Handbook for the Computer Security Certification of Trusted Systems

NRL Technical Memorandum 5540:082A, 24 Jan 1995

Naval Research Laboratory, Code 5540, Washington, D.C. 20375-5337

For additional copies of this report, please send e-mail to
landwehr@itd.nrl.navy.mil, or retrieve PostScript via
http://www.itd.nrl.navy.mil/ITD/5540/publications/handbook
(e.g., using Mosaic).

SECURITY PENETRATION TESTING GUIDELINE:

A Chapter of the
Handbook for the Computer Security Certification of Trusted Systems

TM-8889/000/01

Prepared for:

Center for Secure Information Technology
Naval Research Laboratory (NRL)
Contract: N00014-91-C-2183

by:

Clark Weissman

for:

Unisys Government Systems
12010 Sunrise Valley Drive
Reston, Virgina 22091-3498

ABSTRACT

Penetration testing is required for National Computer Security Center (NCSC) security evaluations of systems and products for the B2, B3, and A1 class ratings of the Trusted Computer System Evaluation Criteria (TCSEC). This guideline is a definitive statement of what constitutes good penetration testing, where it fits in the DOD Standard Software Engineering and TCSEC life cycles, and how it is done according to the best available practice, the Flaw Hypothesis Methodology (FHM). A review of the TCSEC assurance products is presented, as they form evidence of a chain of reasoning on the compliance of the target system to a given evaluation class, and against which penetration testing is mounted. Flaws in the evidence are the products of penetration testing. To exemplify the methodology, results of past experience are provided throughout. The guideline concludes with a short review of new R&D approaches broadly considered penetration testing. An extensive bibliography is provided of work in the field, as are a set of Appendices that provide practical management guidance in planning and performing penetration testing.

FOREWORD

This document is intended for Executives, Strategic Planners, Program Managers, Technical Directors, Engineers, and interdisciplinary folks wishing to acquire a view of Computer Security, an increasingly complex set of requirements in the information systems business. It is one guideline on penetration testing in a Navy handbook series on security certification of computer systems to operate in a multi-level secure (MLS) manner according to current DoD security standards, The Rainbow Series.

The Rainbow Series of technical security guidelines are produced by the NCSC of the National Security Agency (NSA). Collectively, the documents consolidate knowledge about the degree of trust one can place in a computer system to protect sensitive information, and organizes this knowledge into useable criteria for evaluating a computer system's ability to resist unauthorized use. The landmark 1983 document, "The Department of Defense Trusted Computer System Evaluation Criteria, TCSEC," (CSC-STD-001-83), replaced in December 1985 as the DoD security standard DoD-5200.28-STD, is the "Orange Book," the color of the covers for the first book in the Rainbow Series. The TCSEC established for the first time a rating scale for secure operating systems, from minimal (C1) to high (A1) trust, based on security policy, protection features mechanisms, and assurance measures. Subsequent publications in the Rainbow Series have amplified the requirements of the TCSEC, and extended its application for networks, database management systems, and distributed applications. The Rainbow Series of knowledge and guidance has become influential in shaping domestic military, government, and commercial development, procurement, and products. It has also stimulated international security standards in Canada, England, France, Germany, and the Netherlands.

ACKNOWLEDGEMENTS

This guideline would not be possible without the encouragement of the Navy, particularly Judy Froscher, John McDermit, and Charles Payne of NRL/Code 5542, and Dick Stewart, Unisys NRL Program Manager. I thank Harvey Gold, Doug Hardie, Dick Linde, and Dick Stewart for critical review and feedback on the draft text. I also thank Marshall Abrams and Harold Podell for encouragement and review of an earlier essay on penetration testing scheduled to appear as a chapter in their forthcoming textbook [WEIS94]. The suggestions of these professional colleagues improved the guideline immeasurably, however, the remaining "flaws" are mine alone. Lastly, the material owes much to all the TCSEC evaluators who struggle anonymously and go unsung.

Clark Weissman
12/1/92;
revised 10/30/93

CONTENTS

FIGURES AND TABLES

1.

1. INTRODUCTION

A security threat exists when there is the opportunity, motivation, and technical means to attack: the when, why, and how. Penetration testing deals only with the "how" dimension of threats. It is a requirement for high-rated secure systems, ratings above B1 of the Secure Computer System Trusted Evaluation Criteria (TCSEC), that penetration testing be completed without discovery of security flaws in the evaluated product, as part of a product or system evaluation [DOD85, NCSC88, NCSC92]. Unlike security functional testing, which demonstrates correct behavior of the product's advertized security controls, penetration testing is a form of stress testing, which exposes weaknesses, i.e., flaws, in the Trusted Computing Base (TCB).

Of all the security assurance methods -- including layered design, proof of correctness, software engineering environments (SEE) -- only penetration testing is holistic in its flaw assessment. It finds flaws in all the TCB evidence: policy, specification, architecture, assumptions, initial conditions, implementation, software, hardware, human interfaces, configuration control, operation, product distribution, and documentation. It is a valued assurance assessment tool.

Among the important lessons presented in this guideline are the following:

- Testing is a posteriori (analysis) not an a priori (design) assurance method.
- Penetration testing is best at finding flaws not booty.
- A flaw is an unspecified exploitable capability in B2, B3, and A1 systems.
- Penetration testing is holistic; it finds flaws in policy, spec, code, operations.
- Good planning is needed: goals, resources, skills, TCB evidence, schedule.
- Penetration testing begins after the system is under configuration control.
- Residual flaws are those remaining after management approval to operate.
- Flaw Hypothesis Methodology (FHM) is a widely used penetration approach.
- FHM: Flaw Generation, Confirmation, Generalization, & Elimination stages.
- 20 years of experience shows C1-B1 systems have weak resistance to attack.
- Formal design methods are a new form of a priori penetration analysis.
- A model Work Breakdown Structure (WBS) for penetration testing is given.

This guideline is organized in five main parts, plus Appendices and Reference sections. These sections define what good penetration testing is all about (Section 1), where it fits in the product life cycle (Section 2), how it is done (Section 3), some results from past experience (Section 4), and future frontiers (Section

5). The guideline addresses two principal audiences, managers, and technical evaluators. Managers will gain much from reading Sections 1, 2, 5, and 6. Evaluators should read the whole document, but focus on the "how to" Sections 3, 4, and 6.

1.1 What Is Penetration Testing?

Penetration testing is one method of evaluating the security strength of a Trusted Computing Base (TCB). It is a pseudo-enemy attack by a friendly evaluation team on a computer system of interest to discover ways to breach the system's security controls, to penetrate the security perimeter of protection to obtain sensitive information, to obtain unauthorized services, or to cause damage to the system that denies service to legitimate users. It is a novel form of testing, which attempts to discover features, functions, and capabilities of the system that are unspecified and often unknown to its developers and users. It finds capabilities that can be exploited to breach security. These extra capabilities are "flaws" in the specifications, design, implementation, operation, or documentation of the system. Penetration testing finds security flaws and complements security functional testing, which confirms the correct behavior of the specified security features, functions, and capabilities. Because it tests "what is not there," i.e., there are no specifications for flaws, penetration testing develops novel ways of preparing test cases. Essentially, hypothetical specifications are prepared from which tests are derived. However, most tests are paper and pencil "thought" experiments, like German Gedanken experiments in early 17th and 18th Century physics[1]. Some tests are live exercises similar to functional testing. The penetration testing described here is largely based on the author's Flaw Hypothesis Methodology (FHM), the earliest and most widely used approach [WEIS73].

Traditional methods of function testing and repair are poor strategies for gaining assurance of TCBs. The "hack and patch" approach to assure secure systems is a losing countermeasure method because the hacker need find only one flaw, whereas the vendor must fix all the flaws. FHM penetration testing is not hack-and-patch, but a methodical, holistic method to test the complete, integrated, operational TCB -- hardware, firmware, software, and human interfaces -- and expose as many flaws as established as a test goal, (see Section 3.2.2). It is an empirical review of design coherence from abstract design theory through implementation to operational practice. It is peer review of all the TCB assurance evidence. It is one of many novel methods of satisfying assurance requirements. It works. It finds flaws.

1.2 What Is The Purpose/Goal Of Penetration Testing?

There are many possible goals of penetration testing, which must be clearly stated in test plans before testing begins, else resources will be squandered. The primary goal is to satisfy TCSEC requirements for B2, B3, or A1 assurance. Penetration testing provides independent validation of security trustworthiness of a system when performed by an impartial, competent evaluation team. Penetration testing is a useful vendor design and development tool, particularly in anticipation of product submission for evaluation per TCSEC. Many other goals have been satisfied by penetration testing. It has been used as "shock therapy" to convince skeptical managers of their vulnerability to attack threats. It has been used in University teaching of systems/security engineering [HEBB80, WILK81]. It was important in identification of

[1] Galileo's famous experiment of dropping different weight balls from the Tower of Pisa to prove that all objects fall at the same rate, was a Gedanken experiment. He argued logically that, if an unsymmetric dumbbell-shaped body split while falling, the heavier piece would not spontaneously increase in falling speed since there were no new forces acting upon it. Gravity would act alike on both pieces, and by extension, on all free falling bodies.

generic weaknesses of a system architecture that lead to improved designs [ATTA76, BELA74, MCPH74, SPAF89].

Penetration testing is not a game between employees and management, students and teachers, citizens and government, though such examples by hackers are rampant in the popular press. Penetration testing should not be part of Navy System Acceptance Tests. Acceptance Tests show that a product meets its specifications. Security functional testing can be included in acceptance tests. But penetration testing is an open ended test without specifications. No sensible, credible vendor could bid on a task without end. Penetration testing is inappropriate as a coercive force on vendors.

Penetration testing can not prove or even demonstrate that a system is flawless. It is an empirical method and unable to say much about undiscovered flaws. It is as thorough and comprehensive as the talent, knowledge, skill, and diligence of the team members. It can place a reasonable bound on the knowledge and experience required for a penetrator to succeed. That knowledge applied to countermeasures, can restrict the penetrator's access below this bound, and therefore, give a degree of assurance to operate the system securely.

1.3 What Makes A Good Penetration Test?

A consistent test philosophy is basic to good penetration testing. A philosophy that focuses efforts on finding flaws and not on finding booty or other hidden targets adds professionalism to the tests by placing a premium on thinking instead of scavenger-hunt searches. Flaws are found when the system protection mechanisms are breached. B2, B3, and A1 candidate systems must employ a reference validation mechanism (RVM) for satisfactory TCB protection. A TCB is an amalgam of hardware, software, facilities, procedures, and human actions, which collectively provide the RVM security enforcement mechanism of the Reference Monitor [ANDE72].

A Reference Monitor mediates every access to sensitive programs and data (i.e., security Objects) by users and their programs (i.e., security Subjects). It is the security policy mechanism equivalent of abstract data-type managers or type enforcers in strongly-typed programming languages such as Modula, and Ada. The Reference Monitor software is placed in its own execution domain, the privileged supervisor state of the hardware, to provide tamper resistance from untrusted user/application code (i.e., untrusted Subjects). The Reference Monitor software is often called the "Security Kernel," and is complemented with trusted processes (i.e., trusted Subjects) performing authorized policy violations, memory sanitization or label creation, for example. The Reference Monitor needs to be small and simple in its architecture to allow human evaluation for
correctness and for assurance that only the authorized security policy is enforced. The resistance to penetration testing of this triad of policy, mechanism, and assurance of the Reference Monitor is the basis for the high rating of the TCB. A comprehensive test of the triad is fundamental to good penetration testing theory.

A comprehensive penetration test plan improves the odds for achieving good penetration testing. A penetration test plan establishes the ground rules, limits and scope of the testing. The test team identifies what is the "object" being tested and when the testing is complete. For a commercial product, penetration test planning can begin whenever the vendor and the Navy agree, and the TCSEC evidence package is ready. For a system in development, penetration testing is a part of the overall security plan, normally prepared in the early phase of the program, e.g., by System Requirements Review (SRR). Since the

TCSEC evidence package will not be complete until after Formal Qualification Testing (FQT), penetration testing by the Navy evaluators would begin after FQT. Informal and optional penetration testing by the vendor could begin anytime there is solid design evidence. Evidence consists of architectural analysis of RVM at Preliminary Design Review (PDR), flaw generation at Critical Design Review (CDR), and code testing during System Integration and Test (PQT/FQT). (These phases and products are presented in detail in Section 2.) The Navy should cooperate with the vendor during such optional penetration testing for the reasons noted above. It will also make the formal penetration testing run more smoothly.

A key requirement for the penetration test plan is defining the posture of the simulated attacker. Is the evaluator to play the role of an insider or an external unauthorized user? For B2 and higher systems, the worst-case assumption is hostile attack from authorized users inside the security perimeter who exceed their authorization. For authoritative attack simulation the penetration team must be highly qualified and professional as discussed in Section 1.4. Lastly, the plan should define the limits of the penetration test; when is the test complete? Normally, the open-ended nature of flaw searching concludes when resources or people are exhausted. However, a time limit can also be applied. These and many other questions about penetration testing procedures and tasks are covered in later sections, particularly Sections 2 and 3.

1.4 What Makes A Good Tester?

Good people in an integrated team make for good penetration testing. Desirable characteristics for the team include experienced penetration testers, people knowledgeable of the target system, creative folks with bizarre ideas on associations of software modules, software development methods and tools, operating systems' control structure, resource allocation, input/output, human interfaces, and memory management. Successful testers are individuals who are detail oriented, careful thinkers, and persistent. A key requirement is for ethical, mature professionals who can protect proprietary, sensitive vendor data, particularly residual flaws in the system.

It is important that penetration testing employ evaluators who have the highest ethics since they will be given access to proprietary data and uncover extremely sensitive system vulnerabilities. Penetration testers must be non-antagonistic toward the vendor to encourage cooperation. They must protect proprietary information and vendor product investment so their testing can yield an improved security product. Test results and discovered flaws during penetration testing must be kept strictly proprietary and not made public by the test team.

Evaluator-developer cooperation is a prerequisite for good penetration testing. The vendor must be assured of the professionalism of the testing staff and the protection of his proprietary data rights. Most vendors have considerable investment in their products, are quite suspicious of the motives of the test team, and are anxious about the results of the penetration testing, which if leaked could kill a product. A good penetration team will keep the vendor apprised of the team's activities with frequent progress meetings and results feedback. Good vendor rapport is built by adding vendor personnel to the penetration testing team, with prime responsibility given for guiding the team through TCSEC evidence of the RVM, and for training briefings and clarifications of the RVM design. Early cross training of the penetration team and the vendor's staff is an excellent way to build cooperation, knowledge, and trust. The penetration team teaches the vendor about the TCSEC assurance evidence requirements and the FHM. The vendor staff teaches the penetration team about the target product, its development tools, test cases, and internals documentation. Also, vendor-provided equipment, software, and test tools demonstrates the vendor's commitment to the evaluation, further building a cooperative spirit. Never allow antagonism

between the vendor and team members to develop. Antagonism is difficult to avoid since vendor and evaluator have competing goals, finding zero defects versus finding flaws, respectively. Also, the vendor must provide a "frozen" copy of the system being tested which agrees with the product under configuration control. The vendor must not "fix" flaws in the test copy as that would invalidate configuration controlled assurance evidence.

Penetration testing is exhausting work. It is careful, detailed "destructive" analysis of thousands of lines of complex source code in a short period of time. The work burns out professionals rapidly. It is best for penetration team members to rotate to other constructive activities after about six months of testing. Fresh evaluators should be brought in at the start. Penetration testing is an excellent systems training vehicle when junior members are mixed with seasoned professionals [HEBB80], and new folks can be motivated to participate for this experience.

1.5 Who Is Responsible For Penetration Testing?

The simple answer is, the evaluators do the formal penetration testing [RUB86]. Standard commercial-off-the-shelf (COTS) products, are submitted to the National Computer Security Center (NCSC) for evaluation and publication on the Evaluated Products List (EPL). In that case NCSC performs the penetration testing.

The complex answer is, penetration testing will be performed by all parties with a vested interest. A COTS product will be penetration tested during product development and assurance evidence preparation by its vendor for self appraisal well before submission to NCSC for EPL consideration. For custom systems contracted by the Navy, the Navy will define who is responsible for penetration testing as part of the contract and the system security accreditation task. The accreditation will involve the contractor, the Navy end user, and the Navy Designated Approving Authority (DAA), who has ultimate responsibility for residual risks in a given operational environment.

Responsibility for penetration testing gets even muddier for the trusted application component developer, e.g., a secure database management system (DBMS) vendor, a new player on the security scene. In theory the application developer prepares separate component assurance evidence, which shows how the application TCB is a component TCB subset of the system TCB. It also shows how the component TCB subset meets the criteria of Evaluation By Parts (EBP) of the distributed Reference Monitor [TDI91]. It is clear that the component can not operate without its sibling components, therefore, penetration testing must involve all the components of the full TCB. However, if all the other components are rated (previously penetration tested), is the whole retested, or just the new component, its interfaces, and its environment assumptions? EBP theory is new and untried at this writing. Component vendor assurance evidence must claim success of the integration. The evaluator's predicament and answer to the question of retesting the whole TCB, awaits future practice and experience.

2.

2. PENETRATION TESTING IN THE DEVELOPMENT LIFE CYCLE

Penetration testing occurs late in the development life cycle when there is sufficient product to be tested. For custom DOD systems, the life cycle is based on MIL-Standards. DOD-STD-2167A is the standard methodology for military system software development and the life cycle model used in this handbook [DOD88]. DOD 5200.28-STD, the Trusted Computer System Evaluation Criteria (TCSEC), is the independent regulation guiding the evaluation of systems for trust [DOD85]. These two methodologies are not harmonized to the consternation of all security and program managers. Some work has been done to address the blending of these two standards, particularly in the Strategic Defense Initiative Organization (SDIO) programs [RADC90, GE91]. It is beyond the scope of this guideline to do more than comment on this harmonization, however, penetration testing is consistently treated as "testing" in the later stages of both development methods.

DOD-STD-2167A describes the development life cycle as the series of major processes, reviews, and deliverable products shown in Table 2-1.

Table 2-1. DOD-STD-2167A Summary

Process	Review	Product
System Requirements Analysis	SRR	SSS
Software Requirements Analysis	SDR	SRS, SDP
Preliminary Design	PDR	SDD, STP
Detailed Design	CDR	SDD, STD
Coding and Computer Software Unit (CSU) Testing	CSU	Code
Computer Software Component (CSC) Integration and Testing	TRR	CSC, STD
Computer Software Configuration Item (CSCI) Testing	CSCI Tests	CSCI, STR
System Integration and Testing	PCA, FCA PQT, FQT	SPS, O&S

Where:

SRR = System Requirements Review
SSS = System Subsystem Spec
SDR = System Design Review
SRS = Software Requirements Spec
SDP = Software Development Plan
PDR = Preliminary Design Review
SDD = Software Design Document
STP = Software Test Plan
CDR =Critical Design Review
STD = Software Test Descriptions
TRR = Test Readiness Review
PCA = Physical Configuration Audit
FCA = Functional Configuration Audit
STR = Software Test Results
SPS = Software Product Spec
O&S = Operation and Support Documents
PQT = Preliminary Qualification Tests
FQT = Final Qualification/Acceptance Tests

The TCSEC is not organized around development process as is DOD-2167A. Rather it is a set of criteria, design principles, and development practices for achieving evidence of trust. Table 2-2 lists the TCSEC processes and/or products.

Table 2-2. DOD 5200.28-STD Summary

Principle/Process	Product
Philosophy of Protection	Concept of Operation (CONOPS), Security Architecture, Security Policy
Policy Modeling	Formal Policy Model
Formal Design	Formal Top Level Spec (FTLS), A1 Only
System Design	Descriptive Top Level Spec (DTLS)
Model Correspondence	Formal Policy Model to FTLS Map, A1 Only
TCB Implementation	Code
Code Correspondence	DTLS/(FTLS A1) Map to TCB
Covert Channel Analysis (CCA)	Document Channels and Their Disposition
Functional Testing	TCB Test Plan, Procedures, Results
Security Testing	TCB Penetration Test Plan, Procedures, Results
Evidence Documentation	Above plus: Trusted Facility Manual (TFM), Security Features User's Guide (SFUG), Configuration Management Plan (CMP)

2.1 How Does Penetration Testing Relate To Other Life Cycle Products?

Penetration testing is one part of security testing in the TCSEC. Security testing will be performed by the developers at CSCI Testing, when the code and documentation are frozen and placed under Configuration Management. Results can be part of the STR, probably a subsidiary document or appendix because of the sensitivity and/or classification of the results, i.e., vulnerabilities. Plans for penetration testing are documented in the STP and reviewed at the PDR and CDR.

When penetration testing occurs for custom systems (i. e., non COTS) is controversial. From the vendors view, penetration testing is performed by the evaluators **after** the system is accepted (FQT) and before it goes operational as part of accreditation testing. The government wishes penetration testing **before** or as part of acceptance testing. The issues deal with compliance responsibility, money, and security. If penetration testing is performed after acceptance testing and security flaws are found, is the vendor obligated to fix the flaws? Who pays for the repairs if they are significant? Furthermore, the system can not be used operationally until the accreditation tests are complete, which could be significantly delayed by the repairs. The government believes security accreditation is a system requirement the vendor embraces when he bids and wins a program, similar to performance, schedule, and pricing requirements. The vendor has equally valid arguments. Unlike performance, schedule, and pricing requirements, the

solution of which he controls, penetration testing is performed by the government, and is an open ended task. Often the vendor has a "marching army" of its staff awaiting acceptance test completion. It is difficult for the vendor to price and schedule the tests and the possible repairs. Penetration testing before acceptance would be expensive for the vendor and the government. Resolution of this dilemma is beyond this guideline. Recent examples have had **before** testing on cost-plus contracts and **after** testing on fixed-price contracts.

Two popular models of software development are the "Waterfall" method [ROYCE70], and the "Spiral" method [BOEHM86]. They deal with distinct phases for requirements, design, specification, implementation, test, integration, and operation as described in DOD-2167A. Whereas the Waterfall model sees these as serial phases, the Spiral model focuses on risk reduction by building iterative prototypes, where each prototype shakes out the requirements, specifications, and performance weaknesses. By its nature and prerequisites, penetration testing comes so late in the life cycle that the significant flaws it uncovers have a major ripple impact on the development process. Other than implementation flaws, fixing flaws requires repair of specs (SSS, SRS, SDD, SPS), code (CSU, CSC, CSCI), documentation (SDP, STP, STD, SPS), and the extensive TCSEC assurance evidence: code-to-specification and code-to-model correspondences, covert channel analysis, code proofs (A1 systems), regression tests and repeat security tests (STD, STR). The Spiral model is a better fit for high TCSEC class secure systems as the iteration allows feedback for correcting the ripple effect of flaws discovered on earlier iterations. Three iterative loops of the specification-code-test spiral have been typical experience with development of such systems before they are ready for security evaluation.

Figure 2-1 reflects a relationship between DOD-2167A and TCSEC processes and products as expressed in the SDI work for Rome Labs [RADC90]. It is representative of similar experience on other programs, but not without controversy regarding when TCSEC evidence is generated. However, the SDI-suggested correspondence of processes and products of the standards are shown as typical of software trust engineering to exemplify the relationship necessary for practical application of the two methods. Feedback loops of the Spiral method are not show to simplify the presentation.

<table>
<tr><td colspan="10">DOD-5200.28-STD (TCSEC) Methodology</td></tr>
<tr><td>Product & Events</td><td>Protection Philosophy</td><td>Model</td><td>FTLS</td><td>DTLS</td><td>Correspon-dences Analyses</td><td>CCA</td><td>Function Testing</td><td>Security Tests</td><td>Security Evidence</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Product</td><td>SSS</td><td>SRS</td><td>SDD, STP</td><td>STP</td><td>Code</td><td>Code</td><td>STD</td><td>STR</td><td>O&S</td></tr>
<tr><td>Events</td><td>SRR</td><td>SDR</td><td>PDR</td><td>CDR</td><td colspan="2">Coding & CSU Tests</td><td>TRR</td><td>CSCI Tests</td><td>PCA, FCA</td></tr>
<tr><td colspan="10">DOD-STD-2167A Methodology</td></tr>
</table>

Figure 2-1. DOD-STD-2167A and DOD 5200.28-STD Correspondence

Vendors of commercial products intended for evaluation and addition to the Evaluated Products List (EPL) follow a development process analogous to that of DOD-2167A. The vendor may conduct penetration tests as part of his development. Government evaluators of a product intended for the EPL conduct penetration testing after the product has completed the vendor's FQT.

2.2 What Information Is Used To Develop Penetration Tests?

Product test cases are prepared from the product's functional specifications. In penetration testing we are looking for deviations from functional specifications, for capabilities not specified, i.e., for (dis)functional specifications of flaws. In the final analysis it is the code that captures the requirements, specifications, and design, and the code actually tested on the operational hardware. However, there are no (dis)functional specifications for the penetration tests; equivalent "specifications" must be developed as part of the penetration test plan. TCSEC B2 and better systems require their developers to produce a coherent collection of information on the trustworthiness of the system that will convince the evaluators to grant the highest rating class. The accreditation process is a social process judging the adequacy of the system for the application. The social process has analogy to a jury trial with the user agent (e.g., procurement) acting as the prosecution lawyer, (We'll uncover any product weaknesses.), the vendor acting as the defense lawyer, (Product meets all the buyer's security requirements.), the evaluators playing the jury role, (The evidence shows ...) , and the DAA being the ultimate judge, (Product security strength is/is not acceptable for the application threat environment). For such a model of the accreditation process, penetration testing becomes a part of the "evidence" of the system's security, and also a consumer of such evidence in generating the (dis)functional specifications for penetration testing.

2.2.1 RVM Chain Of Reasoning

The TCB is the reference validation mechanism (RVM) that mediates all accesses by Subjects to Objects. The basis of that mediation is defined as the Security Policy, the permission rules for access. Rules can be as simple as a list of users allowed access, attached to the object at the discretion of the owner, i.e., Discretionary Access Control (DAC). Or the rules can be based on management mandated sensitivity labels, where Clearance must dominate Classification labels, i.e., Mandatory Access Control (MAC). The Bell-LaPadula policy is the most widely cited in DOD applications [BELL76]. The Security Policy is a most critical piece of evidence for penetration testing. It defines success. When a flaw is confirmed it is because of a failure of policy or of policy enforcement.

Hardware provides the means to isolate the TCB from tampering by non-TCB (i.e. untrusted) user code. The correct use of this hardware by the TCB during service procedure calls, I/O and clock interrupts, error handling routines, start up and shutdown is part of the security architecture evidence. Security architecture should clearly show the TCB boundary, also called the "security perimeter" between trusted and untrusted domains. The security architecture is another key piece of evidence, and is penetration tested for correct design, implementation, and operation. Furthermore, the security architecture must show that the TCB is always invoked and never bypassed for any reason, else it is unable to provide self-protection against unauthorized modification, or to control access per the security policy.

Protection is achieved if the facility is closed, the hardware remains unbroken, operators follow correct procedures, the TCB is unflawed, the security architecture is sound, and the security policy is appropriate

for the environment and correctly implemented. The objective of security assurance is to show security enforcement, i.e., trust, and dependence of one piece of evidence on another in a coherent chain. Coherent in both the mathematical sense of logical implication, and in the social sense of being clear to deliberative bodies of evaluators and accreditors. When a flaw is detected in the chain of evidence, it is a case of the proverbial "For want of a nail ..." Penetration testing examines the evidence for "missing nails."

2.2.2 Concept of Operations (CONOPS)

Security is a "weak link phenomenon," in which many trusted parts play a role in the protection philosophy of the system. Security is only as good as its weakest part. A global system view of how the different parts play together is captured in the Concept of Operations (CONOPS) document. CONOPS provides a macro view of the roles of trusted parts of the TCB under different user and operational scenarios. It is not a complete description or specification for any given part, but is a complete description of the whole system and the interdependence and interplay of the parts. It is an analysis technique and a pedagogic device for surfacing incomplete system design and component interfaces.

Generation of the CONOPS begins with a time line analysis of system security operations -- boot load, system security officer (SSO) login, SSO loads and updates subject permissions and object restrictions (e.g., access control lists, ACLs, security labels, read/write/append rules), SSO audit analysis, SSO alarm response, and SSO system shutdown. Superimposed on this time line are typical user security actions -- login, program execution, file access, device and I/O access, mail and communications interaction, and logout. Further superimposed on the system time line are the actions of special users, e.g., system administrators, maintenance and repair, database administrators, system and network managers, etc. Each has security-relevant actions that must be part of the CONOPS. Each of these human actions triggers internal system security enforcement modules. These are identified on the time line with their necessary data flow, databases, and hardware. Lastly, unscheduled events are listed and considered to occur at the worst of times for the system. Error conditions, equipment failure, communications breakdown, corrupted data, overloads, etc. are mapped to various time line events. Collectively, these events describe the CONOPS, the theoretical behavior of the security elements of the system that counter the expected threat scenarios.

The CONOPS is prepared early in the system development life cycle, usually by SDR or PDR. It grows in importance as the system is implemented and is a key document in the development of security and acceptance tests. It contributes significantly to defining the roles of all the system users, the labels of data, access rules, and the TFM and SFUG evidence documents. It is a necessary piece of evidence for assessing trust.

2.2.3 All The B2, B3, Or A1 Evidence

Table 2-2 gives a comprehensive list of the key security principles of the TCSEC, their associated processes when applicable, and the evidence products that result. Section 2.2.1 describes evidence common to the DoD-STD-2167A and the TCSEC. This section completes the review of evidence products unique to the TCSEC.

Security policy, CONOPS, security architecture, and security policy model collectively contain the security requirements for the system. The security policy model maps the security policy rules into the system subjects, objects, access rules, and security functionality as seen by the user and his application programs at the security perimeter. The security policy abstraction is made more concrete in the security

policy model. Files, memory segments, I/O ports are identified objects, and logon/logoff, create/delete objects, read/print to ports are subjects, probably trusted processes of the TCB. The security policy model is the beginning of the formal system design process and is captured in a Formal Top Level Specification (FTLS) for A1 class systems by PDR. The FTLS is often expressed in one of the formal specification languages -- Enhanced Hierarchical Development Methodology, Gypsy, and Ina Jo [KEMM86, NCSC88]. For B2 and B3 systems, which do not require an FTLS, the preliminary design is captured in a Detailed Top Level Specification (DTLS) and frozen by CDR.

A variety of security analyses in the TCB evidence collection provide the assurance that the design can resist attack. Foremost of these is the security architecture design analysis discussed above. Less well known are the various correspondence implementation analyses. These show the consistency between the implementation stages: requirements, specification and code. These analyses map the correspondence between the early and later development products; for example, between the FTLS and the code. The mapping is two way -- specification-to-code, and code-to-specification -- because, if there is specification without corresponding code, we have an incomplete implementation of the requirements or design and, if there is code without corresponding spec, we have possible malicious code, e.g., Trap Door, Trojan Horse, Virus, or Worm. There is a significant amount of code in the TCB that is not represented in the FTLS or DTLS because it is not part of the visible security interface, the security perimeter. The code-to-specification correspondence analysis must account for all such code. The A1 formal proofs are a mathematical demonstration of the correspondence of the FTLS to the security policy model, i.e., the correctness criteria and initial conditions.

Covert channel analysis (CCA) seeks out TCB mechanisms that may be shared among untrusted security partitions. Examples of shared mechanisms include common equipment, status variables, buffers, scheduling queues, and semaphores. These shared mechanisms may be used as covert "state" variables and modulated to transmit slow levels of unauthorized covert communications. The penetration testing team uses the vendor's CCA evidence to look for flaws: channels not closed or inadequately repaired, and new channels. Some tools exist that help in CCA of the FTLS and can be used by the evaluators [KEMM83].

Security functional test plans, procedures, and results are useful TCB evidence for penetration testing perusal and for reuse in the actual penetration tests.

2.3 What Happens To The Results Of Penetration Testing?

Self examination is the primary purpose for the vendor performing a penetration test. It gives him an assessment of a probable TCSEC rating class on the Evaluated Products List (EPL). The results are proprietary and are used to fix problems. In some cases the flaws found during penetration testing of an earlier product can lead to significant architectural changes in future versions of the product to achieve a higher evaluation class.

Penetration testing can be a research tool to better understand generic design flaws and possible countermeasures. The Multics experience is a case in point [KARG74].

Penetration testing performed by the government evaluators produces lists of residual flaws, a description of the security strengths, and an EPL evaluation class. The detailed results are shared with the vendor and kept proprietary. A joint vendor and evaluator public bulletin is released on the EPL rating achieved. The summarized results, called a "security profile," are used by the end-user senior management to assess the

risk of using the evaluated system in a given environment. If the residual flaws can not be repaired, or can not be fixed in a timely manner, supplemental protection may be recommended, e.g., improved physical protection, or higher user clearances levels. The risk management recommendation is presented to the Designated Approving Authority (DAA) for approval to operate the system with its residual flaws and added protection for the specific application environment.

3.

3. PERFORMING PENETRATION TESTING

There can be many goals for penetration testing, ranging from security assurance [DOD85] to systems training [HEBB80, WILK81]. For this guideline, penetration testing focuses only on the goal of generating sufficient evidence of flawlessness to help obtain product certification to operate at a B2, B3, or A1 security assurance level.

3.1 For What Are You Looking?

We are looking for flaws not exploiting flaws to find dummy target files. An anecdote characterizes the difference between seeking flaws and booty. A penetration test at a highly sensitive agency in the early 80's, had as its objective accessing a dummy target file on a system considered by the agency to be well protected. After considerable preparation, the penetration team was allowed physical and terminal access to the system and given a week to do its testing. Within the first hour, expected flaws were found, "trap door" code was planted to bypass access controls, and searches were begun for the dummy file. A runaway printing loop in the target computer blocked use of the test terminal because the bypass patch the team inserted in the TCB was off by a few bytes. The print loop made the activities visible to the agency watchdog, who stood in awe as another terminal was used to patch the patch and stop the runaway print terminal. All this patching was performed on-line, bypassing the system security access controls without authority or permission. The team had effective control of the operating system one hour after they began. The agency immediately stopped the penetration testing "game," being convinced that it was vulnerable and needed a serious penetration testing assessment of its flaws.

Penetration testing is best employed to explore the broad capabilities of the object system for flaws against security policy, rather than in a gaming situation between the vendor and the penetration team trying to violate access restrictions to an identified protected object -- "hack-and-patch". Dummy target acquisition penetration goals waste effort by forcing the test team to prepare and debug a break-in, rather than focusing their energy on proven methods of finding flaws.

International travel requires passports and visas to control crossing borders. In banking, turning account balances into cash, is a form of boundary crossing called "conversion." Control is imposed at the human interface. The Reference Monitor defines a "security perimeter" between itself and untrusted user application code, and between different user processes. Flaws within the untrusted application code have no impact on the TCB and are of little interest to security or penetration testing. Flaws internal to the security kernel are of interest to security, but they can not be exploited unless the security perimeter is breached or caused to malfunction (e.g., fool the system into giving unauthorized access). Therefore,

much of penetration testing focuses on flaws in the design, implementation, and operations integrity of the security perimeter, the control of the boundary crossings of this critical security interface.

The variety of the TCB evidence described in Section 2.2 are documents of the TCB policy, model, design, implementation, test, and operation. The analysis of the adequacy of the evidence can be overwhelming. Figure 3-1 shows the evidence implication chain: the operation is secure according to proper procedures, facility management, and reliable hardware for the implementation (code); the code implies its secure specs (DTLS and/or FTLS); the specs are correct per the model; and, the model satisfies the security policy and requirements of the end user. Reasoning logically, secure operation depends on secure hardware, code, specifications, model, policy, and requirements, and in that order of dependency.

TCB Operation -1-> Hardware -2-> Code -3-> DTLS -4-> FTLS -5-> Model -6-> Policy

Where:

-1->	Configuration Management, Trusted Facility Manual (TFM), Secure Features Users Guide (SFUG), and Concept of Operations (CONOPS)
-2->	Security Functional & Penetration Testing Covert Channel Analysis
-3->	Code-to-Specification Correspondence Analysis
-4->	Code-to-Specification Correspondence Analysis for A1 Class
-5->	Proof of Correctness for A1 Class Specification-to-Model Correspondence for B2, B3 Classes
-6->	Model Axiom Proofs for B2, B3, A1 Classes

Figure 3-1. Evidence Implication Chain

Penetration testing attempts to find "kinks" in the reasoning chain by studying all the TCB evaluation evidence. As in all things, history is a great teacher, and the results of past penetration tests are a major starting place for penetration testing. There is a considerable wealth of published material on penetration studies in Section 7, References.

3.2 How Do You Find Flaws?

At the heart of the TCSEC is mathematical induction, sometimes called the "Induction Hypothesis." It is the theoretical basis of TCSEC security. It argues that:

(1) If the TCB starts operation in a secure state, and the
(2) TCB changes state by execution from a closed set of transforms (i.e., functions), and
(3) each transform preserves defined security properties, then
(4) by mathematical induction, all states of the TCB are secure.

Finding flaws begins with finding weaknesses in implementation of this protection theory -- policy, model, architecture, FTLS/DTLS, code, and operation. The evidence implication chain of Figure 3-1 forms the basis of the flaw search for violations of the Induction Hypothesis. As examples, false clearances and permissions void initial conditions (voids rule 1), bogus code (e.g., Trojan Horse, virus) violates the closed set (violates rule 2), and a large covert channel does not preserve the information containment security properties of functions (spoils rule 3) of the Induction Hypothesis. In practice, the penetration test is a social process managed according to a plan with practical goals, bounds, resources, and schedules.

3.2.1 Develop A Penetration Test Plan

Establishing the test ground rules is a particularly important part of penetration testing and is captured in the penetration test plan, a part of the DOD-2167A STP at PDR for development programs, and a stand alone document for post-FQT government evaluations. The test plan defines the test objectives, the product configuration, the test environment, test resources, and schedule. In particular, the test plan defines the criteria for test completion.

3.2.2 Establish Testing Goal

The ground rules for penetration testing define successful completion. The penetration testing is successfully concluded when:

(1) A defined number of flaws are found,
(2) A set level of penetration time has transpired,
(3) A dummy target object is accessed by unauthorized means,
(4) The security policy is violated sufficiently; or,
(5) The money and resources are exhausted.

Most often the last criterion ends the penetration test, after a defined level of effort is expended. For some systems, multiple independent penetration teams are employed to provide different perspectives and increased confidence in the security of the product if few minor flaws are found.

3.2.3 Define The Object System To Be Tested

Penetration testing can be applied to almost any system requiring TCSEC evaluation. However, C1, C2, or B1 candidate systems are intended for benign environments, protected by physical, personnel,

procedural, and facility security. Systems in these benign evaluation classes are not designed to resist hostile attack and penetration. Such attacks are always likely to uncover flaws. The TCSEC, wisely does not require penetration testing for these systems. Penetration testing is most valuable for testing security resistance to attack of candidate systems for evaluation classes B2, B3, or A1, systems designed to operate in hostile environments.

A system intended for TCSEC evaluation at B2 or higher is delivered with a collection of material and documentation that supports the security claim. (See the description in Section 2.) This controlled collection of security evidence defines the security system to be penetration tested. The evidence must be frozen and unmodified during the penetration testing period to avoid testing a moving target.

In circumstances where the Target Of Evaluation (TOE) can not be tested in a controlled laboratory environment and must be tested in situ, there is a possibility of conflict between users and evaluators. Experience has shown that probing for security flaws may require system halts and dumps by the penetration team, and when tests succeed they yield unpredictable results, e.g., uncontrolled file modification or deletion, or a system crash, which disrupts normal operation. Therefore, penetration testing should be performed on a stand alone copy of the TOE to assure non-interference with real users of the system.

When the object system is a network, the TCB is distributed in various components, the whole collection of which is called the network TCB, (NTCB). As noted in the TNI, penetration testing must be applied to: (1) the components of the NTCB, i.e., the partitions of the NTCB, and; (2) the whole integrated network NTCB [TNI87]. Therefore, the TNI Mandatory, Audit, Identification & Authentication, and Discretionary (M-A-I-D) network components must be penetration tested individually and collectively -- individually during the component evaluation, and collectively during the network evaluation.

In a similar manner, trusted applications, e.g., a trusted database management system (DBMS), must be penetration tested individually as a component, and collectively with the operating system TCB on which it is dependent, according to the "evaluation by parts" criteria of the TDI [TDI91].

3.2.4 Posture The Penetrator

When an actual test is required to confirm a flaw hypothesis, a host of test conditions must be established, which derive directly from the test objectives and the test environment defined in the plan. These conditions derive from the security threats of interest and the posture of the "simulated" antagonist adopted by the evaluators. Will it be an "inside job," or a "break and entry" hacker? These assumptions demand different conditions for the test team. The test conditions are described as "open box" or "closed box" testing, corresponding to whether the test team can place arbitrary code internal to the system (open box) or not, restricted only to externally stimulated functional testing (closed box). The TNI testing guideline calls these "black box" (functional) and "white box" (internal) testing, corresponding to closed box and open box testing, respectively [NCSC88b]. Open box penetration testing is analogous to CSU testing, where access to the internal code is possible, and closed box penetration testing is analogous to CSCI testing, where code modules are an integrated closed whole. In open box testing, we assume the penetrator can exploit internal flaws within the TCB and work backwards to find flaws in the security perimeter that may allow access to the internal flaws. In the case of a general purpose system such as UNIX, open box testing is the most appropriate posture. For special purpose systems, such as network NTCB components, which prohibit user code, e.g., where code is in ROM, closed box penetration testing, by methods external to the product, is analogous to electrical engineering "black box" functional testing.

In closed box testing the penetrator is clearly seeking flaws in the security perimeter and exploiting flaws in the implementation of the Interface Control Document specifications, (ICD). Open box testing of the NTCB is still a test requirement of the vulnerability of the network to Trojan Horse or viral attacks.

3.2.5 Fix Penetration Analysis Resources

It was believed that finding flaws in OS VS2/R3 would be difficult [MCPH74]. However, another study claimed "The authors were able, using the SDC FHM, to discover over twenty such 'exposures' in less than 10 man-hours, and have continued to generate 'exposures' at the rate of one confirmed flaw hypothesis per hour per penetrator. ... Only the limitations of time available to the study governed the total number of flaws presented" [GALI76].

Penetration testing is an open-ended, labor-intensive method seeking flaws without limit. The testing must be bounded in some manner, usually by limiting labor hours. Small teams of about four people are most productive. Interestingly, penetration testing is intense, detailed work that burns out team members if regular rotation of the evaluators is not carefully managed. Evaluators should be encouraged to participate in penetration testing with reward opportunities to pursue new technologies that promise better secure systems. Experience shows the productivity of the test team falls off after about six months. Therefore, a penetration test by four people for no more than six months -- 24 person months, is optimal. Composition of the test team must include people knowledgeable in the target system, with security and penetration testing expertise. Much time is spent perusing the security evidence. However, there must be liberal access to the target system to prepare and run live tests. The team needs access to all the TCB creation tools -- compilers, editors, configuration management system, word processors -- and a database management system (DBMS) to inventory their database of potential flaws, and to store their assessments of the flaws.

3.3 Flaw Hypothesis Methodology (FHM) Overview

COMPUSEC's (Computer Security's) raison d'etre is to automate many of the security functions traditionally enforced by fallible human oversight. In theory a trusted system should perform as its security specifications define, and do nothing more. In practice most systems fail to perform as specified, and/or do more than is specified. Penetration testing is one method of discovering these discrepancies.

3.3.1 Evidence Implication Chain

For trusted systems to be rated B2 or better, the trust evidence must show that theory and practice agree, that the implication chain is correctly satisfied at each step. Penetration testing seeks counter arguments to the truth asserted by the evidence; i.e., it seeks to establish the evidence is false, or incomplete. A flaw is such a counter argument. A flaw is a demonstrated undocumented capability, which can be exploited to violate some aspect of the security policy. The emphasis of the FHM is on finding these flaws. It is not on building demonstrations of their exploitation, though such examples may have merit in some cases. Exploitation demonstrations consume valuable resources that can better be applied to further flaw assessment of the implication chain.

3.3.2 Stages Of Flaw Hypothesis Methodology (FHM)

FHM consists of four stages:

(1) *Flaw Generation* develops an inventory of suspected flaws.
(2) *Flaw Confirmation* assesses each flaw hypothesis as true, false, or untested.
(3) *Flaw Generalization* analyzes the generality of the underlying security weakness represented by each confirmed flaw.
(4) *Flaw Elimination* recommends flaw repair, or the use of external controls to manage risks associated with residual flaws.

These stages are shown if Figure 3-2.

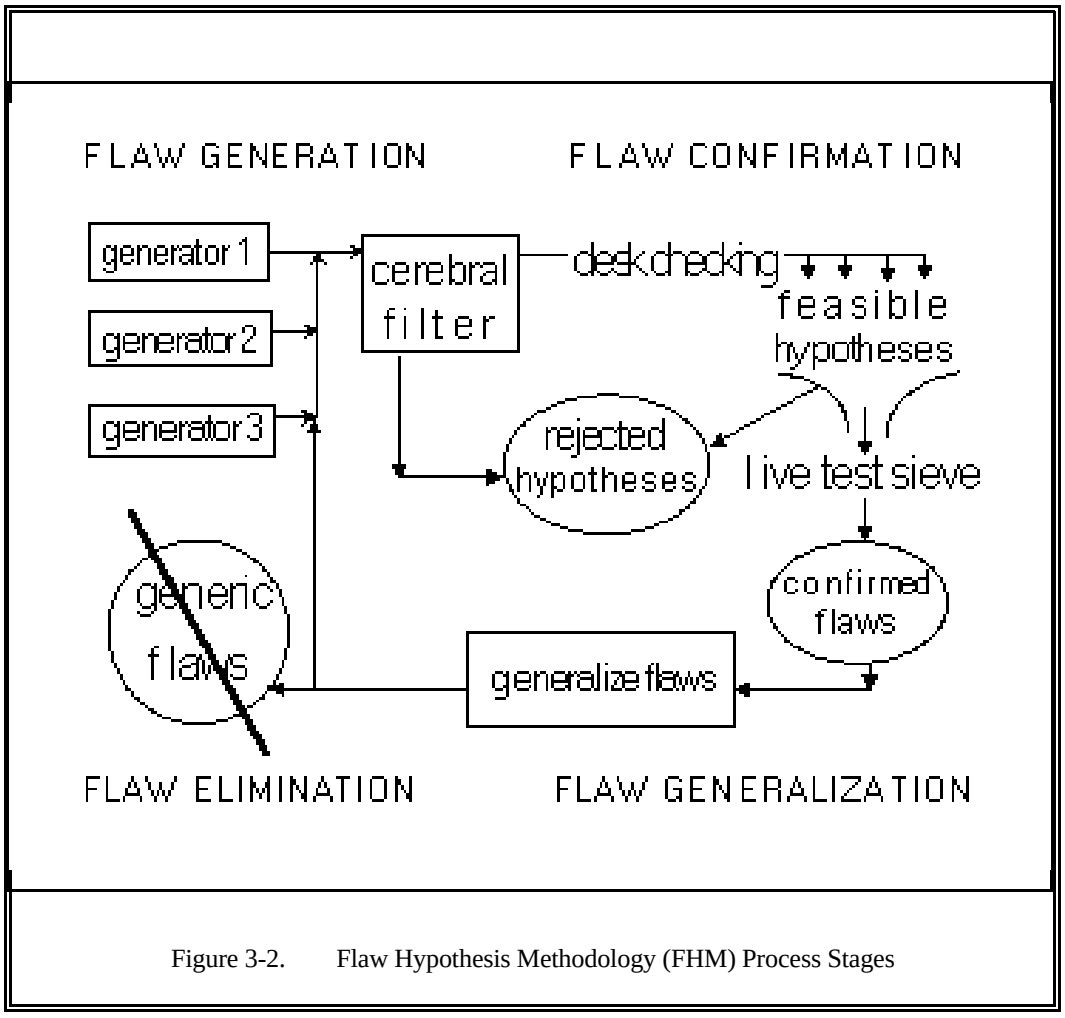

Figure 3-2. Flaw Hypothesis Methodology (FHM) Process Stages

Flaw Generation can be likened to a computer strategy game. In Artificial Intelligence (AI) game software, there is a body of logic that generates "plausible moves" that obey the legal constraints of the game, e.g., assures a chess pawn never moves backward. In like fashion, penetration testing needs a "plausible flaw generator." Flaw finding begins with the evidence implication chain, our experience of security failures of the reasoning chain in other systems, and their potential existence in the target system. The security evidence for the target system is the principal source for generating new flaw hypotheses.

Continuing our AI game analogy, there is a body of heuristic rules the game employs to determine good plausible moves from poor ones. Likewise, in penetration testing Flaw Confirmation, there is human judgment, i.e., a cerebral filter, that evaluates and rates each prospective flaw in terms of its probability of existence and how significantly it violates the security policy. Filtering flaws for confirmation employs desk checking of code, specifications, evidence in documentation, and/or live testing.

The Flaw Generalization stage of penetration testing gives an assessment of our results in progress, the game analogy of "winning" or improving game position. Flaw Generalization assesses confirmed flaws, seeking basic computer science reasons why they exist. For example: in the penetration testing of OS VS2 [LIND75] a simple coding error was traced to a library macro and multiple instantiations of the flaw in the code. Inductive reasoning on the cause of confirmed flaws can lead to new flaws, generators of still more weaknesses.

The Flaw Elimination stage considers results of the Generalization stage and recommends ways to repair flaws. Implementation flaws are generally easier to repair than design flaws. Some flaws may not be practical to repair; slow covert timing channel flaws may be tolerable, for example. These flaws remain in the system as residual flaws, and place the operational environment at risk. However, external countermeasures can be recommended to the DAA for managing these risks, by lowering the TCSEC Risk Index [CSC85] for example.

3.4 Flaw Generation

Flaw Generation begins with a period of study of the evidence to provide a basis for common understanding of the object system. Early in the effort there is an intensive team brain-storming "attack the system" session. Other attack sessions are on-going throughout the penetration test. Target system expertise must be represented in the attack sessions. Each aspect of the system design is reviewed in sufficient depth during the session for a reasonable model of the system and its protection mechanisms to be understood and challenged. Flaws are hypothesized during these reviews. Critical security design considerations are the basis for the penetration team's probing of the target system's defenses. These design considerations become the "plausible move generators" of the Flaw Generation phase. The most productive "top 10" generators are tabulated in Table 3-1.

Each candidate flaw is described on a Flaw Hypothesis Sheet (FHS). (See Appendix 6.2 for details.) The total set of FHS becomes the flaw database that guides and documents the penetration analysis.

Table 3-1. Most Productive Flaw Generators

1. Past experience with flaws in other similar systems.
2. Ambiguous, unclear architecture and design.
3. Circumvention of "omniscient" security controls.
4. Incomplete design of interfaces and implicit sharing.
5. Deviations from the protection policy and model.
6. Deviations from initial conditions and assumptions.
7. System anomalies and special precautions.
8. Operational practices, prohibitions, and spoofs.
9. Development environment, practices, and prohibitions.
10. Implementation errors.

3.4.1 Past Experience

The literature is filled with examples of successful penetration attacks on computer systems [ABBO76, ATTA76, BELA74, BISB78, BISH82, GALI75, GALI76, GARF91, KARG74, MCPH74, PARK75, SDC76]. There is also a body of penetration experience that is vendor proprietary or classified [BULL91, LIND76, PHIL73, SDC75]. Though general access to this past experience is often restricted, such experience is among the best starting points for Flaw Generation. The Navy should have access to these materials, and/or should collect past results and its own new penetration testing experience.

3.4.2 Unclear Design

The design must clearly define the security perimeter of the TCB. How are boundary crossings mediated? Where are the security attributes -- permissions, classifications, IDs, labels, keys, etc. -- obtained, stored, protected, accessed, and updated? What is the division of labor between hardware, software, and human elements of the TCB? And how are all the myriad other secure design issues described elsewhere satisfied [GASS88]? If the secure design cannot be clearly described, it probably has holes. The team will rapidly arrive at consensus by their probing, and uncover numerous flaws and areas for in-depth examination, particularly weakness in the evidence implication chain.

3.4.3 Circumvent Control

What comes to mind is Atlas down on one knee holding up the world. The anthropomorphic view of TCB design gives the numerous protection control structures omniscience in their critical Atlantean role in supporting the secure design. If such control can be circumvented and bypassed, the security can be breached. The security architecture evidence must show the noncircumventability of the control structures. The attack sessions will rapidly identify these omniscient objects, be they password checkers, label checkers, I/O drivers, memory maps, etc. A method of determining their vulnerability to attack is to build a dependency graph of subordinate control objects upon which the omniscient ones depend. Each node in the graph is examined to understand its protection structure and vulnerability to being circumvented, spoofed, disabled, lied to, or modified. If the security architecture is weak or flawed, control can be bypassed. The penetration testing of OS VS2/R3 [SDC76] gives a detailed example of the

use of dependency graphs to determine the vulnerability of VS2 to unauthorized access to Job Data Sets, virtual memory, and password protection control objects.

3.4.4 Incomplete Interface Design

Interfaces are rife with flaw potential. Where two different elements of the architecture interface there is a potential for incomplete design. This is often the case because human work assignments seldom give anyone responsibility for designing the interface. Though modern methodologies for system design stress Interface Control Documents (ICD), these tend to be for interfaces among like elements, e.g., hardware-hardware interfaces, software-software protocols. The discipline for specifying interfaces among unlike elements is less well established. Hardware-software, software-human, human-hardware, hardware-peripheral, and operating system-application interfaces can have incomplete case analysis. For example, the user-operator interface to the TCB must deal with all the combinations of human commands and data values to avoid operator spoofing by an unauthorized user request. Operating procedures may be hardware configuration dependent. For example, booting the system from the standard drive may change if the configuration of the standard drive is changed. All the various error states of these interfaces may not have been considered.

Implicit sharing is now a classical source of incomplete design flaws. Sharing flaws usually manifest themselves as flaws in shared memory or shared variables between the TCB and the user processes during parameter passing, state variables context storage, setting status variables, reading and writing semaphores, accessing buffers, controlling peripheral devices, and global system data access, e.g., clock, date, public announcements. Careful design of these interfaces is required to remove system data from user memory.

3.4.5 Policy And Model Deviations

For B2 and higher evaluation classes, the security evidence includes a formal security policy and a model of how the target system meets the policy. Subjects and objects are defined. The rules of access are specified. For lower evaluation classes, the policy and model are less well stated and, in the early years of penetration testing, required the penetration team to construct or define the policy and model during the attack sessions. However, penetration testing is not required for these classes today.

Consider the adequacy of the policy and the model for the target system. Is it complete? Is it correct policy? Are there policies for Mandatory and Discretionary Access Control (MAC and DAC), identification and authentication (I&A), audit, trusted path, and communications security? Examine the security architecture and the TCB design to see if there are deviations from the stated policy or model. For example, are there user-visible objects that are not defined in the model, such as buffers and queues? Omniscient control objects, as described in Section 3.4.3, should certainly be represented. Are there deviations in the implementation of the policy and model? This consideration receives greater emphasis during Flaw Confirmation, however, there may be reasons to generate implementation flaws during Flaw Generation.

3.4.6 Initial Conditions

Assumptions abound in secure system design, but are not documented well. Evaluation class A1 does better than other evaluation classes because of it more rigorous formal design and use of formal specifications which require entry and exit assertions to condition the state machine transforms. For all evaluation classes the assumptions and initial conditions are often buried in thick design documentation, if documented at all. If these assumptions can be made invalid by the user, or if the initial conditions are different in the implementation from that of the design assumptions -- reality not theory -- the policy and model may be invalid and flaws will exist. The Induction Hypothesis of Section 3.2 begins with "... starts operation in a secure state ..." Initial conditions determine the starting secure state. If the actual initial conditions are other than as assumed in the design, attacks will succeed.

The whole range of user security profiles and administrative security data, IDs, clearances, passwords, permissions (MAC and DAC), define the "current access" and "access matrix" of the Bell-LaPadula policy model [BELL76]. These data are initial conditions. Their correct initialization is a testable hypothesis. Other assumptions and initial conditions need to be established and tested by penetration analysis including, the computer hardware configuration, software configuration, facility operating mode -- periods processing, compartmented, system high, and MLS -- operator roles, user I&A parameters, subject/object sensitivity labels, system security range, DAC permissions, audit formats, system readiness status, and more.

3.4.7 System Anomalies

Every system is different. Differences which may have security ramifications are of particular interest. The IBM Program Status Word (PSW) implements status codes for testing by conditional instructions, unlike the Univac 1100, which has direct conditional branching instructions. The IBM approach allows conditional instructions to behave as non-conditional instructions if the programmer avoids checking the PSW [SDC76]. That is an anomaly. The Burroughs B5000-7000 computer-series Compiler software has privilege to set hardware tag bits that define "capabilities," many of which are security sensitive, such as write permission. The Master Control Program (MCP), checks the tag bit for permission validity. User code does not have this privilege. Code imports can circumvent such checks [WILK81]. That is an anomaly. The IBM 370 I/O channel programs are user programs that can access real memory via the "Virtual = Real" command without a hardware memory protect fault [BELA74]. That's an anomaly. Nearly every software product has clearly stated limits and prohibitions on use of its features, but few define what occurs if the prohibition is ignored. What happens when an identifier greater than eight characters is used? Is the identifier truncated from the left, right, middle, or is it just ignored? Anomalous behavior may not be security-preserving functionality per the Induction Hypothesis theory that can be exploited.

3.4.8 Operational Practices

The complete system comes together during operation when many flaws reveal themselves. Of particular interest is the man-machine relationship, the configuration assumptions, and error recovery. A well designed TCB will have the system's boot process progress in secure stages of increasing Operating System capability. Each stage will check to assure it begins and ends in a secure state. If there is need for human intervention to load security parameters, the human must be identified, authenticated, and authorized for known actions. The evaluator must study the process to see if the design and implementation of the boot process progresses correctly. For example, how is the security officer/administrator authenticated? If via passwords, how did the password get loaded into the initial boot

load? Where does the operator obtain the master boot load? From a tape or disk library? Is the physical media protected from unauthorized access, product substitution, or label switching? If the security officer loads or enters on-line permissions to initialize the security parameters of the system, how does the security officer authenticate the data? If users send operator requests to mount tapes/disks, to print files, or a myriad of other security sensitive actions, how does the TCB support the operator from spoofs to take unauthorized action? If the system crashes, does the system re-boot follow a process similar to the initial "cold" boot? If there is a "warm" boot mode -- a short cut boot which salvages part of the system state -- does the security officer have a role in the boot to ensure the system begins in a secure state? How is the assurance determined?

A common example of an initialization flaw is to discover the system was shipped from the vendor with the "training wheels" still on [STOL89]. This class of flaw has been known to include training files that provide ID-password-authorizations to users so they may train for jobs as security officers, system administrators, database controllers, system operators. These files were not removed by good facility management and system operational practice per the Trusted Facility Manual (TFM), and can be used by unauthorized parties to circumvent security controls.

3.4.9 The Development Environment

Flaws may be introduced by bad tools and practices in the security kernel/TCB development environment. A simple example is the conditional compilation, which is used to generate special code for debugging. If the released code is not recompiled to remove the debug "hooks," the operational system code violates the closed set rule 2, and the secure transform rule 3 of the Induction Hypothesis of Section 3.2, similar to a trap door to circumvent security measures.

Large untrusted reuse and run-time libraries are properties of many programming environments. The TCB may be built using code from the library which finds its way into operational use. All kinds of security flaws may obtain from such environments. If the libraries are not security sensitive, they can be searched for flaws that are exploitable in the operational TCB. If the penetration team can substitute its own code in the libraries, even more sophisticated flaws can be created. Run-time linkers and loaders have similar properties of appending unevaluated code to the trusted object code being loaded to enable code-Operating System communication. If access to such tools is unprotected, similar code-substitution attacks are possible.

A classic attack on an operational system is to attack its development environment, plant bogus code in the source files, and wait for the normal software update maintenance procedures to install the unauthorized code into the operational system object code. If the development and operational system are the same, then the penetration team must mount an attack on the development environment first, particularly the system configuration files. Flaws found there relate directly to the operational system, the source files of which are then accessible and modifiable by the penetrator without authorization. Substitute configuration files give the penetrator a high probability attack and essentially control of the TCB.

3.4.10 Implementation Errors

In any system built by humans, there will be errors of omission and commission. This is not a promising class of flaws to search for, as there is no logic to them. Many are often just typos. Implementation errors

that can be analyzed are those of the IF-THEN-ELSE conditional form. Often the programmer fails to design or implement all the conditional cases. Incomplete case analysis may occur if the code logic assumes some of the predicates are performed earlier. Most often implementation flaws are just coding errors.

Another area for investigation is macros and other code generators. If the original macro is coded incorrectly, the error code will be propagated in many different parts of the system. Similarly, if data declarations are incorrect, they will affect different parts of the code. Incorrect code sequences should be traced back to the source code and the automatic code generators to see if the code error appears in multiple parts of the TCB.

Sometimes there are errors in the development tools that generate bad code. Few configuration management tools provide a trusted code pedigree or history of all the editor, compiler, linker tools that touch the code. Therefore, an error in these tools, which becomes known late in the development cycle and is fixed, may miss some earlier generated modules that are not regenerated. The penetration team may find it fruitful interviewing the development team for such cases.

3.5 Flaw Confirmation

Conducting the actual penetration test is part of the testing procedure developed in the plan. The bulk of the testing should be by Gedanken experiments, thought experiments that confirm hypothesized flaws in the system by examination of the documentation evidence and code. There are three steps to the Flaw Confirmation stage:

(1) Flaw prioritization and assignment,
(2) Desk checking, and
(3) Live testing.

3.5.1 Flaw Prioritization and Assignment

The Flaw Hypothesis Sheets (FHS) represent a comprehensive inventory of potential flaws. Sorted by the probability of existence, payoff, i.e., damage impact, if confirmed, work factor/effort to confirm, and area of the system design, they provide a ranking of potential flaws for each design area from high probability of existence/high payoff (HH) to low probability/low payoff (LL). Usually, only high and medium ranks are studied. The team divides the rank lists among themselves based on expertise in the different system design areas. They move out as individuals on their lists seeking to confirm or deny the flaws. Evaluators share progress and findings at daily team meetings. Management will reallocate staff and FHS to balance the work load. Often, confirmed flaws raise the priority of other FHS, or provide the analysts with insight to generate new FHS.

3.5.2 Desk Checking

The evaluator studies the FHS and the TCB evidence. Code, models, code correspondence maps or dependency graphs are examined to see if the flaw exists. The evaluator must be flexible in considering alternatives, but concentrate on what exists in the actual code and other evidence. The evaluators use code

walk throughs, prior test results, their own insights, and conversations with other team members to reach conclusions on the likelihood of the flaw's existence.

Results are documented on the FHS. Confirmed flaws are flagged in the database for later examination. An evaluator spends a few days, at most, on each flaw. The desk checking continues for weeks, and possibly a few months, yielding a FHS productivity rate of 10-20 FHS/person-month. The work is tedious, detailed, and requires destructive thinking. Occasionally a FHS is of sufficient complexity and interest to warrant a live test, but the investment in the testing process will lower productivity.

3.5.3 Live Testing

Test case design, coding, and execution is expensive and not the preferred FHS evaluation method. However, testing is often the fastest way to confirm complex or time-dependent flaws. In penetration testing, live tests are similar to CSCI functional tests, except the FHS is the (dis)functional spec, and penetration testing may be destructive of the system.

Avoid running tests on the operational system since they can have unpredictable results. Also, the testing is to confirm the flaw, not to exploit it. Test code should be a narrowly focused (by the FHS), quick one-shot routine that is easier to produce if there is a rich library of debug and diagnostic routines.

3.6 Flaw Generalization

When the team assembles for the daily meeting, the confirmed flaws of the day are briefed and examined. Each team member considers the possibility that a confirmed flaw might exist in his area, or if the test code can be used on his FHS. Often a confirmed flaw has only medium payoff value but can be used in conjunction with other confirmed flaws to yield a high payoff. This stringing of flaws together is called "beading" and has led to many unusual high-payoff penetrations.

Deductive thinking confirms a flaw hypothesis. Inductive thinking takes the specific flaw to a more general class of flaws. The team examines the basic technology upon which each confirmed flaw is based to see if the flaw is a member of a larger class of flaws. By this generalization of the flaw, one can find other instances of the weakness, or gain new insight on countermeasures. Inductive thinking proceeds simultaneously with deductive thinking of new instances of the flaw, so that the flaw becomes a new flaw hypothesis generator. Some classic flaws were discovered by this induction, e.g., parameter passing by reference [LIND75, SDC76], piece-wise decomposition of passwords [TANE87], puns in I/O channel programs [ATTA76, PHIL73], and time-of-check-to-time-of-use (TOCTTOU) windows [LIND75]. These flaws are described in Section 4.

3.7 Flaw Elimination

Experts have argued the futility of penetrate and patch, hack-and-patch methods of improving the trust of a TCB for substantial reasons that reduce to the traditional position that you must design security, quality, performance, etc. into the system and not add it on [SCHE79]. However, most human progress is made in incremental forward steps. Products improve with new releases and new versions which fix flaws by patching, work-arounds, and redesign.

The TCSEC requires all known flaws be repaired. The evaluators can suggest to the vendor repair of simple implementation and coding errors, or recommend known generic design flaw countermeasures. After repair the system must be reevaluated to confirm the flaw fixes and to ensure no new flaws were introduced. Reevaluation is a complete repetition of the penetration testing process. However, application of the Ratings And Maintenance Process (RAMP) to B2 and better evaluations may be a possible method to avoid total repetition. This speculation is described in Section 5.3.6. It is impractical for the vendor to fix some flaws. These residual flaws will result in a lower class rating. However, the using agency can prepare a risk analysis that shows the DAA alternative security measures to counter the residual flaws.

4.

4. EXPERIENCE AND EXAMPLES

Flaw Hypothesis Methodology (FHM) has been a cost effective method of security system assurance assessment for over twenty years. Unlike other assurance methods, which focus on more narrow objectives, e.g., formal correctness proofs of design, or risk assessment costs of failures, FHM seeks security flaws in the overall operation of a system due to policy, specification, design, implementation, and/or operational errors. It is a complete systems analysis method which uncovers flaws introduced into the system at any stage of the product life cycle.

4.1 FHM Management Experience

For weak systems in the TCSEC C1-B1 classes, experience predicts a typical penetration team of four people operating for six months will generate about a 1000 FHS and assess about 400 of the highest priority. Some 50-100 of these will be confirmed flaws. That yields a productivity of a flaw every one to two person-weeks. Stronger systems in the TCSEC B2-A1 classes, by definition, must be flawless. However, even these systems have flaws; far fewer of course because of the greater attention to secure system development. Such flaws are repaired, audited, or considered an acceptable risk. Higher flaw rates may signal a lesser evaluation class than B2 is warranted for the target system. Appendices 6.3 and 6.4 provide a model of the tasks, schedules, and person-months of effort to perform a competent penetration test based on this past experience. Specific penetration test planning should adapt these data for the conditions and resources available.

4.2 Taxonomy Of Security Flaws

A taxonomy of flaws is useful if it organizes flaws for the following purposes:

(1) Provides a central reference collection of flaws.
(2) Catalogs flaws for descriptive communication,
(3) Finds repetitive patterns signaling a common fault.
(4) Enhances tool-supported flaw search.
(5) Suggests common attack methods.

An extensive library of flaws is in preparation by NRL which addresses purpose (1) [BULL91]. This section suggests "severity of damage" as a taxonomy, satisfying cataloging purpose (2). Attempts to define generic flaw patterns and build pattern matching tools to aid flaw finding in source code, exemplifies

purposes (3) and (4), and are discussed in Section 5.1. Purpose (5) is a promising taxonomy and is discussed in Section 4.3.

Confirmed flaws are sorted into vulnerability classes shown in Table 4-1. These vulnerability classes are based on the degree of unauthorized control of the system permitted by the flaw, i.e., damage extent. The greatest vulnerability is TCB capture; the machine is under total control of the interloper, (TC -- total control -- flaws). Flaws that permit lesser control are unintentional, undocumented capabilities that violate the policy model, but do not allow total control, (PV -- policy violation -- flaws). Denial of Service (DOS) flaws permit the penetrator to degrade individual and system performance, but do not violate the confidentiality security policy, (DS -- denial of service -- flaws). Installation-dependent flaws are weaknesses in the TCB that obtain from local initialization of the system, such as a poor password algorithm, (IN -- installation dependent -- flaws). They may be TC, PV, or DS flaws as well. Lastly, there are harmless flaws in the sense that they violate policy in a minor way, or are implementation bugs, which cause no obvious damage, (H -- harmless -- flaws). These codes categorize flaws presented in subsequent sections.

Table 4-1. Vulnerability Classes Of Flaws

1. Flaw Gives TCB/System Total Control, (TC)
2. Security Policy Violation, (PV)
3. Denial of Service, (DS)
4. Installation Dependent, (IN)
5. Harmless, (H)

4.3 Taxonomy Of Attack Methods

FHM is a labor intensive method of penetration testing. DBMS, writing, development, and testing support tools are helpful. Automated tools have not succeeded in finding flaws, where the greatest intellectual effort is required. This section provides a representative collection of attack methods found effective in penetration testing by the innovative and skilled penetration teams using the FHM.

4.3.1 Weak Identification/Authentication

Passwords are the cheapest form of authentication. Weak passwords, however, are quite expensive, allowing the penetrator to impersonate key security administrators to gain total control of the TCB, (TC) flaw. In one system, a weak password protected the password file itself. The password was a short English word which took but a few hours of trial and error to guess. A popular program called "Crack" will run for days trying to crack passwords in the UNIX password file [MUFF4a]. The program has a password candidate generator, based on popular passwords, some permutation algorithms, and installation parameters, that encrypts the candidate and compares it with the list in the password file. It

reports successful hits. Since most passwords are initialized by system administration, this attack is an example of an operational flaw and an initial-condition (IN) flaw.

On the DEC PDP-10 and many contemporary machines, there are powerful string compare instructions that are used to compare stored with entered passwords. These array instructions work like "DO-WHILE" loops until the character-by-character compare fails, or the password strings end. It was discovered in the TENEX Operating System that the instruction also failed when a page fault occurred. This turned a binary password predicate -- yes/no -- into a trinary decision condition -- yes/no/maybe -- that enabled piece-wise construction of any password in a matter of seconds. In this case the flaw was a weak password checking routine that permitted the user to position the candidate password across page boundaries [TANE87]. This is an example of a hardware anomaly and an implicit memory sharing (TC) flaw.

On the IBM OS VS2/R3 and similar vintage OS, files or data sets are protected by password. When the user is absent during batch processing, surrendered passwords for the file are placed in the Job Control Language (JCL) load deck and queue file for command to the batch process. The JCL queue is an unprotected database, which is readable by any user process, thus permitting the stealing of passwords. This is an example of a badly designed user-system interface, where system data is placed in the user's address space [MCPH74, SDC75, SDC76]. It is also an example of bad security policy, a (PV) flaw.

In most systems, the user logs into a well designed password authentication mechanism. However, the system never authenticates itself to the user. This lack of mutual authentication permits users to be spoofed into surrendering their passwords to a bogus login simulation program left running on a vacant public terminal. This spoof has been around forever and is still effective. It is an example of a poor user-system interface that yields a policy violation (PV) flaw. Mutual authentication is a necessity in the modern world of distributed computing, where numerous network servers handle files, mail, printing, management, routing, gateways to other network, and specialized services for users on the net. Without it, such services will surely be spoofed, modified, and/or falsified. New "smart card" I&A systems employ mutual authentication countermeasures via encrypted challenge-response tokens [KRAJ92].

4.3.2 Security Perimeter Infiltration

Untrusted code must be confined and only permitted to call the TCB in a prescribed manner for secure access to needed system services [LAMP73]. These boundary crossings of the security perimeter are often poorly designed and result in "infiltration" flaws. A classic example of this is the uncontrolled channel program of the IBM 370. Since channel programs are allowed to be self modifying to permit scatter reads and writes, and the user can turnoff memory mapping, e.g., Virtual = Real, it is possible to write into protected system memory and modify code and/or process management data. Attempts to eliminate these problems by static analysis of the channel programs in VM/370 failed to prevent clever "puns" in the code from continued exploitation [ATTA76, BELA74, PHIL73].

Another example of poor confinement is the Honeywell HIS 6000 GCOS suspend feature that allows a user to freeze an interactive session for a lunch break or longer suspension, and resume later by thawing the program. The design flaw stores the frozen image in the user's file space, including all the sensitive system context needed to restart the code. It is a simple process for a user to edit the frozen image file and modify the context data such that the restarted program runs in system state with total control of the TCB (TC) flaw. This is yet another example of an implied memory-sharing flaw.

Among the most sophisticated penetrations is the legendary breakpoint attack of Linde/Phillips [ATTA76, PHIL73]. It is an excellent example of a beading attack. When a user plants a breakpoint in his user code, the system replaces the user code at the breakpoint with a branch instruction to the system. The system's breakpoint routine saves the user code in a system save area. Later, when the breakpoint is removed, the user code is restored. The breakpoint feature helps the penetrator plant code in the system itself, i.e., the replaced user code; an example of a harmless (H) flaw. It was observed that another existing harmless hardware flaw, a move string error exit, left the address of the system memory map, rather than the address of the string, upon error return. It was possible to induce the string flaw by reference to an unavailable memory page, i.e., a page fault. A third harmless flaw allowed control to return to the caller while the string move was in progress in the called program. The evaluators set up the bead attack by planting a breakpoint at a carefully prepared instruction on the same page as a string move command. They carefully selected a string that crossed a page boundary. They executed the string move, and upon regaining control, released the page containing the end of the long string. That caused a page fault, when the string move crossed the page boundary, at which time the breakpoint was removed. In restoring the pre-breakpoint user code, the system retrieved the saved user code but, because of the harmless hardware string-error flaw, erroneously wrote the user code into protected system memory, specifically the system page map. This unauthorized system modification was possible because a hardware design flaw in the page fault error return left the page address of the system memory map, not the page address of the original user's string. The attack had successfully modified the system map, placing user data in the system interrupt vector table. The attack gave arbitrary control of the TCB. Another subtle flaw in implicit memory sharing, a (TC) flaw.

4.3.3 Incomplete Checking

Imports and exports cross the security perimeter per the TCSEC are either label checked or use implicit labels for the I/O channel employed. Lots of flaws occur when labels are not employed, or employed inconsistently. Another attack exploits interoperability between systems which use different label semantics. The Defense Data Network (DDN), employs the standard IP datagram Revised Internet Protocol Security Option (RIPSO) security sensitivity label [RFC1038]. It differs from the emerging Commercial IP Security Option (CIPSO) standards. Here is a situation ripe for a future security attack and (IN) flaw.

Array-bounds overflow is a particularly nasty attack which is quite pervasive and difficult to counter. The flaw manifests itself in system operation, but its cause is traceable to the development compiler's failure to generate code for dynamic array bounds checking. When the array bound is exceeded, the code or data parameters adjacent to the array are overwritten and modified. In one case the user-entered password was stored adjacent to the system stored password so the two strings (arrays) could be rapidly compared. However, there was no bounds checking. The user simply entered a maximum sized password twice so that it overflowed the user array into the system array creating a password match [BISH82]; a certain (TC) flaw.

Incomplete case analysis leads to flaws. Either the design specification has not considered all the conditions of an IF-THEN-ELSE form, or the programmer goofed. In either event, the penetrator creates the missing condition and forces the code to ignore the consequences, often creating an exploitable state, a (PV) flaw. The IBM PSW flaw of Section 3.4.7 is such an example. The IBM 360 introduced the Program Status Word (PSW) which contained a status condition code for those machine instructions which have conditional execution modes. Many programmers ignore the PSW status code and assume the execution result of the instruction. This is poor coding practice, but a surprisingly frequent occurrence

[BULL91]. Often the programmer believes prior checks filter the conditions prior to the instruction execution and that the data can not cause the unanticipated condition, thus ignoring the condition code. The penetrator must find an opportunity to reset the parameters after the filter checks, but before the conditional code execution. Time-Of-Check-To-Time-Of-Use (TOCTTOU) attacks are exactly what's needed for penetration.

A TOCTTOU flaw is like a dangling participle grammatical flaw in English; the check code is distant from the use code, enabling intervening code to change the tested parameters and cause the use code to take incorrect, policy-violating actions, a (PV) flaw. The attack is a form of sleight of hand. The penetrator sets up a perfectly innocent and correct program, possibly an existing application program, and through multi-tasking or multi-processing, has another program modify the parameters during the interval between check and use. The interval may be small, which requires careful timing of the attack. The flaw is both an implicit memory sharing error and a process synchronization problem. The solution is not to place system parameters in user memory, and/or prohibit interruptibility of "critical region" code [LIND75, MCPH74, PHIL73].

Read-before-Write flaws are residue control flaws. Beginning with TCSEC C2 class systems, all reused objects must be cleaned before reuse. This is required as a countermeasure to the inadequate residue control in earlier systems. However, the flaw persists in modern dress. When disk files are deleted, only the name in the file catalog is erased. The data records are added to free storage for later allocation. To increase performance, these used records are cleared on reallocation, (if at all), not on de-allocation. That means the data records contain residue of possibly sensitive material. If the file memory is allocated and read before data is written, the residue is accessible. A policy of write-before-read counters this flaw, but such policy may not exist, or be poorly implemented. This flaw also appears with main memory allocation and garbage collection schemes. In one example, the relevant alphabetical password records were read into memory from the disk file for the login password compare. After the compare the memory was left unchanged. Using other attacks, such as the array-bounds overflow described above, that residue memory could be read and passwords scavenged. By carefully stepping through the alphabet, the complete password file could be recreated, a (TC) flaw [LIND76].

4.3.4 Planting Bogus Code

The most virulent attacks are those created by the penetrator by inserting bogus code into the TCB, a (TC) flaw. Bogus code includes all forms of unauthorized software, Trojan Horse, Trap Door, Virus, Bombs, and Worms. Fundamentally, flaws that admit bogus code are flaws in the configuration control of the TCB. The flaw may occur any time throughout the life cycle of the TCB. When development tools are uncontrolled, bogus code can be imbedded in the tools and then into the TCB. Ken Thompson's ACM Turing Lecture aptly documented such an attack [THOM84]. But there are easier methods; planting bogus code in the run-time package of the most popular compiler and/or editor. Recent attacks on the Internet were exploitations of poor configuration control. Known flaws in UNIX were not fixed with the free vendor patches. The hacker used the flaws to obtain unauthorized access [SPAF89].

Among the more colorful attacks against human frailty in controlling bogus code, is the Santa Claus attack. It is a classic example of an unauthorized code import, achieved by spoofing the human operator of a secure system. A penetrator prepared an attractive program for the computer operator, who always ran with system privileges. The program printed a picture on the high-speed printer of Santa and his

reindeer. The kind you always see posted at Christmas in and about the computer facility. However, there was a Trojan Horse mixed in with Dancer, Prancer, and friends that modified the operating system and allowed undetected access for the interloper. Before you belittle the computer operator's folly, consider your own use of "freeware" programs down-loaded from your favorite bulletin board. There are as many user and operator spoofs as there are gullible people looking for "gain without pain." Eternal vigilance is the price of freedom from spoof attacks. Also, improved role-authorization controls can limit the damage propagation of such attacks on human foibles.

5.

5. NEW FRONTIERS IN PENETRATION TESTING

This guideline concludes by speculating on future directions of penetration testing. The assumption is: the battle between the TCB developer/operator and the hacker will continue unabated. Better legal protection for users will always trail technology, and the technology will improve for both antagonists. It will be continuous digital electronic warfare between security measures, hacker attacks, user counter measures, and security counter-counter measures; perpetual offense against defense.

The defense, developers of TCBs, are using better methods of designing and implementing trusted systems. Lessons are being learned from past penetrations, both tests and real attacks. The generic flaws are leading to better understanding of security policy for confidentiality, integrity, and service availability, and of the confinement of overt and covert channels when sharing common mechanisms in trusted systems. This understanding is being captured in improved machine architectures with segregated privilege domains or protection rings to reinforce security perimeters and boundary crossings. Hardware that supports safe and rapid context switching and object virtualization. Cryptography is playing a larger role in confidentiality, integrity, and authentication controls. Computers are getting faster and cheaper so that security mechanisms will be hardware-rich and not be performance limiting in secure solutions. Software is getting better in quality, tools, development environments, testing standards, and formalism.

5.1 Automated Aids

Earlier, in the infancy of penetration testing, it was believed that flaws would fall into recognizable patterns, and tools could be built to seek out these generic pattern during penetration testing [ABBO76, CARL75, HOLL74, TRAT76]. Unfortunately, the large number of different processors, operating systems, and programming languages used to build TCBs, with their different syntax and semantics made it difficult and impractical to apply such pattern-matching tools to penetration testing. Considerable cost is required to port or re-implement the tools for each new environment and different programming language. Also, the flaw patterns tended to be system specific.

As modern secure systems focus on a few operating system standards, e.g., UNIX, MS-DOS, etc., fewer programming languages, e.g., Ada and C, and more mature Expert Systems become available, future penetration testing tool initiatives to capture flaw patterns may be more successful. A few specific examples of this trend are beginning to appear.

5.1.1 Virus Antigens

A variety of commercially available anti-virus products recognize telltale patterns on disks prior to main memory loading. These programs contain proprietary algorithms that recognize the signatures of dozens of the viruses that have plagued the industry for MS-DOS, Macintosh, UNIX, and other OS.

5.1.2 CERT Tools

Defense Advanced Research Projects Agency (DARPA) set up the Computer Emergency Response Team (CERT) following the release of the Internet Worm attack of November 1988, with the mission to monitor break-in attempts, and send out security alerts to the internet community. As a clearinghouse of information, it periodically releases security tools for defensive use, which, however, have penetration testing value as well. Two such tools are examples of progress, COPS and CRACK [FARM90, MUFF4a]. COPS is a collection of short shell files and C code that test for known weaknesses in UNIX file systems. CRACK is a password guessing program that contains a collection of cracking algorithms and probable password. It generates candidate passwords, encrypts them per UNIX login and tries to match them in the UNIX password file. Given the password file, it runs in background for a week or so producing a successful hit list.

5.1.3 Intrusion Detection

Considerable research interest has produced a number of prototype intrusion detection tools: Haystack, NIDX, DIDS, IDES, and ISOA [SMAHA88, BAUER88, DIAS91, LUNT92, WINK92]. These analytic tools monitor the TCB's audit records for patterns of abnormal behavior, and flag suspicious conditions to the Security Officer. The intrusion detection model "is based on the hypothesis that exploitation of a system's vulnerabilities involves abnormal use of the system; therefore, security violations could be detected from abnormal patterns of system usage" [DENN86]. Key to the success of these tools is their ability to discriminate between intrusions and legitimate behavior -- the classical Type 1 (false acceptance of an intrusion) versus Type 2 (false alarm for legitimate user) error tradeoff. Expert Systems technology is applied to build heuristic rules of statistically abnormal behavior. Intrusion Detection is so new there is little empirical data on its operational use or success.

5.1.4 Specification Analysis

Formal specs (FTLS) are produced for A1 system as part of their assurance evidence. For these systems a number of interesting tools exist for flaw analysis of the FTLS. Foremost of these tools are the theorem provers, that prove the specifications correct according to their formal policy model [KEMM86]. Failed proofs may indicate potential flaws. (See Section 5.2 for more details.)

Flow analyzers, such as the Unisys Ina Flo or the Mitre Flow Table Generator are tools used on the FTLS to automatically generate lists of potential unauthorized information flows between state variables [ECKM87, KRAM83]. The flow tool automatically analyzes the FTLS states, transforms, and initial conditions and generates all conditions of information flow that lead to overt and covert write-down channels, and potential flaws. Some tools can filter the potential flows against a user-supplied security policy to eliminate all valid flows from those more suspicious.

Most novel are FTLS symbolic execution tools such as Ina Test and Ina Go [ECKM85, WHEE92]. These tools interpret the formal specifications under various initial conditions controlled by the tool user, in a manner similar to a language interpreter. The resulting values of state variables can be observed and

compared against theory (e.g., exit assertions) and expectations. Deviations may be flaws. These tools are novel forms of rapid prototyping of security behavior directly from the FTLS.

5.2 Formal Methods Of Penetration Testing

Formal methods employ rigorous mathematical specifications of the TCB design, assumptions, initial conditions, and correctness criteria, the FTLS. Formal tools take these specifications and generate correctness theorems and proofs of the theorems and the design they portray. It is hoped that these formal methods can achieve similar success at the level of code proofs. A new form of penetration analysis is in progress with new TCB designs -- rigorous formal models of the TCB. These models describe the TCB behavior as state machines, state transition rules, security invariants, initial conditions and theorems that must be proven. Penetration analysis is almost inseparable from the formal design process, producing conjectures of flaws with the model and trying to prove them as theorems. This is a rigorous extension of the FHM. If successful, the correctness proof versus flaw conjecture proof becomes part of the design process, and uncovers problems early in the design, enabling iterative redesign, unlike FHM which often comes too late in the development cycle to permit more than hack-and-patch.

Recent work by Gupta and Gligor suggest a theory of penetration-resistant systems. Their method claims to be "a systematic approach to penetration analysis, enables the verification of penetration-resistance properties, and is amenable to automation" [GUPTA91, GUPTA92]. They specify a formal set of design properties that characterize resistance to penetration in the same framework used to specify the security policy enforcement model -- a set of design properties, a set of machine states, state invariants, and a set of rules for analysis of penetration vulnerability. Five penetration-resistance properties are described:

(1) System isolation (tamperproofness).
(2) System noncircumventability (no bypass).
(3) Consistency of system global variables and objects.
(4) Timing consistency of condition (validation) checks.
(5) Elimination of undesirable system/user dependencies.

Gupta and Gligor contend system flaws "are caused by incorrect implementation of the penetration-resistance properties [that] can be identified in system (e.g., TCB) source code as patterns of incorrect/absent validation-check statements or integrated flows that violate the intended design or code specifications." They further illustrate how the model can be used to implement automated tools for penetration analysis. They describe an Automated Penetration Analysis (APA) tool and its experiments on Secure XENIX source code. Early results from this work indicate that penetration resistance depends on many properties beyond the Reference Monitor, including the development and programming environment, that is characterize as the Evidence Implication Chain in Section 3.1. Though limited only to software analysis of attacks on the TCB from untrusted user code, and leaving significant other system avenues for attack, the work may pave the way for new approaches to building and testing trusted systems and tip the balance in favor of the "good guys."

5.3 Open Issues In Penetration Testing

This section discusses unresolved management and policy issues affecting penetration testing.

5.3.1 Penetration Testing Contracting

Vendors should be encouraged to support their security claims with strong security evidence. Because of the open-ended nature of penetration testing, fixed-price contracts are not appropriate when vendors are expected to conduct penetration testing as a means of self-assessment. Penetration testing is an open ended effort without a completion spec, as the prior material here attests. The effort is over when an agreed criteria is met. A fixed-price level of effort, or cost-plus contract is a more appropriate vehicle for penetration testing.

5.3.2 NVLAP Penetration Testing

The NIST National Voluntary Laboratory Accreditation Program (NVLAP) recommends that third-party laboratories test network protocols. Possibly such an approach can be applied to security accreditation (including penetration testing) as well. Mitre, Aerospace, AF Cryptologic Support Center, NSA already perform similar functions, but not on a commercial basis. The new NIST Federal Criteria divorce security functionality from security assurance. That policy could give NVLAP penetration testing groups responsibility to evaluate vendor secure products. The NVLAP program is quite new; its future success can set the precedent for third-party security evaluation labs.

5.3.3 Penetration Testing and Composibility

A significant open question is: How are evaluators to judge the security of a system composed of individually evaluated component products? Unlike the TNI requirements for a top-down design, the composed system is built bottom-up, and interface specifications will most likely not exist. The TNI composition rules will not apply. For a composed system above B1, each of its components will be B2 or better and will, therefore, have been penetration tested. The ensemble will need a new, system-level penetration test. The composibility controversy is beyond the scope of this guideline, however, FHM is applicable to such composed systems and may be part of the controversy's resolution.

5.3.4 Penetration Testing In Open System Environments

Penetration of networks is today an academic "sport." There is no question that the FHM is applicable and effective. The difficulty is assigning responsibility for fixing flaws when the network is a heterogeneous collection of evaluated products from different vendors. An integrated network, following the TNI or TDI, is not at issue since the responsibility for flaws falls to the integration contractor and there is some guidance for network testing [NCSC88b]. Again, it is beyond the scope of this guideline to establish policy for rating heterogeneous networks, however, the holistic nature of penetration testing makes it among the best methods available for such evaluations.

5.3.5 Penetration Testing Of Trusted Applications

The trusted application must follow the TCSEC TDI to be evaluated. That guideline defines the evidence necessary for evaluation, including penetration testing. Penetration testing must be performed on the OS, the application, and the combined system, the latter two efforts fall to the application vendor. The TDI is new and the secure DBMS vendors are just entering the high assurance arena of penetration testing.

5.3.6 Penetration Testing For Ratings Maintenance (RAMP)

RAMP is defined for lower rated systems (below B2) because they have few development assurance requirements [NCSC89]. High rated systems currently have no RAMP policy; the vendor must reevaluate new releases of the TCB. Penetration testing can be a significant factor in reducing cost of reevaluation, by focusing the penetration testing to those evidence products changed by the new release. It is the Evidence Implication Chain that is being confirmed by FHM, and if items can be demonstrated to be unchanged, considerable simplification of the penetration testing should be achieved. This thesis has been tested successfully on the B2 evaluation of trusted XENIX [MCAU92].

5.3.7 Penetration Testing Of Other Policies

Security is policy specific, but the FHM is not. It is based on the Evidence Implication Chain, where policy comes early in the evidence chain. Integrity, availability, safety are new policies of interest for which penetration testing is applicable. Unfortunately, these policies have no defined standard analogous to the TCSEC, and therefore, they have no definition of evidence, no theoretical foundation comparable to the Reference Monitor, and no evidence chain to examine for flaws. Once such theoretical underpinnings exists, penetration testing will be a valid assurance method.

5.4 Applicability Of FHM To Other Harmonized Criteria

The TCSEC has spurred development of information technology (IT) security evaluation criteria in other countries, particularly in Canada [CTCPEC92] and Europe [ITSEC91]. These have generated competitive pressure for compatible alignment of U. S. commercial security criteria, i., e., "harmonized criteria" as reflected in the draft Federal Criteria [FC92] published by NIST and NSA. Unlike the integrated ratings of the TCSEC digraphs, these newer criteria have split the criteriainto separate functionality and trust requirements; these are further sub-divided in ways particular to each national criteria. *Profiles* can then be defined as specific collections of functionality and trust requirements.

It has been questioned whether penetration testing has meaning for a Target of Evaluation (TOE) based on these newer criteria. The question has importance for this guideline because of the shift of U.S. Federal Criteria to unbundle functionality from assurance. For most of the reasons expressed earlier in this guideline, penetration testing will be required for high assurance evaluations even under these new criteria. The applicability of FHM to the harmonized criteria is discussed here as it will be applied in the 1990's.

5.4.1 Canadian Evaluation Criteria

The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) defines a Security Functional Profile as an open-ended set, an n-tuple, that can describe an infinite variety of product security

requirements. It is at the opposite extreme from the TCSEC and avoids "interpretations" for network, database, or other applications. The CTCPEC divides security functionality into four groups: Confidentiality, Integrity, Availability, and Accountability requirements. Each of these are further divided, e.g., covert channels (CC), MAC (CM), DAC (CD), and reuse (CR); discretionary integrity (ID), mandatory (IM), physical (IP), rollback (IR), separation of duties (IS), and self test (IT); containment (AC), robustness (AR), and recovery (AY); audit (WA), I&A (WI), and trusted path (WT). There are various degrees of increasing strength for each functional requirement division, e.g., CC-3, WA-4. Defined profiles exist for TCSEC digraphs. For example:

Profile 3 = B2 = [CC1, CD2, CM3, CR1, ID/IM1, IS2, IT1, WA1, WI1, WT1]

Eight trust levels T0 - T7 parallel the TCSEC: T0 = D, T1 to T5 = C1 to B3, and T7 = A1. T6 is between B3 and A1, using "semi-formal" detail design methods. Penetration testing is a clear requirement for trust levels T4 to T7. "Flaw hypothesis testing" is required at T1 to T3. Penetration testing is noted through out the trust requirements, but there is no description of what or how it is done. The CTCPEC follows very closely the ideas, definitions, and concepts of the TCSEC and is exactly what the TCSEC would be if the digraphs were broken into their functional and trust components. It is reasonable, therefore, to conclude that a process similar to that acceptable for TCSEC evaluations -- this guideline -- is germane to the CTCPEC.

5.4.2 European Evaluation Criteria

A TOE of the European Information Technology Security Evaluation Criteria (ITSEC) is either a security product or a system. Its penetration testing is therefore comparable in scope to penetration testing in accordance with the TCSEC. However, a TOE's evaluation criteria consists of a 2-tuple, (F, E); one of ten security functionality classes, F, and one of six independent evaluation levels, E. To match the TCSEC classes of FHM interest, we have the following 2-tuples: (F4, E4), (F5, E5), (F5, E6), corresponding to B2, B3, A1, respectively. The first five functional classes of the ITSEC, F1 ... F5 match the six functional classes of the TCSEC, C1 ... A1, with F5 functionality the same for B3 and A1.

The six ITSEC evaluation classes are applied to the TOE Development Process (DP), Development Environment (DE), and Operational Environment (OE), each of which is further divided as follows:

DP: Requirements, Architectural Design, Detailed Design, Implementation

DE: Configuration Control, Programming Languages and Compilers, Developer Security

OE: Delivery and Configuration, Setup and Operation

The ITSEC is so new there is limited practical experience to draw from in looking at the applicability of FHM, so we must look at the published companion document, the Information Technology Security Evaluation Manual (ITSEM) [ITSEM92]. The ITSEM builds on the ITSEC describing how a TOE will be evaluated according to the ITSEC to provide a basis for "mutual recognition" of evaluation certificates by the participating nations. It defines measures for achieving mutual recognition of test results -- reproducibility, repeatability, objectivity, and impartiality. It also describes organizational rules and procedures for the evaluation process. It defines the Information Technology Security Evaluation Facility (ITSEF), the independent laboratory that performs the evaluations -- similar to the NVLAP idea noted

in Section 5.3.2., and the Certification Body (CB), which assures impartial and proper evaluation -- similar to the TCSEC DAA -- and issues the certificate of compliance with the security standards of the ITSEC.
Penetration testing is widely noted in the ITSEM and follows TCSEC practice quite closely. Penetration testing "... is based upon an ITSEC interpretation of the SDC Flaw Hypothesis Methodology ..." ITSEM breaks the four stages of FHM into five sequential test sub-activities: prepare, identify, specify, execute, and follow up. These follow quite closely this guideline, which is directly applicable as written.

Testing for errors and vulnerabilities is required by the ITSEC even at evaluation class E1, retesting of corrected flaws is required at E3, independent vulnerability analysis is needed at E4, and all these requirements are cumulative with higher evaluation classes. These test requirements are quite similar to those addressed by the FHM described in this guideline.

Assurance of a TOE is divided in the ITSEC into correctness and effectiveness. Correctness is based on the six evaluation classes, E1-E6. Effectiveness of a TOE involves a number of considerations: the suitability of the security functionality for the proposed environment, analogous to the TCSEC environment guidelines [CSC85]; whether the functionality yields a sound security architecture; ease of use of security functions; assessment of the security vulnerabilities during development and operation; and the strength of the security mechanisms to resist attack. All these items are "Generators" in the FHM, (see Section 3.4).

The FHM depends on discovering failures in the Evidence Implication Chain, starting with a security policy, (see Section 3.1). The application of FHM to a TOE would require a similar procedure. A TOE has a hierarchy of security policies: a System Security Policy (SSP), a System Electronic Information Security Policy (SEISP), and a Security Policy Model (SPM), corresponding to security objectives, detailed security enforcement mechanisms, and semi-formal policy model, respectively. These policies are tied to the functional classes and form the basis for the correctness criteria for testing. Together with the evaluation classes, an Evidence Implication Chain is formed for a specific TOE, and FHM can be successfully applied.

5.4.3 Federal Evaluation Criteria

Protection profiles are defined by the Federal Criteria for Information Technology Security (FC) as "... an abstract specification of the security aspects of a needed information technology product." It consists of five elements: Description, Rationale, Functional Requirements, Development Assurance Requirements, and Evaluation Assurance Requirements. Like the CTCPEC and the ITSEC, prepackaged profiles can be defined for the TCSEC digraphs. The FC defines the functional and trust sub-elements in a manner similar to the other criteria. These definitions also appear consistent (harmonized) with the CTCPEC and ITSEC.

Penetration testing in the FC is graded into four levels based on the scope, precision, coverage, and strength of the analysis methods used. PA-1 Basic Penetration Testing, is limited to unprivileged users, application program interfaces to the TCB, public documentation, and examples of past flaws. PA-2 Flaw Hypothesis Testing, extends PA-1 with design and implementation documents, source code, specifications, and tests based on the FHM. PA-3 Penetration Analysis augments PA-2 with penetration-resistance verification methods similar to those described in Section 5.2. PA-4 Analysis of Penetration Resistance, augments PA-3 with formal verification proofs of the TCB penetration resistant properties using theorem proving tools as described in Sections 5.1.4 and 5.2. Evidence from these components are

the plans, procedures, and results of the tests and analyses. Again, as with the other criteria, FHM is applicable to the FC.

In summary, FHM should be equally applicable to all the national evaluation criteria and the material of this guideline is germane.

6.

6. APPENDICES

Sections 6.1 through 6.4 are the following four appendices:

- Abbreviations And Acronyms
- Flaw Hypothesis Sheets (FHS)
- Model Penetration Testing Tasking
- Model WBS Schedule, Milestones, And Labor

The appendices are intended to help evaluators and their management with procedures and planning.

6.1 Abbreviations And Acronyms

The following abbreviations and acronyms are used in this guideline:

ACM	Association for Computing Machinery
AF	Air Force
AI	Artificial Intelligence
APA	Automatic Penetration Analysis
A1, B3, B2	Higher evaluation classes in the Orange Book (TCSEC) in decreasing security strength
B1, C2, C1	Lower TCSEC evaluation classes in decreasing security strength
CCA	Covert Channel Analysis
CDR	Critical Design Review, DOD-STD-2167A
CERT	Computer Emergency Response Team, DARPA initiative
CIPSO	Commercial Internet Protocol Security Option, LAN security label format
CMP	Configuration Management Plan, TCSEC
COMPUSEC	Computer Security
CONOPS	Concept of Operations
CPIF	Cost Plus Incentive Fee type contract

CSC	Computer Software Component, DOD-STD-2167A
CSCI	Computer Software Control Item, DOD-STD-2167A
CSU	Computer Software Unit, DOD-STD-2167A
CTCPEC	Canadian Trusted Computer Product Evaluation Criteria
DAA	Designated Approving Authority, TCSEC
DAC	Discretionary Access Control, TCSEC
DARPA	Defense Advanced Research Project Agency
DBMS	Data Base Management System
DDN	Defense Data Network
DIDS	Distributed Intrusion Detection System
DOD	Department Of Defense
DOS, DS	Denial Of Service flaw, FHM
DTLS	Detailed Top Level Specification, TCSEC
EBP	Evaluation By Parts, TDI
EPL	Evaluated Product List, NCSC
FC	Federal Criteria for Information Technology Security
FCA	Functional Configuration Audit, DOD-STD-2167A
FHM	Flaw Hypothesis Methodology
FHS	Flaw Hypothesis Sheet
FP	Fixed Price type contract
FQT	Final Qualification Test, DOD-STD-2167A
FTLS	Formal Top Level Specification, TCSEC
HDM	Hierarchical Development Methodology, also a formal specification language
HH,HM ... LL	FHS priorities from High-High, High-Medium to Low-Low
ICD	Interface Control Document, a part of a system specification
IDES	Intrusion Detection Expert System
ID, IDs	User or component Identification
I&A	Identification and Authentication
Ina Flo	Ina Jo flow analysis tool used in CCA of FTLS
Ina Jo	A formal specification language for FTLS
Ina Test	Ina Jo symbolic execution tool for rapid prototyping FTLS
IN	Installation Flaw
IP	Internet Protocol
ISOA	Information Security Officer Assistant
ITSEC	Information Technology Security Evaluation Criteria, European counterpart to the TCSEC
ITSEF	Information Technology Security Evaluation Facility for ITSEC
ITSEM	Information Technology Security Evaluation Manual for ITSEC
JCL	Job Control Language, IBM computers
MAC	Mandatory Access Control, TCSEC
MCP	Master Control Program, e.g., Operating System
MS-DOS	MicroSoft Disk Operating System for PCs.
NCSC	National Computer Security Center, NSA
NIDX	Network Intrusion Detection eXpert system
NIST	National Institute of Science and Technology, formerly the National Bureau of Standards (NBS)
NRL	Naval Research Laboratory
NTCB	Network Trusted Computing Base, TNI
NVLAP	National Voluntary Laboratory Accreditation Program, NIST

ODC	Other Direct Costs financial category in contracting
O&S	Operation and Support documents, DOD-STD-2167A
PCA	Physical Configuration Audit, DOD-STD-2167A
PDR	Preliminary Design Review, DOD-STD-2167A
PQT	Preliminary Qualification Test, DOD-STD-2167A
PSW	Program Status Word, IBM computers
PV	Policy Violation flaw, FHM
RAMP	Ratings And Maintenance Program, NCSC
RIPSO	Revised Internet Protocol Security Option, DDN security label format
RVM	Reference Validation Mechanism, TCSEC
SEE	Software Engineering Environment
SDD	Software Design Document, DOD-STD-2167A
SDI	Strategic Defense Initiative
SDIO	Strategic Defense Initiative Project Office
SDP	Software Development Plan, DOD-STD-2167A
SDR	System Design Review, DOD-STD-2167A
SEISP	System Electronic Information Security Policy, ITSEC
SFUG	Security Features Users Guide, TCSEC
SPM	Security Policy Model, ITSEC
SPS	Software Product Specification, DOD-STD-2167A
SRR	System Requirements Review, DOD-STD-2167A
SRS	Software Requirements Specification, DOD-STD-2167A
SSP	System Security Policy, ITSEC
SSS	System/Subsystem Specification, DOD-STD-2167A
STD	Software Test Description, DOD-STD-2167A
STR	Software Test Result, DOD-STD-2167A
TC	Total Control flaw, FHM
TCB	Trusted Computing Base, TCSEC
TCSEC	Trusted Computer System Security Evaluation Criteria, NCSC
TDI	Trusted Database Management System Interpretation of the TCSEC
TFM	Trusted Facility Manual, TCSEC
TNI	Trusted Network Interpretation of the TCSEC
TOCTTOU	Time-Of-Check-To-Time-Of-Use flaw
TOE	Target Of Evaluation, ITSEC
TRR	Test Readiness Review, DOD-STD-2167A
WBS	Work Breakdown Structure -- project tasks, labor, schedule

6.2 Flaw Hypothesis Sheets (FHS)

FHM candidate flaws are documented on Flaw Hypothesis Sheets (FHS), one page (record) per flaw. A FHS is intended to be concise and easy to complete as the ideas surface during the penetration testing. The FHS contains seven fields, all text strings.

The first FHS field is PROBLEM ID, a flaw name for identification with the syntax LLLLDD, where "L" = letter, and "D" = digit. We have found it convenient for the letters to encode the evidence item, e.g., "FS" = FTLS, and the initials of the evaluator assigned, with the digits sequencing each flaw candidate per system area or module. Typically, there can be hundreds of flaws, but seldom more that a few per area.

Field 2 is PRIORITY, a two-part ranking of the FHS:

(1) An assessment of the probability of the flaw being confirmed.
(2) The damage impact of the flaw on the protection of the system, if realized.

Both probabilities are measured on a scale of High, Medium, or Low. The combined assessment of HH, HM, HL, MH, ... , LL yields an overall scale of nine for ranking FHS. The ranking is valuable in allocating resources during the Flaw Confirmation stage of penetration testing.

Field 3 is WORK FACTOR, a H, M, or L estimate of the work required to demonstrate a flaw. High work-factor flaws, e.g., cracking encryption codes, are given lower consideration for High effort even if the flaw is ranked HH.

Field 4 is REFERENCE SOURCE, the evidence inventory name or number in the configuration management system from which the FHS was generated. This field's syntax is system specific.

Field 5, VULNERABILITY, is an English description in detail of the potential flaw, its location in the evidence, the environment conditions, the weakness perceived, a characterization of the flaw taxonomy, e.g., residue problem. It should be brief, but sufficient for another evaluator to understand and pursue if the FHS is reassigned. The ten most productive generators (listed in Section 3.4).

Field 6 describes briefly the ATTACK STRATEGY for demonstrating the flaw and for exploiting it if confirmed. Sections 3.4 and 4.3 are useful taxonomies for description. Sufficient detail of the attack should be noted to aid another evaluator, e.g., "trolling for passwords in the print spool file." Suggestions should be noted of closely related attacks, or a family of attacks due to a design, implementation, or operational flaw, e.g., "coding error in macro 26." The FHS attack strategy listed is one suggested approach to confirming the flaw. The evaluator has the last word, and may find alternative and better approaches. It is the result that counts. The approaches tried should be posted in the FHS after the fact so they are not repeated by others.

The last field, Field 7 is ASSESSMENT, which documents the approach(es) tried and the results achieved -- confirmed, not confirmed, partial success, not tried. The assessment should qualify results obtained from: live test, observation, or Gedanken experiment. If there is a formal test report, the result should reference the appropriate sections.

The total set of FHSs becomes the flaw database that guides and documents the penetration testing. The FHSs can be mechanized with a word processor or a DBMS. Most penetration tests adapt the FHS fields to their liking and available tools. Figure 6-1 is an example FHS.

PROBLEM ID:	SOCW-1
PRIORITY:	HH
WORK FACTOR:	L
REFERENCE SOURCE:	TFM
VULNERABILITY:	Training Wheels Operator ID = SO (Security Officer) Password may be preset from vendor.
ATTACK STRATEGY:	Try typical passwords Password = SO = vendor's name
ASSESSMENT:	Confirmed Password = ADMIN Tried various passwords and obtained root privilege.

Figure 6-1. Example Flaw Hypothesis Sheet (FHS)

6.3 Model Penetration Testing Tasking

Table 6-1 gives a model two-level Work Breakdown Structure (WBS) of tasks for a typical FHM penetration test. Adapt the model as necessary to the needs of each penetration testing effort. Most tasks are described in the main text; however, a few need some expansion here.

Table 6-1. Model FHM WBS

	WBS Tasks
1.	Management
	1.1 Penetration Plan Development
	1.2 Team Selection and Management
	1.3 Lab and Facilities Management
	1.4 Program Control & Contracts
	1.5 CM/DM - Evidence, Tests, Results (FHS Database)
	1.6 Customer/Vendor Coordination-Requirements/Materials, Progress Briefs
	1.7 Scheduling & Progress Status
2.	Training & Preparation
	2.1 FHM & Professional (Ethical) Rules of Conduct
	2.2 Penetration Testing Past History
	2.3 Target Of Evaluation (TOE) System Architecture Review
	2.4 TOE TCSEC Evidence Walk Through
	2.5 Tools Familiarity - TOE & FHM
3.	Generation
	3.1 Initial Brainstorming Sessions
	3.2 Dependency Graph Generation
	3.3 Confirmation Strategies & Staff Assignments
	3.4 Building and Ranking FHS Database
4.	Confirmation
	4.1 Daily Results Status Meetings - Replanning Priorities
	4.2 Gedanken Experiments
	4.3 Live Test Planning - Procedures, Tools, TOE Configuration
	4.4 Live Testing
	4.5 FHS Database Updated - Results & New FHS
5.	Generalization
	5.1 Weekly Results Analysis Brainstorming Session
	5.2 Confirmed Flaw Induction Analysis - Understanding Generic Cause
	5.3 FHS Database Updated - Results & New FHS
	5.4 Repeat Generation Stage for New FHS and Insights
6.	Elimination
	6.1 Confirmed Flaws Documented - Expand FHS
	6.2 Customer/Vendor Flaw Presentation
	6.3 Test Customer/Vendor Flaw Fix
	6.4 Regression Testing After All Customer Fixes
7.	Wrap Up
	7.1 Final Report Prepared - Results, Residual Flaws
	7.2 CM/DM Material Properly Disposed Of - Return, Destroy, Store

6.3.1 Management

If not part of a larger program, many penetration testing support tasks must be performed by a management task. The penetration testing will require access to vendor hardware/software, and team utilization of the TOE lab and facilities. Configuration Management/Data Management (CM/DM) is required for all TCB evidence, lab equipment, software, penetration testing tools, FHS database, and test results. The team must assure protection of all proprietary and classified items. Work is heaviest at the start and finish of the effort to plan, initiate, finalize, and complete the penetration testing.

6.3.2 Training

This could be part of Management; however, it is treated separately here to make visible the extent of the training needed on the TOE. For an experienced team, FHM, tools, and history training are not required. Training is an early scheduled effort to establish a common knowledge base for the team.

6.3.3 Elimination

If penetration testing continues long enough for earlier flaws to be repaired by the customer/vendor, these fixes need to be retested. However, the fixed system should be viewed as a new release of the system/product, thereby requiring a regression test of all prior confirmed flaws, and new tests to assure that the fixes did not introduce new flaws. Regression testing is not covered in the Figure 6.2 WBS labor estimates, only fix retesting. Regression testing is not included because the vendor will usually take longer to fix flaws in a new release than the time allotted for the whole penetration test, and because regression testing a new release is essentially a whole new penetration test.

6.3.4 Wrap Up

The final stage is preparing the final report for the DAA or designated recipient, and disposing in a sensitive manner, of all the CM/DM materials collected.

6.4 Model WBS Schedule, Milestones, And Labor

Figure 6-2 presents a model WBS, labor-loaded by person-months (PM) per month for the top-level (level 1) penetration testing tasks. Labor-loadings also show the task scheduling. Management effort is spent throughout the testing, with greater effort at the start and finish. CM/DM is performed throughout the testing. Training is up front for some of the team. Generation makes a big push early, but sees effort over much of the performance period since ideas are triggered by Generalization and other tasks. Confirmation is throughout the effort flowing one month behind Generation. It is 33% of the total effort. Generalization begins about the middle of the effort when confirmed flaws begin to accumulate, and drops off toward the end. Elimination is a vendor task. The effort shown is for the team to coordinate and prepare flaw descriptions for the vendor, and to retest any fixed flaws. Wrap up is also a management task, but separated here because it deals with technical matters, preparing the final report and disposing of all sensitive materials collected in CM/DM during the effort. Overall, management-related tasks -- Management, Training, and Wrap Up -- are about 33% of the total effort. If there exists a permanent

penetration testing organization performing multiple and concurrent penetration tests with experienced folks, the management effort can be halved.

WBS	Task	Months						Totals	
		1	2	3	4	5	6	%	pm
1.	Management	1	0.5	0.25	0.25	0.5	1.5	16.67%	4
2.	Training	1.5	0.5					8.33%	2
3.	Generation	1	1.5	0.5	0.5	0.5		16.67%	4
4.	Confirmation	0.5	1.5	1.75	2.25	1.5	0.5	33.33%	8
5.	Generalization			1	0.5	0.5		8.33%	2
6.	Elimination (*)			0.5	0.5	0.5	0.5	8.33%	2
7.	Wrap Up					0.5	1.5	8.33%	2
	Totals	4	4	4	4	4	4	100.0%	24

* Vendor labor to fix not shown, only team coordination & retests

Figure 6-2. Model WBS Schedule And Labor-Loadings
(All labor shown is in person-months, pm)

7.

7. REFERENCES

[ABBO76] Abbott, R.P., et.al., "Security Analysis and Enhancement of Computer Operating Systems," NBSIR 76-1041, National Bureau of Standards, ICST, Gaithersburg, MD., April 1976.

[ANDE72] Anderson, J.P., et.al., "Computer Security Technology Planning Study, Volume 2," NTIS: AD-772 806, NTIS, October 1972.

[ATTA76] Attanasio, C.R., P.W. Markstein and R.J. Phillips, "Penetrating an Operating System: A Study of VM/370 Integrity," IBM Systems Journal, Vol. 15, No. 1, 1976, pp. 102-106.

[BAUER88] Bauer, D.S. and M.E. Koblentz, "NIDX - A Real-Time Intrusion Detection Expert System," Proceedings of the Summer 1988 USENIX Conference, June 1988.

[BELA74] Belady, L.A. and C. Weissman, "Experiments with Secure Resource Sharing for Virtual Machines," Proceedings of the International Workshop on Protection in Operating Systems, IRIA/LABORIA, Rocquencourt, Le Chesnay, France, August 1974.

[BELL76] Bell, D.E. and L.J. LaPadula, "Secure Computing Systems: Unified Exposition and Multics Interpretation," MTR-2997 Rev.1, MITRE Corp., Bedford, MA., March 1976.

[BISB78] Bisbey, R. II and D. Hollingworth, "Protection Analysis Project Final Report," ISI/RR-78-13, DTIC AD A056816, USC Information Sciences Institute, Marina Del Ray, CA., May 1978.

[BISH82] Bishop, M., "Security Problems with the UNIX Operating System," Computer Science Department, Purdue University, West Lafayette, Indiana, March 1982.

[BOEHM86] Boehm, Barry W., "A Spiral Model of Software Development and

Enhancement," reprinted in Software Engineering Notes,
Association for Computing Machinery Vol 11, No 4, pp. 14-24, August 1986.

[BULL91] Bull, A., C.E. Landwehr, J.P. McDermott and W.S. Choi,
"A Taxonomy of Computer Program Security Flaws," Center for Secure
Information Technology, Naval Research Laboratory, draft in
preparation, 1991.

[CARL75] Carlstedt, J., R. Bisbey II and G. Popek, "Pattern-Directed
Protection Evaluation," ISI/RR-75-31, USC Information Sciences
Institute, Marina Del Ray, CA., June 1975.

[CSC85] Computer Security Center, "Computer Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," CSC-STD-003-85, 25 June 1985,
(The Yellow Book).

[CTCPEC92] Canadian System Security Centre, "The Canadian Trusted Computer Product Evaluation Criteria," Version 3.0e, Communications Security Establishment,
Ottawa, Canada, April 1992.

[DENN86] Denning, D.E., "An Intrusion-Detection Model," Proceedings of the
1986 IEEE Symposium on Security and Privacy, Oakland, CA.,
April 1986, pp. 118-131.

[DIAS91] Dias, G.V., et. al., "DIDS (Distributed Intrusion Detection System) -
Motivation, Architecture, and An Early Prototype," Proceedings of the 14th
National Computer Conference, Washington, D.C., October 1991, pp. 167-176.

[DOD85] Department of Defense, "Trusted Computer System Evaluation Criteria,"
DOD 5200.28-STD, December 1985, (The Orange Book).

[DOD88] Department of Defense, "Military Standard: Defense System Software
Development," DOD-STD-2167A, 29 February 1988.

[ECKM87] Eckmann, S., "Ina Flo: The FDM Flow Tool," Proceedings of the 10th
National Computer Conference, Baltimore, MD., September 1987, pp. 175-182.

[ECKM85] Eckmann, S., and R.A. Kemmerer, "INATEST: an Interactive Environment

for Testing Formal Specifications," Software Engineering Notes,
Vol. 10, No. 4, August 1985, pp. 17-18.

[FARM90] Farmer, D. and E.H. Spafford, "The COPS Security Checker System,"
CSD-TR-993, Department of Computer Sciences, Purdue University,
West Lafayette, Indiana, 1990.
(Software available by anonymous ftp from cert@sei.cmu.edu)

[FC92] National Institute of Standards and Technology, National Security Agency, "Federal Criteria for Information Technology Security," Version 1.0, December 1992

[GALI75] Galie, L.M., R.R. Linde and K.R. Wilson, "Security Analysis of the Texas Instruments Inc. Advanced Scientific Computer," TM-WD-6505/000/00, System Development Corporation, Washington, D.C., June 1975.

[GALI76] Galie, L.M., and R.R. Linde, "Security Analysis of the IBM VS2/R3 Operating System," TM-WD-7203/000/00, System Development Corporation, Washington, D.C., January 1976.

[GARF91] Garfinkel, S., G. Spafford, Practical Unix Security, O'Reilly & Associates, Inc., Sebastopol, CA., 1991.

[GASS88] Gasser, M., Building a Secure Computer System, Van Nostrand Reinhold, New York, N.Y., 1988.

[GE91] "Trusted Software Development Methodology," Volume 1, CDRL A075, GE Aerospace, Blue Bell, PA., 30 September 1991.

[GUPTA91] Gupta, S. and V.D. Gligor, "Towards a Theory of Penetration-Resistant Systems and its Application," Proceedings of the 4th IEEE Workshop on Computer Security Foundations, Franconia, N.H., June 1991, pp. 62-78.

[GUPTA92] Gupta, S. and V.D. Gligor, "Experience With A Penetration Analysis Method And Tool," Proceedings of the 15th National Computer Security Conference, Baltimore, MD., October 1992, pp. 165-183.

[HEBB80] Hebbard, B., et. al., "A Penetration Analysis of the Michigan Terminal System," ACM SIGOPS Operating System Review, Vol 14, No. 1, January 1980, pp. 7-20.

[HOLL74] Hollingworth, D., S. Glaseman and M. Hopwood, "Security Test and Evaluation Tools: an Approach to Operating System Security Analysis," P-5298, The Rand Corporation, Santa Monica, CA., September 1974.

[ITSEC91] Commission of The European Communities, "Information Technology Security Evaluation Criteria (ITSEC)," Version 1.2, CD-71-91-502-EN-C, Office For Official Publications of The European Communities, Luxembourg, June 1991.

[ITSEM92] Commission of The European Communities, "Information Technology Security Evaluation Manual (ITSEM)," Draft Version 0.2, Brussels, Belgium, April 1992.

[KARG74] Karger, P. and R. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD-TR-74-193, Vol. II, NTIS AD A001120, HQ ESD, Hanscom AFB, June 1974.

[KEMM83] Kemmerer, R.A., "The Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels," ACM Transactions on Computer Systems, August 1983, pp. 256-277.

[KEMM86] Kemmerer, R.A., "Verification Assessment Study Final Report, Volume I, Overview, Conclusions, and Future Directions," C3-CR01-86, Library No. SW-228,204, Department of Computer Science, University of California, Santa Barbara, CA., March 27. 1986.

[KRAJ92] Krajewski, M. Jr., "Concept for a Smart Card Kerberos," Proceedings of the 15th National Computer Security Conference, Baltimore, MD., October 1992, pp. 76-83.

[KRAM83] Kramer, S., "The MITRE Flow Table Generator - Volume I," M83-31, Volume 1, The Mitre Corporation, Bedford, MA., January 1983.

[LAMP73] Lampson, B.W., "A Note on the Confinement Problem," Communications of the ACM, Vol. 16, No. 10, October 1973, pp. 613-615.

[LUNT92] Lunt, T.E., et. al., "A Real-Time Intrusion Detection Expert System (IDES) - Final Technical Report," SRI International, Menlo Park, CA, February 1992.

[LIND75] Linde, R.R., "Operating System Penetration," Proceedings of the National Computer Conference, Vol. 44, AFIPS Press, Montvale, N.J., 1975.

[LIND76] Linde, R.R. and R.F. von Buelow Jr., "EXEC-8 Security Analysis," Memo. Rep. 3205, Naval Research Laboratory, January 1976.

[MCAU92] Mc Auliffe, N., "Extending Our Hardware Base: A Worked Example," Proceedings of the 15th National Computer Security Conference, Baltimore, MD., October 1992, pp. 184-193.

[MCPH74] McPhee, W.S., "Operating System Integrity in OS/VS2," IBM Systems Journal, No.3, 1974, pp. 231-252.

[MUFF4a] Muffett, A.D.E., "Crack Version 4.0a, A Sensible Password Checker for Unix," Computer Unit, University College of Wales, Aberwystwyth, Wales. (Software available by anonymous ftp from cert@sei.cmu.edu)

[NCSC88] National Computer Security Center, "Trusted Product Evaluations -

A Guide For Vendors," NCSC-TG-002 Version 1 (Draft), 1 March 1988.

[NCSC88b] National Computer Security Center, "Trusted Network Testing Guideline," NCSC-TG-010 Version 1 (Draft), August 1988.

[NCSC89] National Computer Security Center, "Rating Maintenance Phase - Program Document," NCSC-TG-013 Version 1, 23 June 1989.

[NCSC92] National Computer Security Center, "Trusted Product Evaluation

Questionnaire," NCSC-TG-019 Version 2, 2 May 1992.

[PARK75] Parker, D.B., "Computer Abuse Perpetrators and Vulnerabilities of Computer Systems," Stanford Research Institute, Menlo Park, Ca., December 1975.

[PHIL73] Phillips, R. , "VM/370 Penetration Study Final Report," TM(L)-5196/006/00, System Development Corporation, Santa Monica, CA., October 1973.

[RADC90] "SDI Battle Management Security Study, SDS Secure Development Methodology," RADC-TR-90-353, Vol IV, Rome Air Development Center, Griffiss AFB, NY, December 1990.

[RFC1038] "Revised Internet Protocol Security Options, RIPSO," RFC 1038, Network Information Center. See also RFC 791, and RFC 1108.

[RUB86] Rub, J.W., "Penetration Handbook," The Aerospace Corporation, El Segundo, CA., January 1986.

[ROYCE70] Royce, W.W., "Managing the Development of Large Software System: Concepts and Techniques," Proceedings of the 1970 WESCON, August 1970.

[SCHE79] Schell, R.R. Lt. Col., "Computer Security: The Achilles' Heel of the Electronic Air Force?" Air University Review, Vol. XXX No. 2, January-February 1997, pp. 15-33

[SDC75] "Fujitsu, Ltd. Security/Privacy Report," TM(L)-5593/000/00, System Development Corporation, Santa Monica, CA., October 1975.

[SDC76] "A Security and Integrity Analysis of OS/VS2 Release 3," TM-5662/000/00, System Development Corporation, Santa Monica, CA., April 1976.

[SMAHA88] Smaha, S.E., "Haystack: An Intrusion Detection System," Proceedings IEEE Fourth Aerospace Computer Security Applications Conference, Orlando, FL., December 1988.

[SPAF89] Spafford, E.H., "Crisis and Aftermath," Communications of the

ACM, Vol. 32, No. 6, June 1989, pp. 678-687.

[STOL89] Stoll, C., The Cuckoo's Eqq, Pocket Books, New York, N.Y., 1989

[TANE87] Tanenbaum, A.S., Operating Systems Design and Implementation, Prentice Hall, Englewood Cliffs, N.J., 1987.

[THOM84] Thompson, K., "Reflections on Trusting Trust," Communications of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763.

[TDI91] National Computer Security Center, "Trusted Database Management System Interpretation," NCSC-TG-021 Version 1, April 1991, (The Violet Book).

[TNI87] National Computer Security Center, "Trusted Network Interpretation," NCSC-TG-005 Version 1, 31 July 1987, (The Red Book).

[TRAT76] Trattner, S., "Tools for Analysis of Software Security," ATR-77(2780)-1, The Aerospace Corporation, El Segundo, CA., October 1976.

[WEIS94] Weissman, C., "Penetration Testing," Information Security Essays, Abrams, M.D., S. Jajodia, H. Podell, eds., IEEE Computer Society Press, 1994

[WEIS73] Weissman, C., "System Security Analysis/Certification Methodology and Results," SP-3728, System Development Corporation, Santa Monica, CA,. October 1973.

[WILK81] Wilkinson, A.L., et. al., "A Penetration Analysis of the Burroughs Large System," ACM SIGOPS Operating Systems Review, Vol. 15, No. 1, January 1981, pp. 14-25.

[WINK92] Winkler, J.R., and J.C. Landry, "Intrusion And Anomaly Detection: ISOA Update," Proceedings of the 15th National Computer Security Conference, Baltimore, MD., October 1992, pp. 272-281.

[WHEE92] Wheeler, T., Holtsberg, S. and S. Eckmann, "Ina Go User's Guide," TM-8613/003, Paramax Systems Corporation, Reston, VA., 1992.