NPS-CS-02-004
September 2002

The Center for INFOSEC Studies and Research

white paper

Diamond High Assurance Security Program: *Trusted Computing Exemplar*

Cynthia E. Irvine, Timothy E. Levin, George W. Dinolt

Center for Information Systems Security Studies and Research
Computer Science Department

Naval Postgraduate School

Monterey, California 93943

Diamond HASP Trusted Computing Exemplar

Introduction

Over the past decade, the US Government has not been significantly involved in high assurance Trusted Computing acquisitions and research. During this time, the Government's focus on *commercial off the shelf* procurements 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. 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 high security, high assurance, 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 address this downward spiral, the Center for INFOSEC Studies and Research (CISR) in Monterey, California, proposes a Trusted Computing Exemplar project, that will provide an openly distributed ***worked example*** of how high assurance trusted computing components can be built. Completion of this project will be performed as part of CISR's Diamond High Assurance Security Program (Diamond HASP). We propose four related activities:

1. Creation of a prototype framework for rapid high assurance system development
2. Development of a reference-implementation trusted computing component
3. Evaluation of the component for high assurance, and
4. 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.

Background

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. 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. 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 to analyze and verify all the software to determine whether its behavior is benign.

Instead, 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 base. 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.

The science and discipline of trusted computing 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. 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.

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 Computer Platform Alliance (TCPA). The group is proposing a "hardware based" solution to the digital rights management problem. The approach appears to be one of ensuring that only “certified” code is allowed to operate on a system and to access the copyrighted material.

Although this is a good first step, it is clearly insufficient. There is no understanding of how one should design and architect 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 [Anderson02].

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.

Other agencies have recently funded work that attempts “new” approaches in this area. The goal of the work is to provide tools that ensure that the software is “correct by construction.” 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, CISR is proposing a *trusted computing exemplar* project. The remainder of this document describes the project and its goals.

Project Overview

The purpose of the Trusted Computing Exemplar project proposed by CISR is to provide a worked example to show how trusted computing systems and components can be constructed. The project will produce of 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 Embedded MicroKernel Prototype. A third-party *evaluation* of the component will be initiated during development (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 uninitiated "apprentices" will perform these activities.

The combination of open methodology applied to all project documents and deliverables and the mentoring of project apprentices will provide 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.

Framework for Rapid High Assurance Development

The prototype framework for rapid high assurance development will provide a set of interoperable tools and define a set of efficient, repeatable procedures for constructing trusted computing systems and components.

The toolset will support automated management of high assurance development throughout a product's 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,
- Teamwork and training support, and
- 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.

We expect that 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. Generalization of this framework to support different target technologies and larger projects is a topic for future research.

Trusted Computing Reference Component – The Embedded MicroKernel Prototype

We will develop a high assurance, embedded micro-kernel and a trusted application built on top of the micro-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 very 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 kernel are as trusted base for a multilevel network router, a black/red crypto-separation device, as a communications controller for coalition-enabled mobile devices and as a network authenticator for client within a multilevel network. To demonstrate the kernel's utility, an application system will be developed that will act as a network monitor for closed multilevel networks. The system design will be extensible to open networks with the addition of cryptographic services.

Kernel

The primary security function of the Embedded MicroKernel 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 very 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 (e.g., via an access matrix). We anticipate that the kernel will support a small number of processes, data objects, and I/O devices. There may be a "RAM disk" but probably no hard drive. Below, we provide a few more details about our view of the kernel.

The kernel will create and support a fixed/constant 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.

System memory will be subdivided into units, that we call memory segments or just segments. Access to segments will be controlled by the kernel. The size of each memory segment will be set as part of the system 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. 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 invoked by maintaining consistency of the bindings across system boots.

The kernel will provide mechanisms to handle all asynchronous interrupts. Any I/O will be handled by the kernel and presented to processes via memory segments that may be accessed via the process.

The only way that two processes may communicate (at the kernel level) 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 [Reed79][Levin90].

As part of kernel verification, we will prove security theorems to the effect that a pair of processes can communicate if and 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 path, then the two processes will not be able to communicate.** The notion that there is “no such path” is part of what separates the technology that we propose from much of the work that is currently being done by other groups. Other groups show that there must be a path in certain cases, but they do not prove the absence of unacceptable paths.

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.[1]

The static nature of resource allotment will provide predictable processing behavior, as well as limit the *covert channels* based on shared resource utilization [Kemmerer82]. Since resources are pre-assigned, there will be no runtime allocation that would result in changes visible to other processes.

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

[1] In past security kernel projects [Schell85, Karger90], there has been a struggle, after the system has been designed and initially constructed, to redesign around the “covert channel” problems. For example, covert channels have been closed via complex functional limitations in the software or configuration, or through administrative procedures [Trostle93].

Network Monitor Demonstration

It might appear that such a simple kernel would not be useful. We think that this is not the case for a number of different applications. Part of the reason for this is that the cost of modern hardware is very low. It is now possible to build large complex systems with a large number of simple components, in a fashion similar to the RISC architectures for processors. Rather than try to build the complexity into the underlying component, we propose building simple components with high assurance and use these in combination to build high assurance systems. Our proposed demonstration system will be an example of this approach.

The demonstration system is a network monitor for a multilevel LAN. The micro-kernel with application will be used to provide labeling of packets that are placed on the network and de-labeling and distribution (along with appropriate checks) of packets received from the network (see Figure 1). We call such a component a *network monitor*. Each monitor interfaces between a closed IP network and one or more commercial workstation devices. The monitor labels packets going from the devices to the LAN, and monitors traffic coming in from the LAN to the devices according to the packet and device labels, and removes the incoming labels.

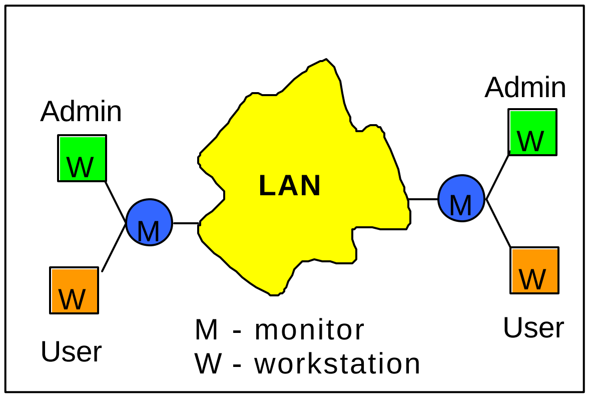

Figure 1. Network Monitor Demonstration System

The monitor has a layered design, consisting of a kernel, a set of trusted processes operating in layer one and an application layer (viz., “layer 2”). Security policy is allocated to each layer in the monitor to minimize the functionality required. The kernel-layer policy will be configured to completely separate the domain of labeled objects (the LAN) from the unlabeled domain (the workstations), while providing a means for trusted transition of packets between the two. Orthogonal to the kernel domains, the “layer 1” policy will separate different partially-ordered “workstation” domains (e.g., e.g., representing integrity or confidentiality policies) from each other, where the ordering determines allowed inter-domain accesses. Applications to manage the movement of data to and from the LAN will run on top of layer 1.

In the demonstration, there are two data integrity domains, *administrative* and *user*, where the user domain is higher in the ordering and lower integrity than the administrative domain. At each monitor, there is a workstation device in each integrity domain. The user domain devices can read either user or administrative data (viz., low or high integrity data) from the LAN, but the administrative domain devices can read only administrative (high integrity) data to keep them from being corrupted by low integrity data.

IP packets are submitted from the workstation devices to the LAN by way of the monitor. The monitor labels each packet according to its source and forwards it to the LAN. When packets addressed to one of the devices enter the monitor from the LAN, the monitor strips the labels and forwards the packets to the destination device, only if the access is allowed by the layer 1 policy. The layer 1 policy uses the packet label and the label of the device to make this determination.

To ensure that labeled and unlabeled data do not mix, which is a precondition for the enforcement of the layer one policy, and to conform to the principle of least privilege[2], separate application processes within the monitor manage different aspects of the system. Separate processes manage I/O for each device and for the LAN, and another *policy-management* process transitions data between the two kernel domains, while ensuring the enforcement of the layer 1 policy. The kernel's static *access control matrix* is configured so that packets do not transition between the LAN (labeled domain) and the workstations (unlabeled domain) without going through the policy-management process.

Future Work

A future extension of monitor functionality will allow communication of network status between monitors, and modification of policy based on that status. For example, in an emergency, all user-generated packets could be dropped to ensure administrative capabilities. Another extension would be to support connection to open (vs. closed) LANs, via cryptographic label authentication. In this case, the monitor would accept packets only if they are labeled and authenticated and would ensure that only labeled and authenticated packets were placed on the network.

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.
- 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.

[2] allocating to each function the least amount of privilege that will enable it to function correctly

- 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.

Once the steps have been completed, we expect that the documentation can be read in a linear fashion. The goal is to present a well structured proof 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 some form of 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.

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 do an EAL7 level evaluation of a computer component or system. CISR will assist in developing the technology, processes and procedures that could be used in the evaluation of a component at that level. It is appropriate for CISR to help develop these processes, since they will be tied to the development and assurance processes that CISR will use to build the demonstration project.

Once the evaluation processes are better understood, the reference component, the microkernel, will be subjected to a third-party Common Criteria evaluation at the EAL7 [CC98] level. As there do not currently exist any generally accepted protection profiles for this type of high assurance system or component, a high assurance protection profile will be developed for use in the evaluation. The EAL7 profile will be based on the TCSEC Class A1 criteria [TCSEC85], and the guidelines available in the Common Criteria documentation. The completed kernel evaluation will provide the basis for a subsequent evaluation of the overall monitor system, including the layer 1 and application functions that implement the extended label-based security policy.

Open 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, such as source code, project plans, and evaluation evidence and reports.

Part of this task 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.

ROUGH ORDER OF MAGNITUDE COST & SCHEDULE

Task[3]	Duration (months)	Cost (thousands)
1. Detailed Project Plan	3	100
2. Development Framework	12	2,000
3. Embedded MicroKernel	30	2,960
4. Domain Policy Layer (1)	24	2,500
5. Application Layer (2)	12	1,500
6. Open Distribution	30	1,500
7. Kernel Evaluation	36	3,000
TOTAL	39	13,560

DELIVERABLES: Open distribution methodology and development framework reports, evaluation reports and evidentiary documents, system manuals for users and administrators, specifications and software components, papers and theses, status reports, final project report

[3] Tasks two through seven overlap and start after task one.

References

[Karger90] Karger, Paul A., Zurko, Mary Ellen, Doug, Benin, Douglas W., Mason, Andrew H. and. Kahn, Clifford E., *A VMM Security Kernel for the VAX Architecture*, Proceedings of the IEEE Symposium on Security and Privacy, IEEE, May 1990

[Kemmerer82] Kemmerer, Richard, *A Practical Approach to Identifying Storage and Timing channels*, Proceedings of the IEEE Symposium on Security and Privacy, IEEE, May 1982

[Levin90] Levin, T., Tao, A., Padilla, S., "*Covert Storage Channel Analysis: A Worked Example*," Proceedings of the National Computer Security Conference, 1-4 October 1990, Washington, D.C.

[Reed79] Reed, David, Kanodia, Rajendra, Synchronization *with Eventcounts and Sequencers*, Communications of the ACM, Vol 22, No. 2, February 1979

[Schell85] Schell, R., Tao, T., and Heckman, M., "*Designing the GEMSOS Security Kernel for Security and Performance*," Proceedings of the National Computer Security Conference, October 1985.

[Trostle93] Trostle, Jonathon T., *Modelling a Fuzzy Time System*, Proceedings of the IEEE Symposium on Security and Privacy, IEEE, May 1993

[Anderson02] Anderson, Ross, *TCPA/Palladium Frequently Asked Questions*, *http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html*, *Fall 2002*.

[TCSEC85] DOD 5200.28-STD. (1985) Department of Defense Trusted Computer Evaluation Criteria (TCSEC).

[CC98] ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation. Technical Report CCIB-98-026, May 1998.

CORPORATE INFORMATION

Center for Information Systems Security Studies and Research
Dr. Cynthia E. Irvine, Director
Computer Science Department
Naval Postgraduate School, Monterey, CA 93943
irvine@nps.navy.mil, 831 656-2239, FAX (831 656-2814)