

One form of "consciousness-raising" involved the almost romanticized activities of tiger teams. Most prominent of these was the USAF ESD team, which included Major Roger Schell, 1Lt. Paul Karger, Ed Burke, and Steve Lipner. In addition to their documented successful penetrations of Multics and DIAOLS, this and other tiger teams seemed always to succeed in penetrating their targeted operating systems.

After a number of documented penetration studies, the Anderson Study made it clear that a *necessary* condition for securing an operating system was hardware that provided, as a minimum, a distinct hardware state for the protection of the security mechanism. Another way of stating this was that the

security relevant instruction set needed to be a subset of the privileged instruction set that could only be executed in supervisor state.

Bob Abbott directed the Research Into Secure Operating Systems (RISOS) study [1] in 1976 which led to a characterization of seven general classes of system flaws:

a. Incomplete parameter validation
b. Inconsistent parameter validation
c. Implicit sharing of privileged/confidential data
d. Asynchronous validation/inadequate serialization
e. Inadequate identification/authentication/-authorization
f. Violable prohibition/limit
g. Exploitable logic error

Matt Bishop and colleagues at Purdue University produced and documented some startling penetration exploitations of commercial Unix Systems in the late 1970s.

Another tiger team, at SDC, was put together by Weissman. In 1972-3, SDC was given a contract to conduct research jointly on the security of IBM's VM/370. Like Multics, VM/370 provided three execution states: one fully-privileged hardware supervisor state for the VM/370 hypervisor, one emulated *virtual* supervisor state for virtual operating systems, and a hardware problem state for user programs. VM/370 hypervisor was small, and much of its design and code was based on a conceptually simple model. The implementation was well-structured for its time (though written in assembler) and was properly structured to resist attack. Unlike Multics' use of special hardware and protection rings, VM/370 had to emulate virtual supervisor state, which was not supported by the IBM S/370 hardware base. The security relevant instructions on the S/370 were a proper subset of its privileged instruction set. The IBM hardware was capable of trapping attempts to execute privileged operations so that the VM/370 hypervisor could legality-check them prior to their execution.

In relatively short order, the joint team succeeded in identifying and exploiting a number of subtle technical flaws in the design and implementation of VM/370 that resulted in their achieving full control over the system. In many cases, the exploited vulnerabilities were not a flaw in coding, but were faithfully-emulated security flaws in the S/370 hardware architecture that were virtualized away from users' direct access. As a result, the VM/370 hypervisor could be conscripted into abetting its own penetration.

In addition to producing a proprietary vulnerability report, the team produced a jointly authored paper for the *IBM Systems Journal*. Dick Linde and Ray Phillips, who led the SDC team, produced a formalization of the approach to identifying potential security flaws, now known as the Flaw Hypothesis Methodology.

Clark Weissman began offering penetration studies as an SDC service. His motivation was to show clients how vulnerable their systems were in hopes of obtaining adequate funding to methodically eliminate identified security vulnerabilities by reworking the penetrated systems. This goal was not achieved, however. The penetrators were altogether too successful. Many clients did not believe the penetration study's results, and remained skeptical until they went through the shock of watching a remote user compromise their system. Some clients went into denial, convincing themselves they were safe because of an attack's sophistication or obscurity -- too often, one heard the phrase "no one would do *that*". SDC was unable to provide a quick, inexpensive "fix" to the flawed systems, and their clients ordered SDC not to reveal uncountered system vulnerabilities. Put simply, the costs of correcting developers' security flaws would be too great for any single client company to bear alone. System vendors, on the other hand, were not faced with overwhelming customer demand that they secure their products.

In the large, customers were after a simple round of quick-fix "penetrate and patch" wherein the system vendor would patch the system once the flaws were identified. The penetrators found that the "repaired" systems were easier to penetrate the next time around, because the patches generally introduced new security vulnerabilities.

So the tiger team activities did not produce a golden age of funded security research and development. However, they did provide an adequate number of worked examples of security flaws from which to glean understanding of those design and implementation techniques that could be most resistant to attack or conscription.

2.5. Secure system prototypes

Beginning with the convening of the Ware Panel, the next decade saw the beginnings of various other computer security-related activities. In part, with tremendous influence from the findings of the Anderson Panel, this activity was spurred by research funding falling out from the Vietnam War and related activities. The U.S. Department of Defense and its agencies were the principal funding sources for research as well as for development.

The private sector developed several security-oriented commercial products and prototypes. Among

the independently-developed products were: IBM's Resource Access Control Facility (RACF) add-on to MVS and the Virtual Machine Facility/370 (VM/370) operating system, and Tymshare's capability-based operating system Gnosis.

The U.S. DoD sponsored system and prototype developments for several prototypes and operational systems, including:

- Multilevel AFDSC Multics
- MITRE UNIX 11/45 prototype
- Stanford Research Institute's Provably Secure Operating System (PSOS) design
- Two attempts at a multilevel secure version of UNIX (Ford Aerospace's KSOS-11 and Honeywell's KSOS-6)
- SDC's Kernelized VM/370 (KVM/370)
- Ford Aerospace's AUTODIN II
- ITT/IBM's SACDIN
- SDC's BLACKER project.

In addition to these, two forays into multilevel database management were conducted. At SDC, Clark Weissman reassigned me to perform computer security research, abandoning my chosen research study on applications of Petri nets. Tom Hinke and I produced a study, model and design, under sponsorship from Rome Air Development Center (RADC), for a multilevel relational database management system that could run under an unmodified AFDSC Multics. My Petri net research proved to have an application and was used in our model as a multilevel secure solution to synchronizing database queries and updates.[15] David Bonyun and colleagues at I.P. Sharp Associates (Canada) produced a multilevel DBMS model for the Air Force Electronic Systems Division. The IP Sharp model was designed to have been implemented within Rings 1 and 2 of Multics and identified various security primitives to support multilevel database management.

The Hinke-Schaefer multilevel DBMS work is noteworthy because its implementation would contain *no* security relevant code and was contractually required *not* to require any modifications to the Multics Security Kernel. It was instead constrained to operate, under the Least Privilege concept, as a completely unprivileged process in user rings.

2.6. Toward system security evaluation criteria

By 1978 researchers and developers had begun to claim that they knew precisely how to implement secure (or "secure enough") systems. While few projects had produced fully operational, well-tuned secure systems, such products were not readily available. Although multilevel AFDSC Multics and ADEPT-50 had been fielded and accredited, there was an understanding that their performance left something to be desired. They were far from being "user friendly". AUTODIN had been accredited for *full* multilevel use (UNCLASSIFIED through compartmented TOP SECRET), the consensus was that if AUTODIN were subjected to recertification and accreditation analysis, it would fail based on contemporary technical knowledge of vulnerability analysis.

The research community had moved forward to achieve a preliminary understanding of covert channel analysis (CCA). The hubris of the moment had led many to claim that with the new secure systems, unauthorized direct access to files, spooling files, printer queues, the address spaces of other processes, etc., would be impossible. Thus, only by timed modulation of various system artifacts could a pair of coöperating Trojan horses communicate with each other in violation of the system security policy interpretation of confinement or the Bell-LaPadula *-property. Various technologies had been created for identifying, measuring and using covert channels [24], and some formal analysis tools had been created [18] to discover covert channels in formal system specifications.

2.6.1. Security kernels bad, TCBs good. For several reasons, many in the security research and development community began to oppose the reference monitor concept and its implementation as a security kernel. For the most part, the criticism focused on the perceived inefficiency of central mediation and context switching forced by the RVM's complete mediation requirement. Many argued against the strict notion of having to validate *every* reference to *every* system object.

One divide-and-conquer strategy, represented in the Bell-LaPadula models, was achieved by having the security kernel apply full policy mediation to every *initial* request or attempt by a subject to access an object in a specific mode. If that mode of access was consistent with policy, a descriptor or token would be generated that the kernel could rapidly consult to allow or reject all subsequent access attempts.

This resolved most of the problem. But it left open the question of how the controller of an object could immediately revoke all or selective access modes to that object. The custom hardware descriptor-based architecture of Multics allowed this to be done immediately. However, in other system architectures, such a feature was deemed too costly, and system

[15] A variation of our technique was independently developed by Reed and Kanodia, and is known as *event counts*.

security policies were modified to have access revocations become effective only on new access requests. For capability-based systems, where a process could endure for days or weeks, this problem resulted in many emotional arguments and dissents.

To some, a more significant issue became apparent with respect to certain modules that were included inside the security perimeter that were not directly related to protection or to supporting the implementation of protection-critical modules. Their inclusion as security relevant code was clear, for their improper operation could lead to a security policy compromise. For example, a resource scheduler or dispatcher could, in principle, operate in a less privileged domain than that of the security kernel. But many argued that a scheduler needed to have access to system-wide information, as scheduling decisions made only within a single security level could result in thrashing or other inefficiencies. The only way a scheduler could view such information under a multilevel security policy model would be if it executed as if it were a system-high subject. But in that case, any request it made to dispatch a specific subject could be misused to signal information as a covert channel in violation of the *-property.

Indeed, no matter how it was structured, every multilevel system had to have some internal processes that allocated or modulated global system resources. Covert channel analysis techniques showed that such processes could always be conscripted to violate information flow confinement requirements—even when such processes were implemented correctly (*i.e.*, in full conformity with their specifications). There was growing awareness of this problem in KVM/370, which called them the Global Processes and in KSOS and the SCOMP, where they were called Non-Kernel Security-Related processes (NKSR). In all cases, their direct verification against the constraints of information flow analysis was impossible.

The Bell-LaPadula models had provided for the notion of *trusted subjects* whose functionality required transferring information between classified containers in apparent violation to the *-property. Global processes were less obviously in this class. Isolating such processes to operate in less privileged domains only led to additional context-switching inefficiencies, as nothing could be done directly to ensure that their use would not compromise security. This observation led to two *dénouements*:

1. It was concluded that because of the uncertainties of discretionary access controls, the potential exploitation of covert channels in multilevel systems, and the nettlesome questions of global process efficiencies, a system could no longer be called *secure* but would henceforth be called *trusted*;
2. The term security kernel was scrapped in favor of the neologistic term trusted computing base (TCB).

As was to be seen in the sequel, 'TCB' was a vaguely-defined term, and its adaptation as a concept resulted in abandoning the third requirement of the Anderson Study, conceptual simplicity of the RVM.

2.6.2. Distributed mediation, capabilities, PSOS, and Gnosis. Still, the prejudice against the centralized security kernel concept manifested itself in an altogether different way. It was argued that automated formal code verification (or mechanical "proof of correctness") was closer to becoming available, and soon all operating system code – and then hardware design and implementation – correctness could be established as mathematical fact. Thus, it would be possible to include all required security checking as part of each system module or function, thereby eliminating needless access-checking function calls and their costly context switching. Moreover, there would be no separation of call from function, and hence no need for the access-checking functions to derive or establish the relevant context of the requested operation in concert with the semantics of the application.

And so, a movement gained momentum to design and field systems structured along the lines of distributed mediation and that had *no distinct security perimeter* other than the [most] privileged (or most primitive) part of the operating system itself. The first such research study, the capability-based Provably Secure Operating System (PSOS) [19] project yielded:

- A methodology for the design, implementation, and proof of properties of large computing systems
- The design of a secure operating system using this methodology
- The security properties to be proven about the system
- Formal verification methods and tools that came to be known as the Hierarchical Development Methodology (HDM) and the formal specification language SPECIAL
- Considerations for implementing such a system, and
- An approach to monitoring security and performance.

PSOS was rigorously decomposed into a hierarchical specification that had no upward functional- or data-dependencies. The unique protection mechanism was a *capability*, a form of

unforgeable, immutable token, possession of which granted a set of specific access rights to the object to which it was linked. The PSOS concept yielded considerable new research, but left open the question of how a secure system is to be initially configured, how the first capability was to be created, and how one could algorithmically examine a capability distribution and determine whether or not a system was in a secure state. In addition, there were no efficient means of determining which users possessed capabilities to which objects. Despite the open questions, it was asserted that PSOS and its proven design could implement a secure multilevel operating system.

Norm Hardy, Charlie Landau and Bill Frantz designed the Great New Operating System In the Sky (GNOSIS) [12] while at Tymshare, Inc. GNOSIS, unlike PSOS, was commercially developed and implemented a capability-based time sharing environment similar to that of VM/370's Cambridge Monitor System (CMS) interface. Questions similar to those raised in PSOS remained to be answered in GNOSIS and its successor system KeyKOS.

2.6.3. Lee Panel, NBS 1978. The National Bureau of Standards organized an invitational workshop on standards for computer security and audit. One of its panels focused on standardizing the assessment of security controls in processors, operating systems and nearby peripherals. This panel was chaired by Ted Lee, with panelists Peter Neumann, Gerry Popek, Pete Tasker, Steve Walker, and Clark Weissman [15].

The overall set of metrics divided into four aspects of assurance features and four of protection mechanism. These were displayed as sectors of a set of concentric circles wherein the center circle represented Null Confidence, and containing circles exhibited greater assured protections.

The decompositions were:

1. **ASSURANCE FEATURES**
   - **a. Hardware**
     - i. Software Checks
     - ii. Hardware Fault Detection
     - iii. Design Correctness Formally Verified
     - iv. Fault Tolerant Hardware
   - **b. Software**
     - i. Formal Design Specifications
     - ii. Proven Design Specifications
     - iii. Design Correctness Formally Verified
     - iv. Verified Implementation
   - **c. Development and Testing**
     - i. Penetration Exercise
     - ii. Modern Programming Practices
     - iii. Automated Testing
   - **d. Operation and Maintenance**
     - i. Configuration Management
     - ii. Reverification Aids
     - iii. Read-Only Memory
2. **PROTECTION MECHANISM**
   - **a. Prevention**
     - i. Data Security Enforcement
     - ii. System Integrity
     - iii. Collusion Enforcement
     - iv. Sophisticated Threat (Denial of Service)
   - **b. Detection**
     - i. Audit Recording
     - ii. Security Officer Aids
     - iii. Detection Analysis
   - **c. Authorization Granularity**
     - i. Physical Devices
     - ii. Logical Devices
     - iii. Data Values
   - **d. Policy Interface**
     - i. Passwords
     - ii. Labels and Access Control Lists
     - iii. Security Administration Tools

These levels within these eight sectors were not directly comparable as requirements. Rather, they illustrated growing degrees of confidence in a system's security that would be gained along each of the measures as additional requirements were satisfied moving outwards along the sector's axis from the Null Confidence center. No evaluation methodology was proposed.

2.6.4. Air Force Summer Study. Following the Miami workshop, a month-long Air Force Summer Study in Computer Security was conducted at the Draper Labs in Cambridge, Massachusetts. Evaluation criteria and methods were discussed at the Summer Study, along with additional topics in database security, network security, the utility of formal methods and other assurance techniques. The Summer Study attracted the active participation of security researchers, developers and practitioners from the United States, Canada, the United Kingdom, and Germany. Although much of the Summer Study included status reports on a variety of projects, it was mostly conducted as a workshop in which ideas and proposals were voiced and discussed at length.

Several spirited discussions raised controversies that are yet to be resolved. These included: whether it is possible to verify the security of a system built of composed subsystems; whether it is possible to build a secure multilevel database management system that offers "full functionality"; and whether it is possible to produce a "proof of correctness" for a system that will

be accepted as proof of security among the mathematically sophisticated community [11]. The database management security presentations and discussions showed major problems from the use of inference against the use of formulary-like data-dependant access control policies. Dennis Tsichritsis presented a damning indictment against least privilege multilevel database management systems, such as the Hinke-Schaefer model maligning them as "strait-jacket DBMS".

Participants in the evaluation criteria discussion included Jack Adams (IBM); H.O. Lubbes (NRL); Pete Tasker, Stan Ames and Grace Nibaldi (MITRE); Christian Jahl (IABG, Germany); Clark Weissman and me (SDC). The results of this set of discussions were ripe for refinement.

2.6.5. The Nibaldi Report, 1979. Steve Walker, now in the Office of the Secretary of Defense for C^3I, tasked MITRE to elaborate on the Lee Panel's report's Security Metric. Grace Nibaldi produced a MITRE technical report [20] in October 1979 in which seven levels of protection were stated. These were:

0. **No Protection**: where there is no basis for confidence in the system's ability to protect information.
1. **Limited Controlled Sharing**: where recognition of some attempt to control access is given, but only limited confidence in the viability of the controls is indicated.
2. **Extensive Mandatory Security**: where minimal requirements on the protection policy must be satisfied; assurance is derived primarily from attention to protection during the system design and extensive testing.
3. **Structured Protection Mechanism**: where additional confidence is gained through methodical construction of the protection-related software components of the operating system (i.e., the TCB implementation), and modern programming techniques.
4. **Design Correspondence**: formal methods are employed to verify the design of the TCB implementation.
5. **Implementation Correspondence**: where formal methods are employed to verify the software implementation of the design.
6. **Object Code Analysis**: where object code is analyzed and the hardware support is strengthened.

Significantly, the Nibaldi report opens with a 15-page tutorial section describing and going into issues of "primary factors" (policy, mechanism, assurance) and "supporting factors" such as ease of use and overall functionality. Much of the lore characterizing the R&D community's state-of-the-art is presented in this section, which includes nearly a page on denial of service considerations. Additionally, confinement, detection, coding and design methodologies, auditing, and recovery are presented in an overview. The Reference Monitor Concept is not enunciated, and the term *TCB* is used in lieu of *security kernel* throughout the report, and thus there are no explicit requirements for minimization of either size or complexity of the protection mechanism at the higher assurance levels.

Each of the six protection levels subsumed the requirements of the prior level and had to satisfy general criteria characterizing attributes of Protection Policy, Specific Protection Mechanisms, and Assurance. In addition, a section was provided to address the "residual risk" associated with a recommended operational environment deemed appropriate for the system. The specific criteria are presented in descriptive, rather than prescriptive, terms based on the tutorial's content. For example, the treatment of storage channels from Level 4 reads:

> A specific requirement of the system is that it be able to audit the use of storage channels. These channels might be detected as a result of the formal verification techniques or by penetration analysis; however, they may not be easily removed without affecting the system in an adverse way. By imposing restrictions on the way resources are being shared, the system may no longer be allowed to use an optimal algorithm for resource utilization. The use of such channels can be detected with auditing mechanisms, and the information obtained from the auditing mechanisms can be analyzed later to find the source and seriousness of the channels' exploitation.

The Nibaldi proposal included the then unachievable Level 6 criteria, which offered:

> …a degree of confidence which is only imaginable from today's technology. Any threats at this level would be a result of highly improbable hardware errors, or, more likely, a failure in the personnel, administrative, physical, or communications security provisions…. At level 6, formal analysis of the object code produced by the compiler is required. 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….

From the Pentagon, Steve Walker had put together a few assorted teams of experts from academia and industry with the intention of providing assistance to vendors who were interested in developing trusted products that could be used by the DoD. Ted Lee and I participated in several of these efforts along with a seasoned group of security practitioners like Pete Tasker, John Woodward, Anne-Marie Claybrook, Susan Rajunas, and Grace Nibaldi from MITRE Bedford. Under nondisclosure agreements, the teams were also performing *ad hoc* product "evaluations" using the Nibaldi draft criteria.

One of the products under consideration didn't appear to fit Nibaldi's working criteria at all well. This was Tymshare Corporation's capability-based Gnosis system. Susan Rajunas, who had been leading the evaluation, was particularly articulate about the Gnosis design and strength of its mechanisms. But there were numerous open questions about the definition of secure state, of how one attained an initial secure state, how individual accountability could be established in an environment where capabilities were inscrutable, and where possession of a capability could conceivably be used by a Trojan horse. Rajunas was funded to assemble a workshop to investigate assembling a set of interpreted criteria for evaluating a trusted capability base operating system.

I requested that Earl Boebert, who led a project to develop a system based on PSOS, the Secure Ada Target (SAT), write a paper for an NCSC Conference showing that multilevel security confinement could not be assured in a pure capability based operating system.[4] A year earlier, Paul Karger had written a paper [29] on a design that augmented capabilities to overcome such intrinsic shortcomings.

About this time, I heard Butler Lampson's observation: "Capability based systems are the way of the future—and they always will be."

3. TCSEC publication

In February 1981, the Department of Defense Computer Security Evaluation Center (DOD/CSEC) was authorized under Directive 5215.1 and the DoD Computer Security Center (DOD/CSC) was formed at the National Security Agency (NSA) in July of that year. Melville H. Klein and Colonel Roger Schell were designated as Director and Deputy Director. The Center grew from the DoD's Computer Security Initiative. The DoD was aware of the growing cost of procuring and maintaining its special-purpose computer systems—systems that became increasingly difficult to maintain as manufacturers discontinued hardware lines and developers moved on to new projects. Over time, internals knowledge about these systems evaporated and, critical as they may have been to the national security, they became fragile and unreliable. Hence, the Center was formed to implement the strategy of encouraging the widespread availability of trusted products produced and maintained by system vendors. These trusted products would be evaluated gratis by the Center and placed on an Evaluated Products List that could be used by vendors in their advertising and by procurement officers in their purchase specifications.

When I arrived as Chief Scientist early in April 1982, Dan Edwards was directing the Standards and Products organization, with Mario Tinto responsible for product evaluations; Steve Barnett directed the Application Certifications organization.

3.1. The evolution of TCSEC drafts

Prior to my arrival at the Center, work had begun on transforming the Nibaldi proposals into draft evaluation criteria. Paragraphs characterizing selected requirements had been written, and there was general agreement as to a general feeling of what was salutary and what was lacking among mechanisms and assurance techniques. But, at best, there were more open technical issues than resolved ones.

I had been unofficially involved in this UNCLASSIFIED process while my clearance was being finalized. Our writing group's principal members were: Roger Schell, Dan Edwards, Mario Tinto, Jim Anderson, Pete Tasker, Grace Nibaldi, and myself.

3.1.1. First draft: powder blue. On my arrival at the Center, we still lacked a unified document. The previous week's workshop on capability based systems had failed to shed adequate light on how their evaluation criteria could be structured. I received the welcomed news that Sheila Brand was going to be joining the Center, probably in May to lead the Standards organization. I also learnt of a controversy articulated by the MITRE Corporation over how evaluation criteria should be structured. The Nibaldi proposal consisted of a strictly-ordered set of seven requirement-subsuming evaluation classes ranging from no protection through attributes beyond the state of the art. While Nibaldi's work could be accepted as a refinement of the Lee panel's results, nearly 2-1/2 years had passed since its publication, and critics observed that the Lee panel had not prescribed a strict hierarchy of fully subsuming levels.

A MITRE report written by Anne-Marie Claybrook proposed that products be evaluated against criteria drawn, by the product developer, from sets of policy, mechanism and assurance requirements that are

perceived as desirable for an application. This was described as following a "Chinese-Menu" approach. There were many strong adherents to this position who argued that not all requirements for, say Nibaldi Level 4 need apply to a multilevel transaction-only system that is to operate in an environment that eliminates all remote user capabilities. This proposal is not much different from the contemporary trend that grew from the German IT Security Criteria, the UK's ITSEC, and finally that of drawing "protection profiles" from the Common Criteria. Many strong arguments were presented on the value of flexibility that would come from this approach, and the ability to tailor a system to its envisioned use.

However, others argued that it would lead only to confusion. Roger Schell was the strongest advocate of preserving Nibaldi's structure of a small number of well-ordered levels. His rationale was based on his years of experience in the DoD procurement process. Put simply, procurement officers have expertise in specifying purchases, not in performing comparative assessments of which competing technical ideas best fit a specific application. He argued that a procurement officer needed to be presented with a strictly ordered set of product characterizations that are keyed to the security requirements under which an application would operate. Thus, a procured system that would have to operate in a remote-user environment with CONFIDENTIAL through TOP SECRET data would minimally have to have attained a specified (or higher) degree of "certifiability". This question could be readily answered by having a frequently-updated published Evaluated Products List from which to select and qualify compliant trusted products.

The eloquence of Schell's argument settled the dispute. We agreed rapidly to argue requirement-by-requirement among ourselves and to fit together seven ordered levels of trusted systems criteria, plus one level for products that failed to satisfy the requirements of any level. To maintain our focus, Schell also suggested that we select existing products or developments to characterize each of the levels we were defining, stressing the value of having worked engineering examples for each evaluation class.

Finally, we agreed that we wanted to limit the possibility of new intermediate evaluation classes being introduced over time. We were particularly adamant on there being no new classes added to water down the minimum requirements for each of the obvious divisions we had settled on. This called for a labeling scheme that established minima for each of the four major divisions we had envisioned: Minimal Protection, Discretionary Protection; Mandatory Protection; and Verified Protection. Dan Edwards solved the problem by assigning evaluation classes a digraph rating, patterned on bond ratings wherein the divisions were ordered alphabetically with D<C<B<A and the classes within a division were numbered in increasing order of strength with a natural number. This was done to preclude the introduction of a class less restrictive than the entry level defined for the Division.

The initial class in each Division was intended to be achievable by evolving a product that qualified for the highest class in the preceding Division. This was not expected to be easy, but we believed it would be feasible. Hence, Class <C1> required principally the addition of discretionary access controls to a seasoned commercial product that could separate its users from the operating system's domain of execution. A <B1> system, known to us as "MAC with Training Wheels" would essentially be a <C2> product that performed mediation and labeling on a defined subset of its users and objects; and an <A1> product would essentially be a <B3> product having a formal verified design.

The identified classes and identified worked examples were, as drawn from our Final Draft [8]:

> Class <D>: Common Practice. This evaluation class is reserved for systems that have been evaluated and failed to meet the requirements of a higher class.
>
> Class <C1>: Discretionary Security Protection. This class of systems has some form of mechanism providing individual user authentication, provides nominal discretionary access control among users and data, and is self-protecting. Candidate: UNIX
>
> Class <C2>: Controlled Access Protection. Systems in this class have at least discretionary access control enforced on users. The requirements of this class may be met through the use of [a] security add-on or security overlay package. The principal distinction between Class <C2> and Class <C1> is that Class <C2> requires individual accountability and security-event auditing features. Candidate: RACF, ACF2 and Secure add-on packages for IBM's MVS/370.
>
> Class <B1>: Labeled Security Protection. Class <B1> systems provide mandatory security access control. Discretionary access control suitable for DoD need-to-know protection is provided. The notion of a well-defined TCB appears even though a formal security model is not required. Security marking of data is required. Any serious flaws identified by penetration testing have been removed. Candidate: GCOS, security retrofitted third-generation operating system
>
> Class <B2>: Structured Protection. In this class of systems, mandatory security is extended to all objects visible outside the TCB, and information flow control and confinement channels are addressed. A model of the security policy enforced by the TCB is required. The TCB exhibits deliberate security structuring, and

stringent configuration management controls are imposed. Authentication mechanisms are strengthened, and features to support trusted facility management are provided. Candidate: the commercial version of Multics incorporating the Access Isolation Mechanism (AIM) with class <B3> attention to storage channels.

Class <B3>: Security Domains. The TCB of this class of system supports a defined security model. The principle of least privilege is pervasively applied within the TCB in this class of system. All security relevant code is clearly identified and the TCB is structured to separate security relevant and non-security relevant code into different domains. Evidence that the TCB satisfies the reference monitor requirements is required. Hardware protection or some other form of very convincing argument is used to show that any unexpected software event in a non-security relevant domain cannot affect the software in a security relevant domain. Candidate: the redesign of Project Guardian Multics.

Class <A1>: Verified Design. The main characteristics of systems in this class are that a formal model exists, the top-level user interface of the TCB has been formally specified, and the TCB has been designed and developed in conjunction with formal verification techniques and verified to satisfy the model. Additional (non-verification) evidence is required to show that the TCB fulfills the reference monitor requirements. The facilities and procedures for trusted distribution become requirements here. In May 1982, this class of system appears to be just at the state-of-the-art for practical implementations. Candidate: Kernelized VM/370 (KVM), Kernelized Secure Operating System (KSOS), Honeywell SCOMP, and the Air Force's SACDIN.

Class <A2>: Verified Implementation. Systems in class <A2> use a formal machine checkable methodology to assure that the actual implementation of the system conforms to the verified top-level specifications. Formal hardware and firmware design and analysis become important, to demonstrate that the reference monitor requirements are met, as well as other development environment attributes (*e.g.*, compilers). In May 1982, this class of system appears to be well beyond the state-of-the-art for practical implementations. Candidate: [none was specified].

The writing style in this draft was imprecise and descriptive. The draft explicitly stated that the requirements were intended to apply both to general-purpose and to embedded systems. As an example of the draft criteria's requirement wording style, the storage channel requirement cited from Nibaldi's Level 4 can be found scattered among requirements in Class <B2>:

> Flow Control The TCB of the class <B2> system enforces information flow security (confinement). Information flow security is applied to all objects that are directly or indirectly visible outside the TCB. Control objects (e.g., number of free disk pages) and TCB responses (e.g., out of space) are included as well as objects normally thought of as storage objects.
>
> Audit See class <B1>. In addition, mechanisms are provided to record the use of channels that have been shown to have an exploitable bandwidth greater than some clearly identified small bandwidth.

Because of controversy in the community, the Draft carefully avoided explicitly requiring the implementation of a security kernel. However, even without the wording, it was a clear objective that Class <B3> systems implemented a security kernel in the full minimal sense of the Anderson Study.

3.1.2. Community response. Well, we announced that the first draft would be presented at the Computer Security Initiative Seminar, and we got a packed house. We naïvely assumed that we were close to getting it right, and we requested written comments from the participants. We set a tight review and publication schedule, because developers wanted to have firm criteria to work from as soon as possible. So the Draft invited a first round of comments with a deadline of 1 July 1982, promising a second draft that responded to these comments by 1 August. Anticipated comments received by 1 September would be considered and the final TCSEC would be published in October 1982.

We received a massive response, filling file-drawers with industry and government comments! Sheila Brand, newly arrived at the Center, was buried in correspondence and in requests for visits to discuss the Draft. There were complaints over the Draft's imprecise language as well as its organization. For the most part, the cards and letters were politely supportive of our effort, and offered constructively stated reservations. We received several well reasoned proposals for wording along with rationales. In the Draft, the Criteria came last, following a prefatory introduction and rationale. A majority of reviewers stated they did not want to wade through our prose to get to the requirements, which they wanted to come first.

Potential users wanted each evaluation class's requirements to be self-contained. This would permit the requirements for a single evaluation class to be extracted and published in a procurement specification. This added to the bulk of the TCSEC, but removed

internal cross-referencing. Additional requests came in for a glossary of terms, for guidelines on testing, on covert channels, and on configuring MAC features. The issue of giving extra credit for providing features or assurances beyond those required for a specific access class was also raised.[16]

The group working most closely on the drafts and comments included, in addition to Sheila Brand and myself, close interaction with: Grace [Nibaldi] Hammonds, Pete Tasker, Dan Edwards, Mario Tinto, Roger Schell, Jim Anderson, Ted Lee, Steve Lipner, Clark Weissman, Steve Walker, Larry Noble, Jim Studer, Gene Epperly, Jeff Makey, Warren Shadle, and me. David Bell provided strong contributions after he joined the Center's research organization. We later came to realize that no practitioners of contractual law were involved in the writing group's activities or in the formal review process.

Of necessity, our schedule slipped dramatically and continually. Comments often turned into lobbying. Developers wanted to ensure that their products would get the best possible earned rating and helped greatly to eliminate impossible or ambiguously-worded requirements. A series of drafts was published. These presented numbered lines and employed **bold faced** insertions and ~~strikeout deletions~~ to help reviewers make it through the growing document's evolution. Drafts were published on 15 November 1982 (white cover), 15 January 1983, and a final Draft (in an ugly Olive Drab cover) on 27 January 1983.

Brand maintained a growing file of comments and how each was accommodated. Rejected comments were noted as such and, depending on our perception of the source's credentials, justification for the decision was written into the file and sent to the reviewer. Based on the number of received comments, Brand insisted that references to *dominance* in the security lattice be replaced with explicit wording on the rules for reading and modifying objects. This proved to be a decision that aided readers unschooled in lattice theory and which removed ambiguities the authors did not perceive at the time.

The definition of Class (A2) persisted into January 1983. However, it was finally removed because it made no sense to include a class whose requirements were defined to be beyond the state of the art and that would have to be redefined at some point if code-level formal verification technology ever advanced to a practical state. It was replaced by a page entitled "Beyond Class (A1)" that only gave a characterization of what might be required for higher assurance.

The Final Draft begat a broader stream of comments. Sheila Brand made a management decision on resolving the last of the remaining open issues and advised Mel Klein that the time had come to publish CSC-STD-001-83, The Department of Defense Trusted Computer System Evaluation Criteria. This appeared, in a bright orange cover, with a Forward by Klein, on 15 August 1983. The new "Orange Book" weighed in at 117 pages.

Several trusted products were being evaluated against drafts of the Criteria during this period. At first, vendors insisted that the draft version be identified explicitly in their contract with the Center and identified in the Final Evaluation Report. However, the vendors later realized that evaluation against any but the final published standard would be a mistake because to do so would be to give the appearance of failing to meet one or more of the "real" requirements. So in reality, products were being evaluated against a moving target during the period from early 1982 through 15 August 1983.

3.2. Lack of deeper understanding

We thought we understood what we were writing. We also thought the community would understand what we had written – or at least what we *intended* to have written. That turned out not to be the case. Indeed, for all our belief that we were writing with precision, only experience could show that we weren't.

We were fully dedicated to not introducing faddish requirements and tried to justify each on the basis of an identifiable need and technical justification. We paid considerable attention to the entry-level criteria for each division, intending to ensure that it would be achievable from a well-designed product at the top of the predecessor division. We also tried to limit the nature of the hardest technical challenge in progressing to evaluation classes by constraining our desire to add additional requirements because of strong advocacy for them. This attempt at following a discipline sometimes led to an appearance of random placement of some requirements.

An example of this is the placement of the first change in the DAC requirement after Class (C2) (the angle brackets morphed into parentheses after the Powder Blue Draft). This requirement is introduced at Class (B3)(!) and specifies (a) that there be access control lists, (b) that there be support for both individual and group access controls, and finally (c) that "it shall be possible to specify a list of named individuals and a list of groups of named individuals

[16] Some of us opposed this practice on the grounds that inclusion of some features without supportive assurances would provide a vendor with meaningless hype for advertising a false sense of security. The majority prevailed on this issue.

for which no access to the object is to be given." Why at B3? Well, Dan Edwards produced the following argument. By definition the requirement could not be introduced at B1, because the rule for an entry-level product would be just the addition of labeling of a defined subset of named subjects and objects. It didn't make sense to add the requirement at B2 because of the difficulty inherent in meeting the requirement to control access between *all* subjects and *all* objects and to address the more important covert channel issue and strong penetration-resistance requirements. Given that Class (A1) added the difficult formal specification and formal covert channel analysis requirements which were to be the only change from B3, we had the choice of adding the requirement at C2 or B3 or not to add it at all. We decided that audit was a hard enough addition to the C1 requirement, and that narrowed the choice to B3 or not at all.

3.2.1. DAC algebra. We found out later that we made the wrong choice for Negative Access Control Lists (or NACLs) as the requirement came to be called. We either should not have added them or we should have proposed a model showing the relationship between ACLs and NACLs and submitted it to the community for comment. In fact, we did not realize at the time that there was an issue present for us to disagree on.

The problems we did not identify focused on apparent contradictions between listings and determining which takes precedence. *E.g.*, if a name appears both on a NACL *and* on an ACL, which takes priority? What if the name is in a group that appears on a NACL but the user's name is explicit on an ACL for the object? What if the user *created* the object, gave herself full rights to the object, and creates its NACL as well, but a system administrator (not knowing of the NACL's existence or its organization) then places this user in a group that is on that particular NACL?

This problem came up a couple of times some years after publication of the final DoD Standard 5200.28-STD version of the TCSEC had been adapted. To my knowledge, the issues were never satisfactorily resolved.

3.2.2. Questions of "high-assurance" DAC. By definition, discretionary access control conveys rights between unlabeled (same access class) subjects and objects. Consider a simple Trojan horse attack. If a subject S can read A and modify B, then S can copy the contents of A into B (if this is consistent with the semantics of B, else S can encode A's contents such that they are compatible with B's semantics). But what if there exists a subject S´ who is explicitly listed on a NACL for A or on an ACL for B? Has S violated the access control policy? Or is the system required to modify the access controls on B such that a NACL is created on B that includes S´? If S has control access to B, can S remove that NACL?

Now, under the MAC rules of the *-property, copying the contents of A into B is prohibited unless A and B are at compatible security levels. But is the above transaction a violation of DAC that must be closed at the B3 level? Is it an "obvious" flaw that must be closed? Well… no! Because it is an intrinsic property of information flow within an access class, and it cannot be eliminated.

But this leads to a hard question that is sidestepped in the TCSEC. When the derivable principles of information theory and a security policy are in conflict, which must take precedence? Similarly, when derivable consequences of two aspects of a security policy are in conflict, which much take precedence? To my shock, I learned that some evaluators interpreted only to the explicit wording of the TCSEC, however impossible they might be in context, to take precedence over the mathematical properties of information science.

The silence of the TCSEC on such points has led new practitioners into making assumptions that an A1 system is "more secure" for single-level applications than a C2 system. This is true (because of the ability of a B2 or higher security architecture to fend off large classes of penetration attacks that exploit architectural or implementation flaws). But the full suite of structural and formal A1 assurances has no effect on information confinement within that single access class. Indeed, without the TCB integrity and recovery requirements, a single-level DAC-only implementation of an A1 architecture could fall victim to many attacks that are common in contemporary virus- or worm-bearing e-mails.

3.2.3. Failed "worked" examples. The "worked examples" identified in the Powder Blue Draft got forgotten along the way. Indeed, RACF, the prototype for C2 audit, failed to meet its own defining requirement because IBM understood that its customers wanted to have the ability to turn off audit to improve performance in some system environments. The consequence was that RACF became the only product ever to be "awarded" a C1 rating. Bob Brotzman, then Director of the NCSC, and I were more embarrassed over the situation than IBM's Tom Russell, to whom I presented the C1 Certificate.

A trusted UNIX candidate for B1 or B2 would have been automatically disqualified because its {Self, Group, World} form of access control would fail to meet even the C2 requirement on groups or named

individuals unless it limited its number of possible users to a very small number.

Though satisfying all the security architecture and formal verification requirements for A1, worked examples KVM/370 and another A1 candidate trusted VMM would also have failed. They were deficient in the C2 individual accountability requirement and C1 individual discretionary access control requirement because the security kernel within the virtual machine monitor could not see how accesses were controlled within multiuser virtual machines. Their evaluation team wanted to insist that every virtual machine have no more than a single human user. These products would therefore have received a D rating, despite their architectures and assurances.

3.2.4. Imprecise language. Other forms of imprecise wordings plagued evaluators and developers alike. Long position papers were written to address the meaning of requirements such as the C2 System Architecture requirement: "…Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. **The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements**." Seemingly, questions were invented that put evaluations on hold for indefinite periods. For example, "what are the rules for access by a subject not defined to be under control of the TCB to an object defined to be under control of the TCB?" "Should the TCB be required to audit all accesses by subjects not under the TCB's control?" Decisions were postponed for months at a time, at tremendous cost to trusted product developers and evaluators, until there was near unanimous agreement over interpretive topics as absurd as these.

Similar problems showed up in the evaluations of Multics (AIM) for B2 and the SCOMP for A1. True, technical problems were detected and corrected in these product evaluations, but tremendous time was lost in attempts to resolve imprecisely stated evaluation requirements.

3.3. Becoming a DoD standard

Shortly after the TCSEC's August publication, a movement was initiated within the Computer Security Center to promulgate it as a DoD Standard. Doing so was necessary in order to legitimize the Evaluated Products List as part of the certification, accreditation, and procurement processes.

This turned into a monumental effort. In order to achieve this goal, the TCSEC needed to be acceptable to the DoD's many departments, services and agencies. Copies of CSC-STD-001 were distributed for official comment, and official comments came in a near tidal wave.

Some agencies objected that promulgation of the TCSEC would officially stifle research and experimentation. Others were concerned over wording that could have an adverse effect on their embedded system applications. Portions of the intelligence community objected that the TCSEC's policies did not directly support their multilevel classification and caveat system of compartments, categories, warning notices, and other dissemination and selective-declassification controls. Because B1 was established as the entry level into Division B, one agency insisted on the inclusion of a new class, C3, that would support their policy needs.

Many of us spent a summer and fall in sometimes heated meetings with executives in the Pentagon and at various agencies. Sheila Brand, Mario Tinto, Grant Wagner, and I repeatedly found ourselves having to explain and defend such TCSEC requirements as closing high-bandwidth covert channel discovered in analysis on the one hand and the lack of explicit requirements to build in protections against deadlocks and other denials of service.[17] This is not to suggest that the services and agencies concerns were trivial; many were very sophisticated and few were frivolous. Many comments came from a lack of knowledge or understanding of the vocabulary of computer security technology, understandable since this was largely an arcane field in the 1980s.

More than one agency actively wanted to make the TCSEC optional since there was "no validated computer security threat" and it made no sense to build in costly defenses where there was no validated requirement. But the discussions were very time-consuming and important to all parties, and policy coördination is an intricate prolonged process.

Finally, and as a tribute to Sheila Brand's skills and dedication, Assistant Secretary of Defense (C^3I) Donald Latham signed off on the TCSEC as DoD 5200.28-STD on 26 December 1985.

4. TCSEC reconsidered

As discussed above, products and systems were under evaluation while the TCSEC was going through its drafts and coördination reviews. Many problems were discovered. Some were fixed prior to the 15 August 1983 Standard's publication. These

[17] In addition to robustness largely being a research area, there also were demonstrations that popular reliability algorithms generally introduce exploitable covert channels.

modifications required consensus among the Center's management, senior staff, researchers and evaluators (including contractual support staff from MITRE and Aerospace Corporation and other consultants). However, no changes were made unilaterally (*i.e.,* based only on our own learning experiences) after 15 August 1983. This is because of the Center's top management's decision that we needed to solidify agreement among the services and agencies on existing wording rather than for us to re-open issues that could delay the TCSEC becoming a DoD Standard. Defending criteria we knew to be flawed was a painful and bitter pill many of us had to learn to swallow.

4.1. The education challenge

We had no way of knowing then that not enough of what we thought was reduced to practice *was* reduced to practice. It took publishing the TCSEC and trying to perform evaluations for us to learn this. Simple terms like *module* had very different meanings in different communities and among professionals who learned their trade in different decades. Those of us who had extensive *decade*ant (sic) experience penetrating systems had a very different appreciation for the importance of certain structuring approaches than did staff (or developers) who had never defeated a protection mechanism or written a real-time I/O driver.

There weren't many professionals in the United States who had acquired years of experience working in computer security technology. Many were employed as researchers in universities, in a small number of industrial or government defense research organizations. A few were consultants. This meant that the NCSC needed to institute an intense training program for its evaluators and for developers in industry.

We had intended that our evaluation teams would show vendors creative approaches to solving or working around architectural problems uncovered during an developmental evaluation. For two reasons, this happened rarely. One reason related to the legal context of a representative of the Government advising *vs* giving contractual direction to a contractor—even though the vendors under evaluation were not contractors.

A second problem of advice-giving was more complex. Because the NCSC and its evaluation support contractors hired young graduate computing science graduates, we periodically received complaints from vendors that our evaluators had less experience and expertise than the developers whose efforts we were evaluating. They viewed our evaluators as being too "Ivory Tower" and "wet behind the ears" to give them any useful pragmatic advice. So, yet another challenge for our technical senior staff arose, this time requiring diplomatic crisis management skills. We needed to find an acceptable means of guiding the vendor and evaluator toward a viable solution without making either more defensive than they often were. This was a skill I have never been accused of!

4.2. Send in the lawyers

Early on, Dan Edwards and Mario Tinto decided that the evaluators would need to produce extensive documentation of technical decisions they made during the evaluation process. These decisions would be likened to judicial *case law*.

For a while, various authors of the TCSEC were asked what a given wording or term meant. It was quickly discovered that we were individually not often in close agreement on these concepts, even though we did not think there was a question on the meaning when we were writing the criteria. So evaluators soon learned that something more than the memories of the authors was needed if evaluations were to be performed consistently.

Because of the TCSEC's widely distributed ambiguities and the unanticipatedly creative nature of developers, the Center's evaluators were almost immediately faced with the need to make reasoned and defensible decisions on how TCSEC requirements would be applied to a given situation. This introduced an intricate and time-consuming process known as Criteria Interpretation.

It made good sense for interpretations to be considered carefully prior to telling a vendor of a decision. But the process that became part of NCSC-lore made *slow* look fast in comparison. The average interpretation process went through a number of proposal, review, comment, revision, review and publication phases that together took weeks to months to years. The effect of this process on product developers was devastating. Often, a product development hinged on making a critical decision to steer the architecture in one direction on another. A wrong decision would be extremely costly to correct later. But evaluation teams could furnish no advice (or at least no advice that might not later be refuted) until the interpretation process terminated.

There was one other unanticipated consequence of interpretations being treated as case law. This became known as *criteria creep*. It seemed that interpretations always added new requirements rather than simply clarifying existing ones. This practice effectively extended the criteria that had to be satisfied by on-

going evaluations, and it added significantly to the cost and time required to complete the evaluation process.

The result was very painful. I, and other seniors in the Center, received angry communications from vendors demanding a speedy decision. Evaluators resented any intervention on our part, since we lacked the context from which the evaluators were working—and they did not want to have their authority undercut.

In at least two cases, those of an A1 and a B2 product evaluation, I took it upon myself to coerce NCSC senior management to break a stalemate and to accelerate the decision to grant or deny awarding an evaluation rating to the product. The problem here was that the evaluators had less guidance to work from to decide that they had indeed performed all of the needed validations and need not perform any more; and the vendor needed to see some revenue from their trusted product or to disband their development team. There were also some occasions where the evaluation process was so slow and mired down in interpretations that a vendor killed a project because the hardware had become obsolete.

4.3. Send in the lawyers

Another consequence of lengthy evaluations was that the evaluated product was several maintenance releases behind the commercially available product. The NCSC needed to find a means of evaluating incremental product evolutions without performing a complete re-evaluation of the product. Ultimately, the Ratings Maintenance Program (RAMP) came out of this process for products in the lower evaluation classes. Consensus was not reached during my tenure in the Center on how RAMP could be applied to products evaluated at or above the B2 level.

4.4. Rainbows

As interpretations came out, so also arose requests for guidance on numerous topics ranging from password selection to the algebra of DAC. So Sheila Brand instituted a process of publishing booklets on a range of evaluation-related topics. Even as the TCSEC was being written, the late George Jelen raised the question of how one could determine which evaluation class was required for a given processing environment. His scholarly dissertation [14] provoked many animated discussions inside the Center. It resulted in Roger Schell's introduction of the *risk range* concept and an algorithm for addressing Jelen's question. This was published as one of Brand's Rainbow Series (so-called because each volume's cover had a unique color). Many volumes in the Rainbow series were scholarly and are still valuable.

As use of the TCSEC became more common, additional questions provoked writing additional volumes. The Rainbow series became something of a self-perpetuating institution. Two definitive entries in the Rainbow series, the *Trusted Network Interpretation* (TNI) and the *Trusted Database Interpretation* (TDI) were published to controversy over an intense writing period. Each addressed portions of the problem of how to apply the TCSEC to the construction of a multilevel network or a multilevel database management system (or *vice versa*). Sometimes divisive controversy surrounded publication of each of these. Several of us argued against publication of either on the grounds that the state of knowledge for building multilevel networks and database management systems was far less developed than the theory and techniques behind building monolithic trusted operating systems.

A somewhat disguised worked example of a counterexample to an A1-compliant TNI architecture was published by Schaefer, et al. in [23].

4.5. A1 is the answer; what was the question?

One unanticipated problem showed up as the TCSEC was becoming accepted by the Military departments and agencies. These came from the procurement officers for whom we had insisted on the simple seven-class structure. Simply put, once there was a B2 product on the Evaluated Products List (commercial Multics), a procurement officer balked at the Air Force specifying a requirement for a B2 or equivalent operating system. The procurement officer, on learning that there was essentially no way for any product other than Multics to comply with the solicitation and its time limit, declared the requirement to be anticompetitive. He refused to allow it.

The other unanticipated problem was likely intrinsic to human nature. A large number of procurements and development contracts came out specifying use of an A1 product. At least one demanded an A2 product so that it would not become obsolete! In many cases, the systems would be operating in a closed environment at system-high, so all users were assumed to be cleared for almost all information on the machine. A B1 product would suffice for the application. But the customers for these products wanted the very best system money could buy, and not some inferior system that failed to meet the highest military standards. We found ourselves having to advocate the use of products from lower levels of the EPL (particularly necessary at the time, as there was not yet an entry above B2!).

Finally, there was customer resistance—often intense resistance. Trusted systems were not known for having user friendly interfaces. Consequences of the *-property and simple security condition proved to be confusing and frustrating to new users. This proved to be embarrassing. We found it impossible to influence the Center's Director or most of its office chiefs to use *any* commercial product that was listed on the EPL—including the Center's flagship B2 Dockmaster system (Multics-AIM) or even the C2 Windows-NT during the Center's "C2 by 92" drive.

5. Reflections and lessons learned

Publication of the TCSEC was, in retrospect, an important step in promoting research and development of trusted operating systems. Vendors would resist producing trusted systems unless they would be evaluated against a published, established standard. Beginning in the late 1970s, several funded and commercial efforts were underway to produce trusted systems or tools for the construction/verification of trusted systems. So the TCSEC or something very much like it needed to be published, and publication needed to happen no later than the 1980s.

Unfortunately, there was a shortage of adequately educated and experienced developers of trusted operating systems. While there were several laboratory prototypes, only Multics (AIM) had a developed user community, and Multics was the only robust security product on the open market. Multics provided both multilevel security capabilities and a structured set of advanced integrity controls. But it was not widely available, its hardware base was not as popular as the less costly IBM or DEC mainframes, and its user interface was not as friendly as the increasingly popular, but vulnerable, UNIX.

Unfortunate, also, was the consequence of the lack of experienced trusted system developers who were willing and able to be evaluators. Many wanted to create a product rather than to "look over someone else's shoulder." The lengthy and overly cautious evaluation and interpretation process ended up killing off vendor participation and trusted product development. This was largely because of the uncertainty of the costs and time associated with getting a product evaluated.

We failed to think of asking experienced procurement officers to review our wording, and no one aggressively thought of making sure that we had written a sufficient and complete glossary of technical terms and concepts. Indeed, the TCSEC's glossary was something of an afterthought, and it was not given the careful attention that the main body of the text was given. This oversight was a significant cause of the lengthy interpretation process.

Another significant problem was our neglecting to write down what we considered to be obvious: the fact that we, the principal authors, did not consider all features and assurances to be created equal. In a bad paraphrase of George Orwell, "Some assurances *are* more equal than others." Had we stated, *e.g.*, that individual accountability under DAC is less significant than assured individual accountability under MAC, many bitter and divisive diversions would have been avoided – and possibly more A1 products would have been produced.

I do not question the wisdom of our decision to limit the TCSEC to its seven all-or-nothing classes rather than taking the Chinese-menu approach that was advocated at the time. I think this was the right thing to do. In that sense, I consider the TCSEC to be an improvement over the criteria created afterwards, particularly the swollen and confusing Common Criteria with its extensible myriad of Protection Profiles and I think it is harmful to "roll your own if you don't like what's there." True, this puts the interpretation in front of the evaluation, but it also has the capacity of producing a huge number of slightly different policies or assurances that will be very difficult for sophisticated consumers to compare or accurately comprehend.

I am very much bothered by the way the industry has moved. Today, a generation after the début of the DoD Computer Security Initiative and the publication of the TCSEC, there are essentially no commercially available trusted systems in use offering protection equivalent to a equivalent to a B2 Multics or the A1 M-component GEMSOS. [30]

Instead, there are bloated, untested, feature-laden interoperating untrustworthy less-than-C2 products that are self-penetrating. Their alleged kernels consist of millions of lines of highly privileged code written by teams of people who've never met their coding counterparts. The illusion of system security is provided by software encryption algorithms that can often be coaxed to reveal their keys to a skilled interloper. Add-on security gadgetry in the form of pattern-matching virus scanners and restrictive firewalls belie the vendors' claims of mature security architectures. And, of course, the periodic announcement of urgent several megabyte security patches only emphasizes the tawdry state of today's commercial offerings.

Never has compromising a system been easier! Never have so many effective penetration tools been provided off-the-shelf *by the vendor* to the would-be interloper!

Also, one cannot but comment adversely on the current issue of electronic touch-screen voting systems. In at least one state, Maryland, the only legal way to vote is on a system that uses cryptography for some aspects of secrecy, but which is implemented on a version of Windows CE – a foundation that would not meet the unexacting standards of the TCSEC C1 class. Attacks against Windows operating system variants are common place, and the vendor's flagship C2 systems (NT and 2000) require regular security patching because of Internet Malware, with no one questioning the presence of its gaping Active Desktop and other inviting security vulnerabilities. Several security studies were conducted that identified voting system security flaws, and of these several could be exploited through a prepared attack. The fact that there is no permanent and immutable audit trail and recovery system has been discussed and dismissed by the manufacturer and by the state election board.[18] Most recently, the Maryland court system has dismissed concerns over the machines' security on the grounds that the system is not going to be connected to hackers, need not to withstand "military style attacks," and so where is the security threat? Surely, no one would want to invest expensive technical effort into controlling the results of a national election! O where have we heard these questions before?!

The TCSEC was written and emended by the skilled computer security practitioners of the late 1970s and early 1980s. The derivative criteria, though written by large committees of skilled personnel, reflect the fact that they were written by committee, and with the goal of harmonizing protection philosophies rather than establishing more focused requirements and guidelines.

It is doubtful that any vendor is going to produce a completely new operating system in the current internetworked environment. For commercial viability, it appears that operating systems need to accommodate everything from real-time wireless gaming to play-on-demand multimedia presentations. With technology moving computer usage away from previous trends (*i.e.*, computation and data processing), it appears that a new paradigm is needed for security engineering in today's environment. Back to basics just doesn't seem to be practicable any more.

And one can legitimately ask whether there is yet a perceived, validated security requirement.

[18] One excuse I've seen in print claims that even if there were a need for a secured voting system and/or hard copy backup ballot, there is no standard or Protection Profile for either.

6. Acknowledgement

Many people encouraged and helped with the writing of this paper. I would like to thank Dan Thomsen, LouAnna Notargiacomo, Steve Greenwald, and Ken Olthoff for their continuing encouragement and critiques in taking on this task from the cozy pastures of retirement. In particular, I am particularly indebted to LouAnna, who took extraordinary steps to ensure the paper's timely completion. I received valuable assistance in reconstructing the past from Rich Graubart, Ronda Henning, Paul Karger, Ted Lee, Peter Neumann, Roger Schell, and Tom van Vleck. Thank you, dear friends!

7. References

[1] Abbott, Bob, J. Chin, J. Donnelley, W. Konigsford, S. Tokubo, and D. Webb, "Security Analysis and Enhancements of Computer Operating Systems," Technical Report NBSIR 76-1041, ICET, National Bureau of Standards, 1976.

[2] Anderson, James. P., *Computer Security Planning Study,* Electronic Systems Division, USAF Report ESD-TR-73-51 in two volumes.

[3] Bell, D. Elliott, and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," Tech. Report MTR-2997 Rev 1, MITRE Corp., March 1975.

[4] Boebert, Earl, "On the Inability of an Unmodified Capability Machine to Enforce the *-Property," *Proc. 7th DOD/NBS* Computer Security Conf., 1984.

[5] Brand, Sheila, ed., *Trusted Computer System Evaluation Criteria*, Final Draft, 27 January 1983, 109 pp. as C1-FEB- 83- S3-25366. DoD Computer Security Center.

[6] Corbató, F. J., and V. A. Vyssotsky, "Introduction and Overview of the Multics System", 1965 *Fall Joint Computer Conference.*

[7] Department of Defense, *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, 26 December 1985.

[8] DoD Computer Security Center, *Trusted Computer System Evaluation Criteria*, Draft, 24 May 1982, 43 pp.

[9] DoD Computer Security Center, *Trusted Computer System Evaluation Criteria*, 15 August 1983, 117 pp, as CSC-STD-001-83.

[10] Harrison, M., W. Ruzzo, and J. Ullman, "Protection in Operating Systems,", *Comm. ACM,* vol. 19, no. 8, 1977.

[11] Demillo, R.A., R. J. Lipton, A. J. Perlis, "Social Processes and Proofs of Theorems and Programs," *Comm. ACM,* Vol. 22, No. 5, 1979.

[12] Frantz, Bill, Norm Hardy, Jay Jonekait, Charlie Landau, GNOSIS: A Prototype Operating System for *the 1990's*, Tymshare, Inc., 1979.

[13] Graham, G.S. and P.J. Denning, "Protection – Principles and Practice," *Spring Joint Computer Conference, AFIPS Conf. Proc.*, 1972.

[14] Jelen, George F., *Information Security: an Elusive Goal*, Program on Information Resources Policy, Harvard University Center for Information Policy Research, April 1984.

[15] Lee, Theodore M. P., "Processors, Operating Systems and Nearby Peripherals: A Consensus Report," appearing as Section 8 of Ruthberg, *op. cit.*, 1980.

[16] Lipner, Stephen B., A Comment on the Confinement Problem, Proc. 6th Symp. Operating Systems Principles, 1975

[17] McLean, John, "Reasoning About Security Models," *Proc 1987 IEEE Symp. Security and Privacy*, Apr. 1987.

[18] Millen, Jonathan K., "Security Kernel Validation in Practice," *Comm. ACM,* vol. 19, no. 5 (May 1976), pp. 243-250.

[19] Neumann, Peter, Larry Robinson, Karl Levitt, R.S. Boyer, and A.R. Saxena, "A Provably Secure Operating System: Final Report," Stanford Research Institute Report, June 1975.

[20] Nibaldi, Grace H[ammond], *Proposed Technical Evaluation Criteria for Trusted Computer Systems,* MITRE Report, M-79-225, 25 October 1979.

[21] Ruthberg, Zella, Audit and Evaluation of Computer Security II: System Vulnerabilities and Controls, NBS Special Publication No 500-57, MD78733, April 1980.

[22] Schaefer, Marvin., "Symbol Security Condition Considered Harmful," *Proceedings 1989 IEEE Computer Society Symposium on Security and Privacy,* pp. 20-46, May 1-3, 1989.

[23] Schaefer, Marvin, W. C. Barker, C. P. Pfleeger, "Tea and I: an Allergy," *Proceedings 1989 IEEE Computer Society Symposium on Security and Privacy,* pp. 178-182, May 1-3, 1989.

[24] Schaefer, Marvin, R. R. Linde, *et al.*, "Program Confinement in KVM/370," in *Proc. ACM National Conference,* Seattle, October, 1997.

[25] Vyssotsky, V.A., F. J. Corbató, and R.M. Graham, "Structure of the Multics Supervisor.", *AFIPS Conf Proc.,* vol. 27, part I, 1965.

[26] Walter, K. G, W. Ogden, F. Bradshaw, S. Ames, and D. Shumway, "Primitive Models for Computer Security, ESD-TR-74-117, Air Force ESD, Hanscom AFB, Mass, 1974.

[27] Ware, Willis H., ed. Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security, R-609-1, reissued by the RAND Corporation, 1979

[28] Weissman, Clark. Security Controls in the ADEPT-50 Time Sharing System. In *AFIPS Conference Proceedings*, volume 35, New Jersey, 1969.

[29] Karger, P.A. and A.J. Herbert. "An Augmented Capability Architecture to Support Lattice Security and Traceability of Access". in *Proceedings of the 1984 Symposium on Security and Privacy,*. pp. 2-12, 29 April - 2 May 1984.

[30] National Security Agency Trusted Product Evaluation Report, Gemini Trusted Network Processor (GTNP), available at http://www.radium.ncsc.mil/-tpep/epl/entries/CSC-EPL-94-008.html