Proceedings of the 5th IEEE Systems, Man and Cybernetics
Information Assurance Workshop, United States
Military Academy, West Point, NY, 10–11 June 2004, pp. 109-115

The Trusted Computing Exemplar Project

Cynthia E. Irvine, *Senior Member, IEEE*, Timothy E. Levin, *Member, IEEE*,
Thuy D. Nguyen, George W. Dinolt *Member, IEEE*

Abstract— **We describe the Trusted Computing Exemplar project, which is producing an openly distributed worked example of how high assurance trusted computing components can be built. The TCX project encompasses four related activities: Creation of a prototype framework for rapid high assurance system development; Development of a reference-implementation trusted computing component; Evaluation of the component for high assurance; and Open dissemination of results related to the first three activities. The project's open development methodology will provide widespread availability of key high assurance enabling technologies and ensure transfer of knowledge and capabilities for trusted computing to the next generation of developers, evaluators and educators.**

I. Introduction

Over the past decade, neither the private sector nor the US Government have been significantly involved in high assurance Trusted Computing acquisitions and research. During this time, the focus on commercial off the shelf procurements by these sectors has helped to fuel explosive advances in commercial technology, but it also contributed to the lack of progress in the ability of commercial systems to appropriately protect themselves and the data with which they are entrusted. No new high assurance (viz. TCSEC Class A1 or Common Criteria EAL6/7) systems have been fielded. While industry has been driven to supply the latest technology at the fastest pace, it has not been motivated, either internally or externally via customer demand, to produce highly trustworthy computing systems. As a result, the National Information Infrastructure is weak; there are no contemporary high security, high assurance (i.e., high robustness (see inset below)), off-the-shelf products available that can be used to strengthen it; and the National capability to design and construct such trusted computer systems and networks has atrophied.

To help address this, we have established the Trusted Computing Exemplar (TCX) project, that will provide an openly distributed worked example of how high assurance trusted computing components can be built. The TCX project encompasses four related activities:

- Creation of a prototype framework for rapid high assurance system development;
- Development of a reference-implementation trusted computing component;
- Evaluation of the component for high assurance; and
- Open dissemination of deliverables related to the first three activities.

The project's open development methodology will provide widespread availability of key high assurance enabling technologies and ensure transfer of knowledge and capabilities for trusted computing to the next generation of developers, evaluators and educators.

C.E. Irvine, T.E. Levin, T.D. Nguyen, G.W. Dinolt: Naval Postgraduate School, Monterey, CA.
This material is based upon work supported in part by the Office of Naval Research and other government sponsors. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

Robustness

The term *robustness* is currently used in the context of the Common Criteria [1] to describe what in the TCSEC [2] was associated with the notion of assurance. In the TCSEC assurance and functionality were bound in each of the ordered Divisions and Classes. Since the Common Criteria allow many more combinations of assurance and functionality, robustness provides a loose indicator of how a system might be viewed in terms of a assurance-functionality combination. A very *high robustness* system would be akin to TCSEC Class A1.

II. Trusted Computing

Trusted computing is special. It addresses not only the problems of frontal attacks, but also that of subversion: the equivalent of software moles in the system. Frontal attacks can take the form of Trojan Horse applications and other assaults from outside the system. On the other hand, moles are manifested as trap doors or other artifices placed within the system [3], [4], [5]. They can be triggered by conditions determined by the adversary and, when active, can be used to launch any attack the adversary desires from within the security perimeter [6]. The safeguards required to protect systems from trap doors go beyond those required to protect against frontal attacks. Trusted computing encompasses the science and engineering required to specify, design, implement, and maintain components in which we have a high level of confidence against both system subversion and frontal attacks.

To protect against frontal attacks, systems must be designed and implemented without exploitable flaws. The systems must be designed to constrain access to information and confine any damage resulting from the execution of malicious application software. This approach acknowledges that it would be infeasible and, in many cases, mathematically intractable [7] to analyze and verify all the application software to determine whether its behavior is benign. Most analysis techniques fail when confronted with the possibility of malicious artifices [6].

Current ambiguities surrounding the terms assurance, necessitates a precise definition of high assurance in the context of the Trusted Computing Exemplar. Systems must meet not only sound security criteria, but also be built in such a way that it is possible to verify the protection mechanisms provided. Verification techniques such as those described in the Common Criteria [1], result in verified protection, a term we will use here.

Thus our definition of a *high assurance* system follows.

A system constructed with verifiable protection provides high assurance that its security properties are correct and complete, and it allows nothing beyond its specification. It will be designed to have no exploitable security flaws and will be constructed so that external inspection and analysis confirms that the protection mechanisms are correct, complete and do nothing more than advertised.

The size and complexity of typical large systems prohibit attempts to verify them in toto. An alternative approach is where the properties of the system that must be verifiable are limited to a small, complete and non-bypassable component to which enforcement of security policy has been allocated. By minimizing system size and complexity the problem of verified protection becomes "tractable" [8].

The system is "proved" to enforce the security policy and to protect itself from penetration using a combination of formal methods and rigorous argument. Self protection is demonstrated through analysis of the design. A formal model describing core security policy and properties is mathematically proven to be sound and correct. All elements of the system are demonstrated to be both necessary and sufficient for enforcement of the policy. To achieve this demonstration, the implementation is systematically traced to the policy. This confirms there is no extraneous code and the system is shown to do what it supposed to do and *nothing more*.

Thus, our engineering approach is to divide systems into two parts:

1. An internal trusted computing component that implements the protection mechanisms and
2. External software that is not trusted.

The external software is considered potentially malicious and its execution is circumscribed by the trusted computing component. Using this approach, the effects of any malicious activity can be controlled and audited.

Protection against trap doors also requires a structured approach to system construction so that the trusted computing base of the system can be subjected to analysis that will ensure the absence of internal subversion. It is the proof of absence of unspecified functionality that distinguishes trusted computing from safety and other forms of high confidence computing.

III. Motivation for High Assurance Systems

The science and discipline of trusted computing as defined above has been neglected for well over a decade. The government and major contractors are now uncertain about the cost and complexity of high assurance computer projects. We lack developers who can create trusted computing systems, as well as public domain worked examples upon which new projects could be modeled.

The challenge of constructing highly secure systems is exacerbated by the dearth of tools to support the process. At the highest robustness levels, formal methods must be used in the system specification and analysis. Many of the tools specifically intended to support both formal methods and trusted systems development are either no longer available or lack ongoing support and modernization. We need to develop a new generation of tools, base hardware and software for developing trusted computing systems, and we need concrete public domain examples of all the steps needed to satisfy high assurance requirements, including end-to-end formal proofs of policy enforcement, so that others can see how high assurance systems are constructed.

Recently, commercial organizations have begun to realize the need for trustworthy computing. Much of this appears to be driven by the need for Digital Rights Management, the protection of copyrighted information in computer systems. One current push for a solution comes from the Trusted Computing Group (TCG) [9]. The group is proposing a hardware based solution to the digital rights management problem. Microsofts Next Generation Secure Computing Base (NGSCB, previously called Palladium) presents a specialized virtual machine monitor that will partition the system into two domains, trusted and untrusted [10]. The approach appears to be one of ensuring that only developer-signed code is allowed to operate on a system and to access the copyrighted material.

Although this is a good first step, it is insufficient. There is no understanding of how one should architect and design the trusted portions of the system to provide assurance of its correctness, nor is there an understanding that one must limit the trusted portions to a small, analyzable portion of the whole system. See, for example, a recent discussion of this topic [11].

There have also been some recent government-sponsored workshops and development in this area. These workshops have emphasized various *ad hoc* mechanisms for analyzing software for specific errors such as buffer overflow and aliasing.

Although this approach is also a good step, it too is limited in scope. The approach seems to ignore much of the past work that has been done in formal modeling of the (security) properties of systems as well as program specification and verification. It does not address how one might ensure that a system does not have a trap door or meet a particular security property.

Some work in formal methods provides proof of what the code will do, e.g. proof carrying code [12], yet it provides no insight for those interested in knowing what the code will not do and encounters composability issues.

Other recently funded work attempts new approaches in this area. The goal of this work is to provide tools that ensure that the software is correct by construction. [13] The approaches are primarily software engineering in nature. The results of the work are software development tools that may be useful in the verification of some of the properties of the system that would be useful. The tools do not address the high-level security architecture and design issues that are required for protection against moles or other subversion attacks. There are no examples that show the complete process that needs to be coherently integrated to produce a secure system, including hardware, software, processes, etc.

To address the need for a high assurance secure system as well as the absence of public domain worked examples in this realm, the Center for Information Systems Security Studies and Research (CISR) at the Naval Postgraduate School is proposing a trusted computing exemplar project. The remainder of this paper describes the project and its goals.

IV. Project Overview

The purpose of the Trusted Computing Exemplar (TCX) project is to provide a worked example to show how trusted computing systems and components could be constructed. The project will produce four related outputs:

1. A high assurance development framework,
2. A trusted computing component,
3. Evaluation of the component, and
4. Open distribution of project deliverables.

A prototype high assurance development framework will be created first, and then used to produce a reference implementation trusted computing component, the TCX kernel. A third-party evaluation of the component is planned (e.g., once the high-level design documentation is written). The documentation, source code, development framework and other deliverables will be made openly available as they are produced. Co-located teams composed of a combination of seasoned trusted computing veterans and "apprentices" are performing these activities.

The combination of open methodology applied to all project documents and deliverables and the mentoring of project apprentices will result in transfer of trusted computing technical know-how to a new generation of trusted computing professionals. Furthermore, the public availability of the high assurance development framework and the reference trusted computing component will provide technology transfer of key enabling technologies to the commercial, government, and open-source communities.

A. Framework for Rapid High Assurance Development

The prototype framework for rapid high assurance development is intended to provide a set of interoperable tools and define a set of efficient, repeatable procedures for constructing trusted computing systems and components. Our current framework reflects the requirements of the Common Criteria, Version 2.1 [1]. Current research results emerging from this aspect of the work includes, for example: documentation tools, security engineering techniques, etc.

The toolset will support automated management of high assurance development throughout a products lifecycle, including the following development functions:

- Specification of security properties
- Design specification
- Verification that security properties are self-consistent
- Verification that a design meets its target security properties
- Code development
- Verification that an implementation meets its target (and contains no unspecified functionality)
- Configuration management of specifications, software, tools and processes
- Specification-based testing
- Evaluation support through document structuring
- Teamwork and training support
- User document development

The framework will support the dissemination of project deliverables using a philosophy similar to the open source approaches. This open methodology will include mechanisms for continuous contribution, evaluation and distribution of the various project configuration items and deliverables.

Our approach for constructing the prototype development framework will be to interconnect, through software mechanisms and well-defined procedures, various state-of-the-art but stand-alone tools, each of which contributes to one or more of the required functions such as those described above.

The development framework will be scaled to the size of the project. It will reveal those aspects of such frameworks that must be tailored to meet specific project requirements.

B. Trusted Computing Reference Component Separation Kernel

We will develop a high assurance, separation kernel and a trusted application built to be hierarchically layered [14] on the TCX Kernel as a reference implementation exemplar for trusted computing. Because the product as well as the process will be showpieces for trusted computing development, high assurance methodologies and techniques will be applied during the entire lifecycle (viz., design, implementation, distribution, and maintenance phases). The goal is to produce a small, portable component that will take advantage of modern hardware support, where applicable, and that will provide users with correct security operation and an *a priori* assurance against system subversion.

Some potential applications of the TCX Kernel are as a trusted base for a multilevel network router, a red/black crypto-separation device, a communications controller for coalition-enabled mobile devices and a network authenticator for remote clients. To demonstrate the kernels utility, a high assurance network authenticator will be developed as the application portion of the Trusted Computing Exemplar.

C. TCX Kernel

The primary security function of the TCX kernel will be to enforce process and data-domain separation, while providing primitive operating system services sufficient to support simple applications. The embedded focus of the kernel drives several high-level design characteristics. The kernel will be small but complete with respect to policy enforcement. It will have a static runtime resource configuration and its security policy regarding access to resources will be based on static process/resource access bindings, that are subject to offline configuration. We anticipate that the kernel will support a small number of processes, data objects, and I/O devices. A RAM disk can be constructed from storage resources, but there is no plan to support a hard drive. Below, we provide a few more details about our view of the kernel.

The kernel will create and support a fixed number of application level processes established by the system configuration. Application processes will be scheduled in a round robin fashion with each process being given a predetermined amount of time, set by the configuration.

The process/resource access binding mechanism within the kernel will allow the assignment of specific modes of access, such as modify and observe, by which a process may access a particular resource. A variety of policies can be represented by this method, including one indicative of lattice of security domains [15].

The static nature of the bindings means that the policy represented by the set of bindings will be enforced globally and persistently until the system is taken offline. A mandatory access control policy could then be supported by maintaining consistency of the bindings across system boots.

The kernel will provide mechanisms to handle asynchronous interrupts. Any I/O will be handled by the kernel and presented to processes via memory segments. The only way that two processes may communicate will be through shared memory segments that at least one can write and the other can read. Different pairs of processes may use different segments. Simple process synchronization primitives will also be provided, that can be implemented to be demonstrably free of covert channels [16], [17].

As part of kernel verification, we will prove that a pair of processes can communicate only if there is a path of processes with appropriately shared memory that connects the two processes. The major result will be that if there is no such explicit path, then the two processes will not be able to communicate.

Our strategy is that by incorporating static resource allotment and various forms of separation early in the development process, we will both greatly simplify the design, and minimize the amount of post-design modifications required to address covert channels. The static nature of resource allotment will provide predictable processing behavior, as well as limit the covert channels based on shared resource utilization [18]. Since resources are pre-assigned, there will be no runtime allocation that would result in changes visible to other processes. (Note that in past security kernel projects, e.g. [19], [20], there has been a struggle, after the system has been designed and initially constructed, to redesign around the covert channel problems [17], [21]. For example, covert channels have been closed via complex functional limitations in the software or configuration, or through administrative procedures [22].)

Our goal in the "proof" of system security is to show how the various hardware mechanisms are coupled with the implementation of the kernel to achieve the desired separation and covert channel properties.

D. Trusted Path Extension (TPE) Application

To demonstrate the utility of the TCX kernel, we will develop a high assurance application that will use the security features of the kernel to assure its secure processing. The demonstration system is a high assurance network authentication device, which provides a trusted path extension (TPE) for communicating security critical information with a remote secure server, such as in the MYSEA distributed security architecture (see Figure 1) [23]. This device interfaces between a specially configured COTS workstation and the network. The device provides trusted path authentication and negotiation services with which users can establish trusted sessions with the server. Once logged on through TPE, user sessions at the workstation may interact with the secure server using the negotiated security parameters. For this architecture, the workstation does not

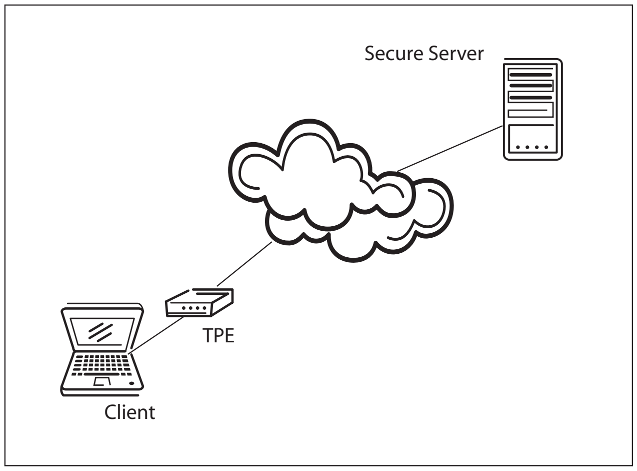

Fig. 1. MYSEA System Architecture: Client-TPE-Server.

need to be trusted to support security critical services.

Communications on the path between the TPE and the secure server are cryptographically protected, both during session negotiation and during the user session. Trusted Path Extension functions are implemented as trusted application programs of the TCX kernel, which is hosted on a handheld style hardware platform (other form factors such as PCMCIA and PCI cards are also being considered). The TPE has a layered design, consisting of the kernel (in the most privileged execution domain, 0), a set of trusted functions operating in an execution domain with intermediate privilege (1) and an application layer (in an unprivileged execution domain, 2). Functionality and security policy is allocated to each layer to minimize the trust required in any given module. The kernel layer is configured to separate the workstation domain (i.e., workstation data packets, and their processing) from the network domain, while providing a means for trusted transfer of packets between the two domains. The execution domain 1 policy allows data transfer to occur only after session-level negotiation has occurred. Application programs in the least privileged domain provide logic for data movement between input and output devices and buffers, subject to the restrictions imposed by the more privileged execution domains, 0 and 1.

E. TPE Hardware Form Factor

Hosting of the TPE on a separate, small form-factor device provides several benefits. First, the device processor and memory are physically separate from the workstation, so that there is no question of interference from user processes. Second, trusted path functions need to have absolute control of the screen and keyboard to ensure that the user is communicating directly with the trusted computing base. Since the TPE device has its own screen and keyboard, we can ensure that trusted path I/O functions are completely controlled without having to analyze the potentially complex mechanisms of the workstation screen and keyboard. Third, the *secure attention key* design, through which the user signals that s/he desires to communicate with the TCB, is straightforward relative to attempting to utilize keystrokes from a standard workstation keyboard.

F. The Development Process

The goal of our work is to provide very high assurance that the system will meet its security objectives and will not contain any mechanisms (software) that will permit subversion. To accomplish this we propose the following steps:

- Develop a written description of the security policy of the system.
- Construct a formal (mathematical) model of the security policy that reflects salient abstract characteristics of the system.
- Present an argument that the model of the security policy is an accurate representation of the policy, and verify that the model is self-consistent.
- Develop a top level specification of the kernel using the model defined above.
- Verify that the top level description of the kernel satisfies the security model.
- Develop a test plan that is driven by the top level specification for both presence and absence of security functionality in the implementation.
- Develop an implementation of the system based on the top level description.
- Provide a code correspondence that shows that the implementation is a faithful representation of the top level specification. We will use as much automation in this step as is practical.
- Test the system to validate that it is correctly implemented.
- Provide a covert channel analysis.
- Provide the appropriate controls on the documentation, verification processes, software, etc. to ensure that they cannot be subverted during the construction and delivery process.

These steps constitute an iterative rather than a sequential process. As steps in the lower levels of the process are expanded, incompleteness, inconsistency or some other form of weakness in their realization may become evident. The results of steps above will then be revised to take into account the new information.

The goal is to present a well structured demonstration that the system implements its security policy. We intend that the documentation will help to provide the appropriate level of assurance. Of course, that assurance also depends on the implementation of the hardware, the correct configuration, etc.

Each of these steps has a deliverable that is a description, a specification, a proof, code, etc. These paper deliverables will be part of the materials made available to the outside world, and will include overview materials that show how they fit into the development process, how each document relates to the other documents and how the particular document was constructed.

The process we describe above is similar to the system development processes used to build any large system. The major differences are in the integration of the formalisms used to provide the assurances that the system meets its (security) goals and the rigor required by the Common Criteria for high robustness documentation.

Verified protection, by definition [1] requires a highly controlled and restrictive development process. Since it is the objective of the Trusted Computing Exemplar project to make its results available and useful to a broad audience in government, industry and academe, the effort uses openly available tools when possible.

G. Reference Component Evaluation

An independent security evaluation is required to provide confidence in the assurance claims made for a trusted component. Currently, no U.S. organization we are aware of is prepared to perform or oversee a complete EAL7 level evaluation of a computer component or system. CISR will assist in developing the technology, processes and procedures for use in the evaluation of a component at that level.

Once the evaluation processes are better understood, the reference component, the TCX kernel, will be subjected to a third-party Common Criteria evaluation at the EAL7 [1] level. As there do not currently exist any generally accepted protection profiles for this type of high assurance system or component, we are now working as part of a team to develop an EAL6+ protection profile for a generic class of "separation" kernels. Our protection profile effort will contribute to the development of a *security target* for evaluation of the TCX kernel. The profile and security target will draw upon the TCSEC Class A1 criteria [2], and the guidelines available in the Common Criteria documentation. The completed kernel evaluation will provide the basis for a subsequent evaluation of the overall TPE system, including the application functions that implement the extended security policy.

We have just completed the development of Common Evaluation Methodology guidelines for configuration management at EAL5 through EAL7 [24]. It is expected that this work will help begin to fill in the gaps in guidance above EAL4 for high assurance systems.

H. Open High Assurance Methodology

Utilizing the open methodology tools and procedures developed in the High Assurance Development Framework (see above), the deliverables and outputs of the Trusted Computing Exemplar Project will be made available to the public. This will include such items as source code, project plans, and evaluation evidence and reports.

Part of our work will be to document for distribution the development framework, including its tools, methods, techniques, and social model. By making available the various high assurance internal engineering, evaluation and development framework documents, the Exemplar project will provide previously unavailable examples of how-to for high assurance trusted computing.

V. Related Work

A few security-oriented projects have made a large body of work available for perusal by others. The most notable is Multics [25]. The documentation, source code and all papers was available to users of the system. Unfortunately, this wealth of information was available only to those with a Multics site, the expense of which was prohibitive for most educational institutions. Despite the care with which the Multics system was constructed, it was not high assurance and had significant security vulnerabilities [3].

In contrast, several efforts received considerable attention through papers and presentations, but did not provide open access to the high assurance process. For example, the PSOS high assurance effort [26], although designed for security and, was never evaluated due to the lack of a complete, documented implementation. Systems such as SCOMP [27], KSOS [28], the VAX VMM Security Kernel[29] and GEMSOS [19] efforts were proprietary, so their internals have not been available for general inspection.

VI. Conclusion and Future Work

We have presented the Trusted Computing Exemplar, a small high assurance system that serves several purposes. First, it is intended to be a pedagogical resource for instruction in constructive security. Second, it will result in a separation kernel that can be used to support security critical functions and can be extended for enforcement of additional security policies. Third, the project will produce new tools and processes for high assurance development.

Considerable future work is planned for the TCX project. For example, the development framework being created to support the TCX is tailored to the magnitude of the project. Generalization of this framework to support different target technologies and larger projects is a topic for future research.

References

[1] *ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation.* No. CCIB-99-031, International Organization for Standardisation, version 2.0 ed., August 1999.

[2] *Department of Defense Trusted Computer System Evaluation Criteria.* No. DoD 5200.28-STD, National Computer Security Center, December 1985.

[3] P. A. Karger and R. R. Schell, "Multics Security Evaluation: Vulnerability Analysis," Tech. Rep. ESD-TR-74-193, Vol. II, Information Systems Technology Application Office Deputy for

Command and Management Systems Electronic Systems Division (AFSC), Hanscom AFB, Bedford, MA 01730, 1974.

[4] P. Myers, *Subversion: The Neglected Aspect of Computer Security.* M.S. thesis, Naval Postgraduate School, Monterey, CA, 1980.

[5] K. Thompson, "Reflections on Trusting Trust," *Communications of the A.C.M.*, vol. 27, no. 8, pp. 761–763, 1984.

[6] E. A. Anderson, C. E. Irvine, and R. R. Schell, "Subversion as a Threat in Information Warfare," 2004.

[7] D. Spinellis, "Reliable Identification of Bounded-Length Viruses is NP-Complete," *IEEE Transactions on Information Theory*, vol. 49, pp. 280–284, January 2003.

[8] R. R. Schell, "Computer Security: The Achilles' Heel of the Electronic Air Force," *Air University Review*, pp. 16–33, January-February 1979.

[9] TCG, "What is the Trusted Computing Group?," in *https://www.trustedcomputinggroup.org/home*, (downloaded 20 March 2004.), 2004.

[10] P. England, B. Lampson, J. Manferdelli, and B. Willman, "A Trusted Open Platform," *IEEE Computer*, vol. 36, pp. 55–62, July 2003.

[11] R. Anderson, "TCPA/Palladium Frequently Asked Questions." Fall 2002.

[12] G. C. Necula, "Proof-Carrying Code," in *Proccedings 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (Paris, France), pp. 106–119, ACM Press, January 1997.

[13] J. McDonald and J. Anton, "SPECWARE - Produciing Software Correct by Construction," Tech. Rep. KES.U.01.3, Kestral Institute, 3260 Hillview Avenue, Palo Alto, CA, 2001.

[14] E. W. Dijkstra, "The Structure of the "THE"-Multiprograming System," *Communications of the A.C.M.*, vol. 11, no. 5, pp. 341–346, 1968.

[15] D. E. Denning, "A Lattice Model of Secure Information Flow," *Communications of the A.C.M.*, vol. 19, no. 5, pp. 236–243, 1976.

[16] D. Reed and R. Kanodia, "Synchronization with Eventcounts and Sequencers," *Communications of the A.C.M.*, vol. 22, no. 2, pp. 115–123, 1979.

[17] T. E. Levin, A. Tao, and S. J. Padilla, "Covert Storage Channel Analysis: A Worked Example," in *Proc. National Computer Security Converence*, (Washington, DC), pp. 10–19, October 1990.

[18] R. A. Kemmerer, "A Practical Approach to Identifying Storage and Timing Channels," in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, (Oakland, CA), pp. 66–73, IEEE Computer Society Press, April 1982.

[19] R. Schell, T. F. Tao, and M. Heckman, "Designing the GEMSOS security kernel for security and performance," in *Proceedings 8th DoD/NBS Computer Security Conference*, pp. 108–119, 1985.

[20] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn, "A VMM Security Kernel for the VAX Architecture," in *Proceedings of the IEEE Symposium on Research on Security and Privacy*, pp. 2–19, IEEE Computer Society Press.

[21] J. Wray, "An Analysis of Covert Timing Channels," in *Proceeding of the IEEE Symposium on Research in Security and Privacy*, (Oakland, CA), pp. 2–7, IEEE Computer Society Press, May 1991.

[22] W.-M. Hu, "Reducing Timing Channels with Fuzzy Time," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, (Oakland, CA), pp. 8–20, IEEE Computer Society Press, May 1991.

[23] C. E. Irvine, T. E. Levin, T. D. Nguyen, D. Shifflett, J. Khosalim, P. C. Clark, A. Wong, F. Afinidad, D. Bibighaus, and J. Sears, "Overview of a High Assurance Architecture for Distributed Multilevel Security," in *IEEE Workshop on Information Assurance (to appear)*, 2004.

[24] M. Gross, "Common Evaluation Methodology Configuration Management Guidance for High Assurance Systems," Master's thesis, Naval Postgraduate School, March 2004.

[25] F. J. Corbato and V. A. Vyssotsky, "Introduction and Overview of the Multics System," in *Proceesings of AFIPS FJCC*, pp. 619–628, 1965.

[26] P. Neumann, R. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, "A Provably Secure Operating System: The System, Its Applications and Proofs," Tech. Rep. CSL-116, SRI International, Menlo Park, CA, May 1980.

[27] T. V. Benzel, "Analysis of a Kernel Verification," in *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, (Oakland, CA), pp. 125–131, IEEE Computer Society Press, April 1984.

[28] T. Berson and G. Barksdale, "KSOS - Development Methodology for A Secure Operating System," in *AFIPS Conference Proceedings, Vol. 48*, pp. 365–371, 1979.

[29] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn, "A Retrospective on the VAX VMM Security Kernel," *Transactions on Software Engineering*, vol. 17, pp. 1147–1165, November 1991.