This concise overview of secure system developments summarizes past and current projects, deciphers computer security lingo, and offers advice to prospective designers.

The Best Available Technologies for Computer Security

Carl E. Landwehr
Naval Research Laboratory

For more than a decade, government, industry, and academic centers have investigated techniques for developing computer systems that can be *trusted* to enforce military security rules. A good deal has been learned about the problem, and a number of approaches have been explored. But what useful advice can be given to those about to embark on a system development? If the application requires a trusted system, how should the developer proceed? The purpose of this article is to summarize past experience and to guide developers.

A reader new to an area must often master its jargon and history before he can be comfortable with the subject. Unfortunately, the history is often described using the jargon and vice versa, making it difficult to get started. This article describes both the terms used in the development of trusted systems and the history of development efforts. A glossary of some of the more significant buzzwords in computer security begins on the next page. The Department of Defense has developed terms to describe modes of operating trusted systems and criteria to be used in evaluating those systems; these are explained briefly in the following section. The history of many projects is capsulized in two tables in the next section with additional notes in Appendix A. The reader is encouraged to skip back and forth among these parts according to his needs. Readers desiring a general introduction to computer security problems are referred to Sections 1-4 of Landwehr.[1]

Although this article focuses on security in military systems, most of the ideas presented also apply to commercial systems that must protect data from disclosure or modification. The key questions in building trusted systems are "What rules must the system enforce?" and "How can we assure that the system enforces them?" Some of the rules to be enforced may differ among military and commercial systems, but the approach to developing a system that can enforce them will be similar.

There is no simple recipe for creating a system that can be trusted to enforce security rules. To be trusted, a computer system must reliably enforce a specified policy for accessing the data it processes while it accomplishes the functions for which it was built. Since software engineering has as its goal the production of reliable, maintainable programs that perform according to their specifications, the best techniques developed for software engineering should be the cornerstone of efforts to develop trusted computer systems.

The principal recommendations to developers are that they (1) consider the security requirements of each system as part of its user-visible behavior, rather than as a separate set of requirements; (2) continue to think about security throughout the design and implementation of the system; and (3) use the best available software engineering technology.[2]

DoD modes of operation and evaluation criteria for trusted systems

Within the Department of Defense, classified information is labeled with a *sensitivity level* (Confidential, Secret, or Top Secret) and with a (potentially null) set of *compartments* (e.g., NATO). An individual is permitted access to sensitive information only if he has a clearance for its sensitivity level, authorization for all of its compartments, and a *need-to-know* or job-related requirement to use the information.

At present, the DoD uses four modes of operation to accredit systems processing classified information:*

- *Dedicated.* All system equipment is used exclusively by that system, and all users are cleared for and have a need-to-know for all information processed by the system.

* Additional constraints can be placed on systems processing compartmented information.

0018-9162/83/0700-0086$01:00

Computer security buzzwords

Terms often used in discussing trusted systems are listed below with definitions and comments. For more complete information on the systems mentioned, refer to the "Projects and Trends" section and to Appendix A.

Bell-LaPadula model. A security model based on the reference monitor concept, the Bell-LaPadula model has been widely applied in prototype DoD systems. Its two principal rules are the *simple security condition*, which allows a subject at a given security level to have read access only to objects at the same or lower security levels (no read up), and the **-property*, which prevents a subject from having write access to objects at lower security levels (no write down). It also provides for *trusted subjects* that are not bound by the *-property. When a trusted subject is implemented as a process in a system, it is known as a *trusted process*. (For more complete descriptions see Landwehr[1] or the article by S. Ames, M. Gasser, and R. Schell in this issue.)

Capabilities. A capability is usually defined as an unforgeable ticket that grants its holder a specific kind of access to a particular object: capabilities can be implemented as virtual addresses with additional bits that define allowed access modes. Capabilities provide a mechanism for controlling access to objects, not for implementing security policy. The provision of security depends on the system design, which may use capabilities.

Enforcement of access controls is decentralized in most capability systems—that is, once a capability is passed to a process, that process is free to pass it on to other processes. Preventing these "tickets" from getting into the wrong hands can be difficult, and the evidence suggests that, lacking appropriate hardware, capability-based systems perform poorly.

Current hardware based on capabilities includes the Plessey S250, the IBM System 38, and the Intel 432. System developments using capabilities include the UCLA data secure Unix kernel (and hence the SDC communications kernel), kernelized VM/370, and the PSOS, or provably secure operating system.

A *descriptor* is like a capability except that it has meaning only within the context of a single process: it is meaningless to pass descriptors between processes.

Confinement. The problem of preventing a program from leaking sensitive data is called confinement. For example, a program granted access to a file of Confidential data by one user might surreptitiously copy the data and transmit it to another user. Channels used to signal data to other users can be quite subtle. If information is transmitted through system storage by file names (a *storage channel*) or by varying the system load over time (a *timing channel*), the information could be obtained by a third, unauthorized party.

A program is said to be confined if it is impossible for it to leak data. In practice, storage channels can be detected, and thus eliminated, through analysis of formal program specifications; timing channels can be restricted, but their elimination usually implies severe performance penalties.

Databases. A database is a system of records composed of attribute-value pairs. Although designs have been proposed for trusted database management systems, the problem of providing both information sharing and protection against unauthorized disclosure has resisted an engineering solution.

None of the trusted-system development efforts described here deals successfully with the complexities of securing a shared DBMS, although NRL, Mitre, and others are researching aspects of this problem. A recent study for the Air Force[3] describes both problems and prospects in this area.

Encryption. Encryption is useful for protecting data that must be stored on or transmitted over media that cannot be protected against unauthorized monitoring. In some cases, it can be used to authenticate the information's source or integrity. Since protection of the data depends on the secrecy of the key, the protection and distribution of keys are critical factors.

Conventionally, *link encryption*, which implies encryption and decryption by each network processor, is used for data flowing over a specific physical path (link). In *end-to-end encryption* a message is enciphered once at its source in a network of processors and not deciphered until it reaches its final destination. *Public key cryptosystems* use different keys to encrypt and decrypt a message so that one of the keys can be publicly held. (See Denning[4] for details on these recent developments.)

The projects described in this article use a more general approach than encryption to protect information; they attempt to secure the operating system that would control the access to keys. One exception is Recon guard, which uses encryption to assure the integrity of labels attached to data.

Guard. A processor that provides a filter between two systems operating at different security levels or between a user terminal and a system is called a guard. Typically, a system that contains sensitive data also includes less sensitive data, which could be made available to users not authorized to access the system as a whole. A guard processor can monitor user requests and sanitize or eliminate unauthorized data from responses, but it may also require a human to act as watch officer. Examples include ACCAT guard, LSI guard, Forscom guard, RAP guard, Recon guard, and the message flow modulator.

Kernels. A security kernel is a hardware/software mechanism that implements a reference monitor (defined below), but the term has also been used to denote all security-relevant system software. Implementations usually include one component called the kernel, which enforces a specified set of security rules, and other components called trusted processes. These processes are trusted not to violate security, although they are not bound by all of the security rules. A rigorous application[5] of the definition would probably include trusted processes as part of the security kernel, but keeping the kernel small enough to assure its correctness formally is difficult under this interpretation.

Many projects have sought to demonstrate the practicality of the security kernel approach; the results so far are mixed. Systems using this approach to support general-purpose programming, particularly to emulate existing operating system interfaces (see Mitre secure Unix, UCLA data secure Unix, KSOS, and KVM/370), have been less successful, on the whole, than those that have applied it to support small, special-purpose applications, such as communications processors (see SDC communications kernel, PPSN, Sacdin). Performance has been a serious problem for several of the former efforts; some have provided only 10 to 25 percent the performance of their nontrusted counterparts. The latter efforts have been more successful at producing systems with adequate performance.

Some argue that these problems come primarily from choosing capabilities, rather than descriptors, as the underlying abstraction for the implementation, and from using hardware that does not support numerous segments with per-process descriptor lists or have enough execution domains. In the KSOS effort, performance problems were also attributed to the fact that a single call to the Unix emulator could trigger multiple calls to the security kernel, with each kernel call incurring the overhead of a context switch. To improve performance, revisions to the security communications processor, called Scomp, and to KSOS will provide simple operating-system interfaces to the kernel in place of Unix emulators, but they will still define Unix-compatible system call interfaces. (For further information on security kernel design and implementation, see the article by Ames, Gasser, and Schell in this issue.)

Measures. No useful quantitative measures exist at present for defining relative system security, and none of the projects listed in the tables has addressed this problem seriously. The only useful measures seem to be the difficulty of penetration or the rate of unauthorized information flow out of the system under specified conditions. The DoD Computer Security Evaluation Center is establishing criteria for defining the level to which a computing system can be trusted to enforce DoD security requirements.[6]

Network architectures. Network security is concerned with both links and nodes. From the standpoint of military security, key questions include:

- Can the security labels on data arriving at a network port be relied on to be correct?
- Can the network nodes preserve these labels and detect whether they have been altered?
- Can third parties monitor traffic on the links?

Off-the-shelf technology in this area is primarily oriented toward link encryption, but efforts to build systems utilizing end-to-end encryption and nodes that can enforce security rules have met with some success (see Sacdin, RSRE PPSN, Autodin II, COS/NFE). The WWMCCS Information System project[7] may contribute some new solutions to these problems. (See also "Protocols.")

Penetration studies. A study to determine whether and how the security controls of a computer system can be defeated is called a penetration study. We know of no serious attempt to penetrate a particular system that has not succeeded. Information on penetration studies applied to the systems listed is sparse, but an SDC study of the VM/370 (CP/67) system, based on virtual machines, did conclude that it was significantly more difficult to break into than some other systems.

Protocols. Procedures for communicating between two or more nodes of a network are called protocols. The major security problem is how to limit the unauthorized flow of information through the control fields (e.g., for routing and error control), which must often be transmitted unencrypted. (See also "Network architectures.")

Reference monitor. A reference monitor is a computer system component that checks each reference by a subject (user or program) to an object (file, device, user, or program) and determines whether the access is valid under the system's security policy. To be effective, such a mechanism must be invoked on every reference, must be small enough so that its correctness can be assured, and must be tamperproof.[5] (See also "Kernels.")

Risk assessment. A risk assessment characterizes a specific installation's assets, threats to them, and system vulnerability to those threats. The most difficult problem in any risk assessment is determining what value to attach to the data at risk. Since risk assessments consider conditions at a specific site, the most general assessment the developer can make is an analysis of the vulnerabilities of the system as a whole. Vulnerability analyses may be sensitive, since they may reveal exploitable system flaws.

Security model. A system security model defines the security rules that every implementation must enforce. It may reflect the demands of a general security policy on a particular application environment. A security model can act as a basis both for users to understand system operation and for system design. If stated formally and used as a basis for formal specification proofs, the security model rigorously defines system security.

Of the projects described in this article, the Mitre kernel, Multics security enhancements and kernel, KSOS, Scomp, guard, Military Message Experiment systems, Autodin II, KVM/370, and Sacdin, are all based on the Bell-LaPadula model (see above). The UCLA kernel implemented controls on capabilities and allowed defining specific Bell-LaPadula rules in a policy-manager module outside the kernel. Application systems, such as MME and guard, that have been built on systems enforcing the Bell-LaPadula model have included trusted processes that could downgrade information in controlled ways. Recently, there has been interest in building systems based on security models derived from particular applications (e.g., NRL SMMS,[9] message flow modulator, COS/NFE).

Specification techniques. These techniques describe system behavior. Specifications that are structured to allow mathematical analysis are called *formal specifications*. Parts of several systems described in this article have been specified formally. Typically, they use a layered approach that refines a relatively abstract top-level specification to produce a second-level specification (and sometimes a third as well). Code is based on the lowest level specification.

Cheheyl et al.[10] introduce some of the systems and tools that have been used to verify security properties, including the specification languages Special, Ina Jo, Gypsy, and Affirm. The languages Euclid, Ada, and Pascal have also been used to write specifications. There is evidence (e.g., KSOS and Scomp) that trained programmers can write formal specifications and that automated tools can expose some security flaws. However, the major benefit for security seems to be that humans—system developers, reviewers, or the specifier himself—can understand a formal specification, despite its often forbidding appearance, and can find security flaws by reviewing it manually. This finding is partly a comment on the state of formal specification and verification tools: the ones currently available, though improving, are research vehicles, not production-quality aids to software development.

Trusted. A component is said to be trusted if it can be relied on to enforce the relevant security policy. Thus, a trusted process is one that will not violate security policy, even though it may have privileges that could allow it to do so (see "Security kernel"). Presumably, there must be some way of establishing confidence that a program is in fact trustworthy. According to the DoD CSEC (Reference 6, p. 105), a *trusted computing base* is "... the totality of protection mechanisms within a computer system —including hardware, firmware, and software—the combination of which are responsible for enforcing a security policy."

Verification techniques. Verification techniques are methods for determining that two specifications are consistent. In the context of computer security, one may wish to verify that a system specification is consistent with (i.e., enforces) a certain security model, or that a lower level specification corresponds to (is a correct refinement of) a higher level specification. If both specifications are formal, mathematical techniques, including automated theorem proving, can be used to accomplish *formal verification*.

Most of the comments on specification techniques apply to verification techniques as well. Verification of security properties of top-level formal specifications seems to be within (but just barely) the current state of the art (see Scomp, KSOS), but verification that program code (in some compilable language) corresponds to some higher level specification is very difficult with available tools and usually requires considerable human intervention. The state of the art is reflected in the message flow modulator project.

Despite apparent progress in research, present verification tools are some distance from practical use in system development. The benefits from attempting verification are software engineering ones: the verification process forces the designer/implementor to think hard about what he is doing and leaves a well-structured, precise documentation trail for others to review.

Cheheyl et al.[10] describe four current systems for verifying (or helping the user verify) properties of specifications: the Boyer-Moore theorem prover, the Gypsy theorem prover, the interactive theorem prover, and the Affirm theorem prover. Other systems now in use include the Shostak theorem prover, the Stanford Pascal verifier, and Compion's Verus.

Virtual machines. This system structure provides each user with a simulated version of a bare machine on which he can load and run his own copy of an operating system. Each user can be isolated in his own machine, with communication among machines provided by the *virtual machine monitor*, which simulates the multiple virtual machines on the physical machine. KVM/370 is the prime example of a trusted system based on this approach; Share-7 is another. Each virtual machine can be operated at a separate security level, and part of the VMM can act as a reference monitor to control communication among users on different virtual machines. Since this organization helps isolate individual users at separate security levels, it is the best available approach if isolation is a primary requirement. However, the overhead required to pass information between different virtual machines becomes an important consideration in systems with extensive communication among users and across security levels.

Virtual memory. A virtual memory structure maps program-generated addresses into physical ones. The mapping mechanism permits several user programs to reside in main memory simultaneously without interference. Because the mapping device participates in every memory access, it can also act as a part of a reference monitor if access to the implementing registers can be controlled. This is one tool for organizing computer systems that a developer of a new trusted system should not forego. However, the PDP-11's small virtual address space and its approach toward device virtualization have been persistent problems in many of the efforts listed.

- *System high.* All equipment is protected in accordance with requirements for the most classified information processed by the system. All users are cleared to that level, but some users may not have a need-to-know for some of the information.
- *Controlled.* Some users have neither a security clearance nor a need-to-know for some information processed by the system, but separation of users and classified material is not essentially* under operating system control.
- *Multilevel.* Some users have neither a security clearance nor need-to-know for some information processed by the system, and separation of personnel and material is accomplished by the operating system and associated system software.

Definitions of these modes are provided in DoD Directive 5200.28.[11]

The DoD Computer Security Evaluation Center is developing criteria for evaluating the suitability of computer hardware/software systems for processing classified information.[6] Because of their likely influence on future developments and because they provide a useful framework for categorizing many of the systems discussed below, they are introduced briefly here. The criteria frequently refer to a system's *trusted computing base*, or TCB, which is defined as the protection mechanisms of that system (hardware, firmware, and software) that are responsible for enforcing a security policy. The CSEC criteria will probably be revised as they are used, but their general structure will persist.

The CSEC criteria define four hierarchical divisions, designated D, C, B, and A. Within each division except D, there are additional numbered classes; the higher the class number, the greater the trust that can be placed in the system. Thus, class C-2 systems provide more safeguards than C-1 systems. Within each class, requirements are stated for (1) the security policy enforced, (2) the ability of the system to attribute security-related actions to responsible individuals (accountability), (3) the assurance that the system operates in accordance with its design, and (4) the documentation provided. The divisions are intended to represent major differences in the ability of a system to meet security requirements, while the classes within a division represent incremental improvements. Thus, a system originally evaluated as, say, C-1 might be enhanced to meet the requirements for C-2, but would be unlikely to meet those for B-1 without major changes.

Division D, minimal protection, is reserved for systems that fail to meet the criteria for any of the other divisions; thus, it consists of systems that have been evaluated and found least trustworthy.

Division C, discretionary protection, covers systems that can be relied on to implement need-to-know controls within a single security level or compartment but cannot reliably separate, say, Secret information from users with only Confidential clearances. They must provide audit mechanisms to account for user actions that affect security. Commercial systems that do not allow security levels to be specified for user files, but do have well-designed mechanisms for users to control access to files, would be likely to fit in this division. Class C-2 provides for access control and auditing mechanisms with a finer resolution than those for C-1.

Division B, mandatory protection, includes systems that incorporate the notion of security levels, can label data on the basis of these levels, and can protect these labels against unauthorized modification. Systems in this category would be expected to segregate, say, Confidential users from Secret data. The developer is expected to provide a security model on which enforcement is based and to demonstrate that a reference monitor has been implemented. There are three subclasses: B1, labeled security protection, does not require that the security policy be stated formally or that storage and timing channels be addressed; B2, structured protection, adds these requirements and others related to the system structure, configuration control, and documentation of the system design. In addition to other constraints on system structure, B3, security domains, requires that the TCB exclude code not essential to security policy enforcement, calls for a descriptive top-level specification of the TCB and a demonstration that it is consistent with the security model, and requires support for a security administrator.

Division A, verified protection, is for systems with functions essentially like those required in class B-3, but with additional assurance that the system design correctly reflects the security model and that the implementation corresponds to the design. This assurance is to be obtained in class A-1 (verified design) through the use of formal specification techniques and formal verification that the specifications correspond to a formal model of security policy, and enhanced in class A-2 (verified implementation) through formal verification that the source code correctly implements the specification.

How these criteria will be used to determine the suitability of particular systems to operate in the modes listed above remains to be seen. The evaluation of a system and its certification to operate in a particular environment are conducted separately. It seems unlikely, however, that a system with a relatively low rating, say, C-1, would be certified to operate in multilevel mode.

Projects and trends

This section summarizes projects over the last 10 to 15 years, including those underway, that have built or are building computer systems intended to enforce security rules. Because of space limitations, not all efforts in this area are enumerated, and some projects that could be described separately have been combined. The focus is on publicly documented systems that produced (or plan to produce) at least a demonstration of operation, though a few study efforts of particular significance are included. Directions that systems now in the planning stages are likely to take are discussed at the end of the section.

* "Essentially" is not further defined in the official documents. In practice, controlled mode seems to be applied to systems that operate in a multilevel mode but bar users with clearances below some specified level. It has also been applied to systems that operate over less than the complete range of sensitivity levels (e.g., systems that contain data only from Confidential through Secret).

Structure of the tables. The following questions were asked about each system:

- When did its development begin?
- Who sponsored (i.e., paid for) it?
- Who built it?
- What were its security goals?
- What approach was used to reach these goals?
- Were formal specifications used? If so, in what language were they written? Who wrote them?
- Was any verification done? If so, what tools were used?
- On what hardware was the system built?
- Which programming language or languages were used?
- If built, did the system perform adequately for some practical use?
- If installed, was it certified for operation with classified data? If so, in what mode?
- What rating might this system receive under the DoD CSEC evaluation criteria?
- Was/is it installed? Where?
- What lessons were learned?

The answers to these questions (except the last) are given in Tables 1 and 2. Table 1 covers projects that are now complete; Table 2 covers those that are still in progress. Within each table, projects are ordered roughly according to when they were initiated. If a particular entry is enclosed in brackets, the information represents a plan or intention, rather than an accomplishment. A question mark appended to an entry indicates the information is uncertain or unknown.

Appendix A summarizes noteworthy aspects and lessons learned about the projects listed. The reader is cautioned that some of this information is subject to change (particularly for projects underway or planned) and parts of it are the personal conclusions of the author. The bibliography (Appendix B) provides additional sources

Table 1. Completed projects to develop trusted systems.

Project	Initiated	Sponsors	Builders	Goals	Approach	Formal Spec.	Verification
Adept-50	1967	DARPA	SDC	General-purpose time-sharing with security	High-water-mark model, labeled objects	No	No
Multics Security Enhancements	Early 70's	AF, Honeywell	Honeywell, Mitre	General-purpose time-sharing with security	Retrofit checks for Bell-LaPadula model	No	No
Mitre Brassboard Kernel	Early 70's	AF	Mitre	Prototype security kernel	Bell-LaPadula model as basis	Yes	Manual
UCLA Data Secure Unix	Early 70's	DARPA	UCLA	Unix with security	Security kernel plus Unix emulator	Yes	Some
Military Message Experiment	Late 70's	DARPA, Navy	ISI, BBN, MIT	Multilevel secure message system experiment	Simulated security kernel interface built on pseudokernel	No	No
Share-7	Mid-70's	Navy	FCDSSA	General-purpose timesharing with security	Based on virtual machine monitor architecture	No	No
Secure Archival Storage System	1978	Navy	Naval PG School	Secure archival file system	Multi-microprocessor-based kernel	No	No
Damos	1979	Christian Rovsing	Christian Rovsing	Operating system for communications	Security kernel with trusted processes on capability architecture	Vienna Defn. Lang.	No
Autodin II	Late 70's	DCA	Western Union, CSC, FACC	Multilevel secure packet switch	Security kernel architecture	Yes Ina-Jo	Yes ITP
SDC Communications Kernel	Late 70's	DoD	SDC	Multilevel secure packet switch	Tailor UCLA Unix security kernel	No	No
Message Flow Modulator	1981	Navy	Univ. Texas	Filter message traffic	Trusted processes directly on hardware, code verification	Gypsy	Gypsy

of information on the projects listed, but since the literature consists largely of technical reports, some may be difficult to obtain.

Comments on the projected CSEC ratings. A comment is in order concerning the estimated DoD CSEC evaluation classes listed for these systems. The ratings listed should be considered only as estimates reflecting the author's knowledge of the systems and the evaluation criteria; they are not based on the kind of exhaustive review that an actual evaluation would entail. Nevertheless, they should provide some indication of how the criteria might be applied.

One anomaly worth noting occurs in the estimated evaluation of the message flow modulator, or MFM. Generally, systems that have employed formal techniques for design and implementation achieve higher ratings than those that have not. Despite the fact that all of the source code for the MFM has been mathematically verified against a set of assertions, it has an estimated rating of C-2. The reason for this rating lies in the criteria's bundling of specific security requirements and requirements for assurance. A system cannot be rated above division C unless it meets certain requirements (e.g., labeling of objects with security levels), regardless of the level of assurance that the system correctly implements its specifications. The MFM, because of its intended use, does not need to meet most of the criteria required of a class B system, yet it requires a level of assurance that corresponds to the requirement imposed on a class A system.

Recent developments. Systems now in the planning stages include several efforts to build trusted network interfaces. These interfaces would assure that security labels transmitted to or from the network were not modified improperly and would prevent network devices from receiving information marked at levels for which they were not authorized. Trusted interface units could make

Hardware	Prog. Lang.	Perf.	Cert.	Est. Eval.	Installations
IBM/360	asm MOL/360	Yes	System high	B-1	Pentagon, CIA,SDC
Honeywell 6180 DPS8/70M	PL/I	Yes	Controlled	B-2	Pentagon AFDSC [DoD CSEC]
PDP-11	SUE-11	Demo	No	A-1	Mitre
PDP-11	UCLA-Pascal	Demo	No	A-1	UCLA
PDP-10	Bliss BCPL.?	Yes	System high	B-2	Cincpac
AN/UYK-7	CMS-2	Yes	[Controlled]	B-1	FCDSSA sites
Zilog 8000	asm	Demo	[Multilevel]	B-3	Naval PG School
CR80	?	Yes	System high	B-2	?
PDP-11	C asm	?	[Multilevel]	B-3	2 test sites
PDP-11	UCLA-Pascal	Yes	[Multilevel]	B-2	SDC.DoD
LSI-11	Gypsy	Yes	[Multilevel?]	C-2	[OSIS]

Abbreviations used in Tables 1-2

Notes:

? data unknown or uncertain
[] enclosed data indicates plans, not accomplishments

Abbreviations:

AF	Air Force
AFDSC	Air Force Data Services Center
asm	Assembly language (for machine indicated)
BBN	Bolt Beranek and Newman, Inc.
Boyer-Moore	Boyer-Moore theorem prover (SRI)
CIA	Central Intelligence Agency
Cincpac	Commander-in-Chief, Pacific
CSC	Computer Sciences Corp.
DARPA	Defense Advanced Research Projects Agency
DEC	Digital Equipment Corp.
Demo	System built as prototype or demonstrator only
DCA	Defense Communications Agency
FACC	Ford Aerospace and Comm. Corp.
FCDSSA	Fleet Combat Direction Systems Support Activity
Forscom	Forces Command (Army)
ISI	Information Sciences Institute
ITP	Interactive theorem prover (SDC)
MARI	Microprocessor Applications Research Institute (England)
MOL/360	Machine Oriented Language for IBM/360
NASA	National Aeronautics and Space Administration
NB	System never built
NC	System not yet complete enough for evaluation
NSA	National Security Agency
RSRE	Royal Signals and Radar Establishment (Malvern, England)
SDC	System Development Corporation
SDL	System Designers, Ltd. (England)
SLS	Second-level specification
SRI	SRI International
TLS	Top-level specification
VMS	Operating system for DEC VAX computer
WIS/JPM	WWMCCS joint program manager
WSE	WWMCCS system engineer
WWMCCS	World-Wide Military Command and Control System
3LS	Third-level specification

possible a multilevel secure network, to which host machines operating at different security levels could be connected.[7] The Army is currently procuring designs for its Military Computer Family operating system that are intended to allow multilevel secure operation. Finally, there is increasing interest in developing security models for particular applications. The MFM, as just noted, enforces a somewhat different notion of security than that

Table 2. Projects underway to develop trusted systems.

Project	Initiated	Sponsors	Builders	Goals	Approach	Formal Spec.	Verification
KVM/370	1976	DARPA, AF, DCA	SDC	General-purpose time-sharing with security	Retrofit security kernel to virtual machine simulator	Yes Ina-Jo	Yes ITP
PPSN (SUE)	1977	RSRE	RSRE	End-end-encryption packet switch	Kernel implementing virtual machines	Some	Some Manual
KSOS	Late 70's	NSA, DARPA, Navy	FACC, Logicon	Production prototype; secure UNIX	Security kernel with Unix emulator, trusted processes	Yes Special	Yes Boyer-Moore
Scomp	Late 70's	Honeywell, DARPA, DCA, NSA, Navy	Honeywell	Production prototype; secure UNIX	Security kernel with Unix emulator, trusted processes, hardware assistance	Yes Special	Yes Boyer-Moore
Sacdin	Late 70's	AF	ITT, IBM	Secure communications processor	Security-kernel-based architecture	Yes TLS Special	Yes TLS Boyer-Moore
Guard	Late 70's	Navy, DARPA	Logicon	Sanitize, filter between DBMSs	Trusted processes on KSOS	Some Gypsy	Some Gypsy
COS/NFE	Late 70's	DCA	Compion (DTI)	Multilevel secure network front end for WWMCCS	Security kernel (Hub), trusted modules	TLS SLS Ina-Jo	TLS SLS ITP
DEC OS Security Projects	1979	DEC	DEC	Add security to VMS, Tops-20	Retrofit Bell-LaPadula security checks (Tops-20, VMS); build kernel (VMS only)	[Yes Kernel only]	[Yes Kernel only]
Forscom Guard	1980	DCA, WIS/JPM	Logicon, Honeywell	Filter traffic between host and terminals	Trusted processes on security kernel	Yes Gypsy	?
LSI Guard	1980	Navy	I.P. Sharp	Guard system for single user	Trusted processes on bare hardware	Yes Euclid	[Yes]
PSOS	1980	NSA	FACC, Honeywell	Secure capability-based operating system	Formally specify and verify entire OS	Yes	Yes Manual
RAP Guard	Early 80's	NASA	CSC, Sytek	Filter terminal-host communications no operator	Trusted processes on trusted task monitor	[TLS Special]	Yes Manual
SDC Secure Release Terminal	1981	SDC	SDC	Trusted release station (guard)	Trusted processes on bare hardware	TLS SLS 3LS Ina-Jo	TLS [SLS] ITP
Recon Guard	1981		Sytek	Guard between network and database	Trusted processes, encryption-based authentication	No	No
GSOS	1982	Gemini Corp.	Gemini Corp.	Secure real-time operating system	Security kernel architecture	[TLS SLS]	No
Distributed Secure System	1982	RSRE	SDL Ltd., MARI	General-purpose multilevel secure local net	Trusted network interface	[Yes Euclid]	[Yes]

defined by the Bell-LaPadula model. Work at the Naval Research Laboratory is investigating the use of application-based security models in the development of military message systems.[9]

Hardware	Prog. Lang.	Perf.	Cert.	Est. Eval.	Installations
IBM 4331	Jovial J3	Demo	[Multi-level]	A-1	Mitre. SDC
PDP-11/34	asm Coral 66	Yes	?	C-2	RSRE
PDP-11	Modula	No	[Multi-level]	A-1	Logicon. Mitre
Honeywell Level 6	UCLA-Pascal C	NC	[Multi-level]	A-1	Mitre. DoD CSEC. Logicon
IBM Series 1	asm	NC	[Multi-level]	A-1	[SAC sites]
PDP-11	C.Modula	NC	[Multi-level]	B-1	?
PDP-11	Pascal	NC	[Controlled]	A-1	[Demo only]
DEC-20 VAX-11	?	NC	[Multi-level]	B-1-A-1	[DEC]
Scomp	C	NC	[Multi-level]	B-3	Forscom
DEC LSI-11	Euclid	Yes	[Multi-level]	B-3	[Navy]
Honeywell (new)	Ada	NB	[Multi-level]	A-1	
Intel 286 VAX-11/730	?	NC	[Controlled]	A-1	[NASA]
LSI-11 [Intel 8086]	Modula	Yes	[Multi-level]	A-1	SDC. (DoD)
Intel 8086	Pascal	NC	[Multi-level]	B-3	
Intel 286	PLM PL/I	NC	[Multi-level?]	B-3	?
PDP-11	[Euclid]	NC	[Multi-level]	A-1	[RSRE]

Advice for the developer

What are the lessons for the developer of a new system? It is important to distinguish between technologies that are available and useful today and research approaches that appear promising but are unproven. At present, no technology can assure both adequate and trustworthy system performance in advance. Those techniques that have been tried have met with varying degrees of success, but it is difficult to measure their success objectively, because no good measures exist for ranking the security of various systems.

Listed below, under the appropriate phase of the system development cycle, are the best available approaches for incorporating system security. Many of them simply represent good software engineering practice.

Requirements. Because security requirements affect the entire structure of the system software, their advance determination is crucial. Security is not an add-on feature. It must be incorporated throughout a system, and the statement of system security requirements must reflect this fact.

A requirements document should state what modes of operation are needed for the system initially and whether future operation in other modes is planned. However, if the requirement for security is isolated and stated without setting the context appropriately ("The system shall be secure"), bidders may view the security requirement separately from other system requirements and propose undesirable solutions (e.g., solutions based solely on physical security).

The criteria for evaluating trusted systems[6] being developed by the DoD CSEC should also be helpful in this phase. A project manager might use these criteria as a way of specifying security requirements that correspond to the needs of his system.

From the start, the system developer should consider how the system's security requirements affect its user-visible behavior. In this way trade-offs can be made where required, and a coherent design, integrating the needs for function and security, can be obtained. If this procedure is not followed, bidders may claim that they can build the system, and later, when they are well into development, discover that requirements conflict.

An illuminating example is that of the military database system that normally contains only unclassified data. During crises, some of its contents may become classified. If the requirement to handle classified data was not implemented initially, its users must either build a duplicate system, attempt to retrofit security to the existing system, or operate in a manual mode during crises.

One way of explicitly integrating security requirements with functional requirements is to specify the system's flows of information (especially the flows of sensitive information) and flows of authorization. Look especially for problems in the user interface, in operations that cause information to flow into or out of the system, and in places where the classification of information could be changed.

Trade-offs in providing security should be identified and assessed as early as possible. For example, physical controls, such as locks and guards, and computer hardware and software controls are alternative techniques for protecting information stored on computers. If they are not assessed as alternatives early in the system development, however, physical controls will be the de facto choice, because they are much easier to provide after the fact than hardware and software controls. Unfortunately, physical controls frequently restrict user functions more than comparable software controls.

Trade-offs exist within the software and hardware design as well. Many of these need to be addressed only in the design, but the system requirements should provide guidelines on the granularity of protection needed, critical information that requires special protection, acceptable bandwidths for leakage channels, and the like. The designers must face these problems whether or not guidelines are provided; it is in everyone's interest that their decisions be informed, not ad hoc.

Design. To develop a trusted system, security must be considered early and often during the design phase. As with requirements, security cannot be added to an existing design. The most prominent design strategy at present uses a security kernel, but a security kernel can impose significant performance burdens, especially if applied to inappropriate hardware. The most successful kernels have been tailored to support particular applications.

Whether a kernel-based structure is chosen or not, a system designer would be well-advised to do the following:

(1) Study system functions, focusing on requirements for the flow of sensitive information and the interface with the user. Specifically, note under what conditions sensitive information is disclosed or modified, has its classification changed, or enters or leaves the system. Include mechanisms to audit functions that might leak information, change its classification, or change its set of users.

(2) Construct a simple model of the flow of information and authority within the system. The model need not be formal, but it should be brief, precise, and simple enough to be understood by both implementors and users.

(3) Keep the model and the design consistent. If one requires changes, make corresponding changes to the other.

(4) Develop a hierarchical set of design specifications. Include a top-level specification that reflects the basic functions of the system and the information flow model, and a program specification of sufficient detail so that outside reviewers (and new personnel) can review the code structure and assess its information flows and authorization mechanisms. As the design is created, have it reviewed at regular intervals by both potential users and individuals knowledgeable in computer security. The kernelized secure operating system, Honeywell's secure communications processor, SDC's communications kernel, and other projects have used this approach with good results. Formal specifications may be helpful, but their use should be contingent on training the implementors to read and update them competently. The software tools now available for formal specification and verification are still primarily research vehicles, although this situation may change within a few years. One of the major efforts at the DoD CSEC is to collect these tools, improve them, and make them available to the computer security community.

(5) Choose hardware that reduces security problems. Generally, suitable hardware provides good mechanisms for isolating different computations, simple and efficient ways for controlling the flow of information between isolated contexts, and a uniform way of treating different kinds of objects. The following features will help in the implementation of trusted systems: virtual memory with controlled access to the mapping registers; a device interface that offers the possibility of uniform treatment of memory, files, and devices; and the ability to change addressing contexts rapidly. Several machine states that restrict access to critical portions of the instruction set will not be required if all data accesses are mediated by the virtual memory mechanism, but they can be used as another way of protecting critical operating system data (e.g., the contents of the mapping registers).

Implementation. The implementation language should have a well-understood, well-supported compiler. Some languages (e.g., Gypsy, Euclid) have been intentionally designed so that programs can be verified, and verifiable subsets have been proposed for others (e.g., Pascal, DoD's Ada).

Experience indicates that disciplined use of a conventional language with a compiler that has been exercised over a broad range of programs is preferable to using a relatively untried compiler and a "verifiable" language. Progress in practical compilers for verifiable languages is expected to continue, however, and developers should keep up-to-date. Language structures that lead to ambiguities (aliasing) in code (e.g., Fortran Equivalence) should be avoided because they make it hard for both human readers and automated tools to detect the true information flow in a program.

In practice, crucial facilities in a compiler include those that allow the parts of large programs to be compiled separately and those that enable generation of efficient object code, so that security overhead is not compounded. Assembly language should be assiduously avoided.

If it becomes necessary for the code to deviate from the specifications, the specifications should be updated in parallel with the code changes. Good coding practices benefit the security of the system, as well as its maintainability, and should make errors easier to find. Careful attention should be paid to configuration control.

Verification and testing. Automated verification that a formal, top-level specification obeys the Bell-LaPadula security model is within the state of the art but is far from routine. The developer must understand just what is demonstrated in such a verification, namely, the correspondence between two formal statements: one about the system security model and one about the system design. Vir-

tually all applications based on the Bell-LaPadula model have required trusted subjects of some kind, and it has been difficult to formalize the properties these trusted subjects enforce. Consequently, formal proofs that the system enforces the axioms of the Bell-LaPadula model typically do not apply to the trusted subjects. Assurance that the trusted subjects preserve system security requires either a separate model, specification, and proof or reliance on manual methods.

Automated tools (e.g., Ina Jo, Gypsy) have been successfully applied to the verification of trusted subjects and to systems based on security models other than the Bell-LaPadula model. However, these applications are far from routine, and demonstrating that the chosen formal security model corresponds to the user's intuitive notion of security can be a difficult task.

Automated verification of the security properties of substantial amounts of source code is beyond the state of the art. One of the largest pieces of code verified to date is in Gypsy, a Pascal derivative; it consists of well under 1000 lines (excluding those not producing executable code). These techniques hold promise for the future, but there is substantial risk in employing them in a system development that is to start next week. The verification of particular properties of a specification or small pieces of code can be done by hand or, if written in suitable languages, with machine assistance.

Despite this perhaps pessimistic assessment of the state of the art, there are benefits in adopting a formal approach. Formal verification necessitates a formal design specification, and constructing this specification requires the designer to be precise and to think his problem through very carefully. Thus, design flaws are more apt to be discovered prior to implementation. Although a few security flaws have been found during formal verification, far more have probably been removed because the designer had already noticed them in the formal specification. Finally, as noted in the comments on design, the implementors must be capable of reading and updating the specification. An airtight specification that is ignored by the implementors is worse than useless, since it may lend unwarranted credibility to the system's security.

Thorough testing continues to be a necessary partner to the careful design and implementation of systems that will be operated in a multilevel secure mode. When test plans are constructed, specific attention must be given to testing the system's security properties. If possible, the developer should arrange for penetration tests and estimate the bandwidths of leakage channels that cannot be eliminated.

Operation. From the standpoint of security, the relevant aspects of system operation are the controls over changes to the system software and configuration and the conscientious use of the system security mechanisms. Configuration control is of central importance in the operation of a trusted system, and the design of the system itself should assist in this task. The maintenance of various levels of specification, good coding practices, and use of high-level languages all help.

One often neglected aspect of operations is the monitoring of the audit trails that are routinely collected. Usually they are too voluminous (and boring) for adequate manual checking, but automated tools for this purpose need careful scrutiny. Defeating the tools becomes equivalent to defeating the auditing controls.

No philosophers' stone can turn a given system into a trusted version of the same system. The advice given above boils down to four main points:

- Consider security requirements in conjunction with the user-visible behavior of the system.
- Think about security throughout the design and implementation of the system.
- Use the best available software engineering technology.
- Be skeptical; lots of "modern" ideas don't work yet.

References

1. C. E. Landwehr, "Formal Models for Computer Security," *ACM Computing Surveys*, Vol. 13, No. 3, Sept. 1981, pp. 247-278.
2. *Software Engineering Principles*, notebook for NRL software engineering course, 1981. Available as AD-A113-415, National Technical Information Service, Springfield, Va.
3. "Report of the 1982 Air Force Summer Study on Multilevel Data Management Security," Air Force Studies Board, National Academy of Sciences, Washington, D.C.
4. D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Mass., 1982.
5. J. P. Anderson, "Computer Security Technology Planning Study," Vol. I, ESD TR-73-51, Oct. 1972, p. 14. Available as AD-758 206, National Technical Information Service, Springfield, Va.
6. "Trusted Computer System Evaluation Criteria (Final Draft)," DoD Computer Security Evaluation Center, Ft. Meade, Md., January 27, 1983.
7. D. P. Sidhu and M. Gasser, "A Multilevel Secure Local Area Network," *Proc. 1982 Symp. on Security and Privacy*, Order No. 410, IEEE-CS Press, Los Alamitos, Calif., April 1982, pp. 137-143.
8. C. R. Attanasio, P. W. Markstein, and R. J. Phillips, "Penetrating an Operating System: A Study of VM/370 Integrity," *IBM Systems Journal*, Vol. 15, No. 1, Jan. 1976, pp. 102-116.
9. C. E. Landwehr and C. L. Heitmeyer, "Military Message Systems: Requirements and Security Model," Memorandum Report 4925, Naval Research Laboratory, Washington, D.C., Sept. 1982.
10. M. H. Cheheyl, M. Gasser, G. A. Huff, and J. K. Millen, "Verifying Security," *ACM Computing Surveys*, Vol. 13, No. 3, Sept. 1981, pp. 279-340.
11. DoD Directive 5200.28 of Dec. 18, 1972, first amendment, change 2, April 29, 1978.

Appendix A—Project Summaries

Following are brief summaries of the projects listed in Tables 1 and 2, ordered as they appear in the tables.

1. Projects completed

Adept-50. Adept-50 is based on a formal security model, often called the high-water-mark model. It included the "basic security condition"—that is, no user can read information classified higher than his clearance. With some authorization, users could cause write downs—that is, move information classified at one level to a lower level. The ease of defeating its authorization controls is a matter of some debate. Installed in the Pentagon, CIA, and SDC, the Adept-50 ran for several years and was certified for system-high operation only.

AFDSC Multics (Multics security enhancements). This Air Force project applied the Bell-LaPadula model to Multics: classifications were attached to Multics objects, and checks were installed to enforce the model's rules. It yielded the first production system that supported the Bell-LaPadula model. No effort was made to isolate the security-relevant code or restructure Multics into a kernel.

The original Multics operating system (and the Multics hardware) paid close attention to protection, though not to security. The most significant innovations were rings of protection, in which inner (lower numbered) rings are more privileged than outer rings, and access control lists to control user access to files (segments). The architecture generalized the concept of a two-state (supervisor and user) machine to that of an n-state machine, with one state for each ring. The current Honeywell DPS 8/70M supports $n = 8$. In practice, nearly all of the original Multics privileged software was in the innermost ring. Verification and security models were not considerations in the original Multics development, but its sophisticated protection architecture and layered design made it a good candidate for adding security controls.

The system was installed in 1974 at the Air Force Data Services Center in the Pentagon and has been certified since December 1976 to operate in a two-level mode; the system stores some information classified at Top Secret and supports some users cleared only to Secret. These modifications, known collectively as AIM, for access isolation mechanism, are included in the standard release of Multics. They are enabled only at the customer's request, however, and only military customers have done so to date.

Following the AIM development, Honeywell and Mitre studied whether the Multics operating system could be restructured as a security kernel. At the same time, an MIT study of the Multics supervisor found that a considerable reduction in code operating in Ring 0 was possible. Funding was cut, however, and no Multics security kernel was built.

Mitre brassboard kernel. A prototype security kernel developed for a PDP-11, the Mitre brassboard kernel was based on the Bell-LaPadula model. Performance was poor, but good performance was not a project objective. Both top-level and low-level specifications were written, and extensive manual verification of the kernel operations and the correspondence between specification levels was performed. An attempt to build a Unix emulator for this kernel led to construction of a second kernel with functions more suited to Unix requirements.

UCLA data secure Unix. This prototype security kernel for a PDP-11 paralleled Mitre secure Unix, but the Mitre effort was not completed.

The UCLA architecture was based on an implementation of capabilities for the PDP-11. A prototype was developed and fitted with a Unix user interface, but performance was very slow. A separate, data-security-based model was developed for this prototype, and the Bell-LaPadula model could be enforced by a policy-manager module running outside the kernel. Efforts to keep the kernel small resulted in a fair amount of non-kernel code with security-relevant functions (e.g., the policy manager). Hence, comparing code sizes of the UCLA kernel and others must be done carefully.

Much effort was expended on formal verification, in cooperation with ISI and the Xivus theorem prover. The effort seems to have been hampered initially by the lack of a high-level system specification, which apparently stemmed from the philosophy that only verification of the lowest level assembly code was important. Nevertheless, higher level specifications, and functions to map the code to them, were eventually constructed. Ultimately, the verification of 35 to 40 percent of the kernel led developers to claim that, given sufficient resources, verification of the entire kernel was feasible.

Message systems for the Military Message Experiment. The MME evaluated operational use of an electronic message system by a military organization. Three message systems were developed in competition: Hermes by BBN, Sigma by ISI, and MIT-DMS by MIT. The ISI system was the one used.

Sigma was designed as though it were running on a security kernel, although the underlying system was not kernel-based. A trusted job was introduced to allow operations that violated the *-property of the Bell-LaPadula model. Sigma required users to confirm activity that could cause insecure information flows, but in practice, most users confirmed each operation requested without understanding or thinking about its implications.

Share-7. This Navy system developed by Fleet Combat Direction Systems Support Activity and implemented on the AN/UYK-7 computer is in operational use at approximately 16 FCDSSA sites. The system, based on a virtual machine architecture, provides a virtual AN/UYK-7 to each of its users. The Share-7 monitor enforces the system's security properties; only the monitor runs in the machine's privileged state. The executive is initiated by the monitor and communicates with users. A Data General Nova provides a trusted path to terminals, and the system uses trusted processes as well. Written in CMS-2, it is now undergoing security certification procedures for operation in the controlled mode (originally, certification for multilevel mode had been sought). It is one of the first systems the Navy has nominated for evaluation by the DoD Computer Security Evaluation Center.

Secure archival storage system. The SASS project developed a kernel for the Z8000 microprocessor. Its overall goal was to apply security kernel technology to a network of microprocessors that store multilevel files. A core-resident kernel and illustrative applications were developed; a full file server was not completed. Detailed specifications were written in PLZ, and the implementation used a structured assembly language. The developers found preliminary operations to be within their range of expectations for a single-chip processor. No formal specification or verification was performed, but close attention was paid to information flows in the implementation.

Damos. An operating system developed by a Danish company, Damos runs on Christian Rovsing's own CR80 computer. The computer and the operating system are described as capability-based, and the operating system includes implementations of a security kernel and trusted processes. These concepts seem to have been developed by CR independent of US efforts in security kernels. Damos is a successor to an earlier system, Amos, and is intended to provide fault-tolerant operation in communication system applications. Systems in which CR80s are or will be used include (1) NICS Tare, a NATO system for automating paper tape relay operations, (2) FIKS, a Danish DoD system for message, packet, and circuit switching, and (3) Camps, a NATO-SHAPE message system for communication centers.

Autodin II. This was to have been a packet-switched communication network for military use, provided by the Defense Communications Agency. Western Union was the prime contractor, with Ford Aerospace and Computer Sciences Corporation as software subcontractors. The Autodin II RFP called for

a kernel-based system for packet switching without adequately defining "kernel" or the requirements for formal specification and verification. There were many problems in its development, including a court fight over the defintion of "formal specification." Eventually, FACC wrote the code and then a specification in Ina Jo. SDC participated in verifying this specification, using ITP. The system passed its security and system tests, but it was not installed due to factors unrelated to security.

SDC communications kernel. This project modified the UCLA prototype PDP-11 kernel for use in communications applications (packet switching, etc.). The code is written in UCLA Pascal and compiled using the UCLA Pascal-to-C translator. The kernel was heavily modified and tailored to the application; no operating system emulator was used. Although performance of the installed system seems satisfactory, Pascal-to-C is not recommended for further use; it takes about five minutes of unloaded PDP-11/70 time to compile each of 60 submodules, or five hours for the entire system. The requirement for specification verification was dropped early in the project; a requirement for verifiable code motivated the choice of Pascal-to-C, but the only actual verification has been manual examination of the specification for information channels.

Message flow modulator. A flow modulator is a system that receives messages, applies a filter to them, transforms messages that pass the filter, and transmits the resulting messages to a destination. A flow modular with a null transform is like a guard, and a flow modulator with a null filter and an encryption algorithm for a transform is like a standard encryption device.

A group at the University of Texas has developed a simple flow modulator, using Gypsy and the Gypsy tools, and has proven properties related to the system's functions and its security. The proofs are unusual in that they have been conducted at the level of the actual code input to the Gypsy compiler; this work represents the state of the art in code verification of operational military systems. The system has been delivered to the Navy, which may deploy it as part of the Ocean Surveillance Information System.

2. Projects underway

KVM/370. This is an SDC project to install a kernel underneath the IBM VM/370 operating system (a virtual-machine-based system). It has been demonstrated on the IBM 4331, IBM 4341, and Amdahl V7/A. Current estimates are that it provides approximately one-quarter the performance of standard VM, a reduction from an earlier estimate of one-half.

KVM seems to be the only current project dealing with the complexities of a large-scale timesharing system (though some argue that this is really a small system running on large hardware). The design provides multiple virtual machines, each at a different security level, with restricted communication between machines handled via shared minidisks. At present, KVM supports only a limited set of peripherals, and changing system directories or the basic hardware configuration requires a system generator. Support for auditing and for a system security officer have not been implemented.

A version of the system is currently operated at Mitre, but without additional development effort, it appears that the system will only be useful for demonstration.

PPSN SUE. This a security-kernel based packet switch for supporting end-to-end encryption in the pilot packet switched network, or PPSN, at the Royal Signals and Radar Establishment, Malvern, England.

The kernel—called SUE, for secure user environment—isolates virtual machine environments so that independent virtual machines can process plaintext and encrypted data. The kernel also provides a controlled path for passing routing and connection management information between virtual machines. The kernel is said to occupy from 2K to 7K words, depending on the configuration, and was designed for this application. It operates on PDP-11/34 hardware, with specially designed support for I/O devices. It was programmed in assembler and Coral 66; some verification performed after the fact led to design changes.

SUE 2 is said to generalize the initial system to support a range of network security applications requiring a policy of isolation of low-level environments and controlled sharing on top. SUE 2 is intended to be the first of a line of common building blocks for network security applications; it is also known as the SCP 1, or secure communications processor.

KSOS. Also known as KSOS-11, the kernelized secure operating system represents an attempt to build a commercial prototype kernel-based system on a PDP-11/70. An emulator on top of the kernel was to provide a Unix interface to users. Full verification of the security properties of the kernel top-level specification (written in Special) and demonstration proofs of correspondence of code (written in Modula) to specifications were planned. The contract with FACC was terminated in December 1980.

Although performance with the Unix emulator and user software layers was poor, verification of flow properties of the kernel top-level specification (using the Boyer-Moore theorem prover at SRI) was successful. The code proof demonstrations were done by hand. The kernel by itself may prove to be a useful base for applications, but the emulator duplicated some kernel functions and performed poorly in combination with it.

The Navy is funding Logicon to enhance the FACC product, making the kernel smaller and faster and building a kernel interface package to replace the original emulator (see "Scomp"). The development of an interface to handle Unix system calls is still a possibility. The formal specifications, as well as the standard system (B-5) and detailed (C-5) specifications, are to be kept consistent with these modifications.

Scomp. The secure communications processor, also known as KSOS-6, was intended to parallel KSOS-11, using Honeywell Level 6 hardware. The government funded development of a hardware box—called SPM, for security protection module—and the kernel software specification and development. The SPM monitors transfers on the bus without CPU interference, providing faster mediation and enhanced virtual memory/capability structure over the standard Level 6. The kernel and hardware are complete, and kernel performance on the hardware is now being tested.

The original plans called for the development (funded by Honeywell) of a Unix emulator as in KSOS-11, but the revised plans call only for a minimal Scomp kernel interface package, called SKIP. A package to map Unix system calls into SKIP calls is now under construction. The kernel was specified in Special, and verified using SRI tools. The verification of the final top-level specification was substantially completed, but a few modules were too large for the SRI tools. Renewed efforts applied to this task as part of an evaluation of the system by the DoD CSEC have apparently succeeded. No code proofs are planned. The kernel is coded in UCLA Pascal, compiled via Pascal-to-C. A C compiler for Level 6 was built by Compion (formerly DTI). Trusted processes are to be specified in Gypsy, but the current contract does not cover their verification.

Scomp hardware and software has now been delivered to several sites, including Los Alamos, Mitre, Logicon, and General Electric. Providing an SPM for the 32-bit DPS-6/94, an extension of the Level 6 series, appears feasible.

Sacdin. Originally named Satin IV, this was to be a packet switching net for SAC, but the Air Force was directed to use Autodin II (now the Defense Data Network) for the network backbone. The Satin IV RFP was thought to be better (with respect to security) than Autodin II; the Sacdin development is somewhat less ambitious than the original plan. IBM is working as a subcontractor to ITT, building a kernelized system. A top-level specification has been written and verified, but the implementation is in assembly language (for IBM Series 1 computers) and no code proofs are planned. Mitre is attempting to construct a mapping between the specification and the code.

Guard. Guard (originally ACCAT guard) provides a trusted path between two database systems for networks that operate at two different security levels. It supports operators who monitor requests from the low database to the high database and sanitize the responses. The trusted process that performs downgrading was specified in Gypsy and verified by Mitre.

Guard was initially planned to run on top of the KSOS Unix emulator, but current plans are to implement it directly on the KSOS kernel as modified by Logicon. A prototype version was developed to run on unmodified Unix. Future plans include automating most of the operator functions in the system and developing a Scomp-based version.

This was the first of the now numerous guard systems to be initiated.

COS/NFE. Compion is building a trusted network front end for a World-Wide Military Command and Control System H6000 host and an Arpanet-like packet-switching network, using their proprietary Hub executive. According to Compion, the Hub executive is a security kernel. Both SDC, using Ina Jo and ITP, and Compion, using its own Verus, have written top-and second-level specifications for the Hub executive. Both companies have verified security criteria based on a strict separation of security-level sets. The Hub structure includes 12 trusted modules; each processes data at several security levels. SDC has written and verified top- and second-level specifications for each module. Code is being written in Pascal for a PDP-11/70, though the Hub also operates on VAX and Motorola 68000 hardware. Compion has not substantiated earlier claims concerning the performance of the system.

DEC operating system security projects. In 1979, DEC developed prototype security enhancements for the VAX/VMS operating system on the VAX-11/780. Discretionary and nondiscretionary controls enforcing the Bell-LaPadula model were added, auditing was improved, and a number of security flaws were corrected. The target was a system that might fit into DoD CSEC evaluation class B-1 or B-2. These changes may be included in future VMS releases.

A DEC customer, Sandia National Laboratories, has undertaken security enhancements to VMS using its Patch utility. A project to enhance security in Tops-20 is in the study phase at DEC. DEC is also building a research prototype security kernel for the VAX-11 architecture that is intended to be compatible with VAX/VMS. It will not be a general-purpose kernel; in order to keep it as small and simple as possible, the prototype will be carefully tailored for its intended applications. Current plans call for a formal specification and for verification that this specification is consistent with the Bell-LaPadula model. The goal is a system that will be in DoD CSEC evaluation class B-3 or A-1.

Forscom guard. The Army's Forces Command is developing a guard processor that will filter traffic between World-Wide Military Command and Control System users and a WWMCCS host. The motive is to ensure that users not cleared for all WWMCCS data obtain only data for which they are authorized. The system requires a single human operator who screens traffic for all of the terminals. Forscom guard programs, unlike those of the initial guard system and LSI guard, include substantial information about likely user activities to enable more accurate filtering.

Logicon produced an experimental version that was tested successfully in an Army exercise. Because KSOS was unavailable, this version employed trusted processes on top of standard Unix. Logicon and Honeywell are now building a production version utilizing the Scomp kernel and hardware; the goal is to eliminate the need for a human operator.

LSI guard. LSI guard implements a subset of guard functions directly on DEC LSI-11 hardware (without a kernel) in Euclid. It allows only a single operator (other guard processors can have multiple operators), and it is not cognizant of the databases. The specification and implementation, both in Euclid, have been completed by I.P. Sharp; verification is awaiting development of new tools for this purpose.

PSOS. An initial specification for a provably secure operating system, or PSOS, was written by SRI in the mid-to-late 1970's. This document was used as part of a product description for a two-phase contract awarded in early 1980 to FACC as prime contractor, with Honeywell as a subcontractor.

The goal was a moderate- to large-scale general-purpose computer system that would be formally verified to meet its specification. The first phase called for the design of a secure operating system using capability-based addressing and tagged architecture; the second phase, if awarded, was to be for development. It was unclear whether code proofs would be done. Honeywell bid 32-bit minicomputer hardware that looks like a next-generation Level 6 with SPM (see "Scomp"). The contract was terminated in May 1981. Honeywell is conducting a related hardware design effort under a new contract awarded in 1982.

RAP guard. The restricted-access processor is a guard system for NASA that will control traffic between unclassified terminals and a host that contains classified data. Like Forscom guard, the RAP guard system will monitor queries and responses that are highly structured, so human operators will not be required for sanitization.

Computer Sciences Corporation is the prime contractor, with Sytek as the subcontractor for security. Sytek is to develop a formal security model and a top-level formal system specification using Special. This specification is to be verified to enforce the security model. The verification will probably be manual, since the automated tools available for verifying security properties of Special specifications apply only to the Bell-LaPadula security model.

The implementation planned is trusted processes on a trusted task monitor; both of these are new developments. Possible hardware is a set of 10 single-board computers and a DEC VAX-11/730 for auditing functions. Required operational date is January 1985.

SDC secure release terminal. This system, developed in-house by SDC, has functions and design similar to the LSI guard. A terminal user sanitizes or reviews incoming data and then allows the data to be transmitted. Trusted code runs directly on a bare machine. Top- and second-level Ina Jo specifications have been written and verified; code verification of the Modula implementation is planned. Currently, the system runs on a DEC PDP-110 (a VT-100 with LSI-11). Ultimately, it is intended for the Intel 8086-based Burroughs B-20 workstation.

Recon guard. This project would allow users to have network access to a database containing some information for which network users are not authorized. The approach being taken is to attach an authenticator (similar to a checksum) to each database record, based on the contents of that record. If either the authenticator or the record is modified, recomputing the authenticator will reveal that a change has occurred. A modification to both the record and authenticator that preserves their correspondence is assumed infeasible, since the computation of the authenticator is based on a secret key.

The guard processor allows queries to enter the database and monitors responses as follows: The authenticator on the returning record is recomputed and checked. If the new value fails to match the stored value, the record has been altered and is not returned. If it does match, then the information in the record (including the security marking) is reliable. If the user making the request is cleared for the level indicated by the marking, the record is returned to him.

The secret key is shared by two systems: one system generates the authenticators for records being added to or updated within the database, and the other checks the authenticator-record correspondence on records returned in response to queries. No operator is required.

The system is being developed by Sytek. Hardware includes several Intel 8612 single-board computers. Although no formal specification or verification has been done, an informal security model was written in English, and constraints were imposed on the use of Pascal so that the source code might eventually be verified.

GSOS. The Gemini secure operating system uses the company's GC-16-4 computer, which employs the Intel 80286 processor. This processor is similar to the 8086, but adds four hardware protection rings and segmentation based on Multics.

The kernel, operating system, and applications will operate in separate rings; the design is said to draw heavily on the Multics security kernel design and on the design of SASS. Top-level and second-level specifications have been written in a variant of Gypsy. No formal verification is planned, but an informal flow analysis has been completed. The kernel is being written in PLM.

The initial offering is intended to support CPM/86 on the same hardware with tools for generating and loading applications in PL/I or other CPM-supported languages. The initial operating system supports dedicated or embedded applications rather than general-purpose programming; designs have been proposed for guard and real-time communications applications.

DSS. A distributed secure system is being developed for the Royal Signals and Radar Establishment by System Designers, Ltd., and the Microprocessor Applications Research Institute at Newcastle Upon Tyne. It is to be a general-purpose multilevel secure system based on physically separate processors connected to a local area network. Trusted interface units to the network will incorporate real-time security kernels and both trusted and untrusted software. The system is to be Unix-based with PDP-11s and DEC personal computers.

Appendix B—Bibliography

The following list will direct interested readers to further information about each system listed in Tables 1-2. It is not intended to be a complete list of references on these systems. Although we have tried to include references that are generally obtainable, several of these projects are documented only in technical reports whose availability cannot be guaranteed. References are given in the order that the systems are listed in Tables 1-2.

1. Projects completed

Adept-50

Weissman, C., "Security Controls in the ADEPT-50 Time Sharing System," *AFIPS Conf. Proc.*, Vol. 35, 1969 FJCC, AFIPS Press, Arlington, Va., pp. 119-133.

Multics

Organick, O., *The Multics System: An Examination of its Structure*, MIT Press, Cambridge, Mass., 1972.

Whitmore, J., et al. *Design for Multics Security Enhancements,* Air Force Electronic Systems Division ESD-TR-74-176, Dec., 1973. For information on security mechanisms available to users of the current commercial version, see *Multics Programmers' Manual Reference Guide,* Honeywell #AG91, Revision 2, Honeywell Information Systems, Inc., Waltham, Mass., March 1979.

Schiller, W. L., *Design and Abstract Specification of a Multics Security Kernel,* Mitre ESD-TR-77-259, Mitre Corp., Bedford, Mass., Nov. 1977 (NTIS AD A048576).

Mitre brassboard kernel

Schiller, W. L., *Design of a Security Kernel for the PDP-11/45,* MTR-2709, Mitre Corp., Bedford, Mass., June 1973.

Mitre secure Unix

Woodward, J. P. L., and G. H. Nibaldi, *A Kernel-Based Secure UNIX Design,* ESD-TR-79-134, Mitre Corp., Bedford, Mass., May 1979.

UCLA data secure Unix

Popek, G. J., et al., "UCLA Secure Unix," *AFIPS Conf. Proc.*, Vol. 48, 1979 NCC, AFIPS Press, Arlington, Va., pp. 355-364.

Military Message Experiment

Wilson, S. H., N. C. Goodwin, and E. H. Bersoff, *Military Message Experiment Final Report,* NRL Report 4456, Naval Research Laboratory, Washington, D.C.

Share-7

"Share 7/Security Design," Fleet Combat Direction Systems Support Activity (FCDSSA), San Diego, Calif., Feb. 1980. (This is an informal document, not an officially published report.)

SASS

Cox, L. A., and R. R. Schell, "The Structure of a Security Kernel for a Z8000 Multiprocessor," *Proc. IEEE 1981 Symp. on Security and Privacy*, Order No. 345, IEEE-CS Press, Los Alamitos, Calif., Apr. 1981, pp. 124-129.

Damos

Hvidtfeldt, A., and A. Smitt, "Manufacturers' Efforts in Computer Security: Christian Rovsing," *Proc. Fourth Seminar on the DoD Computer Security Initiative*, NBS, Gaithersburg, Md., Aug. 1981.

Autodin II

Bergman, S., *A System Description of AUTODIN II,* MTR-5306, Mitre Corp., Bedford, Mass., May 1978. For a short summary, see I. Lieberman, "AUTODIN II: An Advanced Telecommunications System," *Telecommunications*, May 1981, pp. 43-48.

SDC communications kernel

Golber, T., "The SDC Communication Kernel," *Proc. Fourth Seminar on the DoD Computer Security Initiative Program,* NBS, Gaithersburg, Md., August 1981.

Message flow modulator

Good, D. I., A. E. Siebert, and L. M. Smith, *Message Flow Modulator Final Report,* Institute for Computing Science TR-34, Univ. of Texas, Austin, Tex., Dec. 1982.

2. Projects underway

KVM/370

Gold, B. D. et al., "A Security Retrofit of VM/370," *AFIPS Conf. Proc.,* Vol. 48, 1979 NCC, AFIPS Press, Arlington, Va., pp. 335-342.

PPSN (SUE)

Barnes, D. H., "Computer Security in the RSRE PPSN," *Proc. Networks 80,* Online Conferences, June 1980.

KSOS

McCauley, E. J., and P. J. Drongowski, "KSOS: The Design of a Secure Operating System," *AFIPS Conf. Proc.,* Vol. 48, 1979 NCC, AFIPS Press, Arlington, Va., pp. 345-353.

Scomp

Fraim, L. J., "SCOMP: A Solution to the MLS Problem," *Computer,* Vol. 16, No. 7, July 1983, pp. 26-34.

Sacdin

System Specification for SAC Digital Network (SACDIN), ESD-MCV-1A, ITT Defense Communications Division, Nutley, N. J., 1978.

Guard

D. Baldauf, "ACCAT GUARD Overview," Mitre Corp., Bedford, Mass.

COS/NFE

Sutton, S. A., and C. K. Willut, *COS/NFE Functional Description,* DTI Document 389, Compion Corp. (formerly Digital Technology, Inc.), Champaign, Ill., Nov. 1982.

DEC secure OS projects

Karger, P. A., and S. B. Lipner, "Digital's Research Activities in Computer Security," *Proc. Eascon 82,* IEEE, Washington, D.C., Sept. 1982, pp. 29-32.

Forscom guard

"FORSCOM Security Monitor Computer Program Development Specification Type B-5," Logicon, Inc., San Diego, Calif., Feb. 1981.

LSI guard

S. Stahl, *LSI GUARD System Specification (Type A),* MTR-8452, Mitre Corp., Bedford, Mass., Oct. 1981.

PSOS

Neumann, P. G., et al., *A Provably Secure Operating System: The System, its Applications, and Proofs,* second ed., CSL-116, SRI International, Menlo Park, Calif., May 1980.

RAP guard

Contact RAP Project Manager, code 832, Goddard Space Flight Center, Greenbelt, MD 20771.

SDC secure release terminal

Hinke, T., J. Althouse, and R. A. Kemmerer, "SDC Secure Release Terminal Project," *Proc. 1983 Symp. on Security and Privacy,* IEEE-CS Press, Los Alamitos, Calif., Apr. 1983.

Recon guard

Anderson, J. P., "On the Feasibility of Connecting RECON to an External Network," James P. Anderson Co., Fort Washington, Pa., Mar. 1981.

GSOS

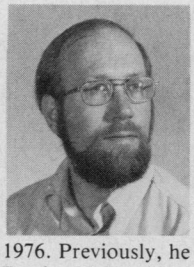
"GSOS Security Kernel Design," first edition, Gemini Computers, Inc., Carmel, Calif., 1982.

DSS

Rushby, J. M., and B. Randell, "A Distributed Secure System," *Computer,* Vol. 16, No. 7, July 1983, pp. 55-67.

Acknowledgments

For asking the question that inspired this paper, I thank D. L. Parnas. The information on which this report is based came from individuals too numerous to list here, but in whose debt I remain. Parnas and C. Hietmeyer provided reviews of earlier drafts that led to significant improvements. R. Schell suggested including the tentative DoD CSEC evaluation classes, and the referees and editors suggested numerous improvements in the presentation. H. O. Lubbes of the Naval Electronics Systems Command and S. Wilson of NRL provided encouragement and support. The responsibility for all opinions (and any remaining errors) is, of course, mine.



Carl E. Landwehr is head of the Software Research Section in the Computer Science and Systems Branch of the Information Technology Division at the Naval Research Laboratory in Washington, D.C. In addition to computer security, his research interests include software engineering and system performance modeling, measurement, and evaluation. He has been with the Naval Research Laboratory since 1976. Previously, he served on the computer science faculty of Purdue University, and he has also taught computer science at Georgetown University. He is a member of ACM, IEEE, and Sigma Xi. He holds a BS degree from Yale University in engineering and applied science, and MS and PhD degrees in computer and communication sciences from the University of Michigan.