Trusted MINIX: A Worked Example

Albert L. Donaldson
ESCOM Corporation
12206 Waples Mill Road
Oakton, VA 22124

John W. Taylor, Jr.
General Electric M&DSO
P.O. Box 8048
Philadelphia, PA 19101

David M. Chizmadia
National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755

ABSTRACT

The Trusted MINIX system is being developed to provide a worked example of C2 security mechanisms and assurances based on MINIX Version 1.5. MINIX is a small UNIX-like operating system for the PC/AT workstation, originally developed by Andrew Tanenbaum as a teaching tool for operating systems classes. Although the computer system will generally be used by only a single user at a time, MINIX was designed for multi-user, multi-tasking operation. From this perspective, the security modifications required for Trusted MINIX are essentially the same as for any multi-user system. However, MINIX was designed with a more modular internal structure than the monolithic UNIX kernel, and this structure affects how security features are added to MINIX. This paper gives an overview of the worked example, both from historical and technical perspectives.

1. Background

Trusted MINIX has its roots in the National Computer Security Center's (NCSC) Rating Maintenance Phase (RAMP)[1] class. A portion of the RAMP class involves security analysis of system changes in order to be sure that the changed system remained consistent with the TCSEC requirements. Early RAMP students attempted to analyze generic changes in a context free environment. Unfortunately, it was was difficult to analyze these changes to the level of detail necessary to provide a useful exercise.

The NCSC concluded that the context of a specific system, with specific changes to that system, were needed to provide a useful class exercise. The difficulty lay in choosing the correct system on which the exercise should be built. A proprietary operating system would not suffice for a multi-vendor class, nor could the NCSC choose a system that would serve as an implicit endorsement of an evaluated system (or system currently in evaluation). The system needed to be conceptually simple enough that students with different operating system backgrounds would be able to grasp the core concepts with little difficulty.

MINIX was chosen as the example system. Unfortunately, standard MINIX does not meet all the requirements of any class of the TCSEC. Therefore, the NCSC embarked on a vigorous campaign to ‘‘pretend’’ that MINIX met the C2 requirements. Auditing and testing magically appeared in discussions about the system and high level design documentation was written. Even though this provided a context upon which to scrutinize system change, the context was internally inconsistent. The students were quick to realize this, and this fact detracted from the benefits gained by the specific context.

A determination was then made by the NCSC to have MINIX built to meet the C2 requirements. However, during the course of creating the class exercise, some interesting discoveries were made: the RAMP class need not be the only beneficiary of a worked example. Since the example system would meet all C2 requirements, it could be used to provide trusted system vendors with examples of TCSEC documentation, such as design documentation and *Trusted Facility Manual* (TFM). If building MINIX also entailed RAMPing MINIX, a *Rating Maintenance Plan* would also be

[1] Briefly, RAMP is the process by which vendors maintain their NCSC ratings on subsequent product versions. For more information, refer to the NCSC's *Rating Maintenance Program Document* [1].

available. These documents, when used in conjunction with the corresponding “Rainbow Series” guide, could provide the vendors of trusted systems with the necessary tools and examples to create documentation meeting the TCSEC requirements. It is anticipated that this may, in fact, speed the evaluation process by increasing the productivity of both vendors and evaluators.

Another application of a worked example is that it can be used for internal training of evaluators without the risks associated with the “trial-by-fire” scheme currently in use. Lastly, the worked example would fill the void in the RAMP class by providing a consistent, evaluated system upon which to perform change security analysis. With these as possible beneficiaries of the worked example, the flavor of the requirements changed slightly. No longer was having the best C2 system features top priority; rather, C2 assurances, in particular, documentation, took a leading role.

In September 1989, the NCSC contracted with ESCOM Corporation to develop the Trusted MINIX operating system and its documentation. The contract called for ESCOM to develop the system for use as a non-proprietary “worked example” of a trusted computer security product. ESCOM’s role throughout this contract has been that of a commercial vendor developing a candidate C2 product rather than a contractor developing data items.

2. Technical Overview

Trusted MINIX was developed as a Controlled Access implementation (C2) of MINIX 1.5 for the IBM PC/AT workstation. MINIX [2] is a multi-user, multi-tasking operating system designed to be compatible with UNIX[2] (Version 7) from the user’s perspective, but with a more modular internal structure, and with widely available, published source code. The most recent release, Version 1.5, is designed to provide POSIX system call compatibility.

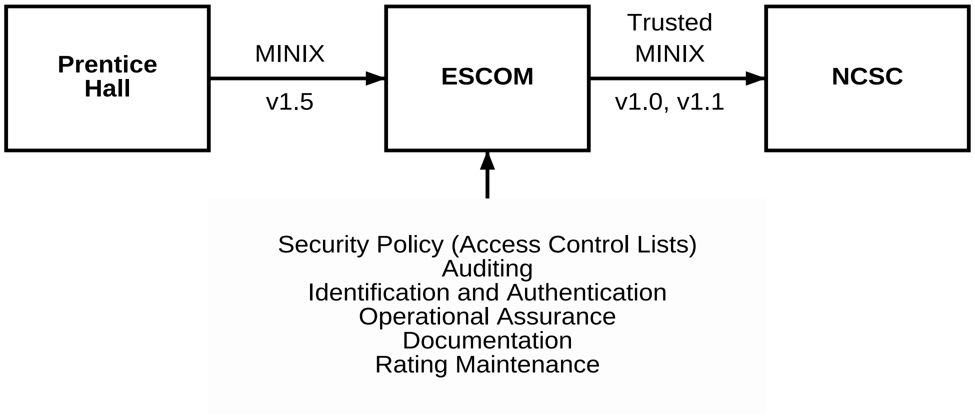

Figure 1. Trusted MINIX Product Concept.

The *Trusted Computer System Evaluation Criteria* (TCSEC) [3] requires C2 systems to include mechanisms to make users individually accountable for their actions through login procedures, auditing, and resource isolation. As shown in Figure 1, Trusted MINIX also provides access control lists (ACLs), a B3 security mechanism. Extensive user, administrator, test, and design documentation have been developed, and a subsequent revision to the system was performed in accordance with the RAMP requirements of the Trusted Product Evaluation Program.

Although Trusted MINIX has been designed specifically for the IBM PC/AT workstation, both MINIX and Trusted MINIX run without software changes on other hardware-compatible systems using the Intel 80286, 80386, and 80486

[2] UNIX is a trademark of AT&T Bell Laboratories.

processors. Trusted MINIX preserves the general architecture of standard MINIX and is source and binary code compatible with most existing MINIX programs.

During this project ESCOM performed several tradeoffs regarding the design of the Trusted MINIX system. In addition to obvious factors such as completing the system within the allocated budget and schedule, primary considerations included:

(1) Succinctly meeting the C2 requirements.

(2) Providing high quality system and user documentation.

(3) Providing a conceptually simple approach in which mechanisms are straightforward, easy to use, and easy to understand. A security mechanism that is not used because it is too cumbersome or is not well understood can actually reduce the security of a system.

(4) Following the original MINIX project goals of modularity, readability, and smallness.

3. Trusted Computing Base

Trusted systems are trusted to protect information – that is, to allow access to data only in accordance with the system's access control policy. Trusted MINIX enforces a discretionary access control (DAC) policy which allows individual users to control access to their data on a "need to know" basis. Trusted MINIX also provides individual accountability by requiring proper identification and authentication of the user before giving access to the system, and by providing the capability for a privileged user to audit security-relevant events within the system. The trust provided by the Trusted MINIX system depends equally upon the proper operation of the DAC mechanisms, individual accountability for system users, and assurances that the system is developed properly.

The parts of the system that collectively provide this trust are referred to the Trusted Computing Base (TCB). As defined by the TCSEC, the TCB is "the totality of protection mechanisms within a computer system -- including hardware, firmware and software -- the combination of which is responsible for enforcing a security policy." The Trusted MINIX TCB includes the system hardware, and firmware, and critical software such as the kernel, device handlers, Memory Manager (MM), File System (FS), and other utilities, commands, and system software. Although modularity is not a requirement for class C2 TCBs, the Trusted MINIX system is internally structured into well-defined and largely independent entities.

Figure 2 shows the overall structure of the Trusted MINIX system. All software in Layers 1 through 3 is included in the TCB, along with certain privileged user programs in Layer 4. Version 1.5 of MINIX includes over 150 user commands, a relatively small number of these that run with superuser privilege or are necessary for system administration are included as part of the TCB.

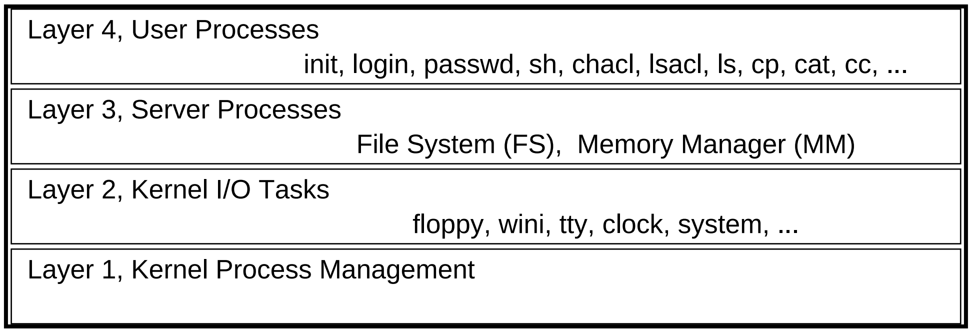
Layer 4, User Processes
init, login, passwd, sh, chacl, lsacl, ls, cp, cat, cc, ...
Layer 3, Server Processes
File System (FS), Memory Manager (MM)
Layer 2, Kernel I/O Tasks
floppy, wini, tty, clock, system, ...
Layer 1, Kernel Process Management

Figure 2. Trusted MINIX Internal Structure.

Each of these four layers is characterized by a distinct hardware privilege and execution priority. User processes at Layer 4 have a low privilege (preventing access to memory segments used by lower layers) and a low priority (they will be run only when the lower layers cannot run). Server processes have increased privilege (allowing servers to access user process memory) and increased priority to ensure that they will run before user processes. The Kernel I/O Tasks have even higher privilege and priority necessary to manage the physical hardware within the system, and the Kernel Process Management Layer manages the privilege and priority mechanisms for the rest of the system.

Trusted MINIX runs in 286 protected mode and uses the 286 protection ring mechanism to enforce the privileges associated with each layer. In addition, the kernel sets up 286 segment descriptors for each process, thus providing separation of processes (even at the same privilege level).

Although the primary purpose of the TCB is to enforce the system's policy, only a relatively small portion of the TCB (kernel, FS, MM) is directly involved with making access control decisions. This portion of the TCB implements the *reference monitor concept* (TCSEC, Section 6.1), that is, it enforces the authorized access relationships between *subjects* and *objects* of a system.

3.1. Subjects

The TCSEC defines the term ''subject'' as including all active entities such as persons, processes, or devices that cause information to flow within a system or affect the state of the system. As shown in Table 1, the only subjects defined for the Trusted MINIX system are user processes. All users are eventually represented by one or more processes, usually including the shell interpreter *sh*. As with UNIX, each process is identified by a unique process identifier (PID). User traceability is provided by maintaining effective and real user and group IDs (UID, GID, respectively) for each process.

Table 1. Trusted MINIX Subjects and Objects.

MINIX Entities	Subject	Object	Named Object	Storage Object	System Object	Public Object
User Processes	x	x	-	-	-	-
MINIX Files:						
- Regular Files	-	x	x	x	-	-
- Directories	-	x	x	x	-	-
- Device Special Files	-	x	x	x	-	-
- Named Pipes (FIFOs)	-	x	x	x	-	-
Unnamed Pipes	-	x	-	-	-	-
MINIX Messages	-	x	-	-	x	-
Disk Blocks	-	x	-	x	-	-
Memory Buffers	-	x	-	x	-	-
Registers	-	x	-	x	-	-
System Clock	-	x	-	-	-	x

3.2. Objects

The term ''object'' is defined by the TCSEC as a passive entity that contains or receives information. As shown in Table 1, there are various types of objects, including named objects, storage objects, system objects, and public objects, with different requirements for protection.

Named objects can be individually addressed, read, and written by arbitrary user subjects. The TCSEC requires the TCB's access control mechanisms to protect all named objects within the system. Named objects within Trusted MINIX include all files within the MINIX file system, including regular files, directories, and devices. These files can be directly manipulated at the TCB interface, may be destructively written by multiple users, and can serve as a channel for information between users.

Storage objects can be read and written by user subjects. The TCSEC requires the TCB to remove residual information from storage objects before allocating them to another subject. For Trusted MINIX, storage objects also consist of MINIX files, directories, and devices.

System objects are protected entities internal to the TCB (for example, firmware, process table, and inode table) that cannot be used for direct communications between subjects. Since they cannot be used to transfer information from one user to another, they do not need to be explicitly addressed by the access control policy. The Trusted MINIX message mechanism allows user processes to communicate with the kernel, MM, and FS processes. This message passing mechanism is a form of inter-process communications (IPC), but it needs no further access control mechanism because user processes are not permitted to send messages directly to other user processes.

Public objects are objects such as the system clock that can be read but not modified by the normal user. Since they cannot be used to transfer information from one subject to another, they do not need to be addressed by the access control policy.

4. Discretionary Access Controls

Because there is only one type of named object (MINIX files) to be considered, the Trusted MINIX DAC design is much less complicated than for other systems. Other systems (such as System V UNIX) allow direct interactions among user processes via IPC or shared memory, but these mechanisms are not available in Trusted MINIX.

The DAC policy for Trusted MINIX is enforced entirely within the FS program based upon an ACL stored in the inode of each file system object. This provides stronger protection against corruption than if the information were stored in a data file or other visible object.

The Trusted MINIX ACL consists of up to eight elements, with each element specifying access permissions for a user ID, group ID, or all other unspecified users on the system. Each element identifies the same permission set (read, write, execute/search) provided by standard MINIX. As shown in the following example, the Trusted MINIX ACL allows access to be specified for multiple users (charlie, lucy, hagar) and groups (peanuts, kudzu):

	Type	UID or GID	Permission
1	USER	charlie	rwx
2	GROUP	kudzu	r-x
3	GROUP	peanuts	r-x
4	OTHERS		--x
5			
6	USER	lucy	r-x
7	USER	hagar	---
8			

Figure 3. Example of Trusted MINIX ACL Structure.

4.1. Compatibility

The predominant consideration for other DAC (TRUSIX, P1003.6) [4,5] working groups has been to provide backwards compatibility with existing software and user interfaces. While such compatibility is important for vendors supporting customers running binary-only applications, it is a relatively low priority for Trusted MINIX.

ACLs and modes represent two distinct discretionary models, and the combination of the two models results in complex interactions that must be understood not only by the developers but also by the users of the system. This makes such a system less effective in enforcing the organization’s security policy.

Consequently, ESCOM has implemented a pure ACL approach without mode permissions. This approach has been described as “the cleanest model theoretically because its discretionary control is based on a single, powerful model” [6]. In addition, it is relatively easy to map the existing MINIX system calls that use mode permissions into an ACL approach, and this provides compatibility for existing binary programs.

4.2. ACL Evaluation

Access control is enforced when a process attempts to introduce a file into its address space by opening the file. When this happens, FS will determine whether the requested form of access (for example, read-only, read-write) is to be granted by checking the process’ identity (UID and GID) against elements in the file’s ACL.

Although ACL elements are not stored in any particular order, they are evaluated in order of most-to-least-specific elements. That is, the ACL will be searched first for matching USER entries, then for matching GROUP entries, then finally for an OTHERS entry. The permissions associated with the first matching entry are used to determine access. If there is no matching element (USER, GROUP, or OTHERS), the requested access is denied. This approach allows access to be explicitly denied for an individual user, even if that user is a member of a group that is allowed access.

4.3. Object Creation

Perhaps the most important issue during object creation is how to set the access control permissions for the new object. Trusted MINIX has replaced the standard *umask* mechanism with an ACL inheritance mechanism where each new object receives its initial ACL from the ACL of the parent directory. As described in other references, this approach is probably the most natural for user and shared project directories, since files inherit permissions from the containing directories.

Using this approach, newly created files inherit an ACL that is derived from the parent directory ACL and the requested permissions from the calling program[3]: The ACL for the new object is copied from the parent directory ACL. An element is created for the owner of the object (if the owner is not already listed on the ACL) with the owner permissions requested by the calling program. All other elements inherited from the parent directory ACL are ANDed with either the group or others bits from the calling program.

This approach is used to initialize the ACL for all newly-created objects. However, certain programs needed to be modified to change this initial ACL to comply with the historical usage of the program. For example, *cpdir -s* duplicates permissions from the origin directory to the destination directory and *tar* provides an option to save and restore ACL information.

[3] For compatibility with applications using MINIX mode permissions, these are the low-order nine bits of the file mode passed to *creat()*, *mkdir()*, or *mknod()*.

5. Object Reuse

The protection philosophy for object reuse is to ensure that the authorizations and contents of reusable objects are properly initialized before the objects are made available to a new user. The TCSEC requires the following mechanisms to be provided for storage objects:

(1) Revoke previous authorizations to the object

(2) Overwrite or clear any residual information remaining in a physical storage location before allowing another user to have access to the object.

The storage objects listed in Table 1 include not only the named objects addressed by the DAC policy (files, directories, etc.), but also lower-level items such as disk blocks, memory buffers, and device registers that may contain residual information. Standard MINIX satisfies the TCSEC C2 requirements for object reuse without any changes other than documenting the mechanisms.

6. Identification and Authentication

The DAC mechanisms described above depend upon the principle of individual accountability. The Trusted MINIX system provides individual accountability by requiring proper identification and authentication of the user before giving access to the system.

6.1. Protecting Encrypted Passwords

Trusted MINIX provides a protected (or "shadow") password file (/etc/tcb/passwd) to prevent general users from being able to read encrypted passwords. The public file (*/etc/passwd*) is still available, but the password field is not used.

6.2. Password Selection

The Trusted MINIX *passwd* program has been modified to filter out certain "weak" passwords, as described below:

(1) Passwords must have a length of at least six characters.

(2) Passwords must contain a non-alphanumeric character (for example, a punctuation mark or a mathematical symbol).

(3) The new password must differ from the previous one.

(4) Trusted MINIX disallows access to the system if the user has a null password. This ensures that the system administrator will set up a password for a new user.

Instead of implementing automatic password aging, Trusted MINIX provides a date field in the protected password file that is changed each time the password is changed. This information can be used by the system administrator to review the current age of users' passwords on the system.

6.3. Login

The Trusted MINIX *login* program authenticates each user's identity before allowing access to the system. It performs the following new functions:

(1) Require a password to be entered, even if the login name is bad. Standard MINIX does not ask for a password if the login name is bad; this allows the user to find valid login names more quickly.

(2) After successful authentication, notify users who login successfully of the date and time of last login and the number of unsuccessful attempts since then. This information is copied from the */etc/tcb/lastlog* file, which

maintains information about each user's last login and unsuccessful attempts.

(3) Update the */etc/tcb/lastlog* file with the new port, time, and failure information.

7. Audit

In addition to strengthened identification and authentication mechanisms, Trusted MINIX supports the principle of individual accountability by providing the capability for a privileged user to audit security-relevant events within the system.

7.1. Audit Events

The requirement for auditing at C2 includes: ''use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events.'' Trusted MINIX provides for auditing the following types of events:

Table 2. Auditable Events

Type of Event	Location	I&A	Object	Admin	Other
User Commands					
login	login	x			
su	su	x			
lpr	lpr				x
passwd	passwd				x
System Calls					
fork, exec	MM		x		
open, close	FS		x		
creat, mknod	FS		x		
link, unlink	FS		x		
chroot	FS			x	
stime	FS			x	
chown	FS			x	
mount, umount	FS			x	
audit	FS			x	
kill	MM				x
aclctl	FS				x
setuid, setgid	MM				x
privilege override	FS, MM			x	
failed I/O	FS				x

The second column shows the location where the audit event will be generated, either a trusted user program, server, or the kernel, depending on the type of audit. For example, the *unlink()* system call is implemented within the FS server by the *do_unlink* procedure.

Privilege override allows auditing of events where an operation succeeds only because it is requested by the superuser (UID 0). These situations occur in various system calls, for example, *open()*, *chown()*, *acl()*, *kill()*, and *setuid()*.

7.2. Audit Architecture

As described above, the audit function in Trusted MINIX is necessarily distributed throughout the TCB, with the majority of audits being generated within FS and (to a lesser extent) MM.

Because of this modularity already inherent in the operating system, there were a number of design alternatives for implementation of the audit collection function. The preferred approach was to implement a new audit server at the same level as MM and FS, however, ESCOM implemented the audit collection function within the FS server in order to avoid problems with potential deadlock between FS and an external server. Because the FS and MM servers are ‘‘single-threaded’’ (that is, they only process one transaction at a time), there is the possibility of a synchronization deadlock. Andy Tanenbaum [7] observed that the current MINIX implementation relies on there only being two servers, with limited, one-way (MM-to-FS) interaction between the two servers. Recording audits within FS also makes for optimal performance, since well over half of the system calls are performed within FS. The collection and writing of local audit information within FS consists of a simple procedure call.

7.3. Selective Audit

The TCSEC requires a means to selectively audit the actions of users based on individual identity. This can either be done by pre-selection (audit only the selected users) or post-selection (scan the audit trail for events matching the user). Trusted MINIX allows pre-selection of the types of events to be recorded. The audit reduction tool (*auditfmt*) allows post-selection of audit entries matching a particular user ID, group ID, or inode. *auditfmt* is designed to operate as a filter on an audit file (not necessarily the current audit file), and send to standard output audit records matching the specified criteria.

One of the areas where Trusted MINIX differs from other audit implementations is that the individual instrumentation points send all audit events to FS, even if the event will eventually be discarded. This centralized approach to audit selection has some minor performance implications, but is more modular, easier to modify, and conceptually cleaner than distributing the decisions to each of the instrumentation points.

8. Documentation

Perhaps the most useful aspect of Trusted MINIX is the example documentation. All documentation was written with the assumption that the reader has a user level understanding of standard MINIX. The documentation can be broken into four subgroups: design, test, user, and RAMP.

8.1. Design Documentation

As shown in Figure 4, six documents were written to satisfy the TCSEC design documentation requirements. This is in addition to previously existing documents that were also used to satisfy the requirements.

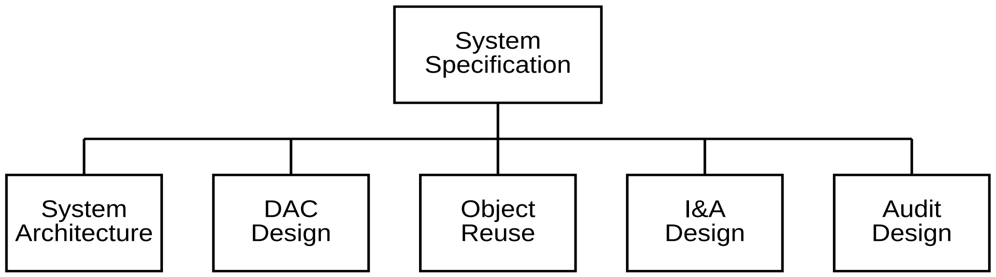

Figure 4. Trusted MINIX Design Documentation

The first document is the *System Requirements Specification*, which identifies the requirements for security features and assurances built into Trusted MINIX system, and describes the relationships among the features. There are four subsystem design documents, which discuss in more detail the functional requirements, the design of the system, and implementation specifics. Unfortunately, these documents did not adequately cover the assurance requirements of the TCSEC, nor did they discuss how the security features fit within the Trusted MINIX system components (such as the kernel, file system, and memory manager). Therefore, the *System Architecture* design document was written to cover these specific issues using a structured breakdown of the Trusted MINIX system. Additional commercially available documentation has been identified for use in documenting the hardware design and implementation.

8.2. Test Documentation

The test documentation consists of a single document covering the design of the Trusted MINIX security relevant tests, the expected results from those tests, and the actual results of the tests.

8.3. User Documentation

The Trusted MINIX user documentation consists of a *Trusted Facility Manual* and a *Security Features Users Guide*. The TFM discusses the issues associated with installing and administering a Trusted MINIX system. It discusses the use of the trusted administrator shell, the use of the auditing mechanism, and various other administration functions and details. The SFUG leads a user through the logon process and explains in detail the security features provided by the system, as well as the users role in system security. Both documents contain manual pages for the security features referenced in the body of each. The SFUG also includes the remaining manual pages not related to security.

8.4. RAMP Documentation

The RAMP documentation consists of all that is needed for a single cycle of Rating Maintenance. This includes a *Configuration Management Plan* and *Rating Maintenance Plan*. The *Configuration Management Plan* takes the view of managing change, as opposed to the *Rating Maintenance Plan*, which takes the view of managing releases. Other RAMP documentation includes the configuration management evidence necessary to support RAMP, as well as a *Rating Maintenance Report* and supporting documentation.

9. Conclusions

The work done by ESCOM to develop the Trusted MINIX system and its associated documentation represents only one side of the effort required to developed a complete worked example, since it only covers the evaluation process from the product developer’s side. The other side of the worked example involves the evaluation of the Trusted MINIX system and the development of the documentation associated with that evaluation. This documentation includes the *Preliminary Technical Report* (PTR), the *Initial Product Assessment Report* (IPAR), the *Evaluation Test Plan*, and the *Final Evaluation Report* (FER). The PTR is a cursory analysis of the proposed system (either a design or an existing untrusted base) to determine how feasible it is to complete a trusted product evaluation. The IPAR is based on a detailed technical analysis of the developer’s design and user documentation and describes how the system satisfies the requirements of the TCSEC. The IPAR is both the blueprint for the actual product evaluation and the basis of the FER. During the evaluation, the team prepares and runs security tests beyond those done by the developer, these test are documented in the Evaluation Test Plan. Finally, the team produces a report for public release that describes how the product satisfies the TCSEC requirements.

During the development of Trusted MINIX, a team of NCSC evaluators worked with ESCOM to define the security issues and identify possible solutions, in the same way the NCSC usually works with trusted product developers. At the completion of the contract, this team will produce the IPAR, ETR, and FER. At this point, the worked example will be complete with respect to the product evaluation process. The final aspect of the project will be validating the evidence developed under the contract to show the the system maintained its trustedness as it evolved from release 1.0 to 1.1. This will provide the RAMP element of the worked example. When the worked example is complete, the NCSC will publish it as a series of 11 documents as part of the technical guidelines program (aka the Rainbow Series).

It is expected that the Trusted MINIX will bring the following tangible benefits to the Information Security community:

(1) By providing an example of what the NCSC is looking for in terms of design and user documentation, it will allow product developers to better determine the level-of-effort required to complete the evaluation.

(2) It will provide the basis for quicker and more effective evaluator training.

(3) It will provide a preliminary validation mechanism for fine-tuning the RAMP requirements.

(4) It will provide the consistent example needed to effectively train Vendor Security Analysts, and

(5) It will provide a low-cost and possibly ‘‘fun’’ means for anyone interested in information security to experiment with basic trust technology, both at home and at school.

The use of Trusted MINIX as a worked example doesn’t stop here. Work is already underway to use Trusted MINIX as the basis for three follow-on efforts: a B-level worked example; integration into a networked/distributed computing environment; and as a baseline for a trusted system portability study. As we and others gain more experience with Trusted MINIX, we expect to find even more ways to use it to advance the state-of-the-art in Information Security.

10. Acknowledgements

The authors wish to acknowledge the work performed by Brian Beattie, John Hare, Tom Welsh, and Karl Nyberg on this contract. Discussions with trusted UNIX vendors, particularly Michael McChesney of SecureWare and Tim Ehrsam of Addamax, were helpful to ESCOM in formulating initial concepts for this system. Finally, ESCOM wishes to acknowledge the guidance and assistance within the NCSC and the Trusted MINIX evaluation team, particularly the assistance by James Goldston before the contract was awarded.

REFERENCES

(1) *Rating Maintenance Program Document*, National Computer Security Center, NCSC-TG-013.

(2) *Operating Systems Design and Implementation*, Andrew S. Tanenbaum, Prentice-Hall.

(3) *Department of Defense Trusted Computer System Evaluation Criteria*, National Computer Security Center, December 1985.

(4) *Rationale for Selection of Access Control List (ACL) Features for the UNIX System*, National Computer Security Center, 18 August 1989.

(5) *Portable Operating System Interface for Computer Environments, Trusted System Extensions*, Draft 3.

(6) *On Incorporating Access Control Lists into the UNIX Operating System*, ‘‘Proceedings UNIX Security Workshop,’’ 1988, Steven M. Kramer, SecureWare, Inc.

(7) Personal communication with Andy Tanenbaum, 5 February 1990.