2001-06

Analyzing input/output subsystem security in Windows CE

Pereira, Barbara A.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ANALYZING INPUT/OUTPUT SUBSYSTEM SECURITY IN WINDOWS CE

by

Barbara A. Pereira

June 2001

Thesis Advisor: Cynthia E. Irvine
Second Reader: Paul C. Clark

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Analyzing Input/Output Subsystem Security in Windows CE			**5. FUNDING NUMBERS**
6. AUTHOR(S) Barbara A. Pereira			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

In the past few years, mobile handheld devices have emerged as an exciting new tool for accomplishing everyday tasks. Devices with the Windows CE operating system provide flexibility for the designer in the form of customizable modules and components. With wireless capabilities and a familiar user interface, Windows CE devices are becoming popular for such tasks as inventory control and information retrieval. By enhancing the self-protection of the operating system, handheld devices could be used in more demanding environments. This thesis reviews the security redesign of operating systems and explores the applicability of such redesign to the Windows CE operating system. The existing security mechanisms in Windows CE are described, and the operating system itself is critically examined for security weaknesses, especially in the Input/Output subsystem area. Recommendations are made for improving the self-protection of Windows CE. Future work is suggested in two areas: analyzing other Windows CE subsystems in terms of security, and developing a method of authenticating a Windows CE device to a server.

14. SUBJECT TERMS Operating Systems, Handheld devices, PDA Security, Windows CE			**15. NUMBER OF PAGES** 116
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

ANALYZING INPUT/OUTPUT SUBSYSTEM SECURITY IN WINDOWS CE

Barbara A. Pereira
B.S. Electrical Engineering, University of Missouri - Columbia, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2001

Author: Barbara Pereira
Barbara A. Pereira

Approved by: Cynthia E. Irvine
Cynthia Irvine, Thesis Advisor

Paul C. Clark
Paul Clark, Second Reader

Dan C Boger
Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the past few years, mobile handheld devices have emerged as an exciting new tool for accomplishing everyday tasks. Devices with the Windows CE operating system provide flexibility for the designer in the form of customizable modules and components. With wireless capabilities and a familiar user interface, Windows CE devices are becoming popular for such tasks as inventory control and information retrieval. By enhancing the self-protection of the operating system, handheld devices could be used in more demanding environments. This thesis reviews the security redesign of operating systems and explores the applicability of such redesign to the Windows CE operating system. The existing security mechanisms in Windows CE are described, and the operating system itself is critically examined for security weaknesses, especially in the Input/Output subsystem area. Recommendations are made for improving the self-protection of Windows CE. Future work is suggested in two areas: analyzing other Windows CE subsystems in terms of security, and developing a method of authenticating a Windows CE device to a server.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the expert guidance of Cynthia Irvine, throughout the entire thesis process. I will never forget the importance of CAMs, AMs, or green eggs and HAM, and the other enlightening things that were learned in her class lectures. Paul Clark also deserves my thanks, for his patience and willingness to listen. Other members of the NPS faculty also have my appreciation for their excellent instruction, and I believe they know who they are. Finally, I would like to dedicate this work to my husband, Mike, for everything, and to my parents, who knew I could do it.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. WHY CHOOSE WINDOWS CE ?

Windows CE is an embedded operating system sold by Microsoft and customized by manufacturers to fit the needs of their consumers. It is a versatile operating system, and it has been used in many devices from industrial controllers to personal digital assistants. Windows CE is even used in cell phones and television set-top boxes. With all the flexibility it has to offer, not to mention its compatibility with Windows-based desktop computers, Windows CE is a good choice for many tasks commonly performed in government and military organizations.

On the other hand, with Palm OS devices accounting for approximately sixty percent of the sales of personal assistant type devices, one might wonder why Windows CE was chosen as the focus of this study. Pocket PCs, running a version of the Windows CE operating system, offer color display screens, have faster StrongARM processors, and present a familiar Microsoft Windows style interface. They are powered by rechargeable lithium ion batteries, rather than the AAA batteries found in most Palm OS devices. The Windows CE strategy is to offer a multifaceted array of services, rather than simply provide basic functionality such as a calendar or task manager.

With Windows CE, a user can work with Microsoft Word and Excel documents, listen to MP3s, and read digital books. However, the Windows CE operating system was not designed with security in mind. Due to the unique constraints and challenges presented by the hardware of most embedded devices, the foremost design objectives

were a small memory footprint and enhanced system performance. As with many things in life, compromises must be made between security and performance. The designers of Windows CE heavily favored performance over security, but it is not too late to take a look back at the system and determine how its self-protection might be improved.

B. OBJECTIVES

This thesis will explore the organization of Microsoft Windows CE 3.0. The operating system components are examined in detail, while keeping security issues in mind. Penetration testing conducted upon Windows CE will be discussed. Obvious security weaknesses in the operating system will be addressed, and suggestions for improvements will be made.

With the advantages of a security enhanced operating system, hand-held devices could be used in more demanding environments, such as onboard a Naval vessel. The aim of this thesis is to take a first look at methods for improving the self-protection of Windows CE. A more robust operating system could be utilized confidently in many governmental tasks requiring the use of hand-held devices.

Some type of control needs to exist on the process of upgrading the operating system or applications on a Windows CE device. Without controls, malicious software might be introduced or the system itself might be corrupted. Rather than allowing anyone to modify system critical software, some restrictions should exist. Trusted subjects would be allowed to upgrade the system or add new applications dynamically, providing new features without compromising the device's usability. The Windows CE operating system is used for embedded systems, possibly safety critical, in which its self-protection may be important.

C. THESIS ORGANIZATION

This thesis consists of five chapters, and two appendices. Chapter I introduces the topic, and provides some motivation for the study. Chapter II discusses important concepts related to the redesign of operating systems for security, and explores the implementation of security in a few modern operating systems. Chapter III describes the security mechanisms that are currently present in Windows CE. Chapter IV explains the Flaw Hypothesis Method. It also describes the penetration testing performed upon Windows CE, and the lab configuration required to support the tests. Chapter V examines the organization of the Windows CE operating system, while focusing on the I/O and communication modules. Recommendations are made for improving the self-protection of the system, and suggestions for future work are proposed. Appendix A describes Platform Builder and other Windows CE development software, as well as the CEPC reference platform. Appendix B outlines a selection of software analysis tools that could be used in analyzing the Windows CE source code.

THIS PAGE INTENTIONALLY LEFT BLANK

II. OVERVIEW OF SECURE OPERATING SYSTEMS

A. INFORMATION PROTECTION

Computers are a part of our everyday life, and they are used in many ways that require some level of information protection. Airline reservation systems, e-commerce customer records, credit bureau databases, law enforcement information systems, and hospital records are all examples of information stored on computers that should be protected. Computer mechanisms can be used to enforce an appropriate security policy that safeguards the sensitive information. The term security is used to denote the mechanisms and techniques that control who may use or modify a computer or the information stored on it.

In order to avoid known design problems, it is necessary to review the historical approaches to enhancing the security of an existing operating system. We know that a secure system should implement the Reference Monitor Concept [AND72], so that the system is self-protecting, always invoked, and capable of being analyzed. In this context, security refers to the mechanisms and techniques that control who may use or modify the computer or its information.

A potential security violation is the unauthorized release of information. This includes traffic analysis, in which an intruder observes the patterns of information use and infers some content from those patterns. It also includes the release of sensitive data or programs.

The unauthorized modification of information is another potential security violation. An intruder might be able to change some vital stored information, possibly without even seeing it.

Denial of computer and network services to authorized users is also a potential problem. The system could be flooded with specific requests, so that it is unable to respond, or the system could be made unresponsive in general. A simple power outage could also cause a denial of service.

In the above cases, unauthorized means that the action is happening without the permission of the person controlling the information, and possibly outside the constraints imposed by the system itself. Some techniques commonly used to prevent security violations are labeling files with lists of authorized users, verifying a user's identity by a password, shielding the computer to prevent the interception and interpretation of electromagnetic radiation, locking the room containing the computer, controlling who is allowed to modify the computer system's hardware and software, and certifying that the hardware and software are actually implemented as intended.

There are eight well-known design principles that apply to security protection mechanisms [SAL75]. These principles are economy of mechanism, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, and psychological acceptability.

Applying *economy of mechanism* means that the system design will be kept as small and simple as possible. A small system is easier to verify that it works according to the specifications. A simple, straightforward system lends itself to a thorough examination of its correctness.

Fail-safe defaults in a system ensure that the default action is to refuse permission and deny access. Thus, even if the system defaults are used, the system is still safe from unauthorized access. A default to allow access might go unnoticed until it was used by an intruder to penetrate the system. But, a default set to deny access would be noticed as soon as an authorized user tried to legitimately access the information. Specific permissions could be provided to authorized users to allow their tasks to be performed.

To have a secure system, every access to every object must be checked for authority. In other words, we need *complete mediation* between subjects such as processes, and objects like files or resources. If an object could be accessed without any checks, the system security would be bypassed as if it didn't even exist. Access control must be present system-wide, not just in a particular system component.

A system should not rely on the secrecy of its design for its security. An *open design* allows many people to examine the system for flaws or inconsistencies. As a result, the system can be made stronger. This is much better than just ignoring system weaknesses in the hope that no one will ever find them out. The security mechanisms should not rely on the ignorance of potential attackers to offer protection.

The design principle of *separation of privilege* guards against a single point of unauthorized access. In order to access an object, more than one condition must be met before access is allowed. For example, it would be like having two keys owned by two separate people to open one protected safe with the treasure inside. The two people must cooperate to open the safe, and if one person loses his key, the safe's contents are not compromised.

In order to limit the damage caused by an accident or error, every program and every user of the system should operate using the least set of privileges needed to complete the job. This is known as the design principle of *least privilege*.

Least common mechanism limits the mechanisms that are common to more than one user and depended upon by all users. Each shared mechanism represents a channel between users and must be designed very carefully to avoid compromising security, often unintentionally.

The final design principle is *psychological acceptability*. No matter how great the system is, people will not use it if it is strange, confusing, or even ugly. The ways available to interact with the system should match the user's ideas of how it should work. The system should be intuitive, and forgiving of unexpected user input. The system needs to match its representation of security features to what the user believes is needed. This mapping will minimize user errors and frustration.

The framework for a secure system can be built by using these design principles. This thesis investigates how these principles apply to a redesign of the CE operating system, and the CE file system, serial I/O, and synchronization mechanisms in particular.

B. THE CONCEPT OF A TRUSTED COMPUTING BASE

The Anderson Report [AND72] introduced the idea of a reference monitor. The reference monitor is responsible for system security by enforcing authorized access relationships between subjects and objects within the system. Subjects are active entities such as processes or threads. Objects are passive entities like files or database records. An implementation of the reference monitor concept is called a reference validation

mechanism. The reference validation mechanism must be tamper proof, always invoked, and small enough in size to be analyzed and tested to ensure its completeness.

A security kernel exists in computer systems that are designed and implemented strictly according to the reference validation mechanism requirements. Systems with the Microsoft Windows CE operating system do not have a security kernel. Most systems, including those running Windows CE, implement a reference validation mechanism as part of a general-purpose mechanism whose size and complexity make it very hard to analyze and test for assurance. Therefore, most security mechanisms present are just stuffed into various locations in the operating system, rather than being organized and modular. The TCSEC also describes the notion of a Trusted Computing Base, or TCB, to apply to the majority of computer systems, which do not have security kernels.

A TCB is defined as the part of a system which contains all the elements of the system that are responsible for supporting and enforcing security policy. According to this definition, non-security related mechanisms can reside within the TCB. However, all security critical mechanisms must be contained inside the TCB. When the TCB contains non-security mechanisms, it is generally accepted that such a system does not meet the common standards for high assurance.

To achieve controlled access protection of a system, the TCB enforces isolation of security related objects. The TCB must ensure that reusable objects do not contain residual data when they are allocated. The TCB also maintains audit information on the use of identification and authentication mechanisms, object access, object deletion, and the activities of users, system administrators, and security administrators. The TCB must

isolate protected resources to ensure that access controls and audit requirements are enforced.

C. HISTORIC EXAMPLES

In any engineering task, it is always good to take a look back at previous related designs so that mistakes are not repeated. Good design approaches can be reused to jump-start project development, and major pitfalls can be avoided by noticing those that have trapped people in the past. Two major security redesign projects of the 1970's were the Multics kernel redesign, and the KVM/370 redesign.

1. Multics Kernel Redesign

In the paper "The Multics Kernel Design Project" [SCH77], Michael Schroeder, David Clark, and Jerome Saltzer discussed the work that was done to make the Multics kernel more secure. Multics stands for Multiplexed Information and Computing Service. It was a mainframe timesharing operating system that was developed in 1965. Originally, Multics was a joint research project by MIT's Project MAC, Bell Telephone Laboratories, and General Electric Company's Large Computer Products Division. The system evolved into a commercial operating system that was sold by Honeywell. Multics was used in many sectors of the population, from education to government and industry.

The main objectives of the design project were to demonstrate security for a large, working operating system, and to show that Multics could enforce a security policy. A few troubling problems were encountered along the way. Today, decades later, we still have the problem of just how to evaluate and understand large amounts of source code that were written by many people over time. Another problem was how to express all of the security mechanisms in a simple, verifiable model. Then, as now, ad hoc security

mechanisms existed outside the operating system, and were hard to incorporate into the implementation of a security kernel.

People agreed that some type of integrity audit was essential before the Multics operating system could be used to protect sensitive information, such as military data existing at different security classification levels. To make Multics more secure, the kernel had to be simplified so that an evaluator could understand it. By simplifying the kernel, the risk of overlooking a threat to system security was reduced. A second subgoal of the project was to provide a set of security functions that could be described by a simple formal model. By describing the security functions in a formal model, their correctness could be verified more rigorously than an informal review would permit. This formal model provided more assurance that the security functions were behaving in accordance to the defined security policy. For more details on the model used, see the paper "The Multics Kernel Design Project" [SCH77].

The Multics kernel design project explored some major issues, such as the possibility of being evaluatable, the actual usefulness of the formal model, and the impact of the security redesign on overall system performance. Some questioned whether it was even possible to evaluate a system of such a large size. There was a concern that the cost of using a formal model might outweigh the model's usefulness to the project as a whole. Finally, system performance was also a design compromise between security mechanisms and responsiveness. There would not be a large consumer market for a system that was very secure and yet very slow.

Instead of developing their own formal model, the Multics team used the MITRE model of sensitivity levels and compartments [BEL73]. This model constrained the flow

of information by preventing information to be written down from a higher security level to a lower level. The model also prohibited reading information at a higher security level than the current security level. AIM, the Access Isolation Mechanism, was a set of security features that was added to Multics. AIM implemented the labeling of all information, and enforced security checks at all points where information could cross level or compartment boundaries.

At that point in the Multics design project, several goals were pursued. Work was done to add AIM to Multics, and to develop prototypes to implement security functions in better, more efficient ways. More detailed formal system specifications were developed, and then verified. The central supervisor was reimplemented to allow more verification.

The new, easier-to-review version of Multics with AIM was named Kernel/Multics. One of the difficult parts of the project was to certify that Kernel/Multics complied with the documented specifications. In order to do that, program verification tools were used, the source code was audited, user testing and error reporting were conducted, and penetration testing of the system was performed. All of these methods were employed to verify the integrity of Kernel/Multics.

Three major redesign proposals were made. The first was to remove protected supervisor functions from the kernel if the functions did not absolutely need to be there. The idea was to put only the minimal necessary functions in the protected kernel. However, moving some of the functions out of the kernel had an adverse effect on overall system performance. The second proposal was to implement the protected functions by using the natural separation given by independent processes operating in distinct address

spaces. The third proposal was to use systematic program structuring techniques for the kernel. Ideas from all three of these proposals were used.

Type extension is a method that makes all modules into object managers, categorizes the ways one module can depend on another, and organizes modules into a loop-free dependency structure. This method was used to implement structured programming in the kernel. Each module was an object manager, so the interface to each module defined all the operations on the object type managed by that module. Some examples of object types used in the Multics kernel design are disk records, core blocks, core segments, and page frames. Dependencies between modules can be explicit, caused by procedure calls or interprocess messages that wait for replies. Implicit dependencies are those caused by direct sharing of writable data among modules. In the Multics redesign, it was necessary to remove the dependency loops so that the correctness of each module could be established independently.

In order to identify all the dependency loops between modules, the dependencies were placed into one of five categories. These categories were component dependencies, map dependencies, program storage dependencies, address space dependencies, and interpreter dependencies. Component dependencies exist when a module's objects are composed of another module's objects. Map dependencies occur when a module depends on the managers that provide the objects in which the map of object names to component names is stored. Program storage dependencies are related to where a module's algorithms and data are temporarily stored. An address space dependency means that a module executes in an object's address space, which depends on the module's managers. For interpreter dependencies, a module depends on the module that

implements its interpreter, or the virtual processor on which it executes. By identifying module dependencies, the Multics redesign effort allowed their elimination to produce a loop-free collection of object managers.

So, it was possible to implement the complete Multics kernel functionality through a loop-free structure of object managers by using the rationale of type extension and the five kinds of dependencies. By doing this, it became feasible for a single evaluator to examine the kernel and be assured of its correctness. Kernel/Multics was developed to incorporate the necessary security mechanisms and make the kernel easier to review. The MITRE formal security model of sensitivity labels and compartments was applied to describe Multics system security functions in a clear and consistent way. The Multics kernel redesign proved that an existing operating system could be re-engineered to enhance its security.

2. VM/370 Redesign

The paper "A Security Retrofit of VM/370" explains the design work that was done to add a feasible, formally verified security kernel to the existing VM/370 operating system [GOL79]. A goal for the security kernel was to allow verification with respect to the security policy. It had a tolerable effect on overall performance, and required minimal replacement of existing code. The security kernel was also designed to be backwards compatible with existing applications.

The general redesign strategy was to partition the VM/370 control program into security-relevant and non-security-relevant modules. Modules that were security relevant were those that executed privileged instructions or accessed global system data. The plan

was to use encapsulation of multiple, individual copies of an operating system under a virtual machine monitor (VMM) system to provide a secure OS.

The new, security enhanced OS was called KVM/370. Its system architecture had four main components. The kernel and verified trusted processes operated in the supervisor ring, separated from the rest of the system. Audited semi-trusted processes were used to access virtual addresses. A Non-kernel Control Program (NKCP) existed for each security level, to act as an interface to the user Virtual Machines (VMs). The last component was user Virtual Machines that were each controlled by the appropriate NKCP for the appropriate security level.

Some design tradeoffs were considered for the resource scheduling and management. They could either be done in a system global manner, or on a NKCP local basis. If implemented globally, the size of the kernel and trusted processes would increase. There would be a complicated interface between the NKCPs and the global processes. Verification would be more difficult and expensive, and it would be harder to modify the system. However, performance would improve with this approach. By using a NKCP local basis for resource scheduling, most resource management would be done by the NKCPs. That would simplify the system design, implementation, verification, and interfaces. The problem with NKCP local approach is that system performance would be adversely affected. So, it was decided to use a mixture of the two approaches in order to preserve adaptability and ease of verification.

As part of the system redesign, the kernel and trusted processes were the only parts of the KVM/370 whose formal specifications were formally verified. The formal proof of correctness required that the kernel and trusted processes enforced the security

policy. The proof also required a demonstration to show that there were no unauthorized signaling capabilities, or covert channels, within the semi-trusted processes.

In general, the KVM/370 redesign was based on the principles of least privilege and least common mechanism. These two principles can also be applied to a redesign of the Windows CE operating system. When possible, security related modules should be separated out from the rest of the system. Overall system performance should be considered, and formal verification should be used if needed.

D. SECURITY IN MODERN OPERATING SYSTEMS

Modern desktop operating systems such as Windows NT or Linux do not incorporate a security kernel. However, they do offer various security services and forms of access control. Personal Digital Assistant (or PDA) operating systems such as Windows CE or Palm OS have rudimentary security features and limited self-protection of the operating system itself. As PDA devices are becoming more connected to networks and through the use of wireless technology, the security of such devices becomes a more important issue. It is interesting to take a look at how the different operating systems compare to the design principles outlined in the paper by Saltzer and Schroeder.

1. Windows NT Security

There is a thorough explanation of Windows NT security features in the book *Inside Windows NT*, by David Solomon [SOL98]. Economy of mechanism is proposed in the design of Windows NT, but it is debatable whether it can be applied to an operating system of such a large size. As for fail-safe defaults, NT first checks if the desired access to an object is allowed. The default is to deny access, which is safe in any case.

Mediation is accomplished through the use of Access Control Lists (ACL). The system design is partially open due to the books written about how NT behaves, even though the source code is not available. Separation of privilege is accomplished through the use of different user roles, such as administrator, power user, user, and guest. The principle of least privilege is met by providing various access modes such as read, write, execute, and append. Least common mechanism is observed through the use of various modules for different tasks. The security reference monitor (SRM), kernel, local security authority (LSA), and security accounts manager (SAM) each provide needed features in a modular way. Mechanisms that must be shared are logged. As for using an isolated virtual machine, a real CPU abstraction called the hardware abstraction layer (HAL) with virtual resources is provided. The authentication mechanism is met by the Windows logon, which is flexible enough to incorporate a password, smartcards, or biometrics. For shared information, NT uses an Access Control List when opening objects, and a ticket called a handle when accessing objects. Linear 32-bit addressing is used in Windows NT. NT is not a capability system and has no tag bits. NT security is based on ACLs and permissions. As for protection groups, NT uses system groups and user groups. The authority to change ACLs is given to the owner and the administrator, as well as anyone else that the owner or administrator has given permission to. NT only supports a Discretionary Access Control (DAC) policy. There is an ACL strategy for all objects, and permissions are object dependent. NT does have protected subsystems, such as the local security authentication server (LSASS), but they are not available to users.

2. Windows CE Security

The orgainization and design objectives of Windows CE are discussed in the book *Inside Microsoft Windows CE*, by John Murray [MUR98]. Details of Windows CE

security features can be found in the Microsoft technical article "Creating a Secure Windows CE Device" [ALF00]. Windows CE addresses economy of mechanism by keeping the OS size small, and by introducing modules and components. Since memory space in embedded devices is at a premium, effort was taken to make the OS as small and streamlined as possible. Unnecessary modules or components for a specific device can be removed, simplifying the design. Windows CE does not support fail-safe defaults. There is very little security implemented in its default configuration. There are no access permissions associated with objects, no user identification or authentication, and the default is to run all processes and threads in kernel mode to enhance system performance. There is a mechanism to certify code modules, but it is turned off by default. A unique device identifier may be assigned, but one is not given in the default configuration. There is no mediation at all, since subjects and objects are not assigned access modes. There is a password for the device, but the user must first enable that feature. The system design is open to certain manufacturers who have agreements with Microsoft. Otherwise, the design is partially open, because its high level structure is documented on Microsoft's web sites and in some books. Separation of privilege, least privilege, and least common mechanism do not apply, because a complete set of protection mechanisms does not exist in Windows CE. There is a real CPU abstraction called the OAL, or OEM Adaptation Layer, between the actual hardware and the operating system itself. The OAL serves a similar function to the HAL in Windows NT. As stated before, there is no intrinsic authentication mechanism. However, support does exist for smartcard readers to be added in later. As for shared information, all processes share a common memory space, by default providing no distinction between kernel level and user level memory.

Windows CE does not use protection groups. The only protected subsystem is a portion of the operating system kernel.

3. Palm OS Security

The Palm OS is described in the book *Palm Programming: The Developer's Guide* [RHO98]. The Palm OS provides for economy of mechanism in much the same way as Windows CE. The OS has a small footprint that is well suited for the PDA devices it is typically installed upon. The OS is divided into subsystems called managers, such as the Memory Manager or the Database Manager. Considering fail-safe defaults, the Palm OS originally has its password and auto-lock features turned off. The user is responsible for enabling those features and for marking certain application data as "private". As with Windows CE, there is no mediation because access modes are not defined. User identification and authentication are not implemented. The system design is partially open, because books and online documentation are available that discuss the high-level design and programming interface. Separation of privilege, least privilege, and least common mechanism are not addressed, due to the lack of Palm OS security mechanisms in general. A user authentication mechanism is not provided. No protections groups exist, and only parts of the operating system are protected.

PDAs are used in a radically different environment from that of the typical desktop PC. Rather than sitting for long periods at a desk, PDA users walk around, enter short notes while in the middle of other tasks, and retrieve PDA information as needed on location. The desktop security model might not fully encompass the different ways that PDAs are used. When considering Windows CE security, one should be mindful of how

and where the PDA devices are used, and how its security approach might differ from that of a desktop computer.

III. SECURITY IN WINDOWS CE DEVICES

A. WINDOWS CE SOFTWARE TRUST APPROACH

According to the Microsoft article, "Creating a Secure Windows CE Device" [ALF00], Windows CE 3.0 provides an integrated set of security services with nine main features. The features include providing a "trusted environment" model, the Security Support Provider Interface (SSPI), support for Windows NT LAN Manager, support for the Secure Sockets Layer (SSL), cryptography, a smart card infrastructure that supports the Microsoft Cryptographic API (CAPI), a unique device identifier, a protected kernel configuration, and digital authentication in the dial-up boot loader.

Two Windows CE API functions are provided for Original Equipment Manufacturers, or OEMs, to implement in order to create a trusted environment. Using these functions can prevent unknown modules from being loaded, restrict access to system APIs, and prevent write access to selected parts of the system registry. As one can see, this definition of a trusted environment is not quite the same as having a TCB.

The first provided function is OEMCertifyModuleInit. It is called once for initialization of each RAM executable module to be checked. The function returns either true or false, depending on the success of the initialization. What this function actually does is determined by the needs of the OEM. For example, it could be setting up public keys or using the Loadauth library routines included with Platform Builder.

The second function is OEMCertifyModule. This function allows the OS loader to pass the module code, which could be a DLL or EXE type file, to the OEM for verification that the module is safe to run on the system. The return values for

OEMCertifyModule are OEM_CERTIFY_TRUST, OEM_CERTIFY_RUN, and OEM_CERTIFY_FALSE. OEM_CERTIFY_TRUST signifies that the module code is trusted to perform any operation. OEM_CERTIFY_RUN means that the module is trusted to run, but is restricted from making some selected privileged function calls. OEM_CERTIFY_FALSE means that the module is not trusted and therefore not allowed to run. To find out the trust level of a calling application, dynamic-link libraries can use the CeGetCurrentTrust and CeGetCallerTrust functions in addition to the OEM functions.

It is the responsibility of the OEM to perform the desired checks on the code module in the OEMCertifyModule function. These checks could be a cyclic redundancy check to verify integrity or a public key certificate check. Basically, this function is only as good as the means chosen to verify the code module, which will vary from one OEM to another. Some OEMs may choose not to certify modules at all.

When a dynamic-link library (or DLL) is loaded into the address space of an executable file (or EXE), the trust level of the EXE process determines the final access level. For example, when an EXE with the OEM_CERTIFY_RUN trust level tries to load a DLL that has a higher trust level of OEM_CERTIFY_TRUST, the final trust level of the DLL is lowered to OEM_CERTIFY_RUN. If an EXE tries to load a DLL with a lower trust level than its own, the DLL will fail to load. If the EXE trust level is OEM_CERTIFY_FALSE, the EXE will not run. If the DLL trust level is OEM_CERTIFY_FALSE, the DLL will fail to load. The different combinations of EXE and DLL trust levels are listed in the following Table 3.1 [ALF00].

EXE Trust	DLL Trust	Final DLL Trust
OEM_CERTIFY_RUN	OEM_CERTIFY_RUN	OEM_CERTIFY_RUN
OEM_CERTIFY_RUN	OEM_CERTIFY_TRUST	OEM_CERTIFY_RUN
OEM_CERTIFY_RUN	OEM_CERTIFY_FALSE	DLL fails to load
OEM_CERTIFY_TRUST	OEM_CERTIFY_RUN	DLL fails to load
OEM_CERTIFY_TRUST	OEM_CERTIFY_TRUST	OEM_CERTIFY_TRUST
OEM_CERTIFY_TRUST	OEM_CERTIFY_FALSE	DLL fails to load
OEM_CERTIFY_FALSE	RUN, TRUST, or FALSE	DLL fails to load, and EXE will not run

Table 3.1 Trust Level Combinations

It seems like it would be a good idea to use the two provided security functions to restrict unknown modules from running. However, in order to support third party drivers, the OEMs must digitally sign them, or have the check in OEMCertifyModule always return OEM_CERTIFY_TRUST for all of the chosen drivers or the drivers will be prevented from loading. This seems like a point that would deter the OEMs from choosing to use the two security functions in the first place. The problem of certifying modules that have not yet been written is one that demands careful thought. OEMs would most likely want to support as many drivers as possible, and new drivers might not be available at the time of the Windows CE device manufacture. This could potentially cause difficulties in implementing and using a module certification scheme based upon the functions OEMCertifyModule and OEMModuleInit.

Another point to remember is that the names of the functions OEMCertifyModuleInit and OEMCertifyModule are arbitrary and any names can be used as long as the kernel pointers pOEMLoadInit and pOEMLoadModule in the OEMInit function are initialized to the desired functions.

When a code module has a trust level of OEM_CERTIFY_RUN, it is restricted from calling the following API functions:

SetInterruptEvent
SetSystemMemoryDivision
CESetThreadPriority
ForcePageout
VirtualCopy
LockPages
UnlockPages
SetProcPermissions
SetKMode
ReadProcessMemory
WriteProcessMemory
SetCleanRebootFlag
PowerOffSystem
DebugActiveProcess

In the CreateProcess function, the debug flags DEBUG_ONLY_THIS_PROCESS and DEBUG_PROCESS are restricted as well. The registry architecture in Windows CE 3.0 permits only code modules with a level of OEM_CERTIFY_TRUST (which Microsoft calls trusted applications) to modify protected keys and values. The following registry root keys and their subkeys are protected:

HKEY_LOCAL_MACHINE\Comm
HKEY_LOCAL_MACHINE\Drivers
HKEY_LOCAL_MACHINE\HARDWARE
HKEY_LOCAL_MACHINE\SYSTEM
HKEY_LOCAL_MACHINE\Init
HKEY_LOCAL_MACHINE\WDMDrivers

To protect the registry, Windows CE restricts some applications from invoking certain registry function calls. Untrusted applications, which are those defined by Microsoft as not having the trust level of OEM_CERTIFY_TRUST, receive the return value of ERROR_ACCESS_DENIED if they try to use the registry functions RegSetValueEx, RegCreateKeyEx, RegDeleteKey, and RegDeleteValue.

Everything else in the registry is unprotected. Microsoft recommends that the OEMs place all their important registry information into one of the protected keys. Code modules with a trust level of OEM_CERTIFY_RUN can read all registry keys and values. In fact, all applications have read-only access to all of the keys and values in the registry.

B. UNIQUE DEVICE IDENTIFICATION

Another security service that Windows CE can provide is unique device identification. A DEVICE_ID data structure is provided for the OEM to store a unique identification number for each device. The input/output control IOCTL_HAL_GET_DEVICEID in the OEM adaptation layer (CE's version of the HAL, Hardware Abstraction Layer, in Windows NT) returns the current DEVICE_ID assigned to the Windows CE device. Device identification could be used for billing or security purposes. If using the device id for security, care should be taken that the usage fits into the overall security policy and that the implementation is sound.

Two modes of operation exist for the Windows CE kernel, protected mode and full-kernel mode. Protected kernel mode is available to reduce the vulnerability of the memory storage, but using it will cause overall system performance to degrade. The default mode of operation is full-kernel mode. Windows CE is vulnerable when using

full-kernel mode while running threads, since the security features are bypassed. In full-kernel mode, applications can access any physical memory in the system. This means that the system can be exploited by malicious applications that scavenge privileged information or delete files. Running in full-kernel mode increases performance at the cost of system security. Full-kernel mode can be disabled by setting the second bit of ROMFLAGS in the Config.bib file before building the operating system image. By disabling full-kernel mode, the kernel operates in protected mode, restricting the physical memory access of user level threads.

Windows CE has a dial-up boot loader stored in ROM that can be used to upgrade the OS image file, Nk.bin, using flash memory or a remote server. To help ensure the integrity of the OS image, digital encryption can be used to sign and verify image files. The dial-up boot loader uses the Microsoft Cryptography Application Programming Interface (CAPI) to authenticate data using the asymmetric hashing algorithm CALG_SHA, which produces a 160-bit hash value. The dial-up boot loader extracts the signatures from the manifest file and verifies the authenticity of each OS image file. In the case of authentication failure, the download process is halted and the user is notified. Platform Builder 3.0 provides three tools for digital authentication. Makekey.exe creates a public/private key pair. The program mksigs.exe signs the OS image files. The signatures can be appended to the manifest file by running the addsigs.exe program.

C. APPLICATION LEVEL SECURITY SERVICES

Windows CE also offers application level security services [ALF00]. These services include the Security Support Provider Interface (SSPI), Security Support Providers (SSPs), Windows NT LAN Manager Security Support Provider, Secure

Sockets Layer (SSL), Microsoft Cryptography API (Crypto API or CAPI), Cryptographic Service Providers (CSPs), and Smart Card Service Providers (SCSPs).

SSPI resides in the Secure32.dll module. It is an API designed for obtaining integrated security services for authentication, message integrity, and message privacy. SSPI acts as an abstraction layer between application level protocols and security protocols. The Windows CE SSPI provides access to the DLLs which contain the available authentication and cryptographic libraries. These DLLs are known as Security Support Providers (SSPs), or security packages. The SSPI allows the use of one or more SSPs by any application that desires them. It is also possible for an OEM to write customized security packages and add them to the registry.

According to the book *Network Programming for Microsoft Windows* [JON99], Winsock is the preferred interface for accessing a variety of underlying network protocols and is available in varying forms on every Win32 platform. Winsock is a network programming interface and not a protocol. The Winsock interface inherits a great deal from the Berkeley (BSD) Sockets implementation on UNIX platforms, which is capable of accessing multiple networking protocols. WinInet operates at the session layer. WinInet handles programming Windows Sockets (Winsock), TCP/IP, and Internet protocols.

Figure 3.1 illustrates the relationship between the SSPs, the SSPI, Winsock, and WinInet.

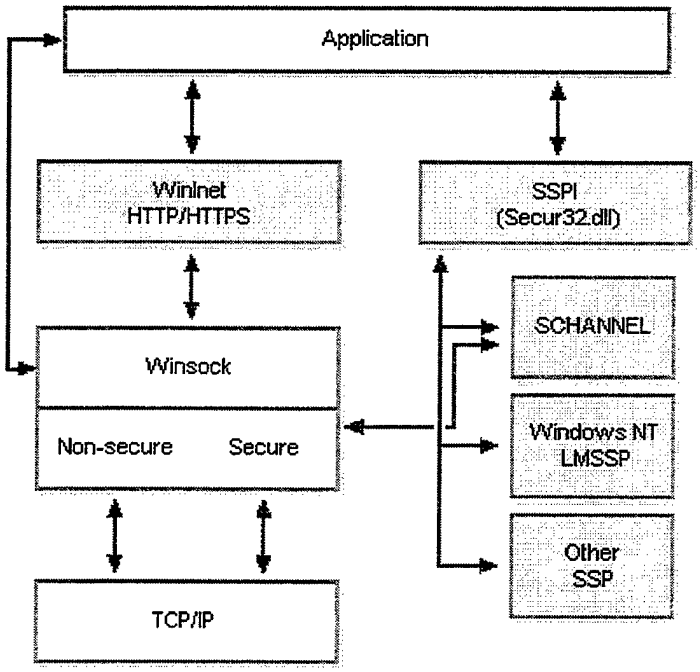

Figure 3.1 Security Support Provider Relationships [From: ALF00]

Windows NT LAN Manager Security Support Provider (NTLMSSP) is available for use with Windows CE. In client-server applications, NTLMSSP can be used by Windows CE applications for user authentication to a NT server. The client application supplies the user name, domain name, and password to the NTLMSSP and the server and client applications exchange tokens to complete the authentication. This type of client-server exchange could take place during an ActiveSync session.

Windows CE, unlike Windows NT, does not support impersonation. This means that Windows CE does not allow an object to acquire the security credentials of an authenticated user or client. Authentication under Windows CE is done at the TCP/IP level only. An authentication check is made the first time a client calls the server. If the client passes the check, no authentication takes place during subsequent calls to the server. In addition, an application may completely disable Windows CE authentication to a server.

Normally, a Windows NT domain controller manages the security credentials of subjects logging on and authenticates a Windows CE client to a network. However, in mobile situations, or when a Windows NT domain controller is not available, a local Windows CE database of user names and passwords can be created for the Windows NT LAN Manager security package to use for verifying credentials. It should be remembered that Windows CE databases are unprotected and any Windows CE application can access the data contained in them. This could possibly lead to compromise of the Windows NT user names and passwords.

Windows CE supports Secure Sockets Layer (SSL) versions 2.0 and 3.0 for providing some measure of secure network communications. SSL is available through WinInet or directly from WinSock. Applications can use secure sockets for transmitting and receiving encoded data. Windows CE maintains a database of trusted Certification Authorities independent of the Crypto API certificate store. Responsibility for verifying that a certificate is acceptable rests entirely upon the applications.

The Crypto API provides services for encrypting/decrypting data, authentication using digital certificates, and encoding/decoding of Abstract Syntax Notation One, ASN.1. ASN.1 is a flexible, abstraction notation that allows a variety of data types to be defined. Simple types such as integers and bit strings can be used to create structured types such as collections of one or more other types. Crypto API is compatible with many CSPs that perform the actual cryptographic functions, such as encryption and decryption, as well as key storage and key protection.

The Microsoft cryptographic system consists of three elements: the operating system, the applications, and CSPs. Applications interface with the OS through the CAPI

layer, and the OS communicates with the CSPs through the Cryptographic Service Provider Interface (CSPI). Figure 3.2 from [ALF00] represents the relationships between the three cryptographic system elements.

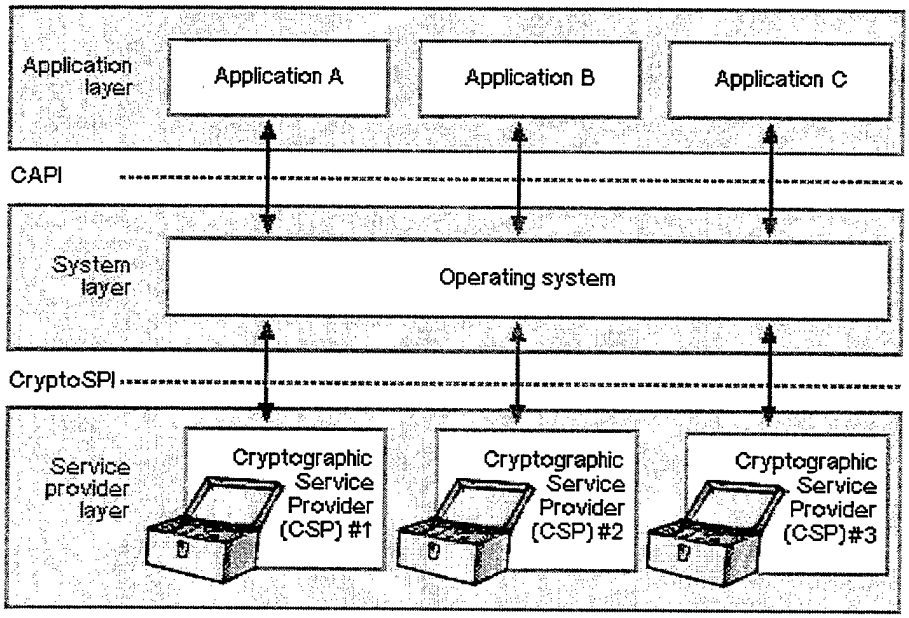

Figure 3.2 Microsoft Cryptographic System Elements [From: ALF00]

Windows CE includes two predefined CSPs. They are the RSA Base Provider, which supports digital signatures and data encryption, and the RSA Enhanced Provider, which supports 128-bit key encryption. Windows CE supports only a subset of the Crypto API 2.0 features present in Windows NT/2000. The supported CAPI 2.0 features are X.509 encoding and decoding of digital certificates, and certificate management. The Coredll module exports CAPI 1.0 functions, and the Crypto32 module exports CAPI 2.0 functions. All of the CAPI functions are defined in the Wincrypt.h header file [ALF00].

If needed, designers can add smart card functionality to their Windows CE devices by supporting CAPI through the use of Smart Card Service Providers (SCSPs). The Windows CE smart card subsystem links the smart card reader hardware and the

applications. Usually, the smart card hardware vendor provides the appropriate SCSPs. Windows CE provides the subsystem components for the resource manager, resource manager helper library, smart card reader helper library, and sample smart card reader drivers. The smart card interfaces are exported by Windows CE, and it is up to the designer to implement the system details and interface with the smart card reader hardware.

Caution should be taken when relying upon the OEM security functions, read-only protected registry keys, device identifiers, protected kernel mode, application security services, and the dial-up boot loader. These security services can be useful, but should not be relied upon to provide complete self-protection of the Windows CE operating system. Much of the security is left up to the OEMs to implement, if they so choose, instead of being enforced by the operating system itself.

For added protection, it might be advantageous to place the Module Certify functions in the operating system, and ensure that those functions are always used. The trust level model could be revised to add more than three levels, and to offer increased granularity. For example, a trust level could be defined that allows reputable sources (trusted code on a Navy server) to update the OS on a handheld device located on a ship. Another trust level could permit the user of the handheld device to update a simple application, such as a text editor, without having the ability to update the OS.

Support for the dial-up boot loader could be completely removed, if that feature was not required. The OS could be built with protected (non-kernel) mode only, to provide more memory protection. By making careful design choices at build time, the self-protection of the operating system configuration could be enhanced.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. WINDOWS CE PENETRATION TESTING

A. INTRODUCTION

Penetration testing is a method used to discover the security strengths and weaknesses of an operating system. Typically, the method is similar to the white-box approach in software testing. In white-box testing, the system design and implementation are known and can be examined to determine if they meet the requirements. In the class penetration testing exercise conducted at the Naval Postgraduate School, the approach taken was a mix of white and black-box testing, due to the fact that the Windows CE 3.0 source code was not available. Black-box testing allows the software engineer to derive sets of input conditions that will fully exercise all of the functional requirements of a program [PRE97]. Black-box testing is done at the interface level, without knowing exactly what is contained "in the box". This is similar to some forms of penetration testing, in that the exact implementation of the system is not always known, but only how it behaves. Black-box techniques are applied in order to determine the presence or absence of classes of errors, so that inferences can be made about the errors in the system as a whole. Penetration testing techniques allow the identification of classes of system security vulnerabilities. Once the vulnerabilities are identified, steps can be taken to eliminate them.

People of very different backgrounds break into computer systems for a variety of reasons: malicious hackers testing their skills, disgruntled employees seeking revenge, or those engaging in episodes of serious espionage. In each case, computer systems are inviting targets for attack. Security testing attempts to verify that the protection

mechanisms built into a system will, in fact, protect it from obvious attacks. In order to try out the protection mechanisms of Windows CE, penetration testing was performed on the Maxall configuration of Windows CE 3.0, installed on three CEPCs in a research lab environment (see Appendix A for more details). The testing was conducted during a three month time span by students of the Naval Postgraduate School enrolled in the course CS 4600, Secure Systems, in the Fall of 2000. The course was taught by Professor Cynthia Irvine.

Penetration testing on Windows CE was done to assess the system's overall vulnerability to attack. Two major goals were to test for correct functional behavior and to examine the penetration resistance of the system. The CE OS was probed for any obvious problems, such as design, implementation, or configuration errors. During the penetration testing exercise, the teams attempted to circumvent the security features of the system, and tried to exploit design weaknesses as well as undocumented interfaces.

B. THE FLAW HYPOTHESIS METHOD

Clark Weissman originated the idea of the Flaw Hypothesis Method (FHM). FHM is explained in the paper by Richard Linde, written when he was employed by the System Development Corporation [LIN75]. FHM was applied to several operating systems, including Adept 50, GCOS, CP67, VM/370, KVM/370, and MVS. It was incorporated into the TCSEC evaluation process of high assurance systems [DOD85]. FHM provides a systematic way to conduct system penetration testing and is composed of six main phases. The phases are as follows:

1. definition of purpose and goals
2. background study
3. brainstorming and flaw hypothesis generation
4. flaw hypothesis verification

5. generalization of system weaknesses
6. documentation of results

FHM is somewhat cyclic in nature, because the phases from background study to hypothesis verification can be repeated as needed or until time constraints are imposed. Results from applying FHM to a system can be used to isolate generic system functional flaws which can point to areas of the OS design needing revision. The success of FHM does depend in part on the experience of the penetration team personnel and their familiarity with the system being tested. However, even for relatively inexperienced teams, FHM provides a logical framework in which to approach the penetration testing exercise. FHM views flaws as undocumented capabilities, and penetration is defined as exploitation of those flaws. The FHM team attempts to systematically discover flaws in the target system.

1. Definition of Purpose and Goals

Phase one of FHM involves defining the purpose and testing goals. First, a baseline must be established. The testers should decide on the criteria for success, based upon the details of their particular exercise. The subject of the analysis should be determined, and the testers should bound the environment and establish the available resources. The purpose behind the penetration testing helps define the entire testing process. The testing can be done for various reasons, such as certification purposes, as part of a risk assessment, or for research. In our case, the course requirements set the goals for our penetration testing. This focused the analysis and defined specific criteria for success.

2. Background Study

Background study is the focus of phase two in the penetration testing methodology. The optimal FHM team would already be experienced with software systems, security products, penetration tools, penetration methodologies, historical penetration information, and known system vulnerabilities. However, a background study should be done in order to understand the target system and its specific configuration. There are many sources for background study materials. System documentation, user documentation, configuration documentation, and bug reports were all utilized in the background study phase. If available, design documentation, discussion with the design team, and implementation source code can also be used in the background study phase to determine what the designers intended for the system. During the background study, potential weaknesses can be found by paying attention to what the designers recommend not to do. Operator commands and messages can sometimes be exploited. All the implications of rarely used functions might not have been considered by the designers. Generic weaknesses might be identified by examining known problems with the system. Without performing the background study, the next phase of the FHM would not be as effective.

3. Brainstorming and Flaw Hypothesis Generation

Phase three consists of brainstorming and flaw hypothesis generation. This is where all the background study, team expertise, and system knowledge combine to produce the best guesses for possible system flaws. Team members who are area experts can educate the rest of the team on an aspect of the system and encourage group discussions. In our penetration testing exercise, team members became familiar with one of five areas and shared their findings with the team. The areas were the Object Store

and File System, Process and Thread Management, Communication and I/O, Memory Management, and the Graphics, Window, and Event Manager Subsystem (GWE). In this phase, Flaw Hypothesis Sheets are generated. Each sheet documents the following information: the flaw hypothesis, problem identification number, the originator who found the flaw, the investigator, references, type of vulnerability, the date of investigation, and the date of flaw confirmation. A Flaw Hypothesis Database is constructed using the data in the Flaw Hypothesis Sheets. The database is essential in completing a successful penetration test. Without a way to organize and sift though the data gathered, it is hard to generalize the flaws into broad, useful categories.

In order to hypothesize attacks, the common methods of system penetration were considered. The teams examined how spoofing, masquerades, unauthorized access, covert processes, pre-existing debugging entry points, and denial of service (DOS) through resource utilization would apply to Windows CE. The teams also searched for design errors, bad design assumptions, and compromises that were made to achieve better performance. As Flaw Hypotheses were generated, each team examined them and discarded the ones that were incorrect or outside the scope of the testing. The remaining hypotheses were prioritized in terms of ease of exploitation, probability of success, and likely payoff.

4. Flaw Hypothesis Verification

The fourth FHM phase consists of the verification of the Flaw Hypotheses that were generated in phase three. *Gedanken*, or thought experiments, were done with the information gathered in the background study. Often, the validity of a Flaw Hypothesis can be determined by carefully thinking through all of the steps of the penetration

experiment. The *gedanken* experiments are the most important phase of the penetration study, due to the fact that most uncovered flaws are found by performing this type of experimentation [LIN75]. The verification phase is often the most difficult part of the entire penetration exercise, so a time limit is usually set for completion. Usually the easily exploitable, high payoff hypotheses are considered first, and sometimes the difficult, low payoff hypotheses are discarded in order to avoid wasting time and effort. Public and custom designed software tools can be used to help in the verification task.

5. Generalization of System Weaknesses

Phase five involves the generalization of system weaknesses. In order to discover hidden trends of system deficiencies, flaws should be generalized so that they fit into broad classes. Problems in one area could possibly be mirrored in another area, so the testers should be aware of this principle and exploit it. Generalization of flaws is a team activity, where an inductive process may be used to generate classes of flaws.

I/O control is often one class of errors. Programs could have unrestricted access to the system through device drivers. In this area, there are often semi-privileged instructions and poorly designed protection mechanisms. Sometimes there are no protection mechanisms at all. Access control is another class of errors. There are often problems with configuration, and compromises between modern and legacy features. Some errors belong in the class of algorithmic flaws, such as badly generated random numbers. Timing errors happen with unsynchronized processes, when temporary objects are unexpectedly modified, and when threads interact in strange ways. Data and resource sharing allow for flaws, because shared memory can allow the bypass of security

mechanisms. Undocumented functionality is another class of flaws, because little known hardware or interface features can be exploited in unexpected ways.

6. Documentation of Results

Phase six of the FHM, the final phase, is the documentation of results. In any scientific testing, documentation is essential for establishing a permanent record of what was done, what was discovered, and what conclusions were drawn from the process. The penetration testing experiment should be repeatable, and the documentation assists in the effort to duplicate confirmed attacks. The documentation can be used as a starting point for other tests, provides justification for incomplete testing, and may be helpful in counteracting the weaknesses found. However, open publication of the findings might be damaging to whoever owns the system, so care should be taken when releasing documentation. There is some unresolved debate as to the best way to deal with known system flaws, with arguments both for and against immediate public awareness. Basically, the release of documentation depends specifically upon the conditions in which the penetration study was done. Some circumstances may require that the documentation becomes proprietary information. In our case, the study was an academic exercise without access to system source code or internal design documentation. Therefore, our penetration study documentation was for the benefit of the students involved.

C. LAB AND TEST CONFIGURATION

For the CS 4600 penetration testing exercise, the class was divided into three teams of four to five students. Each team, working independently, used FHM for conducting penetration tests on the Windows CE 3.0 operating system. Each of the members of a team were assigned one of the following roles: Lab Administrator, Web Searcher, Tools Manager, Database Manager, and Report Editor. In addition to their role

duties, each team member investigated a Windows CE area, such as Memory Management, in great detail.

1. Penetration Team Roles

The lab administrator was responsible for the test hardware and software installation, configuration, and backups. The administrator also handled general team management. The web searcher scoured the Internet for any known Windows CE penetration flaws, and any related flaws that might be extended to apply to the test system. The searcher also looked for useful tools to pass on to the tools manager, and for relevant system background information. The tools manager discovered, evaluated, and in some cases, created tools that could be used in the system penetration effort. The database manager was responsible for designing, implementing, and managing a database to store all of the generated flaw hypothesis information. The report editor was tasked with organizing and editing the final written report at the conclusion of the penetration exercise.

2. Lab Setup

In order to have a baseline test environment for all teams, some preliminary lab setup work was required. First, the teams needed some Windows CE systems to attack. Since this was a penetration testing exercise, and a goal was to gain a deeper understanding of the system operation, the flexible CEPCs were used rather than the consumer-oriented handheld devices. A CEPC is a typical desktop computer, with specific hardware, running Windows CE as its operating system. In most respects, the CEPC behaves just like its smaller cousins. However, using the Microsoft Platform Builder tool, it is possible to build a customized version of CE by choosing components and even adding custom built ones. It is much easier to download new versions of CE to

the CEPCs rather than to the Read Only Memory (ROM) chips on handheld devices. In order to maximize the penetration efforts, the ability provided by the CEPCs to download custom applications and build system configurations was desired.

In the penetration testing lab, three test stations were configured. Each attack team had their own test station. A test station consisted of a CEPC, otherwise known as the target, and a development Windows 2000 desktop PC. The CEPC and development computer were connected by a local isolated network, and a serial debug connection on COM1. For details about the CEPC, see Appendix A.

The following software was installed on the development PCs: Platform Builder 3.0, eMbedded Tools 3.0, the Handheld PC Software Development Kit (SDK), the PocketPC SDK, MS Office, and MS Visual Studio. ActiveSync 3.1, a Microsoft application for synchronizing and transferring files between a desktop PC and a CE device, was also installed on the development PC. No security protections other than the defaults were applied to the development PC. One administrator account was configured, which the team shared. A peculiarity of the Platform Builder application was that to use it, a person must be logged on as an local administrator, preferably as the administrator who installed the software. If a non-administrator account was used, Platform Builder was inaccessible. When a user logged on to an administrator account that differed from the account used for installation, Platform Builder behaved erratically. So, to avoid these problems, Platform Builder was installed with the administrator account given to each team, and everyone was advised to utilize those accounts.

Once the CEPCs and development PCs were configured, the next big challenge was to establish communications between them. Major effort was expended to

accomplish two tasks: downloading a CE image from the development PC to the target, and transferring files between the two systems using ActiveSync and *repllog* (a Windows CE remote networking service). Configuring the CEPC device networking (used for Pocket Internet Explorer, etc.) was quite difficult, and after some effort, was abandoned. Difficulties were also encountered when attempting to use the Platform Builder Remote Tools, which were supposed to allow examination of the CEPC heap and registry, among other features.

Platform Builder can be used to create a platform and build an OS image, as explained in Appendix A. The OS image is transferred from the development PC into system memory on the CEPC. The CEPC is then booted using a boot loader application on a CEPC boot floppy disk. Therefore, a CEPC does not require a hard drive, but sometimes one is added for the convenience of booting a standard CE image. In our penetration testing lab, we had two CEPCs with hard drives, and one CEPC without.

A CEPC can be connected to a development PC using an Ethernet or parallel connection, allowing an OS image to be downloaded to the CEPC. The two options for image download and target debugging are Ethernet or parallel port. In both cases, debug status messages are output from the target via the COM1 serial port and can be viewed with the HyperTerminal application on the development PC. These status messages can be useful when troubleshooting the development PC – target connection. For our test stations, the Ethernet download and debug option was chosen because it was much faster than using the parallel port.

Prior to establishing the development PC – target connection, many sources were consulted. The Platform Builder online help, and the "Microsoft Windows CE Platform

Builder 3.0 Getting Started Guide" [MIC00b] were used as references for the entire connection process. The CEPC vendor, Special Computing, also offered invaluable assistance in setting up the connection. Useful information was also found on the Platform Builder newsgroups, which are listed in the references section of this thesis.

a. CEPC Boot Disk

The first step in the connection process was to create a CEPC boot floppy disk. The following procedure is given by the "Microsoft Windows CE Platform Builder 3.0 Getting Started Guide" [MIC00b].

1. Run Websetup.exe, located in the Program Files \ Windows CE Platform Builder \ 3.0 \ CEPB \ Utilities directory. By default, this application installs Webimgnt.exe in C:\Winnt.
2. Run Cepcboot.144, a disk image file that is located in the Program Files\Windows CE Platform Builder\3.0\CEPB\Utilities directory. The Web Image NT dialog box appears.
3. Insert a floppy disk into the floppy drive on your development workstation, and then choose either Disk A or Disk B to specify the floppy disk drive used to create the book disk.
4. Click Cancel after the boot disk has been created.
5. Verify that the boot disk contains the correct files.

Table 4.1 lists the files that should be contained on the CEPC boot floppy disk.

File Name	Description
Eboot.bin	This is a binary file that is an Ethernet boot loader component
Loadcepc.exe	This is an executable file that loads the boot loader image Eboot.bin
Autoexec.bat	This is a batch application file that must be edited to match your system configuration.
Config.sys, Himem.sys, Command.com	These are required MS-DOS files
Readme.txt	This file contains booting instructions
Drvspace.bin	This file adjusts the settings in the Drvspace.ini file to mount a drive
Io.sys, Msdos.sys	These are MS-DOS system files

Sys.com	This file is an MS-DOS application
Vesatest.exe	This is a DOS executable file. It tests the VGA BIOS on the video card to ensure that it is compatible with the Windows CE 3.0 default display driver. The Readme.txt file included on the boot floppy disk provides additional information about this.

Table 4.1 Files on the CEPC Boot Disk [From: MIC00b]

If the CEPCs were purchased from Special Computing, as in our case, a ready-made CEPC boot floppy disk was provided. In addition to the files listed above, a directory named \DRV\RTL8029 exists on the Special Computing boot disk. A configuration tool for the network card (NIC) is included in that directory, as well as a program *net.exe*, which sets up a NetBEUI connection. NetBIOS Extended User Interface (NetBEUI) is suited for use in small workgroups or Local Area Networks (LANs). It is possible to install a NetBIOS gateway and the NetBEUI client protocol on all remote access servers running Windows 2000 and most Windows networking clients. NetBEUI is not routable, and the only configuration required for the protocol is a computer name. In our penetration testing lab, the NetBEUI protocols were not utilized.

After a CEPC boot disk was obtained, it was necessary to modify the autoexec.bat file on the disk to reflect the test station target configuration of NIC IRQ, base address, and static IP address (if desired). There is supposed to be a way to leave the IP address blank and use Dynamic Host Configuration Protocol (DHCP), but that option was not implemented in our penetration testing lab.

Next, the CEPC network card settings were determined. The easiest way to accomplish that was to use the utilities provided on the CEPC boot disk. Upon system boot, a menu was displayed on the target as follows in Figure 4.1:

```
MS-DOS 6.22 STARTUP MENU
========================

1.    Boot CE/PC (local nk.bin)
2.    Boot CE/PC (ether via eboot.bin with /D:2 (640x480x8))
3.    Boot CE/PC (ether via eboot.bin with /L:1024x768x8)
4.    Boot CE/PC (ether via eboot.bin with /L:800x600x16)
5.    Boot CE/PC (ether via eboot.bin with /L:640x480x24)
6.    Boot CE/PC (parallel device)
7.    Run VesaTest program and list valid display modes
8.    Setup RTL8029 Ethernet Adapter
9.    Clean Boot (no commands)
```

Figure 4.1 CEPC Boot Menu

The menu item **8** was chosen, Setup RTL8029 Ethernet Adapter, which initiated the NIC setup program. Then the network card settings could be observed, including the NIC IRQ and base address. For our CEPCs, the settings were IRQ=9, base=D800. Next, the CEPC boot floppy was taken to another computer (non-CEPC) and the autoexec.bat file was edited to reflect our settings. Any ASCII text editor is suitable for this purpose, but Notepad was used. The IO base address must be specified in hexidecimal, but the IRQ can be either a decimal or a hexidecimal value. After the REM comment block, the next three set lines were changed as follows:

```
set NET_IRQ=9
set NET_IOBASE=0      (value of 0 signifies an auto search for the base address)
                      (D800 could have been used instead since it was known)

set NET_IP=:10.10.10.25      (the static CEPC IP address belongs here)
                             (note: it must be on the same subnet as the
                              Win 2000 development computer)
```

After editing, the file was saved back to the CEPC boot floppy disk as *autoexec.bat*. In editing the file, there is a major pitfall lurking in the "set NET_IP=" line. In spite of being mentioned in the autoexec.bat comments, it was very easy to omit the colon after the equal sign and before the valid static IP address. If the colon is omitted, the CEPC and the development workstation can not communicate, no matter how many times it is tried. So, it is very important that the set NET_IP line is written in the following manner : "`set NET_IP=:10.10.10.25`". The quote characters are not included when typing the line.

The CEPC is not Plug-and-Play enabled, so special care must be taken to prevent device IRQ and IO base address conflicts. In the *readme.txt* file on the CEPC boot disk, the default IRQ and IO base settings for all standard CE device drivers are listed. This list can be checked for conflicts with the CEPC network card settings.

b. Serial Debug Connection

Now that a CEPC boot disk has been created and edited, the next step in downloading a CE image is to establish the serial debug connection between the CEPC and the development computer. CEPC debugging text messages are output via the serial port COM1, and these can be helpful in diagnosing connection problems.

One would think that any available null modem cable would be adequate for the connection between the CEPC COM1 port and the development computer COM1 port. Sadly, this is not the case. A standard for the pin connections of null modem cables does not exist. A generic "null modem" cable will seem to work for the debugging messages, but will have problems later on communicating with ActiveSync. Through a long, tedious process of lab experimentation, Platform Builder online newsgroup

research, and discussions with Special Computing, the correct cable pin-out was discovered. Two 9-pin female connectors were used. The proper cable pin connections for CEPC to desktop computer communication are as follows in Table 4.2. The pins listed in the End 1 column should be connected to the pins listed in the End 2 column.

Cable Connector Pins for End 1	**Cable Connector Pins for End 2**
1 and 6	4
2	3
3	2
4	1 and 6
5	5
7	8
8	7
9 – No connection	9 – No connection

Table 4.2 CEPC Serial Cable Pin Connections

Following the discovery of the correct pin-out, it was learned that a null modem cable matching that description could be commercially purchased. The necessary cable is a Belkin F3B207-10, Pro Series, PC Compatible, DB9 Female/Female File Transfer Cable. Once the cable is connected on COM1, output messages are automatically sent from the CEPC upon system boot. These messages can be viewed with the Windows HyperTerminal application.

c. *Ethernet Downloading*

It is possible to download a Windows CE OS image from the development PC to the target via an Ethernet connection. In order to download an image, three main steps must be performed. These steps are :

1. Set up the hardware connections. Ethernet cables, hubs, and serial cables were used.
2. Edit the *autoexec.bat* file on the CEPC boot disk to reflect CEPC configuration.
3. Configure the connection in Platform Builder on the development PC.

Steps one and two were discussed above. Step three, configuring Platform Builder, was accomplished using the menu options of the Platform Builder Integrated Desktop Environment (IDE). After opening Platform Builder and building a platform, **Target / Configure Remote Services** can be selected from the menu. Under the **Services** tab, Ethernet should be selected in both the Download and Debugger drop-boxes. Figure 4.2 contains a screenshot of the required selections.

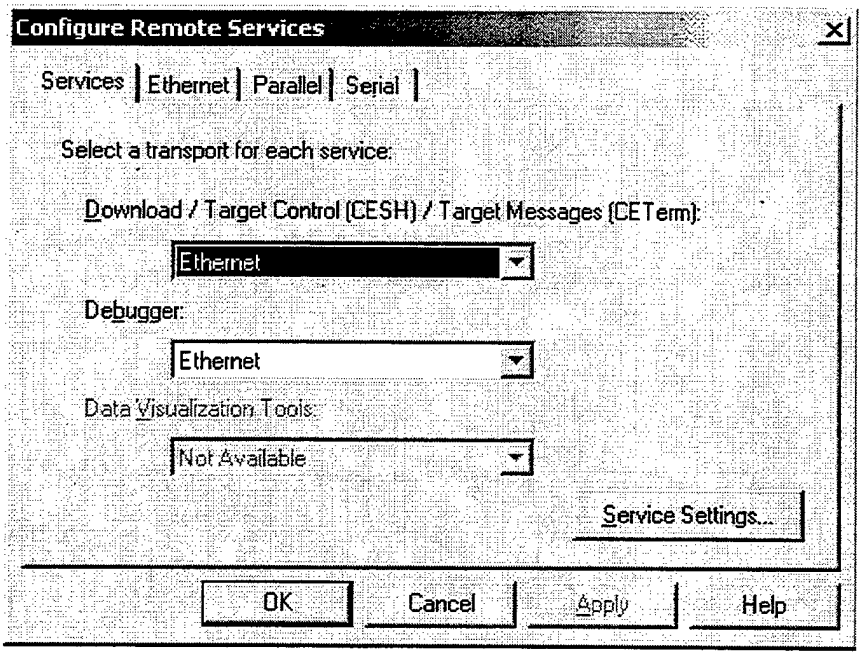

Figure 4.2 The Configure Remote Services Dialog Box [MIC00b]

The Service Settings button allows the configuration of the target messages, target control service, and the kernel debugger. By default, target messages and target control are started upon download. The kernel debugger configuration only matters when a debug version of the OS image has been built. By default, kernel debugger services are not started.

The purpose of the next step was to assist the Plaform Builder software in recognizing the CEPC target so that a device number could be automatically assigned. The **Ethernet** tab should be selected, as in Figure 4.3 below.

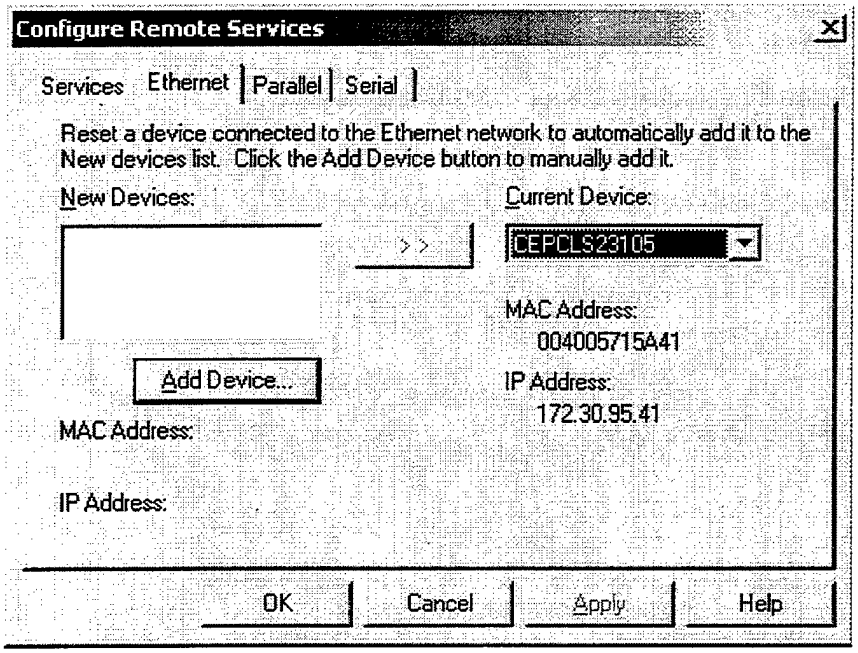

Figure 4.3 The Configure Remote Services Dialog Box, Ethernet tab [From: MIC00b]

After reaching this stage on the development computer, the CEPC boot disk was inserted into the CEPC, and the CEPC was reset by cycling its power. When the boot menu was displayed on the CEPC, as in Figure 4.1, menu choice **3**, "Boot

CE/PC (ether via eboot.bin with /L:1024x768x8)", was selected. This was equivalent to selecting menu choice **9**, "Clean Boot", and typing the command

```
loadcepc /b:38400 /c:1 /e:1:0:9:10.10.10.25 /v eboot.bin
```

at the command prompt. The `loadcepc` command is documented in the Platform Builder online help, and it allows the loading and booting of an OS image onto a CEPC. Figure 4.4 explains some of the common options that may be set with `loadcepc` command switches.

```
loadcepc /b:38400 /c:1 /e:1:0:9:10.10.10.25 /v eboot.bin
     /b = Connection Baud Rate
     /c = Serial Communications Port Used
     /e = NIC card type, IO base, IRQ, and IP address
     /v = Display Additional Status Information
     eboot.bin = file to download from the development PC

Note: NIC card type may be either :    0 for a SMC9000 card
                                       1 for a NE2000 card
```

Figure 4.4 Loadcepc Command Options

When using the `loadcepc` command, the valid baud rates are 9600, 19200, 38400, 57600, and 115200. The default baud rate is 19200. The communications port may be either 1 for COM1: or 2 for COM2:, with the default value of 1. The NIC card type must always be set to 1, for a NE2000 card because the other option is not supported at this time. The IP address is optional if Dynamic Host Configuration Protocol (DHCP) is used. There is a /L switch for setting the display characteristics, but

it was not used directly. Instead, the /L switch was used only indirectly as a choice from the batch file menu.

Following a menu choice or typed command, the CEPC screen will either clear or display 'Jumping to 0x00132B58'. This behavior of the CEPC can be ignored for now. On the development computer, a device number should be displayed in the New Devices text box of the Configure Remote Services Dialog Box, with the Ethernet tab selected. In order to add the device as the current device, the arrow button must be depressed. Now the device number should appear in the Current Device drop-down box. Next, the OK button was depressed to confirm the selection of the current device.

After the CEPC has been recognized by Platform Builder, a Windows CE OS image may be transferred from the development PC to the CEPC. On the development PC, this was initiated by selecting Target / Download Image from the Platform Builder menu, with the desired platform open. HyperTerminal was also started on the development PC in order to view the CEPC debug messages. The next step was to reboot the CEPC, with the boot floppy disk in the a: drive. The appropriate choice was made from the boot menu, which was number 3, "Boot CE/PC (ether via eboot.bin with /L:1024x768x8)". Finally, the CEPC and development PC were communicating, and a progress bar appeared in Platform Builder on the development PC. After approximately ten seconds, the OS image had downloaded to the CEPC, and the CEPC booted up. The CEPC Windows CE 3.0 startup screen displayed, which was a very welcome sight.

d. File Transfer Using ActiveSync

Besides downloading an OS image, another useful action is to connect the CEPC to the development computer via ActiveSync and `repllog`. This allows for the transfer of files between the two systems. In the penetration testing exercise, this file transfer ability was made available to place custom attack programs on the CEPCs. ActiveSync is the desktop PC file synchronization utility, and `repllog` is the remote networking component of Windows CE.

First, a physical serial connection had to be made from the CEPC COM1: port to the development computer's COM1 port using the Belkin cable discussed above. COM 2 ports would work as well, although the Platform Builder debug messages are only output via COM1. Next, the ActiveSync 3.0 software was installed on each development PC. This software can be downloaded from the Microsoft web site, and it is also included on Platform Builder installation disk 11.

ActiveSync presents a wizard-like interface upon its first use, which does not allow access to the connection settings menu until a mobile device is recognized as a valid connection partner. But, the ActiveSync application does not recognize a CEPC as a mobile device. This created a problem in configuring ActiveSync. The problem was circumvented by attaching a more conventional Windows CE device, in this case a Symbol PPT 2700, to its cradle with a serial connection to COM2 on the development PC. The Symbol was recognized by ActiveSync as a mobile device, and a partnership with that device was formed. Next, Connection Settings was chosen from the ActiveSync menu bar. This action opened the Connection Settings dialog box, and the com port setting was changed from COM2 to COM1. Leaving this dialog box open on

the development PC, it was time to initiate `repllog` on the CEPC. Start Menu, and Run were chosen on the CEPC. Next, `repllog` was typed at the CEPC command prompt. Back on the development PC, a screen displayed which asked the user if he or she would like to create a partnership. The choice "NO" was selected in order to connect the CEPC as a guest. Then, it was possible to transfer files between the development PC and the CEPC. After the initial connection was made, one could connect later by simply launching ActiveSync on the development PC and then typing `repllog` at the CEPC command prompt.

The partnership and file synchronization features were not built into the Platform Builder communication components for incorporation into CEPC OS image builds. This means that the CEPC can only connect as a guest, and the development PC will break the connection if a partnership is attempted. There is a Microsoft QFE (Quick Fix Engineering) patch available for download to fix that problem, but it was not used in the penetration testing lab.

To summarize, the lab setup consisted of configuring three CEPCs, three development PCs, and establishing OS image download and file transfer capabilities for each test station. After the initial test station configuration, the responsibility for maintaining and tweaking the setup shifted to the administrator of each team.

D. PROPOSED FLAW HYPOTHESES

After several weeks of background study, the penetration testing teams proposed several flaw hypotheses in each of the core Windows CE areas. Effort was made to classify the flaws into general classes of errors, in an attempt to discover trends in the

system weaknesses. A selection of the flaw hypotheses generated are discussed in the following sections.

1. File System / Object Store

This Windows CE subsystem contains persistent storage such as the CE file system, registry, and property databases.

a) Windows CE file shadowing allows for the creation of a file in RAM that has the same name as an existing system ROM file. For all practical purposes, the newly created RAM file supercedes the existing ROM file. The user can only see the RAM file, and that file is executed instead of the ROM file. This feature was provided for the ease of upgrading system files and drivers. The hypothesis was that an adversary could utilize the file shadowing capability to replace system files with malicious applications. Also, the registry, which contains system settings, could possibly be spoofed in this manner. [AGA00], [BRI00], [CLE00]

b) The property databases store application data, such as names, addresses, and phone numbers. A hypothesis was proposed that an attacker could access or modify a user's personal information through the property databases. The property databases could also be covertly copied over a network during device synchronozation. Currently, there are no access restrictions on these databases. [AGA00]

c) It was suggested that perhaps a malicious user-defined file system could be installed and used to compromise the OS. Windows CE currently provides support for installing additional file systems, but there are no administrative constraints on who is allowed to install these file systems. This implies that

anyone can install file systems on a CE device, potentially gaining full access to the target system and its data. [AGA00]

2. Process and Thread Management

This subsystem contains the process and thread model, and the round-robin, priority-based scheduler.

a) A hypothesis was proposed that since CE has a shared memory space for all processes, it is possible for an application to read the memory area of another process with the call **ReadProcessMemory**. This penetration could potentially lead to the modification or theft of data. [AGA00]

b) Another problem could happen if a file handle reference is moved so that it points to another memory location after the file has been opened. This could allow the file handle to point to some malicious code, or cause a denial of service if a critical file could not be located. [AGA00]

c) One hypothesis is that any process has the ability to terminate any other process. If this is the case, any user application would be able to terminate critical system processes. A malicious process could also terminate an application and start another one spoofing the original. [AGA00]

d) A CE process is not limited in the number of threads it can spawn. Therefore, a hypothesis was that a process could launch a very large number of threads and use up all of the system memory, causing a denial of service. [AGA00]

3. Communication and I/O

This subsystem contains the native and stream drivers, the driver interface to the kernel, networking support, ActiveSync support, infrared support, USB support, and streams to files or consoles.

a) It might be possible for an application to call the **CreateFile** function with a file name that is not null-terminated. This flaw would fall in the category of buffer overflows. If the **CreateFile** function limits the number of characters that it copies into its memory space by reading until it reaches the null-terminator, it could be possible to insert malicious code into the file name string and cause **CreateFile** to overwrite its own return stack. [AGA00]

b) If the function **wvsprintf** is called with mismatched arguments, it might be possible to discover some information in the returned format specifier data structure. This information could be data that would normally not be provided to the calling process. [AGA00]

c) Another hypothesis proposed the exploitation of the CE NetBIOS **net** commands, which are similar to the Windows 95 NetBIOS commands and might share their vulnerabilities. [CLE00]

d) A denial of service attack might be possible by overwhelming the CE device with TCP/IP packets. A program similar to "Win Nuke" could be used. This would be especially devastating for wireless CE devices. [CLE00]

4. Memory Management

This subsystem deals with the allocation and freeing of memory, paging, maintaining the heap, and managing virtual memory.

a) One flaw hypothesis in this area is that the cache manager might be overloaded by preemptively loading unnecessary pages into memory. If successful, system performance could be downgraded, causing many page faults and possibly undermining the stability of the system. [AGA00]

b) Another hypothesis is that the system could be kept busy flushing the cache for long enough that the network connection is lost. If malicious code causes the cache to flush for more than two thousand milliseconds, the network connection could be broken because the system is not responding. This situation could allow a spoof attack, because the system network response could be simulated, when in fact the system was down. [AGA00]

c) It might be possible to mark a file as sequential, allowing intelligent read-ahead to bring in the next 192 KB in memory. The memory read immediately following the file could contain cryptography secrets or other important data. This captured data could be stored and later exported to a desktop PC during ActiveSync of the CE device. [AGA00]

d) By modifying the read input buffer during a read operation, it may be possible to create an unstable system state. The Windows CE Programming documentation advises against doing this, because the result is undefined. This is a timing attack that would have to be repeated five or six times before its effectiveness could be ascertained. [AGA00]

5. Graphics, Window, and Event Manager Subsystem

This subsystem contains the user input system, the event manager, the window manager, raster graphics API, touch screen driver, and font support.

a) The GWE module is responsible for managing the power-saving suspend mode. This could possibly be exploited to turn off the power saver, thus running down the batteries and causing the loss of files stored in RAM system memory. The user might believe that the CE device would turn off normally, but because the timeout interval had been tampered with, the batteries could run out and user data would be lost. [CLE00]

b) Another flaw hypothesis exploits the fact that the CE OS only supports an 8-bit color scheme. Therefore, a denial of service attack would run an application that would set the resolution higher than 8-bit, causing the device display to function incorrectly. [CLE00]

c) Windows CE only supports a 256 color palette. The OS does not include a palette manager, and there are no checks to ensure that palette settings are valid. Malicious code could manipulate the palette to support only one color. This would also cause the display to be unreadable. [CLE00]

d) There is a problem with stale process handles in the GWE subsystem. When applications are terminated, the handles to those processes remain active instead of being released. A 2-bit reuse count indicates how many times a slot in the handle table has been reused. The OS designers claim that the reuse scheme works seventy-five percent of the time. However, the possibility of stale handles exists, and could be exploited to gain improper access to files or memory areas. [BRI00]

6. Miscellaneous Flaw Hypotheses

The CE device password may be easy to crack due to the fact that text input is limited on a CE device. Most users will choose a short password that is easy to enter using a stylus, and will not use symbols, such as @ or $, in a password. These habits might cause the password to be a weak entry point to the system. [CLE00], [BRI00]

Many CE devices include a built-in microphone for dictation and recording sound clips. This hardware could be exploited by a malicious application to record ambient sounds or private conversations while the user was unaware of the fact. [BRI00]

E. PENETRATION TEST CONCLUSIONS

After intensive study, the penetration test teams were successful in identifying potential flaws and vulnerabilities in the Windows CE operating system. Using the Flaw Hypothesis Method, the teams identified the main components of the operating system, discovered potential weaknesses in those components, and tested the weaknesses through gedanken experiments. Finally, the team findings were documented to preserve the system knowledge gained.

Caution is recommended when using Windows CE due to the lack of basic security features, such as authentication or a trusted path. Numerous flaws were found that could be exploited to compromise system security. Also, by adding a mobile CE device to an existing computer network, a weakest link might be introduced to the system as a whole.

Windows CE does offer some process separation features, and allows the use of certificates to install trusted modules at build time. However, these features are

inadequate by themselves and should be incorporated into a coherent security architecture.

V. REDESIGN FOR SECURITY

A. EXISTING DESIGN

Because the Windows CE devices are so versatile and mobile, many governmental uses for these devices can be envisioned. In the rush to incorporate new technology, we should keep in mind how it will affect existing computer systems and networks. Especially, the security impact of the new technology should be examined. Using wireless PDAs may be convenient and time-saving, but the devices could introduce a weakest link in the overall systems security because of the current design of the operating system. By taking a closer look at the components and design of Windows CE, we can make suggestions on how to improve its self-protection.

1. Windows CE Design Goals

The Windows CE operating system was first introduced in 1996. It was specifically designed for supporting embedded applications, which are typically limited in resources. There are many applications for devices based upon the Windows CE OS. A few examples are commercial building automation systems, handheld communications devices, manufacturing process controllers, and medical data acquisition devices.

A major design goal of Windows CE was to support the requirement of minimal memory usage. The size, or footprint, of the operating system itself can be controlled by adding or removing modules to obtain the specific desired configuration. Several of the modules can be further divided into components, for increased flexibility. Hardware resources such as ROM and RAM can be minimized for a particular system design by selecting a minimum set of modules and components. Component modules are defined

by Microsoft as Windows CE modules that include one or more optional components. Examples of component modules include Coredll, Gwes, Filesys, and Ole32. These component modules can be customized to meet the needs of the embedded application designer.

Three key design decisions were made in Windows CE. The first decision was to partition the operating system design into modules, in order to support hardware upgrades and enhance system flexibility. Second, Microsoft decided to build a portable OS that would not depend on just one OEM for the device platform. This allows OEMs the choice of several processors to choose from to fit their project needs. Third, they considered what the system would be used for, and what capabilities might be desired. For example, since it was determined that only a limited number of applications would be running at any given time, the OS was limited to supporting thirty-two processes. Also, in order to relax the level of code redundancy, the OS kernel was allowed the ability to possibly corrupt applications. The designers knew the kernel would not be very secure, but it was viewed as a reasonable compromise. However, some protections were built in so that the applications would have a harder time inadvertently crashing the kernel. For compatibility, the designers decided to support a subset of the Win32 API. This provided Win32 applications programmers a jump start to writing Windows CE applications.

2. Building Windows CE

A Windows CE system can be built using Microsoft Platform Builder (PB). Version 3.0 is the current release at the time of this writing, and it supports the data visualization tool as well as kernel debugging and profiling. Data visualization allows

the developer to visually monitor and track the state of system data while a platform operating system executes. Since Windows CE can be customized to support a small footprint with a subset of desired features built in, the general approach is a modular one. A system is built using a platform, a project, and various modules, which are composed of components. Microsoft defines a module as an executable or dynamic-link library that usually implements self-contained functionality. A module contains components that can be replaced or removed as needed. In order to meet the constraints set for a system's memory requirements, careful thought should be given as to which modules and components to include. PB provides seven sample project configurations that have been tested and are guaranteed to work. These configurations range from Minkern, with a minimal kernel and a very simple application, to Maxall, which is the complete version of CE, including a soft input panel, handwriting recognition, WinNET FTP and communication applications. It is strongly recommended that new systems be based upon one of the provided configurations. There are several environment variables that can be set to modify the pre-tested configurations. The environment variables are contained in the file Cesysgen.bat. It is interesting to note that the password component can be removed from the Minkern configuration, and the security component can be removed from the Mincomm configuration. However, for a dedicated embedded device, the password or security components might not be required

A system generation phase (Sysgen) was added to support componentization. Sysgen integrates a built version of the OS, installed with Platform Builder, that has all the libraries for all the components, the master header files, the module definition files, and a configuration file that specifies the desired components. Three tasks are performed

during the Sysgen phase. The selected and necessary parts of the OS are linked together, the master header files are filtered to take out the parts that aren't exposed by the selected components, and the module definition files (.DEF) are filtered so that only the desired functions are exported.

Componentization was good in that it allowed four major benefits. They are: scalability, the ability for OEMs to make system design tradeoffs, the ability for OEMs to replace code with their own customized version, and minimizing the overhead as new features are added to the system.

Two of the big problems encountered in componentization were dependencies and configuration testing. All of the logical units that already had interfaces were made into separate components, such as the Graphics, Window and Event Manager (GWE) module. Unfortunately, approximately ten percent of the system was not very modular at first and so it was hard to break it into components. For example, someone might want to build a system that had communications without a window manager, but because the system used the PostMessage function and PostMessage used something else in the window manager, the window manager would have to be included anyway. The modules were rewritten for Windows CE 2.0 with the goal of isolating dependencies across the modules.

Componentization also resulted in difficulties in testing all of the possible configurations. For just ten components that could be either included or excluded, the Quality Assurance department would have to test a great number of possible configurations. As the number of components increases, the amount of testing quickly

becomes unmanageable. The Microsoft designers decided to compromise and allow a limited set of component configurations to be chosen.

B. WINDOWS CE KERNEL MODULES

The Windows CE kernel consists of modules and components, which are listed in Table 5.1.

1. Module Listing

Module	Description	Components
Coredll	Operating core dynamic link library	accel_c, coreimm, coreloc, coremain, coresioa, coresiow, coresip, corestra, corestrw, cryptapi, fileinfo, fileopen, fmtmsg, fmtres, lmem, mgdi_c, rectapi, rsa32, serdev, shcore, shexec, shmisc, shortcut, tapilib, thunks, wavelib, wmgr_c
Nk	Windows CE kernel	

Table 5.1 Kernel Modules Table [From: GAR99]

In Coredll, the only required components are coremain, lmem, and thunks. Coremain contains the Windows CE base functionality. Lmem is the local heap. Thunks is the name of the component that handles the older 16-bit code, making the OS backwards compatible. The thunks component takes care of the kernel to Win32 thunking mechanism. The generic thunking mechanism provides functions that allow a 16-bit application to load a Win32-based DLL, get the addresses of its exported functions, call the functions, convert addresses, and free the Win32-based DLL.

C. WINDOWS CE OBJECT STORE MODULES

1. Design Background

The CE object store and file system are quite different from the Windows NT file system. An obvious difference in CE is the lack of any of the types of security protection offered by the NT model.

The CE object store provides persistent storage that is available for use by applications. It is the part of memory not in use by the operating system, and is built on the internal heap using the system RAM, ROM, and optionally mounted PC cards. The maximum size of the object store is 256 MB for CE version 3.0, and 16 MB for previous versions. The object store has three main components: the file system, the registry, and property databases.

There were four major design goals for the heap. The Microsoft designers wanted it to be small to conserve memory. They also wanted the heap to be robust against power loss and crashing. They needed it to be fast, and have a small working set.

A new proprietary file system was implemented in order to minimize overhead and get the most out of the available storage memory. The file system is a lightweight layer on top of the heap that supports files stored in ROM as well as files stored in RAM.

One interesting thing about the file system is that it allows file shadowing. If a file is created in RAM that has the same name as an existing ROM file, the RAM version shadows the ROM file. This means that only the RAM file attributes are seen, and the RAM file is executed instead of the ROM file of the same name. The ROM file is still there, but you can't access it. If the RAM file is deleted, the ROM file attributes will be seen again. Shadowing means that you can hide the ROM file size, but not its name.

The CE registry is similar to the one found in Windows NT, but it is a limited subset. However, it is Win32 compatible. The registry stores data about applications, drivers, user preferences, and other configuration settings. The data stored in the registry is persistent and shared among all applications. In the Windows CE registry, there is no way to deny read access to applications. Therefore, care should be taken when storing data in the registry because there is no real way to protect the data. The registry uses the native logging and transactioning of the heap. Since CE does not support the registry security features, a default security descriptor is assigned to the CE registry keys.

It is very likely that the CE registry is vulnerable to the same types of attacks used on the NT registry. The CE registry shares common functionality with the NT registry, but has none of the NT registry security features, so that will offer possibilities for exploits. The CE registry may be stored on a PC card and copied into a device's system RAM at boot-up. This could offer a possible way to overwrite the valid registry with whatever keys are desired.

The final component of the object store is the property database. Property databases store application specific data, and can be built on the internal heap or on mounted volumes. The property databases are loosely based on the Microsoft Messaging API (MAPI). Examples of applications that use the property databases are: Inbox, Calendar, and Tasks.

In a CE property database, only one level of hierarchy is allowed. This means that a record cannot contain another record. A single record cannot be shared among databases, and it is not possible to lock a database to restrict access. Transactioning occurs after each individual database call.

There are at least three ways to access a property database. The Pocket Access application, CE API functions, and Microsoft Foundation Classes (MFC) can all be used. Active Data Objects (ADO), which are used to access relational databases, may also be used.

D. WINDOWS CE I/O AND COMMUNICATIONS MODULES

Windows CE provides many options for communication with the desktop, the Internet, and other Windows CE devices. The OS supports a variety of Win32 communication APIs for modems, networks, and serial and infrared communications. Smartcards, external file systems, and flash memory can also be incorporated. Native drivers provide support for graphics, the touch screen, and audio, among other things. Stream drivers exist for serial communications. Network and Universal Serial Bus (USB) drivers are also supported.

1. Windows CE Driver Model

The OEM Adaptation Layer (OAL) is the layer between the device hardware and the Windows CE kernel. It is comparable to the Hardware Abstraction Layer (HAL) in Windows NT. When considering the CE driver model, it is somewhat instructive to think of the components being arranged in layers, with hardware at the bottom. Next comes the OAL, and various drivers that interface directly with the hardware. Above the OAL are the CE kernel, the window manager and user subsystem GWE, and the device module. However, the model is not composed of true layers, because the drivers can communicate directly with the hardware, bypassing the OAL, and can also communicate with the kernel and the GWE module. In fact, the native drivers, which include display, battery, keyboard, audio, and touch screen, are a part of GWE. Some driver services are

exported in the Device module to make writing drivers for serial and PC card devices easier.

There are four main categories of Windows CE device drivers: native drivers, stream interface drivers, NDIS network drivers, and USB drivers. The native device drivers are linked with the OS. The stream drivers can be installed at any time. Because exploring all of the drivers in detail is beyond the scope of this thesis, the concentration will be placed on the stream drivers. A summary will be given for the other driver types. Sample drivers of all types are provided in the Windows CE Device Driver Kit.

To understand where the drivers fit into the big picture, it helps to take a look at the system startup sequence. When a Windows CE device is turned on, the hardware is first initialized. Next, the OEM startup code runs. After that, the kernel is initialized, then the OEM init function runs, which is one of the OAL functions inside of the Nk module in Nk.exe. The kernel start up, then the Filesys, GWE, Device, and other modules are loaded. Finally, GWE loads up the native drivers and Device loads the other drivers. User level stream drivers can also be loaded at any time thereafter.

2. Native Drivers

The native device drivers are split into two parts: the module device driver (MDD) and the platform device driver (PDD). These driver models are unique to Windows CE, and are provided for convenience. Usage of these models is not required, but is highly encouraged. The MDD provides the interface to Win32, which allows the OEM to make changes only to the PDD. Through the PDD, the OEM can control the hardware, without having to worry about the software interface from the driver to Win32. So, the OEM has the choice of writing a monolithic driver, or just implementing a

smaller piece that interacts with the MDD. The print and display drivers are monolithic. Each monolithic native driver must export a specific set of functions, called the Device Driver Interface (DDI). The DDI is called by the GWE module at run-time. The layered drivers use another interface called the Device Driver Service-provider Interface (DDSI). That interface consists of the PDD functions that are called by the MDD. The touch panel, keyboard, PC card socket, and USB drivers are layered drivers.

3. Stream Drivers

Stream drivers are a generic type of driver, because they always expose the same functions. This is in contrast to native device drivers, which have a unique interface. Stream drivers are typically used for third-party devices such as bar-code scanners, GPS receivers, and IR receivers. Using the stream driver interface, devices are presented as a type of special file, and use the file system API to interact with applications. For example, the CreateFile function would open a device. Stream drivers are loaded by either the Device module or by specific applications, instead of by the GWE module like native drivers. They are usually implemented as Dynamic-Link Libraries (DLLs) and are located in the \Windows directory.

There is a strict naming convention for stream drivers. The format is three upper-case letters, followed by a single-digit index (1-9, and 0 used for 10), with a colon at the end. For example, "COM1:" is a valid name for the first serial port. The three letters act as a key to access the driver functions, and the digits identify a specific drivers.

There is a user-level process called the Device Manager Module that acts as an interface between the kernel, registry, and stream interface drivers. Its main purpose is to load and unload stream devices as needed. There are three ways that stream drivers can

be loaded. One way is at boot time. Upon boot, Device Manager loads all the drivers listed under the HKEY_LOCAL_MACHINE\Drivers\Builtin registry key. Another way of loading is when a device is connected. For example, when a PC card is connected, the Device Manager calls the native socket driver to get a Plug and Play identifier. That id is then checked against the registry values in the key HKEY_LOCAL_MACHINE\Drivers\PCMCIA. If found, the driver is loaded. If not found, the Device Manager calls the detection functions listed in HKEY_LOCAL_MACHINE\Drivers\PCMCIA\Detect. If one of the functions can handle the device, the Device Manager registers that driver for the device. The last way to load a stream driver is used when an application tries to open an unloaded driver. The application can load the device itself and then open and access it. This last scenario typically happens with devices like digital cameras.

There are two ways to unload stream drivers. If the Device Manager is notified of the disconnection, the corresponding entry is removed from HKEY_LOCAL_MACHINE\Drivers\Active. The Device Manager also calls the DeregisterDevice function to remove the device name from the file system and FreeLibrary to unload the DLL from memory. Alternatively, if an application loaded the stream device, then the application has to unload the DLL from memory on its own. If it fails to do this, some of the valuable memory storage space will be wasted holding a DLL that is not being used.

All stream interface drivers should implement some standard functions. The functions and their descriptions are listed in Table 5.2.

Driver Function	Description
???_Close ( )	Closes the device associated with a handle
???_Deinit ( )	Device Manager calls this to de-initialize the driver
???_Init ( )	Device Manager calls this to initialize the driver
???_IoControl ( )	Sends a device-defined command to the driver
???_Open ( )	Opens a device for reading or writing
???_PowerDown ( )	Powers down the device, if capable
???_PowerUp ( )	Powers up the device
???_Read ( )	Reads data from the device
???_Seek ( )	Moves the data pointer within the device
???_Write ( )	Writes data to the device

Table 5.2 Stream Driver Functions [From: GAR99]

Stream drivers can be used either by a single application at a time, or by multiple applications, depending on whether a handle to the device or NULL is returned by the ???_Open function. This allows the designer to decide which policy best fits the anticipated device usage.

4. NDIS Network Drivers

The Network Driver Interface Specifications (NDIS) drivers used in Windows CE are derived from Windows NT. They allow networking protocols like TCP/IP and IrDA to be independent from a network interface card (NIC). Windows CE supports Ethernet

and IrDA miniport drivers that conform to the NDIS 4.0 implemented in Windows NT. NDIS provides the glue between the network interface cards, the network drivers, and the network protocol stacks. However, monolithic network drivers and full NIC drivers are not supported under Windows CE NDIS. Some other features that are not supported are: general direct memory access, contiguous physical memory allocations, and wide area networking through NDIS. For miniport drivers, Windows CE is mostly source-code compatible with Window NT, so they share almost identical NDIS APIs. This fact speeds up Windows CE driver development time for people who are already familiar with Windows NT miniport drivers.

5. USB Drivers

Universal Serial Bus (USB) is an external bus architecture for connecting USB-compatible peripheral devices, like a mouse or keyboard, to a host computer. USB was not designed for use as an internal bus, but rather as a communication protocol that supports serial data transfers between a host system and its USB-compatible peripherals. Today, you can find USB connectors on common items like joysticks, digital cameras, MP3 players, and PDAs. USB provides a single, well-defined connector type, supports hot plugging and Plug and Play, and provides power-saving and suspend modes. There is currently no support for connecting a Windows CE device as a USB peripheral to a host computer.

6. Module Listing

Table 5.3 describes the modules that are related to common native and stream I/O drivers.

Module	Description	Components
Device	Installable device manager	
Dualio	PCMCIA client driver module for the Socket Communications Dual Serial I/O	
Elnk3	Elink Ethernet driver	
Gwe	Graphics, Events, and Windowing subsystem	** Note ** Only the driver-related components are listed here. Gcache, gwectrl, gweshare, gwesmain, serdev
Inetcore	Windows Internet DLL core	
Ircomm	IrDA communication	
Irdastk	IrDA stack	
Ndis	NDIS network module	
Pcl	PCL printer driver	
Prnerr	Printer port error information and dialog	
Prnport	Printer transport layer	
Remnet	Remote networking	
Rnaapp	Remote networking application support	

Serial	Serial Driver	
Softkb	Soft Input Panel (SIP) device driver	
Sramdisk	SRAM (PCMCIA) card	
Tapi	Telephony API	
Tcpstk	TCP/IP stack	
Trueffs	TrueFFS block device driver	
Unimodem	TAPI service provider for AT command modems	
Usbd	USB module	
Usbmouse	USB mouse driver	
Wininet	Internet API support	
Winsock	Winsock services	
Xircce2	Xircom Ethernet driver	

Table 5.3 Driver Modules Table [From: GAR99]

E. WINDOWS CE I/O SERVICES

Windows CE provides rich support for several communication services. A few of the offered services are ActiveSync, Remnet, and Repllog. Serial and IR communications are also helpful services that can be controlled directly through the use of built-in drivers.

1. ActiveSync Service

ActiveSync allows the user to synchronize the programs and data contained on a Windows CE device with a desktop computer. The service is configurable, but the default is to overwrite the identical desktop computer files with those contained on the CE device. ActiveSync also installs new applications from the desktop computer onto the CE device. In its previous incarnation with Windows CE 2.x, ActiveSync was called "Windows CE Services" and provided much of the same functionality. ActiveSync can store a backup of the CE device, to be restored later in the event of a catastrophe. It also provides the ability to view and transfer files from the desktop PC to the CE device. From a vulnerability standpoint, ActiveSync could potentially provide a channel for captured data stored in a file to be exported to a desktop PC whenever the user synchs the CE device.

2. Remnet Service

Windows CE provides a connection application that can be launched from the Control Panel. The application is called Remnet, and it establishes a connection from the CE device to a host computer. This application cannot be used at the same time as Repllog (discussed below). Remnet allows applications on the CE device to use a direct serial cable connection or dial-up networking to the host computer.

3. Repllog Service

Repllog is another communication service that is used in conjunction with ActiveSync. It cannot be used concurrently with Remnet. Repllog provides connectivity from the CE device to the host computer, and also monitors the connection and provides data synchronization services. ActiveSync executing on a host computer can detect Windows CE devices running Repllog, but it can not detect CE devices running Remnet.

Repllog is a service offered directly by the Windows CE operating system, and must be included at build time.

4. Serial and IR Communications

Several serial drivers are provided with Platform Builder. They are: the native serial driver, Serial.dll; the 16550 Serial UART driver, Ser16550.lib; the PC card serial driver, Ser_card.lib; and the dual serial driver, Dualio.dll. Caution should be exercised when using serial and IR ports, because the information transmitted could be intercepted and analyzed by a malicious application.

F. RECOMMENDATIONS

Windows CE is a customizable, general purpose, embedded operating system that can be used as a base for supporting many everyday tasks. A PDA operating with Windows CE can be used for networking, barcode scanning, maintaining databases, or the playback and recording of sound files. However, the impact of security on the Windows CE operating system was not a concern during its original design. Speed and system performance were the primary concerns, and security of the operating system itself was not a high priority objective. Future work could include analyzing the Windows CE 3.0 source code directly for dependencies and other obstacles to system self-protection.

The utilization of Windows CE devices in government or military environments requires that the devices offer some form of assurance. The devices should be considered as a single component among many that are addressed by a well-designed security policy. Due to the system's flexibility and familiar user interface, Windows CE devices could

become an important asset in such tasks as inventory control, if the proper security precautions are taken.

It is recommended that a streamlined version of the Windows CE OS be designed to meet the customer's specific requirements. The system should be optimized for the tasks it will perform. To reduce system vulnerability, unnecessary components of the OS should be removed. For example, support for external file systems could be removed, thereby controlling the device's ability to mount external volumes.

If there are no plans to update the OS remotely, the Dial-Up Boot Loader (or DUB) could be removed from the OS altogether. If it is used, the DUB should be configured to respond in a safe, non-compromising manner if device authentication fails. The default behavior upon failure is to halt the download process and display a user message. It might be a good idea to record the DUB actions in an audit log on the server as well.

Support for infrared (IR) communication provides a convenient way to transfer programs or data between Windows CE devices. However, there are cases in which data transfer should be restricted, and removing the IR support from the OS would aid in that restriction. Without the IR support, it would be much harder for rogue applications (either malicious or non-approved) to spread from one device to another. This would simplify controlling the Windows CE device configuration.

The Windows CE OS should be built using protected (non-kernel) mode only, as this affords the most protection of the system memory. Using protected mode restricts the physical memory access of user level threads. This decreases the possibility that a user application could undermine the stability of the kernel.

The OEM Certify Module functions should be implemented in order to prevent unknown modules from being loaded, restrict access to the system APIs, and prevent write access to certain parts of the system registry. As currently implemented, the functions provide a superficial attempt at compliance, rather than true protection. This means that all driver modules should not be routinely assigned a trust level of OEM_CERTIFY_TRUST. Drivers and other new code modules would have to be reviewed by an outside source to determine their trust level. After approval, the drivers could be distributed from a controlled source. Any cryptography used in the Certify Module functions should posses an adequate key length, and the algorithms themselves should be free of known flaws. Due to their visibility and the likelihood of attacks, the Microsoft Cryptographic Service Providers might not be the best choice for implementing cryptographic algorithms. In certain environments, however, it is possible that the CSPs might be adequate for meeting design requirements.

Each PDA should be assigned a unique, immutable device ID number. A data structure, DEVICE_ID, is provided for storing the unique device ID. Bounds checking should be done upon the return of a DEVICE_ID, in order to avoid buffer overflow problems. The device ID could be used for some primitive identification and auditing upon synchronization of the device to a trusted server. The decision to download new applications or OS updates to the device from a trusted server could be based in part on the device ID. However, other security mechanisms should be in place, so that the device ID is not the only criteria relied upon.

When a CE device is synchronized with a desktop computer, it uses ActiveSync to communicate via a serial port, USB port, IR port, network connection, or modem.

There is currently no identification or authentication of the CE device or the computer it connects to. Building this authentication into a communications protocol is an area for further research. It might also be possible to create a sort of VPN, or virtual private network, between the CE device and the desktop to prevent the disclosure of data that is sent over the link. Right now, PDA devices are very vulnerable to tools such as PortMon, which can read the transmitted data directly. PortMon is a freeware utility available at the URL www.sysinternals.com.

Finally, the importance of the human element in computer security can not be overstated. With the simplified architecture of the Windows CE OS, anyone who has access to the physical device has access to all of the information stored on it. The CE device should never be left unattended or given to someone who is potentially untrustworthy. A mandatory user password or smartcard system should be used to restrict access to the CE device. A system lock feature should be implemented to suspend the device after a defined period of inactivity. This would help prevent an unauthorized person from using the device.

Overall, Windows CE devices can be used efficiently for performing a variety of tasks. They provide flexibility, mobility, and a familiar user interface. Before deploying CE devices in organizations where security is a concern, some modifications to the generic CE operating system should be made. These changes might include removing capabilities of the OS that are not needed, building the kernel in protected mode only, implementing the Certify Module functions, usage of a unique device ID, strengthening the synchronization protocols, and increasing user awareness of the vital importance of good security practices.

APPENDIX A – PLATFORM BUILDER AND WINCE TOOLS

This appendix discusses some useful software and hardware tools associated with Windows CE. Platform Builder is the integrated development environment for building the Windows CE operating system. The CEPC is a hardware reference platform for downloading and testing Windows CE operating system builds. Embedded Visual Tools provides support for building C++ and Visual Basic applications for Windows CE.

A. MICROSOFT WINDOWS CE PLATFORM BUILDER 3.0

Platform Builder (PB) is a tool for building custom Windows CE operating systems for embedded system devices [MIC00b]. The eleven CD-ROM installation kit includes the Windows CE 3.0 operating system, a set of embedded development tools, an integrated development environment (IDE), support for the Microsoft run-time libraries, the ActiveSync application, and sample code. An Add-On pack containing more debugger tools, a limited version of the source code, and Board Support Packages (BSPs) for various processors can also be obtained from Microsoft.

Nine standard configurations of the Windows CE OS can be built using PB [MIC00a]. The configurations are: Minkern, Mininput, Mincomm, Mingdi, Minwmgr, Minshell, Maxall, IESample, and DUB. Descriptions of these OS configurations are listed in Table A.1.

Configuration	Description
Minkern	a minimal version of Windows CE
Mininput	provides user input and native device driver support
Mincomm	includes serial communications and networking
Mingdi	includes Graphics Device Interface (GDI) support
Minwmgr	provides window management

Minshell	includes almost all components, provides the Task Manager and the Command Processor
Maxall	a fully configured version that includes several communication applications
IESample	a demonstration version that includes Microsoft Internet Explorer browser components
DUB	includes the Dial-Up Boot Loader for downloading and updating the OS

Table A.1 WINCE OS Configurations [MIC00a]

In the Windows CE penetration testing exercise, the Maxall configuration was used. It is a complete version of the CE OS, containing all non-conflicting modules. The following table, Table A.2, was taken from the Platform Builder online help, and it lists the features and components included in Maxall. Similar tables exist in the PB online help for the other standard OS configurations.

Functional area	**Features**	**Component/module names**
Kernel/OAL	Memory, process	Nk
	Compression support	Nkcompr
CoreDLL	Full NLS-capable APIs	Coreloc
	Local heap and memory allocation	Lmem
	Messaging, user input, windowing, and GDI support	Wmgr_c, Mgdi_c
	Communications support for serial communications and TAPI	Serdev, Tapilib
	Crypto 1.0 APIs with two CSPs: Rsabase.dll and Rsaenh.dll	Cryptapi
	WAV API support	Waveapi
	IMM support	Coreimm
	Stdio support	Coresioa(w)
	C runtime support	Corestra(w)
	FileInfo and FileOpen common dialogs	Fileinfo, Fileopen
	Shell API support	Shellapis

	Soft input panel support, including sample IM (English/Korean/Japanese)	Softkb, Coresip
	FormatMessage API support and Message resources	Fmtres and Fmtmsg
	Keyboard accelerator support	Accel_c
	TAPI API support	Tapi, Tapilib
	WAV APIs	Waveapi
	Crypto APIs	Cryptapi
File system	RAM, ROM and IFS support	Fsysram
	Database	Fsdbase
	System registry	Fsreg
	Password support	Fspass
	FAT file system	Fatfs
Device Manager	Generic device driver support	Device
GWES	Messaging, and user input support including support for standard window controls, such as buttons, edit controls, and scroll bars	Msgbeep, Msgbox, Msgque, Uibase, Edctl, Scbctl, Btnctl, Cascade, Clipbd
	Japanese language support, including support for edit controls, IME, and Japanese characters	Hwxjpn, Msime98, Imejpp, Edimejpn
	Korean language support, including support for edit controls, IME, and Korean characters	Edimefek, Mshime97
	Timer message support	Timer
	Clipboard support	Clipbd
	Power management	Getpower
	Notification LED support	Nled
	GDI support, including TrueType, text drawing, palette, and printing support	Mgtt, Mgdrwtxt, Mgpal, Mgprint
	Customizable Touch Screen Calibration UI	Tchui
	Network UI dialog boxes	Netui
	WAV API and PCM manager	Waveapi
	IMM support	Immthunk
	Window and dialog box management	Wmbase, Dlgmgr
	Customizable Startup, Out of Memory, Touch Screen Calibration, and	Startui, Oomui

	Notification UIs	
	User notifications API support	Notify, Notifymin
	Windows CE common controls and common dialog boxes	Commctrl, Cmbctl, Cascade, Btnctl
Communications: Serial	Basic serial communications support	Serdev
	IR Emulated Serial Port support	IrComm
	Serial over PC Card	Serial
Communications: Networking	Winsock APIs	Winsock
	Schannel with SGC support, and SCHNLUSA (128-bit)	Schannel, SCHNLUSA
	TCP/IP and IR	Tcpstk, Irdastk
	Network Driver Interface Specification (NDIS)	NDIS
	DHCP	DHCP
	SLIP/PPP	PPP
	RAS	PPP
	WNet/SMB redirector	Redir, Netbios
Communications: Other	Telephony (TAPI)	TAPI, Unimodem
	Customizable Network UI	Netui
	IP configuration tool	Ipconfig, Route
	IPHelper APIs	Iphlpapi
	NTLM	Ntlmssp
	MSMQ	Mqoa, Msmqapi, Msmqd, Msmqdm, Msmqrt
	Sample network drivers: NDIS	Ne2000, Xircce2
Management	SNMP server	Snmp
	SNMP MIDs	Snmp_mibii, Snmp_hostmib
Device drivers	Display	Display
	PCMCIA Card and socket services	PCMCIA
	Keyboard and mouse	Kbdmsg.lib, mouse
	Battery	Getpower
	Notification LED	Nled
	Touch screen	Tchui

	USB HID driver	Usbhid, Usbmouse
	CardTest, sample PC Card driver	
	NDIS minidrivers	Ne2000, Xircce2
	ATA disk	Atadsik
	SRAM disk	Sramdisk
	Serial, sample	Serial.dll
	Audio	Gsm610, Audio
COM/OLE	COM, OLE, OLE Automation, Istorage or full DCOM/COM support	Com, Olemain, Oleaut32, Docfile, Mfs, Nep
DCOM		
Add-in technologies	Handwriting recognition support	Hwxusa
	Spelling checker	Splusa
	Microsoft Message Queue Server (MSMQ)	Mqoa, Msmqapi, Msmqd, Msmqdm, Msmqrt
	Microsoft Jscript® development software	Jscript
	HTTP Server	Httpd, Httpdadm, Httpdsvc
Windows CE shell components	Command Processor	Cmd
	Control Panel applications	Commctrl
	Handheld PC style shell	Explorer, Asform, Ce shell
Applications	Communications	
	Microsoft® Pocket Internet Explorer Internet browser	Iexplore
	Microsoft® Pocket Word	Pwd_res, Pword, Pwwiff
	Inbox	Labledit, Mailutil, Msgstore
	Help for Windows CE	Peghelp
Debugging	Shell.exe and ToolHelp.dll	Shell, Toolhelp

Table A.2 Maxall Components [From: Platform Builder Online Help]

Environment variables may be set to control the driver configurations. Sample settings are contained in the Cesysgen.bat file. For more information on the environment variables, see the online Platform Builder help.

1. Installation

In order to install and use Platform Builder on a desktop PC, local administrator privileges are required. It is important to use the same administrator account when installing and using Platform Builder. The installation is accomplished through the use of wizards. To install, the user must insert Platform Builder Disc 1 and follow the instructions displayed on the screen. If Ethernet downloads will be used, that choice should be selected at installation time. Note that selecting parallel port downloads will disable local parallel port printing. So, a local printer can not be used if the parallel port download option is selected. Also, to save hard disk space, the unwanted processors can be deselected in the "CPU Selection" dialog box. To build images for the CEPC, only the x86 microprocessor support is needed. Therefore, all other processors can be deselected.

2. Hardware Requirements

The Platform Builder installation requires an x86 desktop PC running Windows NT or Windows 2000 Professional, with Service Pack 5 or later. The recommended processor is one that is equivalent to a Pentium 200 MHz or higher. Other requirements are: 64 MB RAM, 1.36 GB of hard-disk space available for a typical installation or 7.8 GB for the complete installation, CD-ROM or DVD-ROM drive, VGA or higher resolution monitor, Microsoft compatible mouse, bi-directional parallel port, serial port, Ethernet network card (optional), network hub (optional).

3. Building a Platform

The best way to build a platform, or Windows CE OS image, using Platform Builder is to follow the documented procedure in the Platform Builder online help or the Getting Started Guide [MIC00b]. There are four main steps when building a platform. Those steps are:

1. configure the platform
2. make the OS image bin file
3. transfer the platform to a target device (such as a CEPC)
4. debug the platform

Platforms may be built with kernel debugging either enabled or disabled, and release or debug versions of the platforms can be built. The debug versions are larger in size, so care should be taken that enough RAM exists on the target device to hold the OS image. Kernel profiling can also be built into the platform to measure the system performance. Refer to the Platform Builder online help for more details. Generally, the documented procedure was helpful, and no major problems were encountered while building platforms.

B. CEPC

Platform Builder supports the CEPC, which is an x86 hardware development platform. In many situations, the CEPC is more convenient to use, since it is easy to download new OS images as they are developed. It is possible to assemble a CEPC from basic components. The necessary hardware is listed in the Getting Started Guide [MIC00b]. Very minimal information is provided about building a CEPC, and our attempts in that area were not successful. An easier path is to purchase a ready-made CEPC from a third party vendor, such as Special Computing.

C. MICROSOFT EMBEDDED VISUAL TOOLS 3.0

Microsoft eMbedded Visual Tools 3.0 is an applications development suite that allows C++ and Visual Basic programs to be written for Windows CE devices. It is very similar to Microsoft Visual Studio, but allows the development of Windows CE applications rather than Windows NT, 98, or 2000 programs. Embedded Visual tools contains an editor, debugger, and help files, all within a convenient graphical environment. Embedded Visual Tools can be ordered at no cost from the following URL: http://www.microsoft.com/Windows/embedded/ce/tools/emvt30order.asp.

D. SUMMARY

This appendix described some relevant hardware and software tools that are useful for the development and testing of customized versions of Windows CE. Platform Builder and CEPCs are excellent tools for developing a customized operating system, and applications can be developed using Embedded Visual Tools.

APPENDIX B – IMAGIX 4D AND SOFTWARE ANALYSIS TOOLS

This appendix describes various software analysis tools. The tools were investigated in order to find the most suitable application for analyzing the Windows CE source code. One important criteria for tool selection was the ability to map out dependencies between modules in the source code.

A. C AND C++ SOFTWARE ANALYSIS TOOLS COMPARISON

Several reverse engineering and software analysis tools were examined for possible use in analyzing the Windows CE 3.0 source code. Imagix 4D was chosen for its ease of use and compatibility with the Microsoft Visual Studio compiler. Table B.1 compares the tools that were evaluated. The column labeled "Demo Version" indicates the availability of a demonstration version of the tool.

Item	Tool Name	OS Supported	Cost	Demo Version
1	CIAO / CIA	Solaris, SGI	Free for education use	Yes
2	PBS, Software Bookshelf	Linux, Win32, Solaris	Free	Yes
3	Imagix 4D	Win32, Irix, Linux, Solaris, HP-UX	Per installation, approx $2000 per computer	Yes
4	+1ReverseC	Solaris	Contact sales	Yes
5	DMS Software Engineering Toolkit	Win32	Contact sales	No
6	jGrasp	Win32, Linux, Java, and more	Free	Yes

Table B.1 Software Analysis Tools Comparison

B. DESCRIPTION OF SOFTWARE TOOLS

The following is a brief description of the software analysis tools, and a listing of where the tools can be obtained.

1. CIAO / CIA

CIAO / CIA is a reverse engineering tool with a graphical user interface. It allows the user to query and browse structural connections in software and document repositories. It can be found at the URL http://www.research.att.com/~ciao/.

2. PBS

This tool is web based, and operates within a web browser. The user can navigate information representing large software systems. The tool shows the relationships between files, but it is uncertain whether it can permit viewing at the function call level. It can be found at the URL swag.uwaterloo.ca/pbs/.

3. Imagix 4D

This tool can be used to reverse-engineer large, complex, or unfamiliar software written in C/C++. It has a graphical color 3D view of dependencies, supports the creation and printing of reports, and allows integrated text editing of the source code. It can support the parsing of several compiler types through customized compiler configuration files. Information about this tool can be found at the URL www.imagix.com.

4. +1ReverseC

This tool parses C code, and graphically displays the calling structure. It can be found at the URL www.plus-one.com/+1ReverseC_fact_sheet.html.

5. DMS Software Engineering Toolkit

This toolkit provides a set of tools for carrying out custom re-engineering of medium and large-scale software systems. It supports analysis, porting, translation, interface changes, and/or domain-specific program generation. This tool had no demo and their website was rather confusing. The toolkit can be found at the URL www.semdesigns.com.

6. jGrasp

jGrasp makes extensive use of Control Structure Diagrams. The Control Structure Diagram (CSD) is a control flow and data structure diagram that is designed to fit into the space that is normally taken by indentation in source code. The intention of the CSD is to improve the comprehension efficiency of source code and, as a result, improve software reliability and reduce software costs. The purpose of GRASP is to provide this diagram in the context of a full-featured development environment. At this point it would be called a "light" IDE, as there are no integrated debuggers or navigation and related features. This tool is free, easy to use, and provides a good sense of the source code complexity. A complexity profile graph is a visualization of the complexity of a program unit, based upon a profile metric designed to compute complexity at various levels of granularity. Unfortunately, jGrasp does not generate complexity profile graphs for C and C++, but only for Java and Ada at this time. The tool can be found at the URL http://www.eng.auburn.edu/department/cse/research/grasp/.

After investigating these six software analysis tools, Imagix 4D was selected to aid in the analysis of the Windows CE 3.0 source code. Imagix is capable of analyzing very large software projects, and was developed specifically for reverse engineering

legacy code. It has very good support for the Microsoft Visual C++ compiler, which is probably the most similar to the compiler used on the Windows CE source code. Imagix 4D also has a very understandable graphical user interface, and is comparatively easy to use. The selling point of this tool was its graphical representation of dependencies at the file, module, and function levels. This is exactly the functionality that was needed to examine the relationships between modules in the Windows CE 3.0 source code.

LIST OF REFERENCES

A. BOOKS AND ARTICLES

[AGA00] Agar, Chris; Brock, Jerome; Burns, Titus; Stavritis, George; "CS 4600 Penetration Study Report: The Group that Shall Remain Unnamed", Naval Postgraduate School, Monterey, CA, (December 2000).

[AND72] Anderson, J. P., "Computer Security Technology Planning Study.", ESD-TR-73-51, Vol. 1, Hanscom AFB, MA, DTIC-AD-758206, (1972).

[ALF00] Alforque, Maricia, "Creating a Secure Windows CE Device", Microsoft Technical Article, http://msdn.microsoft.com/library/default.asp?URL=/library/techart/winsecurity.htm, (October 2000).

[ATT76] Attanasio, C. R., Markstein, P. W., Phillips, R. J., "Penetrating an operating system: a study of VM/370 integrity." *IBM System Journal No. 1* 102-116, (1976).

[BEL73] Bell, D., LaPadula, L., "Secure Compute Systems." Air Force Elec. Syst. Div. Report ESD-TR-73-278, Vols. I, II, and III, (November 1973).

[BIS78] Bisbey, Richard, Hollingworth, Dennis, "Protection Analysis: Final Report." *ISI/SR-78-13, ARPA ORDER NO. 2223*, (May 1978).

[BRI00] Briggs, Jaime; Glover, Mark; Negi, Chandan; Pereira, Barbara; "CS 4600 Penetration Study Report: The X-Team", Naval Postgraduate School, Monterey, CA, (December 2000).

[CLE00] Clement, Gary; Del Grosso, Michael; McCalister, Jim; Mosley, Harold; Thompson, John; "CS 4600 Penetration Study Report: The HackMasters Group", Naval Postgraduate School, Monterey, CA, (December 2000).

[DIJ68] Dijkstra, Edsger W., "The Structure of the "THE"-Multiprogramming System." *Communications of the ACM, Vol 11, No. 5* 341-346, (May 1968).

[DOD85] DoD Trusted Computer System Evaluation Criteria, 5200.28-STD, (26 December 1985).

[GAR99] Gareau, Jean Louis, *Windows CE From the Ground Up*, Anabooks, (1999).

[GOL79] Gold, B. D., Linde, R. R., Peeler, R. J., Schaefer, M., Scheid, J.F., Ward, P. D., "A Security Retrofit of VM/370." *National Computer Conference* (1979).

[JON99] Jones, Anthony, Ohlund, Jim, *Network Programming for Microsoft Windows*, Microsoft Press, ISBN: 0735605602, (1999).

[LAN94] Landwehr, Carl E., Bull, Alan R., McDermott, John P., Choi, William S., "A Taxonomy of Computer Program Security Flaws, with Examples." *ACM Computing Surveys, 26.3,* (Sept 1994).

[LIN75] Linde, Richard R, "Operating system penetration." *National Computer Conference* 361-368, (1975).

[MIC00a] Microsoft Corporation, "Microsoft Windows CE 3.0 Operating System Configurations", http://msdn.microsoft.com/library/techart/configs30.htm, (Apr 2000).

[MIC00b] Microsoft Corporation, "Microsoft Windows CE Platform Builder 3.0 Getting Started Guide", Part No. X05-69830, (May 2000).

[MUE00] Muench, Chris, *The Windows CE Technology Tutorial*, ISBN: 0201616424, Addison-Wesley, (2000).

[MUR98] Murray, John, *Inside Microsoft Windows CE*, ISBN: 1572318546, Microsoft Press, (1998).

[PAR72] Parnas, D. L., "On the Criteria To Be Used in Decomposing Systems into Modules." *Communications of the ACM, Vol. 15, No. 12*1053-1058, (Dec 1972).

[PRE97] Pressman, Roger S., *Software Engineering – A Practitioner's Approach*, 4th edition, The McGraw Hill Companies, Inc., ISBN: 0073655783, (1997).

[RHO98] Rhodes, Neil, McKeehan, Julie, *Palm Programming: The Developer's Guide*, ISBN: 1565925254, (Dec 1998).

[SAL75] Saltzer, Jerome H., Schroeder, Michael D., "The Protection of Information in Computer Systems." *Proceedings of the IEEE, Vol. 63, No. 9* 1278-1308, (Sept 1975).

[SCH72] Schroeder, Michael D., Saltzer, Jerome H., "A Hardware Architecture for Implementing Protection Rings." *Communications of the ACM, Vol. 15, No. 3* 157-170, (Mar 1972).

[SCH77] Schroeder, Michael D., Clark, David D., Saltzer, Jerome H., "The Multics Kernel Design Project." *Proceedings of Sixth ACM Symposium on Operating Systems Principles* 43-56, (Nov 1977).

[SIB95] Sibert, Olin, Porras, Phillip A., Lindell, Robert, "The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems." *IEEE* 211-222, Oakland, CA, (1995).

[SMI99] Smith, Sean W., Weingart, Steve, "Building a high-performance, programmable secure coprocessor." *Computer Networks, The International Journal of Computer and Telecommunications Networking,* 31, 831-860, (1999).

[SOL98] Solomon, David A. *Inside Windows NT*, Second Edition, Microsoft Press, ISBN: 1572316772, (1998).

[WEI95] Weissman, Clark, "Security Penetration Testing Guideline.", *Handbook for the Computer Security Certification of Trusted Systems Ch. 10*, (Jan 1995).

B. OTHER

Newsgroups

1. msnews.microsoft.com
2. microsoft.public.windowsce.embedded
3. microsoft.public.windowsce.platbuilder
4. microsoft.public.windowsce.platbuilder.beta
5. microsoft.public.windowsce.targeted.device

Microsoft Websites

1. Windows Embedded CE, www.microsoft.com/windows/embedded/ce/default.asp
2. Windows Embedded, www.microsoft.com/windows/embedded/default.asp
3. Microsoft Developers Network Embedded, http://msdn.microsoft.com/embedded
4. Microsoft Newsgroups http://communities.microsoft.com/newsgroups/default.asp?icp=msdn&slcid=us
5. PocketPC, www.microsoft.com/mobile/pocketpc/default.asp
6. Microsoft Developer's Store http://developerstore.com/devstore/pbsourcecode.asp

CEPC Websites

1. Special Computing, http://www.specialcomp.com
2. Real Time Online, http://www.realtimeonline.com/CEPC.htm

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   833 Dyer Road
   Monterey, CA 93943-5118

4. Commander, Naval Security Group Command 1
   Naval Security Group Headquarters
   9800 Savage Road
   Suite 6585
   Fort Meade, MD 20755-6585
   San Diego, CA 92110-3127

5. Mr. Paul Clark 2
   Naval Postgraduate School
   Code CS/Cp
   833 Dyer Road
   Monterey, CA 93943-5118

6. Ms. Deborah M. Cooper 1
   Deborah M. Cooper Company
   P.O. Box 17753
   Arlington, VA 22216

7. Ms. Louise Davidson 1
   N643
   Presidential Tower 1
   2511 South Jefferson Davis Highway
   Arlington, VA 22202

8. Mr. William Dawson 1
   Community CIO Office
   Washington DC 20505

9. Mr. Richard Hale 1
Defense Information Systems Agency, Suite 400
5600 Columbia Pike
Falls Church, VA 22041-3230

10. Dr. Cynthia E. Irvine 2
Naval Postgraduate School
Code CS/Ic
833 Dyer Road
Monterey, CA 93943-5118

11. Capt. James Newman 1
N64
Presidential Tower 1
2511 South Jefferson Davis Highway
Arlington, VA 22202

12. Mr. Carl Siel 1
Space and Naval Warfare Systems Command
PMW 161
Building OT-1, Room 1024
4301 Pacific Highway
San Diego, CA 92110-3127

13. Ms. Barbara A. Pereira 1
48015 Mayflower Dr
Lexington Park, MD 20653