PAT Guidance Working Group

Form and Content of Vendor Test Documentation

NATIONAL

COMPUTER SECURITY CENTER

9800 Savage Road
Fort George G. Meade
Maryland 20755-6000

May 1994

Report No. TBD
Library No. TBD

FOREWORD

The National Computer Security Center is issuing the *Form and Content of Vendor Design Documentation* and the *Form and Content of Vendor Test Documentation* to define process requirements and to provide guidance for meeting the C2-B1 design documentation and test documentation requirements, respectively, of the Department of Defense Trusted Computer System Evaluation Criteria (DoD 5200.28-STD). The guidance contained in the documents provides specific approaches for creating comprehensive views of the vendor's design and test documentation and establishes a set of deliverables needed to follow the approaches.

In addition to the current set of deliverables, the following documents are now required for systems entering the Trusted Product Evaluation Program: the Architecture Summary Document, the Interface Summary Document, and the Test Matrix Document. The *Form and Content of Vendor Design Documentation* establishes requirements for the Architecture Summary Document, the Interface Summary Document, and an expanded Philosophy of Protection. The *Form and Content of Vendor Test Documentation* establishes requirements for the Test Matrix Document.

The goals of the approaches described for vendor design documentation and test documentation are the following: to provide clearly defined, reliable methods for meeting the design and test documentation requirements of the Criteria; to allow the vendor to use as much of the existing system documentation as possible and to quickly identify incomplete or missing documentation; and to mitigate or eliminate significant delays in the evaluation schedule due to the lack of adequate design and test documentation. In an effort to provide specific guidance, recommendations are made concerning the detailed content of vendor design documentation and test documentation. Vendors who follow the guidance closely will more likely complete the evaluation in an efficient manner.

The approaches in the two documents were established by the Process Action Team (PAT) Guidance Working Group (PGWG) with assistance and significant comment from members of the Interpretations Working Group (IWG) and the Trusted Product Evaluation Program (TPEP) community. The efforts of the PGWG are part of a continuing initiative to improve the process and the overall quality of TPEP. Vendors are encouraged to provide feedback on the PGWG documents to evaluation teams or directly to the National Computer Security Center. As experience is gained in the use of the guidance and as sufficient feedback is obtained from the vendor and evaluator communities, the documents will be revised as necessary.

ACKNOWLEDGMENTS

Acknowledgments are given to the primary authors of and contributors to this document: Anne Lindell and Kraig Meyer of The Aerospace Corporation; Brett Borgeson, John Janeri, Roberta Medlock, Ron Watro, and Robert Williamson of The MITRE Corporation. Portions of this document are based on the work of Olin Sibert. In addition, other members of the Interpretations Working Group and of the Trusted Product Evaluation Program (TPEP) evaluation community have provided invaluable guidance and insightful comments on various revisions of this document.

TABLE OF CONTENTS

TABLE OF CONTENTS

FIGURES

This page intentionally left blank

Chapter 1

Introduction

This document discusses security testing and related documentation required for the Trusted Computer System Evaluation Criteria (TCSEC) evaluation process for C2-B1 candidate systems [1]. This document describes each of the required documents; provides a suggested methodology for creating the test documentation; and discusses issues related to security testing, such as recommended Trusted Computing Base (TCB) coverage, the security test suite and test tools, test evidence, and evaluation team testing. Testing activities in the evaluation process itself are discussed in appendix A. This document provides specific recommendations for operating systems. Guidance concerning higher assurance systems and other topics, including networks and databases, will be developed later. The general information in this document should be useful for evaluations of all types.

Some of the material contained in this document is based on *Software Testing Techniques* [5]. The book contains some basic definitions of approaches to testing (e.g., black-box and white-box testing). A more detailed discussion of black-box testing may be found in *An Analysis of Black-Box Testing Techniques* [6]. An approach to gray-box testing is discussed in "A New Security Testing Method and Its Application to the Secure Xenix Kernel" [10].

1.1 Purpose of the Document

The purpose of this document is to clarify the form and content of the test documentation needed before an evaluation can begin. Examples of actual tests used in previous evaluations are provided to illustrate the concepts presented in the discussion. This document is introduced to reduce the delays in obtaining the required test documentation that often impede the progress of evaluations. Vendors who follow the methods explained in this document will increase the likelihood of passing the test documentation requirement and, therefore, will likely avoid the ambiguities and the coverage issues that lead to significant revision, testing analysis efforts, and longer evaluation schedules.

Design documentation for a candidate product is required to support test activity. This document refers to design information as defined in *Form and Content of Vendor Design Documentation* [12]. Of particular importance are the Architecture Summary Document (ASD), which describes the candidate product's organization into logical subsystems, and the Interface Summary Document (ISD), which identifies the TCB interfaces in each of the logical subsystems.

1.2 Purpose of Security Testing

The security testing requirement in the TCSEC for all rating levels states, "The security mechanisms of the Automated Data Processing (ADP) system shall be tested and found to work as claimed in the system documentation." The testing requirement implies that all software in the TCB that enforces or supports a security mechanism must be tested.

The goal in developing security tests is not merely to check adherence to the documented design but to uncover flaws. Flaws allow violations of the security policy of the system and include errors in the design or implementation.

1.3 Security Mechanisms

The TCSEC refers to security "mechanisms" testing. For C2-B1 systems, security mechanisms are contained in the logical subsystems that perform functions that include the following: access checks, defined operations on shared objects (e.g., I/O operations on files which have an associated mandatory sensitivity label), identification and authentication, auditing, labeling output, exportation of objects outside the system, process isolation, TCB self-protection, and the handling (i.e., maintaining the integrity) of security attributes. The last item refers to operator or administrator actions, such as setting up a user account or downgrading information, and any other software in the TCB that is trusted to work correctly.

1.4 What Makes Security Testing Unique?

Functional testing consists of two types: nonsecurity-related functional testing and security functional testing. The bulk of vendor testing is often performed on those functions that are most commonly used (i.e., oriented toward those functions of interest to the end user).

Security functional testing provides a broader, more even, coverage of system security functions. Security functional testing focuses not only on those functions that invoke security mechanisms but also on the least-used aspects of the mechanisms (e.g., overflow conditions), primarily because the least-used aspects often contain exploitable flaws. Sometimes security testing involves testing the boundaries of parameters. Examples of such testing include configuring the system to run with all mandatory levels and categories enabled or examining the results of overflowing the audit log. Security testing also includes negative tests or tests that are designed to show that attempts to violate the security policy are unsuccessful.

1.5 Test Documentation Required Prior to Evaluation

Before an evaluation is begun on a candidate product, all test plans must be completed, including the test matrix documents discussed in the following chapters. In addition, the vendor must provide drafts of the test procedures that prove that significant progress has been made on the procedures and a schedule for their completion to the evaluation team. To complete the evaluation in a timely manner, the vendor is encouraged to complete the test procedures prior to the start of the evaluation and to provide a sample of actual test results. Test documentation found to be significantly lacking in TCB coverage or test construction, such as incomplete test plans, will result in delays in the start of the evaluation process until the open issues have been resolved.

Although the test matrix document, test plans, and test procedures are required documents, only the test matrix document must be delivered in the form and have the content discussed in this document. The vendor may choose the form and content of test plans, procedures, and other test documentation, subject to TCSEC requirements and the decisions of the product evaluation team and the Technical Review Board. However, if the guidance in this document concerning the other test documentation is followed closely, the vendor will have a high likelihood of meeting the test documentation requirements.

Appendix A presents a discussion of test activities that occur over the course of an evaluation. As noted in appendix A, the evaluation team conducts its own set of tests in addition to those contained in the vendor's security test suite. All vendor test documentation and vendor testing must be completed before the start of evaluation team testing.

1.6 Document Organization

Chapter 2 describes the required test documents and introduces a matrix method for constructing security tests. Chapter 3 provides details about how the matrices are constructed and filled in and includes worked examples. Chapters 4 and 5 describe the contents of test plans and procedures, respectively. Chapter 6 provides recommendations for user's guides for test tools. Analysis documents are described in Chapter 7. Appendix A discusses test-related activities that occur during the actual evaluation. Appendix B contains the acronyms used in this document. Finally, appendix C lists the bibliography and references.

This page intentionally left blank

Chapter 2

Overview of Required Documents

This chapter provides an overview of the three required testing documents: the Testing Matrix Document (TMD), one or more test plans, and one or more test procedures. The TMD is essentially an index into the other testing documents. The test plans describe (via test assertions) the items that need to be tested, and the test procedures provide the detailed descriptions of how each test assertion in the plan is tested. In addition, expected test results must be described either in a test plan, test procedure or other document. The relationship between the main documents is illustrated by figure 2.1, and is explained in the remaining sections of this chapter.

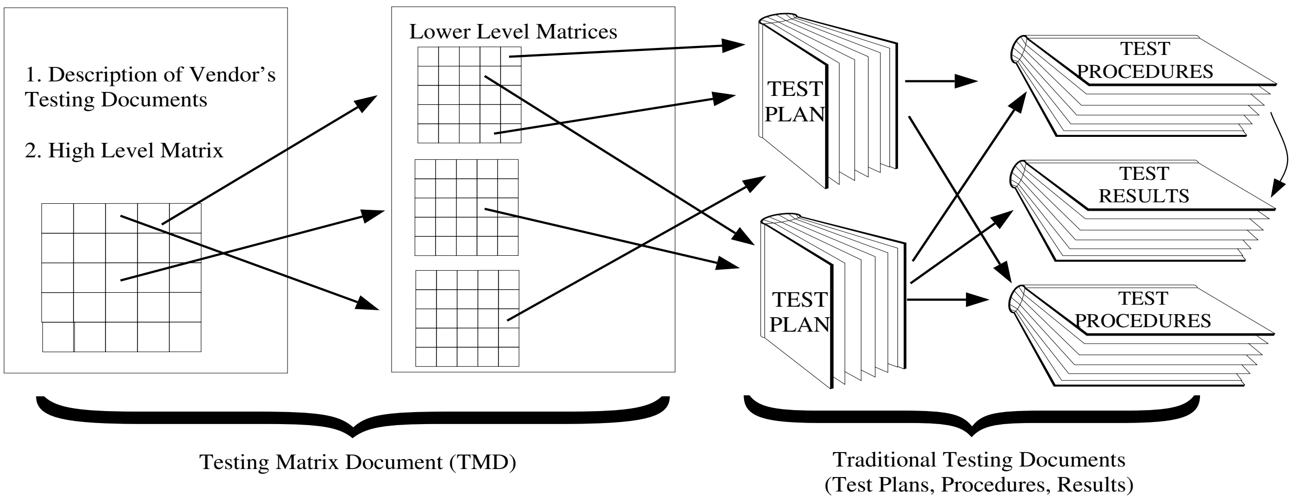

Figure 2.1. Relationship Between TMD, Test Plans, Test Procedures, and Results

The TMD must follow the form and content described in chapter 3. Chapters 4 and 5 logically break the remainder of the required test documentation into two separate documents, the Test Plan and Test Procedures. However, some candidate products may organize the remaining test documentation in a different fashion. For example, a Philosophy of Testing may be included either as part of a Test Plan or as a separate document. Some of the other documents that are commonly used to provide test information are briefly described in section 2.4, Other Documents, and are illustrated in figure 2.2.

In all cases, a discussion of how new tests are integrated into existing test suites and test documents (plans, procedures) and of how the suites and documents are kept consistent with the product must be included in the Rating Maintenance Plan for Rating Maintenance Phase (RAMP).

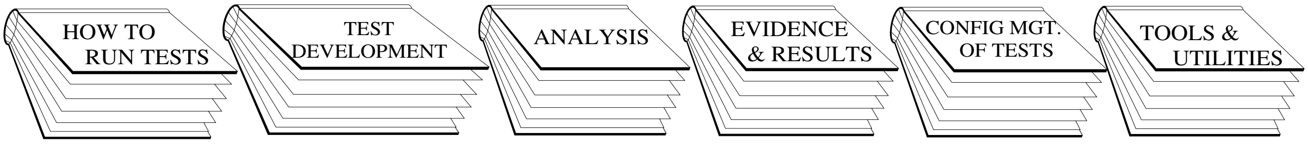

Figure 2.2. Other Possible Testing Documents

2.1 Testing Matrix Document (TMD)

The purpose of the TMD is to organize all testing materials in a standard format. This format allows vendors and evaluators to quickly determine whether each TCB interface is adequately tested and thus whether the product's test plan meets all testing requirements of the TCSEC. The TMD will generally not be the result of a traditional vendor development cycle; rather, it will be done to satisfy TPEP requirements.

The TMD acts as an index to all other testing documents. If a candidate product already has complete test documentation, it should not require significant effort to complete the TMD. If a product has an incomplete test plan, completion of the TMD will allow the vendor to quickly determine the additional information necessary to complete the test plan and to meet the TCSEC testing requirements.

As illustrated in figure 2.1, the TMD starts by describing each item in the vendor's collection of testing documents, including test plans, procedures, guides, analysis, evidence, and any other documents. The remainder of the TMD, which comprises the bulk of the document, contains a large number of matrices and supporting text for the matrices.

There are two types of matrices: high-level coverage matrices, and detailed, low-level matrices. In a given TMD, there is only a single high-level coverage matrix. This matrix gives an overview of how each feature of a candidate product is tested against each requirement of the TCSEC. Each cell in this matrix provides a "pointer" or index to a low-level, detailed matrix. The low-level matrices break each feature down by component TCB interfaces and in turn provide pointers to traditional testing documents that may be organized in any convenient manner.

2.2 Test Plans

The test plan (or plans) describes what is to be tested for the product under evaluation. For each security-relevant TCB interface, the plan has a list of testable assertions and a supporting rationale. The rationale provides the justification for why the list of assertions completely tests the given interface for some TCSEC requirement or requirements. Generally, the plan is based on the design documentation for C2 candidate products and on the informal security model plus the design documentation for B1 candidate products. Chapter 4 describes the form and content of test plans in more detail.

2.3 Test Procedures

The test procedures provide the detailed description of the implementation of each test assertion in the test plan and the information necessary to run each test. Each test assertion from the test plan may be broken down into multiple test cases. Each test case, in turn, is a set of steps performed by a human or automated program. Actual test results may also be included in the test procedures document or they may be in a separate document. Details about the form and content of test procedures are described in Chapter 5.

2.4 Other Documents

Depending on the design of the test plan(s) and procedure(s), some required information may appear in other documents, as illustrated in figure 2.2. These other documents may include guides that describe how test tools are run. A guide that describes how the tests were developed and the methodology for choosing tests may be included. If code or design analysis is used to explain or augment a test, an additional analysis document may be needed. Test evidence must be included that either contains the results of the tests or explains how to find the results of the tests. Finally, if a test suite requires the use of tools or utilities to execute or interpret the test suite, user's guides should be available.

This page intentionally left blank

Chapter 3

Matrix Methodology

The high- and low-level matrices introduced in chapter 2 are the core of the TMD. Each of these matrices is described below, followed by a methodology that can be used to construct them. Numerous example matrices are also shown throughout this chapter.

It is important to understand how the matrices support a product's test coverage argument. The test coverage argument has two aspects: depth and breadth. The matrix method shows that every logical subsystem and interface of the product is "covered" by a test—that is, tested in some manner—and thus forms the basis for a *breadth* coverage argument. The test plans and procedures show that all interfaces are adequately tested and form the basis for the *depth* coverage argument.

The instructions for matrix construction in this section are written assuming a traditional representation of a matrix as a two dimensional array of cells. It is acceptable to employ other representations of a matrix when they more efficiently represented the required information. For example, if a particular matrix contains only a few distinct entries, then a simple list of the entries along with the matrix locations for each entry is acceptable as a matrix description. Vendors that use alternate matrix representations will need to interpret the instructions of this section in the context of their own representation technique.

3.1 High-Level Matrix

The high-level coverage matrix described in this section is the starting point for the technical content of the TMD. It shows that each major logical subsystem of the product is tested for each applicable requirement and tells where to find information about such a test or tests.

Every major logical subsystem of a product from the ISD is listed on the vertical axis, and each applicable requirement class is listed on the horizontal axis, thus forming two sides of a matrix. The subsystems on the vertical axis are product-specific. For a UNIX(tm) candidate product, the subsystems typically include file and memory management and input/output systems. Refer to [12] for more information on identifying logical subsystems.

The requirements classes listed on the horizontal axis of the high-level matrix are taken directly from the set of requirements in the TCSEC that corresponds to the target rating level of the product under evaluation. Some requirements classes may be excluded from the matrix, as described in the methodology section below.

Figure 3.1 illustrates what might appear on the axis of a high-level matrix for a C2 candidate product. The horizontal axis lists each requirement class taken from the TCSEC C2 rating level set of requirements. The vertical axis lists each major logical subsystem of the product being evaluated. Printed in the cell is a reference to a lower-level matrix in the TMD where more detail is provided about tests for the given subsystem or requirement. For example, the matrix of figure 3.1 shows that Section 2.3 of the TMD can be referenced to find more information about testing Discretionary Access Control (DAC) for Interprocess Communication (IPC) mechanisms.[1] The content of these detailed sections is described in the next section.

	DAC	Obj Reuse	I & A	Audit	Sys Arch	TFM
File System						
Memory Management						
Process Management						
IPC	2.3	2.4	2.4	2.4	2.4	2.4
Batch Processing						
Access Mediation						
Identification and Authentication						
Integrity Checking						

Figure 3.1. Example High-Level Matrix for C2 Candidate Product

3.2 Lower-Level Matrices

Lower-level matrices expand the high-level matrix (subsystem, requirement) pair(s) into finer detail. A lower-level matrix is formed as follows: On the vertical axis, each TCB interface that supports the logical subsystem is listed. On the horizontal axis, the applicable TCSEC requirement or requirements are listed.

The example matrices of figure 3.2 show how the high-level cell (IPC, DAC) might be expanded. The vertical axis lists all of the TCB interfaces that support IPC. The figure shows two possible ways to list the requirements. On the right side of the figure, the requirements are broken down into more detail; on the left side, they are taken verbatim from the high-level matrix.

Each cell in the lower-level matrix corresponds to a single TCB interface and a single TCSEC requirement. These may be written as the tuple $\langle$interface, requirement$\rangle$. Printed in each cell of the lower-level matrix is a reference pointer to a list of test assertions; these assertions indicate what must be tested for a given (interface, requirement) combination. The reference must point to a specific section in the vendor document. It is not acceptable to simply point to a multipage document without further reference. An example of an acceptable reference is given for the *shmat* TCB interface[2] in the example matrix on the left of figure 3.2. The example assumes that there is some test plan called the "IPT" test plan; the example shows that details about *shmat*'s DAC tests are given in section 8.7 of the IPT test plan. Whenever test assertions do not exist in a vendor's testing document, they must be written and added to a new or existing vendor's document.

If a logical subsystem contains a large number of interfaces, it may be desirable to create an intermediate level matrix. The intermediate level matrix would list groups of interfaces from the logical subsystem on the vertical axis and contain pointers to lower level matrices that cover each of the interfaces in the group. This amounts to a technique for breaking a very large low-level matrix into a number of parts based on groups of interfaces inside a single logical subsystem.

[1] In this figure, only the row labeled "IPC" is filled in; in an actual matrix, all cells would have entries.
[2] *shmat* is a specific UNIX TCB interface.

	DAC
shmat	IPT 8.7
shmdt	
shmctl	
shmget	
semctl	
semget	
semop	

	DAC (bits)	DAC (ACLs)
shmat	IPT 8.7	IPT 32.1
shmdt		
shmctl		
shmget		
semctl		
semget		
semop		

Figure 3.2. Two Possible Low-Level Matrices for IPC

3.3 Construction of the Matrices

Before attempting to construct the high- and low-level matrices of the TMD, the vendor must be familiar with all existing test documents, in particular, the test plans and procedures (see chapters 4 and 5). Also needed are a list of TCB interfaces for the product and a list of the logical subsystems. These last two items will likely be taken directly from the ISD.

3.3.1 Construction of the High-Level Matrix

Constructing the high-level matrix is straightforward. Simply list each of the product's logical subsystems on the vertical axis of a matrix. A good place to start is from the section or paragraph headings of the ISD [12]. On the horizontal axis of the matrix, list each requirement for the product's candidate class from the TCSEC.

Once an initial set of column headings has been established, the following list may be used to decide which requirement columns may be dropped from the matrix:

- *Documentation Requirements* are met by printed or electronic documents. In most cases, all documentation requirements can be eliminated from the matrix.[3]
- *System Integrity Requirements* refer to diagnostic tests. This requirement can be eliminated from the matrix. However, evaluation teams exercise the diagnostic tests to gain assurance that the test configuration is operating properly and that the diagnostic tests function correctly. System integrity tests can also be used to satisfy aspects of the system architecture requirement (e.g., process isolation). The system architecture requirement is discussed below.
- *Life-Cycle Assurance Requirements* need not be listed in the matrix.
- *Operational Assurance Requirements*, such as system architecture, may or may not be listed in the matrix. System architecture testing is often developed to demonstrate that the TCB executes in a protected domain (i.e., that users outside the TCB domain cannot perform privileged operations or transfer to a privileged state). Another reason to include the system architecture requirement

[3]Note that, in addition to the interfaces described in design documentation, TCB interfaces described in the Trusted Facility Manual (TFM) for use by system administrators are also listed in the ISD and tested to ensure that they work correctly.

in the matrix is the case where TCB interfaces accept parameters that could potentially violate the requirements of domain and/or subject isolation. For example, there may be a way for a subject or user to reference memory outside of its address space (by using absolute addressing or a negative offset). If this is possible, the TCB must be tested through one or more interfaces to ensure that the TCB does not allow a user to use this mechanism to access data that belongs to the TCB or to another user. There is also the concern raised by the system architecture requirement that parameters are passed correctly by TCB interfaces and that error codes returned by internal functions, such as the security decision functions, are correctly passed to the caller. Analysis of the system performed to address the concerns mentioned above determines the appropriate security testing for the system architecture requirement and forms the basis of a rationale that supports the content of the matrix.

Once the initial high-level matrix is formed, the result will look something like figure 3.1. Note that sometimes the same text will appear on each axis. In the example high-level matrix, Identification and Authentication (I & A) appears on the vertical axis because the TCB includes an I & A subsystem with features such as login and password modification. I & A also appears on the horizontal axis because I & A is a specific TCSEC requirement that the product must satisfy at C2 (and above).

DAC	Obj Reuse	I & A	Audit	Sys Arch	TFM

Figure 3.3. Vertical Columns of a Lower-Level Matrix

3.3.2 Creating the Lower-Level Matrices

To begin construction of the lower-level matrices, create one matrix for each horizontal row in the high-level matrix. The vertical columns for each matrix will be similar to those shown in figure 3.3 for each logical subsystem in the product. Next, go through the list of subsystems. For each subsystem, list every security-relevant TCB interface for that subsystem, taken from the ISD. For example, each of the TCB interfaces that support IPC in some candidate UNIX product are listed in figure 3.4. If a TCB interface supports more than one logical subsystem, it must be listed in each of those subsystems' lower-level matrices. Once the TCB interfaces have been listed in the lower-level matrices, you will have a set of matrices, each one similar to the example of figure 3.4.

3.3.3 Filling in the Lower-Level Matrices

The next step is to go through each cell in each lower-level matrix and determine whether test assertions exist for the corresponding (interface, requirement) pair. If an assertion does exist, the location of those assertions is placed in the cell, as shown for (shmat, DAC) in figure 3.5. If the (interface, requirement) pair is inappropriate or not applicable, the cell must refer to an explanation in the test plan that explains why no test assertions are required. If tests have been eliminated through design analysis, refer to the analysis document (see section 7.2, Design Analysis in Lieu of Testing). For example, *shmctl* may be an interface that

IPC Lower-Level Matrix

	DAC	Obj Reuse	I & A	Audit	Sys Arch	TFM
shmat						
shmdt						
shmctl						
shmget						
semctl						
semget						
semop						

Figure 3.4. A First Cut at a Lower-Level Matrix

cannot be used either to obtain or to release an object. Since "IPT 8.8" appears in the cell corresponding to (shmctl, Object Reuse), section 8.8 of the IPT test plan would explain why no test assertions are necessary for this pair.

IPC Lower-Level Matrix

	DAC	Obj Reuse	I & A	Audit	Sys Arch	TFM
shmat	IPT 8.7					
shmdt						
shmctl		IPT 8.8				
shmget						
semctl						
semget						
semop						

Figure 3.5. Partially Filled Lower-Level Matrix

At the end of this process, when each cell in a given matrix has been examined, any cells that are still blank indicate an area of incomplete coverage. To complete the coverage, additional assertions must be written for the (interface, requirement) pair in question. These assertions should be added to a new or existing test plan. Alternatively, if the cell is not filled in, a rationale should be provided to the team that justifies why a particular requirement class does not apply to the interface.

If there are test plans that are organized according to TCSEC requirements, it may be desirable to split the lower-level matrix into multiple matrices, each with one requirement or a portion of one requirement. For example, there may be one matrix that refers only to DAC testing for a set of TCB interfaces (figure 3.2), while a separate matrix refers to Object Reuse testing for that same set of interfaces (figure 3.6). This splitting of matrices may be done at any point during the construction of the lower-level matrix.

Section X
IPC for Object Reuse

X.1. Lower-Level Matrix

	Obj Reuse
shmat	IPT 8.7
shmdt	IPT 8.7
shmctl	IPT 8.8
shmget	IPT 8.2
semctl	IPT 8.3
semget	IPT 8.3
semop	IPT 8.8

Figure 3.6. Detailed Section for Object Reuse

3.3.4 Filling in the High-Level Matrix

Once the lower-level matrices are complete, filling in the high-level matrix is a straightforward task. Each lower-level matrix must appear in its own section of the TMD. For each lower-level matrix, determine the cell or cells in the high-level matrix for which this particular lower-level matrix provides information. Then, place the section number of the lower-level matrix in the cell on the high-level matrix. For example, if it is determined that the lower-level matrix in TMD section 2.3 covers test assertions for (IPC, DAC) and the low-level matrix in TMD section 2.4 covers test assertions for all other IPC cells, fill in the high-level matrix as shown in figure 3.1. If the procedure was followed correctly, there will be no blank cells in the high-level matrix. If there is incomplete test coverage, there will be blank cells in the lower-level matrices but not in the high-level matrix. If there are blank cells in a high-level matrix, a lower-level matrix was omitted during the step of creating the lower-level matrices.

3.4 Trusted Application Interfaces (TAIs)

Vendors are encouraged to identify Trusted Application Interface (TAI)s and provide additional test documentation and results, if appropriate, so that teams may provide more information of interest to system buyers and developers of trusted applications, such as databases, in the Final Evaluation Report (FER). There are three types of TAIs.

The first type consists of those interfaces for which the functions are not thoroughly tested by the security test suite because of evaluated configuration considerations. The most obvious examples are network interfaces that likely return error codes when the system is run in its evaluated (i.e., standalone) configuration. More thorough testing of network interfaces requires physical network connections that may not be considered part of the TCB. Vendors are encouraged to inform the team of the testing activities that are performed to exercise interfaces outside the evaluated configuration. The information supplied to the team could include informal documentation of special test suites and associated test results or, if desired, the matrix approach can be followed to generate test documentation and to develop a test suite. Evaluation teams will reflect their findings in the FER including, time permitting, a spot check of the test suite and the results.

The second type of TAI consists of those system interfaces developed exclusively for use by the trusted applications, such as operator and administrator utilities, that are provided with the system. The interfaces (e.g., libraries) may not have been tested thoroughly, since the interfaces may have been tested only in the context of their use by existing system software. Vendors are encouraged to identify their proprietary interfaces that can be used by developers of other trusted applications. Vendors are also encouraged to test proprietary interfaces separately from the testing via existing applications through the use of the matrix approach. The team will note proprietary interfaces and the extent to which they are tested by the team in the FER.

The third type of TAI consists of interfaces that are intended to be invoked by software running with privileges or user capabilities. The security test suite should already cover "privileged" interfaces. However, the vendor is encouraged to identify the application environment for which each of the privileged interfaces are designed. The information provided to and the tests performed by the team will be reflected in the FER.

This page intentionally left blank

Chapter 4

Test Plan

The Test Plan primarily contains test assertions. Test assertions are statements about the functions of each security-relevant TCB interface that are exercised in the security test suite. Using the matrix approach described in Chapter 3, all security-relevant TCB interfaces are identified and pointers are provided to testable assertions. For each security-relevant interface, a test plan provides analysis that describes why the identified test assertions completely test the security-relevant services provided by the interface.

The following information needs to be provided in the Test Plan and will be discussed in the sections below:

1. Test assertions with analysis
2. List of pointers to test procedures for specific test cases
3. Aspects of the test environment applicable to all test cases
4. Automated tools used with references to descriptions.

4.1 Test Assertions

Test assertions are created by reviewing the TCB design documentation and identifying conditions that are security-relevant, testable, and analyzable. Cells within lower-level matrices point to all test assertions applicable to a single TCB interface that participates in satisfying a single TCSEC requirement. These test assertions are part of the Test Plan.

Although many of the assertions for a TCB interface are derived directly from documentation that describes that particular interface, some assertions may be derived from other design documentation, such as a high-level system description. It is recommended that a reference to the document from which each assertion is generated be provided in the Test Plan in cases where the reference is to a document not already referenced in the ASD or ISD. An assertion reference is helpful to provide traceability and justification for each assertion and the associated tests.

Three examples of test assertions for a UNIX-type system are provided to demonstrate the level of granularity required and the breadth of focus for one set of test assertions. Each example is provided in an attempt to demonstrate that acceptable assertions can take many forms. In Example 1, the symbol $\Rightarrow$ is a reference to a specific test procedure located in a separate document. The cited reference immediately follows one or more brief statements describing TCB behavior to be verified. In Example 2, a textual pseudo-code is provided with a single reference to appropriate test cases; and in Example 3, a structured format is selected for the test assertion mechanism that includes values often found only in test procedures. Each example is a slight modification of an actual acceptable test assertion provided from the test suite for an evaluated product.

Example 1: Test assertions for DAC, Mandatory Access Control (MAC), system architecture functions, and trusted application interfaces for the *creat*(2) TCB interface used to create files.

Features/Options Being Tested:

- A number of tests are performed on the pathname provided to the creat(2) TCB interface to ensure that the calling process has appropriate access to all directories in the path.

  ⇒ Described in table 2-2 of the B test procedure document.

- Verify that memory references passed to the TCB as parameters are checked to ensure that the references are in the calling subject's address space.

  ⇒ Covered in standard test 3, p. 5, of the B test procedure document.

- Verify that the calling process needs DAC write access permission to the parent directory of the file being created. If access is denied, return error code 2.

- Verify that the process needs DAC write access permission to truncate an existing file.

  ⇒ These two are covered in the c.1 standard tests, p. 26, of the B test procedure document.

- Verify that a process needs MAC write access to the parent directory.

- Verify that the process must have MAC write permission to truncate an existing file.

- Verify that if the parent directory of the file being created and process do not have the same sensitivity labels, and the process does not have any MAC-related privileges, a -1 is returned.

  ⇒ The three MAC tests for processes without privilege are covered in the c.2 standard tests, p. 27, of the B test procedure document. MAC-related privilege tests are enumerated in table 4-13 of the B test procedure document.

- Verify that if the directory's label dominates the process' label and the process has MAC-related privileges, the new file inherits the parent directory's MAC label.

  ⇒ This is performed as part of the tests for another TCB interface, in the STM test procedure document, p. 102, and in the c.3 standard tests, p. 27, of the B test procedure document.

- Verify that if the process has the WILDCARD sensitivity label and does not have MAC-related privileges, a -1 is returned.

  ⇒ See standard tests c.3, p. 27, in the B test procedure document.

- Verify that if the process does not have the WILDCARD sensitivity label, the file inherits the process' sensitivity label.

  ⇒ See standard tests c.3 in the B test procedure document.

- Verify that a new file has null potential and granted privileges.

  ⇒ Covered in the c.1 standard tests, p. 56, in the S test procedure document.

- Verify that a new file has a WILDCARD Access Control List (ACL).

  ⇒ Covered in the c.1 standard tests, p. 56, in the S test procedure document.

- Expect error code 3 if the target file exists and is locked.

- Expect error code 4 if the target file is an existing directory.
- Expect error code 5 if the target file is an existing program file that is being executed.
- Expect error code 6 if the process has used up all its slots of open file descriptors.

  ⇒ The preceding 4 tests are covered in the SV test procedure document, p. 5, and the FS test procedure document, p. 22.
- Verify that if the file exists, its length is truncated to 0 and its mode bits and ownership remain the same.

  ⇒ Covered in the SV test procedure document, p. 6.
- Verify that new file's owner is set to the process' effective unique identifier (uid).
- Verify that if directory's set group id bit is not set, the file's group id is set to the process' effective group ID.
- Verify that the file's mode bits are set according to the requested mode bits except that all bits in process' umask are cleared.
- Verify that the "sticky text" mode bit of an existing text file is cleared.

  ⇒ Preceding four tests covered in the FS test procedure document, p. 7.
- Verify that if directory's set group ID bit is set, the file's group ID is set to directory's group ID.

  ⇒ See special test in the B test procedure document, p. 87.

Features/Options Not Being Tested:

- Expect error code 7 if the target file is an existing device file.
- Expect error code 8 if the file system is already full.
- Expect error code 9 if the system-wide open file table is full.

  ⇒ The creat(2) TCB interface shares common code with the fx(2) TCB interface to perform these functions. These assertions are included in the test plan section for the fx(2) TCB interface.

Example 2: Test assertions for the *access*(2) DAC access check when a process requests access to a file.

- If the process has the appropriate privilege, access to the file is granted in the requested mode.
- If the effective user ID of the process is equal to the user ID of the file owner, then the process is treated as the file owner. If the requested access bit is set in the file owner class permission bits, then that access mode is granted; otherwise, access is denied.
- If the effective user ID of the process does not match the user ID of the file owner, access permission is checked in the following order:

  If the effective user ID of the process matches the user ID of a user Access Control List (ACL) entry AND the requested access matches that entry's group access permission bit, then that access is granted.

  If the effective group ID of the process matches any group ACL entry AND the requested access mode bit is set in the file group access permissions and is set in the matching ACL entry, then the access mode is granted.
- If the process is not the file owner or unsuccessfully completes a group match, the process may be granted access to the file if the requested access mode bit is set in the file other class permission bits; otherwise access is denied.

For test procedures, refer to the Test Cases, Binder 4, "Access Control-DAC," Section 4.3.1.

Example 3: Test assertions for the *acl*(2) TCB interface for getting and setting DAC permissions.

- COMPONENT: Discretionary Access Control
- SUB-FEATURE The *acl* system call
- DESCRIPTION: The *acl* call provides for getting and setting discretionary permission information associated with a file. It will allow the file owner or a process with appropriate permission or appropriate privilege to perform the following functions:
  1. Get or set the Access Control List (ACL) information of a file in an atomic operation
  2. Return the number of entries contained in a file's ACL.
- STIMULUS: *acl* is a system call and will be tested by being called within a C program.
- INPUT: The synopsis of *acl* is:

  #include <acl.h>

  int acl(pathname, command, number of entries, acl structure buffer)
- TEST ASSERTIONS:
  - The following three assertions are tested in My-System Test Procedures, Volume 6, DAC, section 1.1.7.

  1. ACL_SET creates or modifies an Access Control List (ACL) and uses the acl structure buffer as input.
  2. ACL_GET retrieves the contents of an ACL by writing into the acl structure buffer.
  3. ACL_CNT counts the number of ACL entries and returns the "number of entries" value.
- For the ACL_SET and ACL_GET commands of the *acl* system call, the acl structure includes the following three fields:
  1. *type*
  2. *ID*
  3. *permissions*
- Twelve values of the type of Access Control List (ACL) will be tested: USER, USER_OBJ, GROUP_OBJ, GROUP, CLASS_OBJ, OTHER_OBJ, DEF_USER, DEF_GROUP_OBJ, DEF_GROUP, DEF_CLASS_OBJ, and DEF_OTHER_OBJ.
- The value of *ID* will be as follows:

  User ID when the value of type is USER or DEF_USER.

  Group ID when the value of type is GROUP or DEF_GROUP.

  The value of *ID* will be unused for all other values of type.
- The permission fields within the acl structure will contain the permissions of the ACL.
- Attempts to set invalid parameters to include type, *ID*, and permission will fail.

4.2 Testing Environment

Before security testing can commence, certain environmental requirements need to be established. Some of these environmental requirements will be constant throughout the testing and should be described in the Test Plan. The Test Plan must clearly identify the version of each TCB software element that has a separate release number. The Test Plan must identify each piece of hardware to be tested by unit and model number. In addition, the Plan must identify the firmware loaded by release number or other identifier that uniquely describes the version to be tested. The Test Plan should also describe any special conditions required to run the security test suite. This could include:

- Editors, compilers, tools, or any non-TCB software that may be required to support testing
- Files that must be present
- Privileged user-ids that must exist and their attributes
- Specific DAC and/or MAC access rights for subjects
- Specific DAC and/or MAC access restrictions on objects
- Hardware that must be installed to run the test suite but is not part of the TCB

All identified hardware will need to be installed and configured and diagnostic tests run to verify that the hardware, as configured, is working properly. The use of any hardware not identified in the ASD to perform security tests is strongly discouraged. Any use of hardware not in the TCB to run the security test suite must be identified to the team. In addition, a rationale must be provided to the team that justifies the use of non-TCB hardware in the running of the test suite.

Once the hardware, firmware, and software of the TCB have been initialized in accordance with the procedures in the Trusted Facility Manual (TFM), the system is said to have reached a secure initial state. All security testing should be done while the TCB is in this secure state. Security testing performed outside the secure state should be noted as such and discussed with the team.

Finally, a user environment must be constructed to perform security testing. Certain users, a predefined file structure, user accounts, privileges, etc., will need to be created. Once this is done, security functional testing can begin.

There are security-relevant functions of the TCB that require an inordinate amount of setup to test or are difficult to reproduce, such as error conditions that arise depending upon the timing of one or more system events. It may be acceptable to omit such error condition testing from the security test suite. However, the vendor should identify all such conditions and state the rationale for omitting the tests from the security test suite. Another alternative is to submit evidence of unit or informal testing of the function in question. With regard to hardware, it is not required that every hardware component be employed in vendor or team testing. However, it is not acceptable to omit hardware components from testing merely because the components may not be available.

The components used for the test configurations should be discussed with the team. The hardware configuration used to run the security test suite should be a representative set of the hardware in the evaluated configuration. For example, the vendor should provide a rationale that justifies the use of a disk drive in the test configuration as sufficient to represent other disk drives in the list of evaluated components.

4.3 Automated Tools

Chapter 6 discusses the use of tools to facilitate security testing. The use of tools is encouraged; however, if tools are used, their use shall be clearly documented in the Test Plan by providing the following information:

- Identification of each tool used, including version numbers, if any
- A description of the purpose of the tool in the security test suite
- Either of the following:
    - A description of how the tool works
    - A reference to a user's manual or other reference describing how the tool works.

This page intentionally left blank

Chapter 5

Test Procedures

The test procedures document represents the detailed documentation for one or more test plans. The purpose of the procedures is to describe how the test plan is implemented. The form of the procedures can be paper or electronic. Regardless of their form, the procedures should give anyone with a working knowledge of the system the ability to understand and execute the entire test suite using the procedures and the supporting documentation. The procedures must not be written such that execution of the test suite requires intimate knowledge of the system or some special expertise.

5.1 Test Cases

The test procedures map each test assertion in the Test Plan to a set of steps or one or more test cases. For assertions that could potentially result in a very large number of test cases, the rationale for determining the number of test cases constructed should be stated in the test procedures. For example, test assertions for an interface that provides for many parameters or many possible combinations of parameter values could potentially lead to an extremely large set of test cases. An argument, based on the vendor's analysis of the implementation, should be formed that supports reducing the set of potential test cases to one that is more representative of the functions of the interface in question.

A test case is a series of steps performed by a human or a program in which the steps can be stated in terms of a chain of inputs, functions, and outputs. With the exception of the actual steps involved in executing the test, test cases are written in prose form. The prose does not have to contain the most minute detail of a test; however, the description must be more readable than pseudocode.

A test case description contains information about all that is pertinent to a specific test. Such information includes the purpose, the execution environment, any dependencies upon other test results, test input, the steps necessary to execute the test, expected results, test output, the procedure for verifying the results, and any cleanup necessary. Each portion of a test case is discussed in more detail in the paragraphs below.

5.1.1 Test Purpose

The test purpose is a short description of the goal of the test. See section 5.2, Examples, for examples of test purpose statements.

5.1.2 Test Setup

There are two types of setup activities: those that establish the global environment for the test suite, and those that establish a particular environment for one or more test cases. The term global environment refers to elements of the test environment configuration that may be the same for the entire test suite, such as the system configuration. The system configuration must be stated in the test plan. In addition, the procedure used to prepare the system for test execution, if one exists, must be documented.

System setup procedures, and those for a particular test case, often require the use of software in the TCB. The setup procedures should reference the TFM for information on how to install and execute TCB software. There is no need to repeat TFM information in the procedures. However, enough information must be provided to successfully execute the test suite. For example, it is not sufficient to simply state that a user account must be created. The specific username for the account must be provided with the system setup procedures (or individual test cases, depending upon the intended use of the account) along with a procedure for the tool or reference to the TCB software (documented in the TFM) used to create the user profile. The specific use of tools must be stated in the system setup procedures or individual test cases (see chapter 6 for information on tools).

Test case setup procedures include items specific to a single test case or a group of tests as opposed to those items that establish the global environment for the test suite. The test setup items must be explicitly stated in the test procedures. Items that are specific to a test case include the following: setting the appropriate classification of objects (e.g., files or devices), creation of special user accounts for testing, use of test tools to perform the setup procedure, inclusion and/or use of special hardware and firmware, and requirements for system boot and/or shutdown. If special hardware or firmware is to be used for a test case, the hardware or firmware must be chosen from the list of evaluated components. If special firmware is used, the loading sequences must be described or referenced.

Other information pertinent to the test case environment often refers to devices, such as checking that a tape is mounted on the proper tape drive, ensuring that the printer has an adequate paper supply, or that certain devices are on/off line. Tasks necessary to initialize a system entity or to perform general housekeeping must also be identified in the procedures. The examples in this section illustrate some of the test setup items.

5.1.3 Dependencies

The success of a test case occasionally relies on the output or successful completion of previous tests. Dependencies between test cases must be clearly stated in the procedures. One of the most common examples of a dependency is an output file that is created or changed by one test and used as input for another. If a test case has no dependencies, the case description must state so.

5.1.4 Input

The input for a test case typically refers to data variables relevant to the test. Perhaps the most common example of a test input is a parameter value to a TCB interface. However, test input must be thought of in a more general sense, including anything used to execute the test (such as data files or tapes). All test input to a test case must be specified in that test case. This can be done in several ways. Specific inputs can be given or a randomization process can be specified, if appropriate, so that actual inputs vary somewhat each time the test is successfully run. A randomization process may create better long-term test coverage.

5.1.5 Procedure

The test case procedure is a list of steps performed during the test. Each test case writeup must provide a level of detail such that it is clear to the reader exactly how to execute the test. If the steps in a test procedure are automated, the actions performed by the test program must still be documented. Most often, every command to be typed by the tester is listed in the test case. In the situation where a test case is automated and the result is not apparent to the interactive user, the procedure can be described in prose rather than in keystrokes (see figure 5.1). On the other hand, if the results of the test are apparent to the interactive user (as may occur in a trusted software test), then the commands issued by the user or via a script must be listed (see figure 5.2). The first two examples, contained in figure 5.1 and figure 5.2, are from the same test suite that used a combination of automatic, scripted, and manual procedures to accomplish security testing. If the test suite is entirely automated, the actual procedures for every test may be generated by the tools used to construct the test suite; therefore, the procedures exist in electronic form. When procedures exist in electronic form, itemizing the test cases may be sufficient as long as there are enough comments provided so that the reviewer can understand what the test is intended to accomplish. However, electronic procedures must be manually verifiable without the need for code analysis. An example of such a test suite is described in section 5.3, Example 3.

5.1.6 Expected Results

The expected results describe the intended outcome of the test. The outcome includes status returned by the TCB interface(s) executed by the test case, introductions of objects into the user address space, creation and deletion of objects, changes in MAC and/or DAC attributes for subjects and objects, audit records generated, and changes in TCB internal data. Expected results must either be generated manually or they may be generated automatically and manually checked to verify they are correct.

5.1.7 Test Output

Test output contains the actual test results and may be in any form, e.g., hardcopy or electronic. Whichever is the case, the test case must specify where the output is located (e.g., the pathname of the results file). The actual results, for example, may be the status message obtained from exercising the TCB interface, the modified attributes of the subject or object of interest, and/or the audit record of the event caused by running the test. Note that it is not necessary to repeat the location of the output for each test that is part of a large group of tests for which output is appended to the same location or file.

5.1.8 Verification

Verification consists primarily of comparison of the test output to the expected results. Verification is usually performed through the use of automated tools or by manual inspection of hardcopy output. When automated tools are used, the comparison method of the tool must be documented (most commonly, a user's guide for the tool will contain the necessary information). For manually generated procedures, each test case must tell the tester how to verify that the actual results match the expected results; the test case description should be explicit about what the tester must look for to verify the results. In addition, the inspection must be a relatively quick and straightforward procedure. The use of tools that aid the tester during test verification is strongly encouraged. Without tools to analyze the results files and/or the audit log, verification will take an inordinate amount of time and may require the additional step of inspecting test suite code to gain assurance that the test suite is functioning properly.

Verification may also include taking steps to ensure that the test was performed as described in the procedures document and/or that the test suite was implemented properly. Test code inspection and keystroke capture are two methods of gaining assurance of the correct performance of the test suite. For example, script tools often provide the capability to echo commands to a log as they are executed by the tester.

5.1.9 Cleanup

Each test case description must mention any cleanup procedures that need to be performed after the test or group of tests has completed. Cleanup procedures include the recording of results files to tape, resetting of devices, dismounting tapes, and terminating of test programs. Test cleanup is important because residual information from previously executed tests may affect the next set of tests to be run. It should be explicitly stated if no cleanup is needed.

5.2 Examples

This section contains examples of test cases and short discussions of the associated test suites. It should be noted that the examples are isolated test cases. That is, the example tests may not adequately test an assertion made in the test plan. Adequacy is assessed through examination of all of the tests for an entity (i.e., more than one test may be needed to test an assertion) and the associated design documentation.

5.2.1 Example 1

The test case contained in figure 5.1 illustrates one test of a TCB interface. The example also illustrates the principle of gray-box testing. The test was performed through the use of a test program that invoked the TCB interface with the appropriate parameters and also performed the required verification (the called routine is available to untrusted software).

5.2.2 Example 2

The test case contained in figure 5.2 depicts a test that exercises a trusted software command. The test was performed through the use of a script that issued the command with the appropriate input. Verification was performed manually by printing the audit file.

Note that the login procedure was not scripted. Knowledge of the login procedure is an example of a valid assumption of the tester's expertise.

TEST NAME get_device_status_01

Test Purpose

The purpose of this test is to ensure that if the device is not readable in the mandatory sense by the test process, the attempt will fail. Thorough MAC testing has been done in the get_device_access test, so it will not be repeated here. Therefore, it is necessary only to exercise single success and single MAC-related failures to demonstrate that this function calls the mandatory_access_allowed function correctly.

Test Dependencies

None.

Test Input

None.

Test Setup and Procedure

The following steps will be performed for this test procedure. The device was previously created and pre-established with the proper security attributes. *(For this test, the device was a terminal which was specified for a group of related tests in introductory text to the section of the test procedures from which this example was taken; the device attributes were specified as part of the system test setup in an associated test user's guide.)*

a. The test executive must set the device access level higher than the access level of the calling process. This will cause the attempt to read the device to fail because the subject access level must be higher than or equal to the object access level.

b. The test executive initiates the calling process to attempt the get_device_status function.

c. Upon return from the test process, the test executive will record the results and write a test audit record for future verification.

d. The test executive will then set the device access level lower than the access level of the calling process, to produce a successful get_device_status call.

Test Output and Expected Results

When a calling process is not properly authorized, get_device_status returns the error condition DEVICE_DOES_NOT_EXIST to the test executive. Because the attempted get_device_status failed, the system should have generated a DEVICE_ACCESS_VIOLATION audit record.

For the combination of access levels for which the get_device_status is expected to work properly, device type, device class, and physical device number will be returned.

Test Verification

The results of this test are verified by comparing the actual error return with the expected error return (DEVICE_DOES_NOT_EXIST). In addition, the system audit file will be examined to ensure that the expected audit record matches what was generated.

For the situation in which a successful get_device_status is expected, a comparison will be made between the expected returned parameters and the actual returned parameters.

Figure 5.1. Test of a Programmatic TCB Interface (Gray Box Test)

TEST NAME cup_01

Test Purpose

The purpose of this test is to exercise the cup command. A user who has change user password capability provides the valid user password and enters the same new password twice.

Test Setup

Create user01 with password user01.

Test Dependencies

None.

Test Input

None.

Test Procedure

Login as user01, Password: user01

(T is scripted test program; S is system response)

```
T:     <System Attention Key>
S:     Enter command
S:     ?
T:     cup
S:     Enter old password
T:     user01 (password will not appear on screen)
S:     Enter new user password
T:     test01 (password will not appear on screen)
S:     Enter new user password again
T:     test01 (password will not appear on screen)
S:     User password changed.
```

Test Output and Expected Results

An audit record indicating AUDIT_CUP will be generated and the user password will be changed.

Test Verification

Ensure the above interactive dialogue. Login as user01 to verify that the password has been changed. The system audit file will be examined to ensure that the expected audit record matches that which was generated.

Figure 5.2. Test of Trusted Software Command

The tests in the test suite are based primarily on the informal security model which specifies the TCB objects and the legal operations that may be performed on each object given the appropriate access. Given the number of objects and the possible permutations of TCB interface parameters and access, it became essential that the security test suite be automated.

Two tools, a test manager and a test generator, were used to construct the tests. The test generator is actually a set of utilities that allow the user to create test scripts and separate programs to execute system service calls or commands. Data files containing the service or command to be executed, along with the specified parameters, are edited and used as input to the test generator. The test generator is also used to generate sequences of commands that perform setup, verification, cleanup, code management, installation, and performance analysis for each test.

When a test has been run for the first time, the log files are inspected manually to be sure the test performed as expected. Once the expected results have been obtained, the log files are used to create benchmark files used by the test manager to compare to the log files generated after each successive run of that test.

Tests are then grouped into test sets. The test manager creates a monitoring program for each test set. The monitoring program executes and monitors the progress of each test as it runs. When the last test has completed, the test manager records the status of each test in a log. To coordinate execution of the entire test suite, there is a menu-driven program that is used by the tester to choose which sets of tests are to be run.

In this example, the "procedures" are the tests that are generated by the test generator. The log files contain the results and provide information enabling the tester to trace back any particular test to the file containing the generation code.

Figure 5.3. Description of Fully Automated Test Suite

5.2.3 Example 3

The test suite discussed in figure 5.3 does not illustrate a test case but rather discusses a fully automated test suite for which the procedures are "created;" that is, the entire test case is in electronic form. The test plan described the security functions tested; however, no formal test procedures document was written for the test suite. Additional test suite documentation was provided by the vendor that described the implementation of the test suite, including which tools were used, which files were used for input and output, and how the results were verified. Documentation that described each specific tool was also provided and the associated files, scripts, and logs were made available to the evaluation team for inspection.

5.2.4 Example 4

The test cases shown in figure 5.4 describe the test procedures for a UNIX command, umask, which is part of the C shell. The umask command is used to set the file-creation mode mask.

TEST NAME umask

Test Purpose

The purpose of this test is to exercise the umask command with a range of accesses to demonstrate umask functions.

Test Setup

None.

Test Dependencies

None.

Test Input

None.

Test Procedure

A check is made to determine whether the file "f.TC" exists. If so, it is deleted. The file is then created. The umask command is tested by setting the umask variable to 0 and then to a walking 1 bit pattern (000, 001, 010, etc.) and to 777 and then to a walking 0 bit pattern (776, 666, 555, etc.), as noted below. A test file is made and the mode is checked. Some invalid umask values are set at the end of the test. In addition to the return from umask of "000," an "improper mask" message is received. There are 22 test cases listed in the table below. For each test case, the commands shown in the "description" column are executed.

Test Output, Expected Results, and Verification

To verify the results of the test, the following command is used to examine the permission mode after the test commands have been typed:

ls -l f.TC

The output from this command is compared to the results in the "expect response" column, and if the permission modes match, the test passes. Otherwise, the test fails.

test#	description	expect response
1	umask 000; touch f.TC	-rw-rw-rw-
2	umask 001; touch f.TC	-rw-rw-rw-
3	umask 010; touch f.TC	-rw-rw-rw-
4	umask 100; touch f.TC	-rw-rw-rw-
5	umask 002; touch f.TC	-rw-rw-r--
6	umask 020; touch f.TC	-rw-r--rw-
7	umask 200; touch f.TC	-r--rw-rw-
8	umask 004; touch f.TC	-rw-rw--w-
9	umask 040; touch f.TC	-rw--w-rw-
10	umask 400; touch f.TC	--w-rw-rw-
11	umask 124; touch f.TC	-rw-r---w-
12	umask 142; touch f.TC	-rw--w-r--
13	umask 412; touch f.TC	--w-rw-r--
14	umask 777; touch f.TC	----------
15	umask 666; touch f.TC	----------
16	umask 555; touch f.TC	--w--w--w-
17	umask 444; touch f.TC	--w--w--w-
18	umask 333; touch f.TC	-r--r--r--
19	umask 222; touch f.TC	-r--r--r--
20	umask 111; touch f.TC	-rw-rw-rw-
21	umask 888	umask = 000
22	umask aaa	umask = 000

Figure 5.4. Test of umask

This page intentionally left blank

Chapter 6

Guide for the Use of Tools

This chapter provides general guidance on the appropriate use of tools for security testing for a TCSEC candidate product. Security functional tests must be designed so they can be repeated (i.e., that the results obtained from each run match the expected results). Input sequences need to be accurate, the TCB module tested needs to perform as expected and it must be possible to clearly determine the outputs that were derived from specific inputs. In general, individual tests should not be dependent on other tests.[1] Destructive tests should not be followed by other tests because TCB data structures and/or the user environment may need to be reconstructed after the destructive test completes.

If security functional tests meet the above criteria, it is often possible to completely automate all or a portion of the tests for parts of the TCB. The use of tools to automate as much of the security testing effort as possible is encouraged as long as the tools used are well-understood, well-documented, and all outputs are manually verifiable.

Automated tools should be under configuration management control so that their functions are documented and clearly understood.[2] Automated tools that are used generally fall into one of five categories:

1. Typical software development tools used during security testing
2. Automated user/operator interaction
3. Emulated terminal interaction
4. Functional test automation
5. Automated test generation

6.1 Software Development Tools and Testing

Software development tools that are used during the security testing of the TCB must be identified. The vendor must demonstrate that the functions performed by the software tool have no effect on the functionality of the TCB code being tested. Editors and compilers are software development tools that have been used during security testing.

Editors may be used to modify input control files for initial setup and/or configuration, input control files to control the flow of a test procedure or groups of tests, and input data files for individual tests. Editors also have been used for post-processing test results from data files as well as the audit log.

[1] This is not meant to restrict test suites in how they are structured. Automated test suites, in particular, can be structured so that the results of previous tests are relied upon by successive tests for more efficient performance reasons.

[2] Automated tools must be under configuration control to satisfy RAMP requirements.

Compilers are used to compile the TCB and tests to prepare for test execution. If the evaluated product is to be distributed to customers as object code, the compiler used to create that object code needs to be the same one used to compile the TCB for testing. If the TCB is distributed as a source product, the compiler used to create the TCB object modules for testing needs to be one compliant with a recognized standard, if applicable, and commercially available or distributed as part of the evaluated product.

The use of tools such as interactive debuggers that require modification of the TCB to install is strongly discouraged. If such tools are used in the security test suite, then a justifying argument must be provided to the team. The argument must explain how the integrity of the test suite and the TCB is maintained with the addition of the tool.

6.2 Automated User/Operator Interface

Tools that provide a scripting capability are used to simulate host-user interaction and help eliminate human error during testing. Test scripts can be designed with all commands, command options, and inputs to test programs planned in advance. Thus, persons analyzing test results are assured that inputs do not vary for multiple test executions.

The system administrator and operator interface tests should be automated whenever possible. This may allow the process of initializing the system to be automated. Using an automated system initialization process ensures a consistent TCB that has specific code modules and data structures for security testing. Automating system initialization also provides an explicit test for the TCB initialization procedures described in the TFM. In addition, operator interaction with running functional tests often can be automated. This provides test implementors the ability to correctly anticipate every operator action.

When automated test scripts are used, test plans must include a description of the tool, its operation, and its general setup. The following information must be provided in the test procedures:

- Security functions being tested
- Name of the input and output files used by the tool
- General discussion from the user's guide
- Test case specific files in the test procedures
- Listing of the input file (scripts)
- Procedures necessary for specific test cases

6.3 Terminal Emulators

Often a vendor can connect two hosts through a communication interface such that one host runs an application program called a terminal emulator, which emulates a certain number and type of connected terminals to the other host. Such an emulator often provides all the controls on a logical terminal available for a physically connected terminal. Terminal emulation programs may provide for user scripts to automate user input. If a terminal emulator is used during security functional testing between hosts, the vendor must be able to demonstrate that the host executing the tests is executing the TCB (hardware, firmware, and software). Only terminal types that are in the evaluated configuration may be emulated. The vendor must understand how the emulator performs and provide a convincing argument to the evaluation team that all security-relevant actions performed by the emulator are identical to those performed by the actual terminal type(s) being emulated.

If the terminal emulator is running in the same host as the TCB, the vendor must demonstrate that all the restrictions above are met and that the terminal emulator is not performing security-relevant actions. Typically, if it can be demonstrated that the terminal emulator is running without user-level privileges (e.g., DAC Read Bypass, System Administrator) and in an unprivileged hardware state (e.g., problem state, user-level) then there is a reasonable argument that the emulator does nothing security relevant. However, if the emulator is running in an unprivileged hardware state but with user-level privileges and/or the emulator is executing in a privileged hardware state, it may be difficult to argue effectively that the emulator is not performing security-relevant actions that will compromise the results of any test of the TCB. If the system is using type enforcement, then an emulator may run in a domain normally reserved for a device or the devices being emulated. In this way, an unprivileged emulator can be used while maintaining the assurance that it can only perform the same actions as those for a real device.

6.4 Functional Test Automation

It is possible and often desirable to completely automate security functional testing or at least the testing of some part of the TCB. Input files can contain scripts, operator commands, and test case inputs. The executing sequence of tests is controlled through an input file or coded in the automated test driver. Outputs are written to files and/or the audit log for post-processing analysis that can be automated as well.

When entire TCB products/modules are tested through completely automated tests, the vendor must provide information that describes test suite implementation details. The vendor must provide the following information in the test plan:

- A description of the automated tool(s) used, including applicable information for the other types of tools previously described (if, for example, a terminal emulator is part of the fully automated test suite).
- Information that provides assurance the tool works as described.

In addition, the test procedures must include all information necessary to run the tests, including the command sequence for the activation of the test suite and the keystroke sequences for manual tests, if appropriate. Finally, the vendor must be able to take selected test scenarios, perform each step manually, and produce identical results with the same inputs.

6.5 Automated Test Generation

Fully automated test generation requires the use of machine readable specifications in a formalized language. These specifications describe both expected product behavior and expected test inputs. The description of expected inputs amounts to a machine-readable codification of the developer's test philosophy. It may tell exactly what the test inputs are, or it may specify a distribution that is used to generate one or more random inputs.

The specifications are used as input to a test-generation tool that creates test scenarios (e.g., in the form of C programs). The specifications may be augmented with "constraints" that limit the number of test cases; this is clearly necessary in situations where exhaustive testing would produce an unlimited number of test cases. All tools and specifications used in test generation must be documented.

Chapter 7
Analysis

There are at least three different types of analysis associated with the development of a security test suite. The first type of analysis performed eliminates redundant tests. That type of analysis is typically included in the rationale that describes how the contents of the test suite were chosen. A second type of analysis is performed in lieu of completely testing some functions. The third type of analysis is that needed to ascertain the outcomes of the security test suite. Typically, analysis associated with the test suite is described in the test plan. However, the analyses mentioned above may require significant effort and detail. Separate analysis documents may be needed.

Some types of analysis must be completed prior to the start of the evaluation to support the approaches and strategies in the test plans and procedures. For example, an analysis team may recommend that logic flow analysis be performed and documented (either in updates to existing test documentation or in a separate document) on portions of the TCB to support a grey-box testing approach. It is also possible that additional vendor analysis will be needed after the team has come to understand the security test suite and the vendor's other testing activities and perhaps found some areas lacking in coverage. In that case, additional analysis documents, if needed, will be completed during the evaluation.

7.1 Test Reduction Analysis

Occasionally, it is possible to reduce or eliminate redundant tests through design and/or code analysis. As an example, consider a system in which there are two or more methods of interfacing to the TCB, all of which result in calls to system services. Analysis may be used to determine which interfaces merely parse parameters and formulate calls to lower-level system services. In such cases, it may be acceptable to thoroughly test only the system services and to perform significantly less testing on the alternative interfaces.

Another example is analysis that demonstrates that MAC checking is performed in one function in the system and that every object access invokes that function. Thorough testing of the MAC implementation may be performed once. Thereafter, testing to be sure each initial object access does in fact call the MAC function correctly may cover the remainder of MAC testing.

Design analysis must be briefly described in the test plan or in the TMD. The analysis must argue that the parse and formulate actions of the untested TCB interface are not tested thoroughly. Analysis may be performed with the aid of a tool, the output of which can be studied by the evaluation team. The evaluation team will perform its own analysis and/or testing to confirm the vendor's analysis.

7.2 Design Analysis in Lieu of Testing

There may be several portions of a candidate product that are difficult to test because the conditions that cause execution of the code may be very difficult to reproduce or require significant amount of overhead to create. Such portions include object reuse scenarios (e.g., page clearing upon deallocation). The evaluation team and vendor may reach an agreement on particular areas of the TCB that are difficult to test. Alternatives to actual testing need to be discussed.

The details of the design analysis employed to determine alternative methods of testing for some portions of the candidate product need not be presented to the team. However, the team may spot check the vendor's analysis or use one or two examples provided by the vendor to understand how the analysis was performed. The vendor is encouraged to include the results of analysis in the test documentation, particularly if an agreement between the team and the vendor was established regarding the use of such analysis.

7.3 Analysis of Test Results

The documentation of test results was discussed briefly on section 5.1.8, Verification. In the case where the output of the security test suite is voluminous, vendors are encouraged to write a tool that can collect and filter the relevant results into a form such that the outcomes of each of the tests can readily be discovered. A results analysis tool can benefit both the vendor and the team in this case. In addition, it may be convenient for the vendor to include the output of the analysis tool as part of the test documentation supplied to the team.

Appendix A

Test-Related Activities in Evaluation

This appendix provides a description of typical testing activities and milestones in the evaluation process and relates test documention to these process components. It is provided in an effort to further clarify the nature and role of test documentation. It is not the role of this document to define the evaluation process and thus the inclusion or omission of an activity or milestone here should not be construed as a constraint on current or future evaluation processes.

A.1 Overview of the Test-Related Activities

Before an evaluation is scheduled by the National Security Agency (NSA), a preliminary review of the product is performed to determine if the vendor and the product have a high probability of successfully obtaining the target TCSEC class. The preliminary review is performed by a team of senior evaluators who will, in relation to testing, determine whether test plans and test procedures exist. The following test-related documentation must be provided to the preliminary review team:

- The TMD, including high- and low-level matrices
- Test plans
- Test procedures, including actual test cases
- Potential hardware configurations.

In addition, to satisfy RAMP requirements, the RM-Plan must discuss configuration management of test plans and procedures.

Beginning with the preliminary review effort and continuing through the evaluation, evaluators must determine if the vendor has sufficiently tested the TCB. Evaluators make this judgement based upon the test documentation, including the TMD, test plans and procedures, and *test evidence* provided by the vendor.

There are two primary purposes for test evidence. First, it serves as proof that the security test suite was successfully run on the system in the evaluated configuration. Second, it shows how the results from the test suite are interpreted to verify that the tests did what they were supposed to do.

Test evidence includes printed materials, documents in electronic form, and demonstrations of tests performed by vendor personnel. Regardless of the form, the evidence must be something that is readily analyzable and understandable by the team. For example, if the raw output from a test suite is large, it will need to be organized or reduced so that the evaluation team can quickly examine the pertinent results. However, the full output may still be required for reference by the team.

At the preliminary review, sufficient test material needs to be presented to show that the vendor's test-related activities are well underway. All test plans must be completed. The test procedures do not all have to be completed by the time of the review. However, the vendor must demonstrate that development of the procedures is in progress by providing existing drafts and a schedule for completion of the procedures. If actual tests have been developed and run on the system, the vendor is encouraged to present the results to the team who will spot check them during the preliminary review.

The test documentation needed for the preliminary review includes most of the printed materials illustrated in figures 2.1 and 2.2, possibly with some documents in electronic form. The preliminary review team will analyze the test documentation to be certain that the TCB test coverage is adequate.

During an evaluation, the vendor provides TCB hardware, firmware, and software for the team to run the vendor test suite as well as tests created by the team. Typically, the system is installed at a vendor location; however, any location agreeable to both the evaluation team and the vendor is acceptable. Evaluation teams must have dedicated access to the complete TCB. The following are the major testing activities that are performed during the evaluation:

- Test Coverage Analysis (TCA)
- Test Readiness Review (TRR)
- Initial Product Assessment Report (IPAR)/Test Technical Review Board (TRB) presentation
- Design and implementation of supplemental team tests
- Verification of TCB initialization
- Verification of vendor test results
- Reevaluation of the vendor test suite
- Completion of a team test report
- Final TRB.

A.2 Test Coverage Analysis

Shortly after the evaluation begins, the evaluation team will perform a Test Coverage Analysis (TCA). The purpose of this analysis is to determine whether the TCB test suite has sufficient tests to demonstrate that all applicable requirements are being met. The TCB is sufficiently tested when each security-relevant TCB interface is tested to determine that it functions as specified in design documentation. The matrix approach described in chapter 3 details a methodology that helps to verify all identified interfaces are tested. This approach facilitates the test coverage analysis process, since it provides a tool for the evaluation team to determine if all security-relevant TCB interfaces have been tested. Before the evaluation team can complete a test coverage analysis, the vendor must have completed the following:

1. Test plans and test procedures
2. Process of using the matrix approach to identify all security-relevant TCB interfaces (i.e., a complete and accurate TMD)

3. Analysis on identified TCB interfaces to remove multiple occurrences of the same interface used to provide the same service (reduces the number of tests).

If the vendor writes each of the three items listed above carefully, there will be reasonable assurance that all security-relevant actions at the TCB interface are tested and that the Test Coverage Analysis (TCA) will be successful.

During the TCA, the evaluation team may determine that the vendor test suite is incomplete in one or more areas. Teams follow up the review of the test suite with vendor discussions about any test suite incompleteness. If the team determines that test suite is significantly deficient, the Initial Product Assessment Report (IPAR)/Test Technical Review Board (TRB) and the team testing activity will not be scheduled until the test suite incorporates identified missing tests.

A.3 Test Readiness Review

An evaluation team may require a Test Readiness Review (TRR). Typically, the TRR occurs within the two weeks prior to the scheduled team testing period. The TRR bears no direct relationship to the IPAR/Test TRB. The TRR may occur either before or after the TRB.

Two team members (typically the team leader and the test coordinator) will arrive at the location where the TCB hardware has been installed. The purpose of this visit is to determine that everything is ready for security testing before the entire evaluation team begins to travel. The items that must be presented by the vendor to complete this review are typically the following:

- TCB hardware configured and operational in the evaluated configuration and hardware integrity tests completed
- TCB software ready for system initialization
- TCB initialization procedures
- The Trusted Facility Manual (TFM) and the Security Features User's Guide (SFUG)
- The TMD, test plan, and all test procedures
- Test tool documentation
- One complete set of design documentation.

The TRR is the final hurdle prior to actual system testing by the evaluation team. During TRR, test evidence must exist to show the evaluation team that the candidate product and its associated test suites are in a suitable form to begin formal testing. The vendor must bring the product up in the evaluated configuration, be prepared to run all of the system tests, and explain the results of the output from the tests.

A.4 IPAR/Test TRB Presentation

Once the evaluation team understands the TCB implementation, an IPAR is written. This report is presented to a TRB to ensure that the team understands in detail how TCB mechanisms work and how the TCB mechanisms are used to satisfy the TCSEC requirements for the candidate class. Additionally, the team must present an argument that the TCA indicates that the vendor has sufficiently tested the system, a completed test plan for team-developed tests, and the conclusions of the TRR, including an indication that the vendor's security test suite is ready to be run by the evaluation team.

A.5 Design and Implementation of Supplemental Team Tests

As part of the evaluation team's security testing, the team develops and runs tests in addition to those supplied by the vendor. The purpose of the team's tests is to uncover potential implementation flaws resulting from unusual or system overload scenarios that may not be discovered by the matrix approach. Some historical examples of evaluation team tests are listed below:

1. Testing all possible system call values to initiate a change in the hardware privilege level and to determine if any undocumented functions exist
2. Overflowing the audit log to determine how much audit information can be lost and how the system recovers
3. Defining the longest possible label and checking to see its effect on banner and page labels
4. Identifying specific boundary conditions (legal or illegal) for a particular TCB interface
5. Creating and implementing scenarios suspected by the team to cause system problems (the scenarios usually derived from portions of the design documentation or system responses perceived to be vague or ambiguous)

The team tests are not typically written to address incomplete portions of the vendor's test suite. However, there are at least two examples where team tests may be incorporated by the vendor. The results of team tests may indicate that additional vendor testing is required. For example, if previously unknown aspects of system interfaces are discovered, the vendor will be required to document the identified functions and update the test suite appropriately. The team may, as part of the actions required to address incomplete testing, recommend that the vendor incorporate the team tests directly into the security test suite. Team-supplied tests that uncover flaws in the TCB may also give rise to test suite modifications. In addition to requiring the vendor to fix the flaw, the team will often recommend that the test (or a similar one written by the vendor) be incorporated to be sure that the fix is retested each time the suite is executed. If flaws are discovered, the team will return for a retest period and rerun selected team tests or the updated portions of the vendor test suite, if appropriate, on the updated system.

Team testing usually requires two to three weeks of dedicated time. In addition to the use of the systems, vendors should plan to provide adequate resources for the team, possibly to include sets of design documentation and office and/or conference room facilities. The team may also need access to system developers or to support personnel for assistance. Once the team has the TCB operational, vendors should only allow other users on the system with the understanding that the team will most probably be running team tests designed to test boundary conditions of TCB interfaces with illegal values or conditions and thus system failures are very possible. No modifications to the system should be allowed while it is being tested by the team. If modifications are made, the team must be able to verify that the evaluated configuration was restored to a secure state prior to further testing.

A.6 Verification of TCB Initialization

The vendor must demonstrate that the process of putting the TCB into operation is reproducible and clearly documented and that each step is unambiguous. At the beginning of the team testing period, the evaluation team will first check that the system is in its evaluated configuration. All of the hardware components used must appear on the list of evaluated hardware known to the team (from the ASD). The team will then perform the steps necessary to install the TCB software and place it into operation, following only the steps described in the TFM. The evaluation team will act in the role of customer to verify that a customer can step through the complete process of taking software as it is shipped and, following the documented steps in the TFM, place the TCB into operation. If the evaluation team cannot successfully bring up the TCB without vendor assistance, the team and the vendor will need to identify the problem. If modifications are required to the TFM because it is incomplete or becasue of test dependencies or TCB modifications, the TFM must be updated so that all customers have a reasonable chance to accurately install the TCB. Once the evaluation team is successful, all tests performed by the team will be conducted on the TCB they initialized.

If the vendor custom configures the system for each customer, it is not necessary for the team to perform TCB initialization. However, even in this case, it is still useful for the team to have direct experience in bringing up the system.

A.7 Verification of Vendor Test Results

The evaluation team will run the vendor's complete security test suite to verify the results provided by the vendor. An automated security test suite facilitates this process; however, if an automated test suite was used by the vendor, the evaluation team must verify all inputs, processes executed, and outputs to ensure that all security-relevant TCB interfaces have been sufficiently tested and that the tests run correctly.

All vendor tests run by the team will use the same test descriptions used by the vendor when performing security tests. The team should be able to run the entire test suite without assistance from the vendor as a demonstration that test documentation is clear and complete. Minor assistance may be required by the evaluation team in the operation of a specific hardware device.

If any tests produce unanticipated results, the vendor must explain the function of the test, the mechanism being tested, why the results were produced and identify whether the error is in the security functional test or in the TCB. The test and/or the TCB must be fixed so that results are predicted correctly, and a successful test can be rerun by the team during a final test session.

The vendor must demonstrate that for each modification to the TCB implementation, a correct modification was generated for the applicable tests as well as the appropriate design document. This information will be used by the evaluation team during retesting efforts. The only acceptable modifications to the test procedures and the system are those changes made to address problems identified by the team.

A.8 Reevaluation of Vendor Test Suite

The evaluation team may uncover implementation errors in the test suite. Once the team has identified all potential problems discussed in the previous section, the vendor and the team discuss the team's findings. The result of this discussion is an agreement that identifies necessary changes to the test suite. Once the vendor completes all necessary changes, the evaluation team and the vendor will schedule a retest session where additions and modifications to the test suite and the system are retested by the team. Any security functional test could be rerun during the retest; however, teams may focus only on modified and added tests or those that uncovered TCB flaws. Analysis and justification can be used to subset the portion of the test suite that must be rerun.

A.9 Completion of a Team Test Report

Once the team completes security testing of the TCB, the team completes a report that provides the following information:

- Description of the logistics involved with and activities performed during the testing effort

  Reference to the environmental requirements in the Test Plan

  Reference to the TCB initialization document

  Reference to the test procedures

- Explanation of the choices for team tests with reference to the results of the TCA activity and other sources
- Description of team tests performed
- Results of the security tests and team tests

  Problems encountered when the team initialized the system

  Problems associated with running vendor and team security tests

  TCB flaws uncovered

  Incomplete tests uncovered

  Determination if any retesting effort is needed

  Action items for the vendor and the team before retesting will be scheduled

- Lessons learned from the testing activity for future evaluation teams
- Conclusions

The contents of the report are discussed between the evaluation team, the vendor, and NSA management concerning the types of test failures encountered. A final version of the report is delivered to NSA, but it is typically not provided outside NSA for public examination.

A.10 Final TRB

After the evaluation team has completed their evaluation of the product design and security testing of the product, a draft FER is completed. To complete the draft FER, the team updates the IPAR to address outstanding TRB issues, to correct technical inaccuracies, to summarize the security testing activities, and to add information obtained from the RAMP Audit [4] conducted by the team. The draft FER is submitted to a Final TRB. Before the Final TRB will recommend assignment of a rating, the TRB will review the FER and the results of security testing to determine that it was properly conducted by the team and that the results meet the Security Testing assurance requirement in the TCSEC.

This page intentionally left blank

Appendix B
Acronyms

ACL	Access Control List
ADP	Automated Data Processing
ASD	Architecture Summary Document
DAC	Discretionary Access Control
FER	Final Evaluation Report
I & A	Identification and Authentication
IPAR	Initial Product Assessment Report
IPC	Interprocess Communication
ISD	Interface Summary Document
MAC	Mandatory Access Control
NSA	National Security Agency
PGWG	PAT Guidance Working Group
RAMP	Rating Maintenance Phase
SFUG	Security Features User's Guide
TAI	Trusted Application Interface
TCA	Test Coverage Analysis
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Criteria
TFM	Trusted Facility Manual
TMD	Testing Matrix Document
TPEP	Trusted Product Evaluation Program
TRB	Technical Review Board
TRR	Test Readiness Review
uid	unique identifier

This page intentionally left blank

Appendix C

Bibliography and References

[1] *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.

[2] *National Computer Security Center Trusted Network Interpretation*, NCSC-TG-005 Version 1, 31 July 1987.

[3] *A Guide to Understanding Security Testing and Test Documentation in Trusted Systems*, National Computer Security Center, NCSC-TG-023, July 1993.

[4] *C2-B1 RAMP Requirements*, Transaction [233] from Announcements Dockmaster Forum, 19 December 1991.

[5] Beizer, B., *Software Testing Techniques*, New York: Van Nostrand Reinhold, 1990.

[6] Woit, D. M., *An Analysis of Black-Box Testing Techniques*, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, May 1992.

[7] Special Section of Software Testing, edited by Richard Hamlet, *Communications of the ACM*, June 1988, Volume 31, Number 6.

[8] Haley, C. J., and Mayer F. L., "Issues On The Development Of Security Related Functional Tests," in Proc. DOD NBS Conference for Computer System Security, National Bureau of Standards, Gaithersburg, MD, September 1985.

[9] *Guideline For Lifecycle Validation, Verification, and Testing of Computer Software*, National Institute of Science and Technology Report, FIPS PUB 101, pages 1-37, Washington, D.C., June 1983.

[10] Gligor, V.D., et al, "A New Security Testing Method and Its Application to the Secure Xenix Kernel," *IEEE Transactions On Software Engineering*, Vol. SE-13, No. 2, February 1987.

[11] Myers, G.J., *The Art of Software Testing*, New York: John Wiley and Sons Inc., 1979.

[12] *Process Action Team Guidance Working Group, Form and Content of Vendor Design Documentation*, National Computer Security Center, May 1994.

[13] *Final Evaluation Report, Verdix Corporation, VSLAN 5.0*, CSC-EPL-90/001, July 1990.

[14] *Final Evaluation Report, International Business Machines Corporation, MVS/ESA Operating System*, CSC-EPL-90/002, September 1990.

[15] *Final Evaluation Report, Wang Laboratories, Incorporated, SVS/OS CAP 1.0*, CSC-EPL-90/004, September 1990.

[16] *Final Evaluation Report, SecureWare, Incorporated, Compartmented Mode Workstation Plus*, CSC-EPL-91/002, January 1991.

[17] *Final Evaluation Report, HFS Incorporated, XTS-200*, CSC-EPL-92/003, May 1992.

[18] *Trusted Product Evaluation Questionnaire*, NCSC-TG-019, May 1992.