Teaching Security Engineering Principles

Cynthia E. Irvine and Timothy Levin
Computer Science Department, Naval Postgraduate School, Monterey, California

Abstract: The design and construction of secure systems cannot be entirely captured in textbooks or class notes, but must be taught as an art which is learned through apprenticeship and practice. This paper describes a course in Secure Systems that uses the Flaw Hypothesis Methodology for penetration testing as a vehicle for motivating and teaching students fundamental principles of security engineering.

Key words: Security Engineering, Education, Principles, Flaw Hypothesis Methodology

1. INTRODUCTION

At our institution a course entitled "Secure Systems" is taught on an annual basis. It is an advanced graduate-level course intended to teach students about the design and implementation of secure platforms. The catalog course description is given below.

> The course covers implementation of protection for both monolithic and distributed secure computer systems. The importance of system architecture to assurance methodologies for security kernels is emphasized. Topics will include the use of protection hardware, the implementation of virtual machines through the effective use of memory management techniques including segmentation and paging, synchronization mechanisms, critical sections, software engineering methodologies as applied to the development of secure computer systems, and configuration management techniques. Critical topics in database security will be discussed.

The course covers a wide range of security engineering principles as well as mechanisms for their realization. As with many courses, students learn

best when they are fully engaged in the topic. To facilitate this engagement, laboratory activities are particularly useful. Our view is that security engineering is relatively immature compared to some other engineering disciplines, and, as a result, may be best taught as a hands-on art form rather than an abstract science.

Laboratory exercises in secure system construction could take several forms. First, the instructor could devise a series of activities that are completely hypothetical with no connection to a real system whatsoever. This may not be particularly realistic and may have both the appearance and pedagogical value of a toy. Another approach would be to inspect a secure system. This would require access to source code for a secure system that had been constructed using the principles the instructor wished to convey. Unfortunately, the known systems that fall into this category are proprietary. A third path would be to add security to an open source system. This could work, but the instructor may need to take great pains to ensure that students distinguish between principle-based security engineering and undisciplined system development that can emerge from a system built using ad hoc construction methods. Finally, students can examine a demonstrably flawed system and study what principles should have been applied in design and implementation. In this case students can examine either a proprietary or an open source system, but instead of constructing the system they deconstruct it.

For our course we chose the last approach. Instead of sending students off to attempt ad hoc penetrations of the target system, use of the Flaw Hypothesis Methodology was employed.

The objective of this paper is to describe the principles we teach in our lectures and to describe how a complementary laboratory exercise based upon the use of a penetration exercise based upon the Flaw Hypothesis Methodology was created. We note that, while the subject course concentrates primarily on the design of secure operating systems, the deconstructive pedagogic techniques described here can be used for teaching design of all sorts of secure systems, including networks and database management systems.

The remainder of this paper is organized as follows. Section 2 provides a review of the major principles addressed in the course. In Section 3 an outline of the motivation for, and objectives of, penetration testing in a security engineering context is sketched. The application of a penetration exercise in the context of our course is described in Section 4. A brief discussion of our activities is given in Section 5

2. PRINCIPLES OF SECURE SYSTEMS

The objective of the class was to teach principles of secure systems. These principles were derived from a combination of experience and the literature [5, 1, 8, 9] and include:

Modularity and layering

At the highest level, modularity and layering are engineering disciplines, the application of which result in systems whose structure is comprehendible. Division of code into well-formed modules provides the opportunity to isolate functions and related data structures into compact logical units. Layering allows the relationships of these units to be better understood, so that dependencies are clear, and undesired logical looping can be avoided. A system built with assurance against internal looping will be less error prone and more resistant to denial of service attacks.

Economy of mechanism

This principle means that the designer should keep the system design as simple and small as possible. A small and simple design will be more understandable, more analyzable, and less prone to errors.

Complete - always invoked

This principle means that a strong mechanism must be effective against all parties and under all circumstances. There are no superusers for whom the mechanism does not apply, and it cannot be tampered with to undermine its effectiveness.

Non-bypassable

If a protection mechanism can be avoided, for example by using an alternate interface to access an object, then the mechanisms will be ineffective, even if it is always present.

Least Privilege

Each entity is given only enough privilege to complete the job at hand and not privileges that might go far beyond those associated with the task.

Fail-safe defaults

When the system is initialized it comes up in a secure state and if the system crashes or an error occurs, the system does not enter a state in which protection mechanisms are no longer in effect. If the user does not specify the security characteristics for an operation, either the system will use secure defaults or it will not perform the operation.

No Security through obscurity

Although a system may not be open source, the foundations of its security should be based upon the quality of its design and the rigor of its engineering process. Since it is hard to know whether or not an adversary knows the details of a protection mechanism, it is hard to know when a mechanism ceases to be effective that has been secured through obscurity.

Usability

Secure systems should be appealing to potential users. If security mechanisms are frustrating or difficult to use, then they will be disabled or avoided.

Attention to shared variables and resources

As the popularity of timing attacks on smart cards and other devices intended to store secrets continues, students must be reminded that design of secure systems requires attention to the adversary's ability to monitor global variables and shared resources [11].

Separation of Privilege

In many circumstances, the division of a privilege between multiple parties will be able to decrease the likelyhood of that privilege being abused. For example, with a dual-keyed safe deposit box, neither the banker nor the customer can access the box surreptitiously.

Privileged Resources and tools

The commands and resources utilized by the control systems should not be available to general users. For example, certain memory locations and processor instructions are reserved for use by the operating system, resulting in a two state architecture for many modern processors. Furthermore, the means through which a process gains entry to the privileged state needs to be strictly controlled.

Per-user resource allocation

In multi-user systems, the resources allocated to one user must be protected from access by another user. Separate process address spaces are an example of the application of this principle.

Information Hiding and Encapsulation

The principle of information hiding means that access to a set of related data is managed by a minimal number of functions. Ideally, these functions reside within the same module. The functions should not be dependent on the internal representation of the data.

Analysis

The engineer must develop system security requirements by applying a threat model. The attacker may view the system from an entirely different perspective than the designer and may easily find simple ways to bypass security. Systems must be examined to find the weakest link in the chain of security mechanisms, for this is the avenue of attack that will be most attractive to the adversary. Analysis of the work factor needed to exploit the weakest link is needed. The notion of work factor analysis is amply demonstrated in cryptography, where cryptanalytic attacks are found to be more attractive than brute force attacks on the key space. Time-of-check-to-time-of-use (TOCTOU) attacks are common and careful design and analysis

are required to construct a system that does not present TOCTOU vulnerabilities.

Engineering Discipline

A course objective is to teach students methodological development processes, such that the other principles can be applied in a careful and complete manner. For example, in the absence of strongly typed programming languages, engineers need to treat data consistently to have semantically clear data access methods. The mere use of object-oriented approaches and object-based languages does not guarantee good engineering (e.g., information hiding); it must be taught and practiced.

In this class the students are taught not only the theoretical notions associated with these principles, but also how they are applied in real systems. The technology of secure computer construction is fairly immature in comparison to such fields as residential home construction, or industrial bridge construction. In those fields, there are "effective procedures" for applying the body of engineering principles to known problems. Furthermore, the construction-industry technologies are mature enough that most new problems can be adapted to the principles and existing solution frameworks in a straightforward manner. The result is that consumers are highly confident that new constructions are "safe." Secure computer technologies are not that mature. In general the security engineering community does not have "effective procedures" for applying our principles to standard or new security problems, and consumer confidence in the security of new products is generally low. Instead of teaching rote procedures for system construction we rely on apprenticeship and experience to pass along the methods for applying our principles. In the absence of generalized effective procedures for secure system design and construction we recognize that security engineering is a form of art, best taught through hands-on experience.

A good jazz musician will tacitly learn music theory and the art of playing. The music student may take lessons for years and never be asked to study music theory as a separate subject. Yet without some internalization of music theory, the musician is a mere mechanic and can neither create new music nor understand the nuances intended by other composers. Even though music theory may not be explicitly taught, it is impossible to improvise effectively without an understanding of how music is constructed and the interrelationships between the notes in major and minor keys. In contrast, if a musician studied only music theory and never practiced an instrument or improvization, then how could music be created?

Security engineering is similar. A secure system cannot be built by staying at the desk and studying theory and abstract systems. Instead those ideas must be practiced on real systems. Both the desk learning and security

engineering benefit from pedagogical guidance. The teacher can guide and accelerate the student through the material and can provide useful graded materials for practicing the security engineering art.

A number of readings, including those previously cited, contributed to the students' appreciation of security engineering [10, 2, 12].

3. FLAW HYPOTHESIS METHODOLOGY

The Flaw Hypothesis Methodology was developed by Linde [6] and has been described in further detail by Weissman [14, 15]. It provides an organized approach to testing a system for possible errors in implementation and is intended to complement other aspects of developmental assurance. The major objectives of flaw testing are to provide assurance of correct functional behavior and to examine the penetration resistance of the system. This testing takes place during the last stages of system development (and is not conducted by customers after the system has been fielded.) To date, the FHM has been shown to be a useful framework for finding system flaws, yet it is not a formula for completeness: the transitive closure of all of the potential flaws in the system cannot be achieved using this or perhaps any other approach.

The problems revealed by testing include: design and implementation errors, user errors, and configuration errors. Design and implementation errors can have a tremendous impact on the system if discovered after it has been fielded. They can be both difficult and expensive to remedy. If documentation and interfaces are inadequate, user errors can render the system vulnerable to attack. Similarly, configuration errors can result from a number of factors and may be easily remedied through improved interfaces, defaults, and documentation. Penetration testing is employed and complements functional testing, which is insufficient for secure systems. Functional testing determines whether a particular requirement is met and if the interfaces and execution are correct. It is not complete, does not test extreme conditions, nor does it reveal anything about hidden functionality, bad configurations or usage. Penetration testing is not needed to reveal the types of flaws found through functional testing.

Penetration testing involves an exercise in attempting to circumvent the security features of the system. Penetrators “do everything wrong” and think “out of the box” and destructively. They assume the worst of the system developers. They ignore the system documentation guidance and warnings regarding parameter limits, use of functions, and configuration. They exploit design weaknesses and explore the vulnerabilities of rarely used functions.

The Flaw Hypothesis Methodology consists of six phases.

Definition

During this phase the penetration team determines the purpose and goals of the penetration exercise. They attempt to understand why they are conducting the exercise. These objectives might include one or more of the following: discovery of interesting threats to the system, support a risk assessment of the system; categorization of attacks relative to security objectives such as confidentiality, integrity and availability; categorization of potential attackers; and enumeration of possible points of entry used to mount an attack on the system.

The team must also determine what will count as success. A penetration exercise can continue forever, so that before starting, the team must define a stopping point. This may be in terms of flaws discovered or time invested, for example.

Background Study

At this point the penetration team attempt to understand the target system. This will enhance the likelihood of making a valid flaw hypothesis. The team will amass as much system documentation as possible, examining manuals, code, reports, papers, etc. Each team member will choose a particular component and become a "team expert" in that area. During this phase, the team is already thinking like adversaries asking "What if?" questions and searching for documented warnings that one "should not" do something to the system.

Brainstorming and Hypothesis Generation

To start this phase of the penetration exercise, each team member makes a presentation to the entire team about his or her area of expertise. This is usually in the form of an interactive lecture. During these lectures and subsequent brainstorming sessions, team members must refrain from judgemental comments and promote a free exchange of ideas. To ensure that no one is left out members should be regularly polled for their opinions. The search for system errors has started and the team will look for errors in a variety of areas: software, firmware, and hardware. They will seek design errors, implementation errors, interface flaws, and configuration errors.

These brainstorming activities result in the generation of flaw hypotheses, often stated as a set of conditions, interface functions, and inputs, that will result in exploitation of a system flaw. The flaw may be minor, however if it is beaded with other flaws, the ultimate result may be a system penetration.

As the flaw hypotheses are generated a flaw hypothesis database is created. Once the database is populated, it is possible for the team to examine the hypotheses to determine if certain errors can be generalized and are applicable to other parts of the system. The general types of errors include I/O control, algorithmic blunders, access control errors,

undocumented functionality, unintended program or data sharing, and timing flaws.

The final step for this phase is to prioritize the hypothesized flaws for subsequent verification.

Hypothesis Verification

Some flaws can be verified through desk checking while others may require the creation or use of special tools. During this phase team members pool their talents to explore the various hypothesis. Verification of a flaw will be recorded in the Flaw Hypothesis Database.

An important part of this phase is frequent interaction with team members for further brainstorming, hypothesis generation, and hypothesis reprioritization.

Hypothesis Generalization

During this phase, the team may derive a taxonomy of flaws that appears to be applicable to the target system. The generalization technique may permit additional flaw hypotheses associated with a flaw class to be generated.

Documentation

The last step for the team is to create a report documenting the results of the penetration exercise. The document will report hypotheses that were successfully verified as well as those that were not found in the target system. The team will provide a rationale for the prioritization of hypotheses for verification. The results of the report may provide guidance to system developers who may choose to take remedial action. There are ethical considerations that must be addressed in a penetration exercise: it may not be wise to broadcast the list of flaws to the newspapers, but instead notification of responsive vendors or of emergency response teams may be appropriate.

4. EXERCISE IMPLEMENTATION

This section describes our use of a penetration exercise as the laboratory component for our course on secure systems.

Prerequisites for this course include Operating Systems, Networks and Introduction to Computer Security. Because the course was not restricted to computer science students a few students from other curricula enrolled. In addition, even the computer science students came from varied backgrounds. Some had undergraduate degrees in electrical or computer engineering, while others had received undergraduate degrees in somewhat less technical information systems programs. Careful assessment of student backgrounds and skills helped to make the laboratory exercise rewarding for the entire group.

4.1 Team Effort

The flaw hypothesis methodology permitted the laboratory experience to be a team effort. Too often work on computers is viewed as the enterprise of the lone genius. For large systems, this is far from the norm. Major software and system development projects usually involve a team whose members are expert in particular areas to contribute various talents to the collective effort. Through the synergistic interaction of these individuals, team creativity and productivity is amplified and large, complex systems can be developed.

Unfortunately, many students emerge from undergraduate and graduate programs with little or no experience in team efforts. In the usual educational experience, students are often sent off to conduct laboratory and homework exercises alone or in pairs. To help remedy this shortfall in their educational experience, the penetration testing exercise was organized as a team effort.

During the first day of class, students were surveyed to determine their background and skills in computer science and engineering. A sample questionnaire is illustrated in Table 1. The information provided by the students allowed the instructor to ensure that each team included members whose collective skills would be sufficient to carry out the penetration exercise. It is imperative that the instructor collect enough information to be able to create well balanced teams.

Students were divided into teams of four or five. Each student was assigned a *subsystem role* and was responsible for becoming the local expert on that particular subsystem of the operating system. These subsystems included:

– Memory Management
– Process Management
– I/O Subsystems
– File System
– Graphical Interface

In addition each student was assigned a *service role* in which he or she provided a functional service to the team. The service roles included:

Tools builder - this individual was responsible for the design and construction of specialized tools needed by the team for specific penetration tests. The individual chosen for this role should have experience programming in both assembly and a high order language.

Report editor - this task involved recording and organizing the material provided by other team members into a coherent report. The report editor might also take the role of team leader.

Table 1. Student Questionnaire

One of the principle activities of this course will be an evaluation of an operating system using the Flaw Hypothesis Methodology. This work will be conducted by a team working on NT and other systems in the Computer Security Lab.

Team assignments will be made so that each team has members with appropriate strengths and talents. This questionnaire is intended to help your instructor make useful assignments.

1. Have you ever programmed in assembly? If so, give us an idea of how much experience you have in this area
2. Do you know C , C++, or Ada. How much experience do you have?
3. Have you ever used MS Access?
4. Have you ever designed a database?
5. Have you ever administered a Windows NT system?
6. Have you ever administered a system accessed by multiple users? If so, describe.
7. Have you ever managed a software development project involving several designers and programmers?
8. Do you know how to backup systems?
9. Have you every backed up a system?
10. Have you ever installed a system? Which one(s)?
11. Have you ever been a member of a software development team?
12. Have you used Linux, BSD Unix, or another flavor of Unix?
13. Have you ever installed Linux, BSD Unix or another flavor of Unix?
14. Have you ever installed a Windows system (any kind - 95, 98, NT, etc.)?
15. Are you good at finding the information you are looking for easily on the WWW?
16. Does using the WWW make you feel as if you are going into information overload?
17. Approximately how many e-mail messages are in your "in" box? Approximately how many folders do you maintain?
18. Do you maintain your bookmarks in a "flat" system or hierarchically?
19. If you downloaded source for a tool, could you compile it and make it run on your home system (assuming that it was targeted for that system)?
20. Do you like to design software?
21. Do you like to implement software?
22. Do you dislike designing and/or implementing software?
23. What is the longest program you have ever written? Was writing it a satisfying experience
24. Is there something else you would like to relate that can help the instructor make an informed team assignment?

Database designer - The Flaw Hypothesis Database was constructed by the database designer. Background in database design was a useful prerequisite for this assignment.

Web surfer and tool collector - This individual was required to find published weaknesses in the target system and existing tools that would help the team with their penetration efforts. Although the web surfer/tool collector was not required to be an expert programmer, a sense of what

constituted a useful tool as opposed to one that would be a waste of time was needed.

System Administrator - This individual was responsible for the management and configuration of both the target system and the development platform used for managing the penetration exercise. Tasks included system installation, configuration management, backups, and user administration.

To supplement the readings previously cited, students were asked to review several classical works on system penetration [3, 4].

4.2 Target System

For the penetration exercise, students examined the Microsoft Windows CE (WinCE) operating system. WinCE is a small footprint operating system intended for use in hand-held devices, embedded systems, and as a diskless system to support thin-client platforms. Because of the proprietary nature of the system, we were unable to provide each team with source code. Instead each team had a development system and a target platform. The target platform was a Windows CE PC obtained from Special Computing.[1] The development systems were standard PCs upon which the WinCE Platform builder had been installed. Using the development platform, students could develop applications, examine header files, download software to the target, and run a debugger. Each team was provided with a development and a target platform.

So that students could understand system internals, two books were used as assigned reading: Inside Windows NT [13] and Inside Windows CE [7]. The former provided students with insight regarding the design and construction of the vendor's flagship operating system. Our initial hypothesis was that parts of the WinCE operating system were intentionally designed for the smaller platform, but that certain subsystems would be lifted directly from the main product. The latter book provided a rather colloquial account of some of the design decisions that went into the creation of the WinCE operating system.

4.3 Approach

During the first week of class the goals for the penetration study were established. Since our student teams lacked experience in many areas, it

[1] WinCE is not designed to be plug-and-play. The operating system must be compiled with the exact configuration needed to run on the target platform. The default target platform is the CE PC

would have been unrealistic to have expected them to do more than scratch the surface of the target system in the twelve-week quarter we had for the entire course. Only a team with considerable knowledge and experience in penetration testing, an understanding of operating systems, and a deep appreciation for security engineering principles could have conducted a truly meaningful penetration exercise in the allotted time.

At the start of the course, some students considered complete reverse engineering of the target system and were dissuaded from this approach. Despite considerable eagerness to start attacking the systems, students were not permitted to begin the actual penetration tests during the first several weeks of class. Instead, they were required to thoroughly study the target system by reading as much material as possible. This reading included not only the books and papers provided by the instructor but materials the students found on their own. During this period students were attending lectures where the principles of security engineering were being described.

Each student prepared a presentation on his or her subsystem and presented it to the team. This proved to be the most intense period of the course for the instructor. Because of the students' inexperience, and for student assessment purposes, the instructor attended every presentation and helped to guide the discussion of the system component. All of the presentations were completed in a period of approximately two weeks and most were scheduled at times outside of both regular class and laboratory hours. With a class of seventeen students each of whom gave a one to two hour lecture to a team, the instructor had to block out most other activities in order to attend these brainstorming sessions.

After the student presentations, the students were allowed to proceed at will. Many had subsequent discussions with the instructor regarding fine points associated with particular system features. In the meantime, normal class lectures on secure system engineering proceeded.

4.4 Student Assessment

Students were assessed based upon the criteria shown in Table 2. Several elements in the table are traditional, however those associated with the laboratory are unique and require additional clarification.

Table 2. Student Assessment Criteria

Component	Portion of Grade
Class participation and demonstration of Insight	10%
Teamwork assignments	20%
Individual work assignment (OS component presentation and FHS tests)	20%
Midterm Examination	25%
Team Final Report	25%

Each student was required to make a presentation on the OS subsystem that he or she had been assigned. The presentations usually lasted between one and two hours. Other team members asked questions during the presentation and the instructor asked questions to guide the discussion.

The penetration study report produced by each team constituted a major portion of each student's grade. Three different grades were given to each student based upon the final report. First each students wrote individual sections describing how they carried out their service roles and how that effort contributed to the overall success of the team. Second, they had to produce a subsystem report describing aspects of the penetration analysis as it applied to their subsystem. For example, they might give a brief overview of the subsystem followed by detailed discussions of the flaw hypotheses applicable to that subsystem. This would include references to the flaw hypothesis sheets as well as test results related to hypotheses. In the case of beaded attacks, students wrote a coordinated section describing how interaction between subsystems or sequential use of those subsystems could result in a successful penetration. Finally, the general quality of the report was graded. Although students were prohibited from assisting each other on their subsystem and service assignments, the application of a grade to the report as a whole motivated students to ensure that their team mates were producing high quality results in the penetration exercise. In a sense, this overall grade truly reflects the real world: when a team is involved in an engineering project, the project either succeeds or fails as a whole and all team members either share in the benefits of a job well done or the implications of a poorly executed project.

5. DISCUSSION AND CONCLUSIONS

This course has been taught twice using a penetration exercise as the laboratory component. In each instance the students remained interested in both the classroom and laboratory material. The lectures taught them the principles that the designers of the target system had neglected to observe. Discovery of flaws based upon a lack of principle-based security engineering validated the students' understanding of those principles and helped to demonstrate that secure systems must be achieved constructively.

Each time the course has been taught, students have been asked for recommendations for improvement. Almost unanimously they have suggested that the course is too short. They feel that during the twelve week span of the class, they are only beginning to appreciate the system and are only beginning to understand various approaches that could be taken by their

penetration teams. Many students did comment that they felt the course helped them to understand how the concepts that had been presented to them in their operating systems class were applied in real systems. Because the target system was different from those studied in traditional operating systems classes, they also noted that they were beginning to see a number of similarities among all of the systems.

The Windows CE operating system does not provide any significant mechanism for its own self protection. This design approach renders the system highly vulnerable to attack. A benefit of this condition is that the students are quickly validated in their penetration exercise. A drawback is that the teams were not confronted with a system designed to create isolation domains for individual processes and to protect the underlying system from tampering by applications.

To summarize, a course in secure systems design and implementation should be based upon fundamental principles. Using those principles as a framework, it is possible to develop exciting laboratory-based penetration exercises using the Flaw Hypothesis Methodology as an organizing technique.

ACKNOWLEDGEMENTS

The authors wish to thank the students who participated in our classes and, in particular, Barbara Pereira, whose efforts to set up and administer the CE PCs contributed significantly to the success of the penetration exercise. We also gratefully acknowledge the support of this work by the United States Navy (N643) and the Defense Information Systems Agency.

REFERENCES

[1] J.P. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. (Also available as Vol. I, DITCAD-758206. Vol. II, DITCAD-772806).

[2] R. Anderson and M. Kahn. Tamper Resistance - A Cautionary Note. In *Second USENIX Workshop on Electronic Commerce*, pages 1-11, Oakland, CA, November 1996.

[3] C. Attanasio, P. Markenstein, and R. J. Phillips. Penetrating an Operating System: A Study of VM/370 Integrity. *IBM Systems Journal*, 15(1):102-116, 1976.

[4] R. Bisbey and D. Hollingsworth. Protection analysis: Final Report. Technical Report ISI/SR-78-13, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, May 1978. ARPA ORDER NO. 2223.

[5] E. W. Dijkstra. The Structure of the “THE” Multiprogramming System. Communications of the A. C. M., 11(5):341-346, 1968.

[6] R. R. Linde. Operating System Penetration. In National Computer Conference, pages 361-367, 1975.
[7] J. Murray. Inside Microsoft Windows CE. Microsoft Press, Redmond, WA, 1998.
[8] D. L. Parnas, On the Criteria to be Used in Decomposing Systems into Modules. Communications of the A. C. M., 15(12):1053-1058, 1972.
[9] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9): 1278-1308, 1975.
[10] M. D. Schroeder, D. D. Clark, and J. H. Saltzer. The Multics Kernel Design Project. Proceedings of the Sixth A. C. M. Symposium on Operating System Principles, pages 43-56, November 1977.
[11] O. Sibert, P. A. Porras, and R. Lindell. The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems. In Proceedings 1995 IEEE Symposium on Security and Privacy, pages 211-222, Oakland, CA, May 1995. IEEE Computer Society Press.
[12] S. Smith and S. Weingart. Building a High-Performance, Programmable Secure Coprocessor. Computer Networks, 31: 831-860, November 1999.
[13] D. A. Solomon, Inside Windows NT. Microsoft Press, Redmond, WA, Second Edition, 1998.
[14] C. Weissman, Security Penetration Testing Guideline, Naval Research Laboratory, Unisys Government Systems, 12010 Sunrise Vally Drive, Reston, VA, tm - 8889/000/01, October 1993. Prepared under contract to NRL.
[15] C. Weissman, Penetration Testing. In Abrams, Jajodia, and Podell, editors. Information Security: An Integrated Collection of Essays, pages 269-296. IEEE Computer Society Press, Los Alamitos, CA, 1995.