

# Windows<sup>®</sup> Internals and Software Driver Development

## Overview

This seminar is designed for software engineers and architects who need to understand the details of the major Windows architectural components, as well as how to create software only kernel-mode drivers that serve as "extensions" to the Windows operating system. The seminar includes a review of basic Windows architecture, and covers the basics of building and debugging kernel-mode drivers for Windows. It then discusses the details of driver structure, including the I/O and PnP subsystems, device discovery, enumeration, and interrupt processing. Other extension mechanisms, such as Object Manager, Process Manager, and Registry callbacks are discussed. The seminar also includes in-depth discussions of the Windows Kernel, process and thread instantiation, and the Memory Manager.

During lab sessions, participants create and modify software-only drivers to perform various kernel-mode tasks. Note that this seminar focuses on the development of drivers that service as Windows "operating system extensions" but do not service any hardware. Thus, while the basic concepts of drivers for devices are discussed, the lab sessions focus strictly on the development of software (i.e. pseudo) drivers which typically use "legacy" or "NT V4" style interfaces (as opposed to PnP).

## Seminar Formats

This seminar is available as a 3 day lecture or a 5 day lab session. Either version can be customized with additional or different topics for presentation at your location. Please contact an OSR Seminar Coordinator for details on arranging an on-site seminar.

## Target Audience

Security researchers, government contractors, engineers involved in security and threat analysis/modeling, who need a solid understanding of key Windows internals and data structures, and the options available to monitor and extend the actions of the Windows operating system. This seminar is also appropriate for engineers actively involved in the areas of intelligence and information warfare or cyberwarfare.

## Prerequisites

This is not a seminar for beginners without knowledge or experience in either operating systems or firmware. Rather, it is an intense, practical, architectural study of specific parts of the Windows O/S, interspersed with practical exercises.

Students attending this seminar will need a good understanding of general O/S concepts and will be well served by starting a basic understanding of Windows O/S architectural concepts. Because it is a hands-on seminar, students need a working knowledge of the C programming language, as well as the basic ability to use a Windows system.

## Seminar Outline

### 1. Windows Operating System Architecture Overview

A brief review of the general architecture of the Windows operating system.

### 2. Kernel Mode and Key Structures

In this module, we discuss fundamentals of Kernel Mode in Windows, including coverage of key object types (Dispatcher, Control, Executive) and data structures (KPCR, KTHREAD, EPROCESS) that Windows uses.

### 3. Device Stacks – The Windows I/O Subsystem

A discussion of how the PnP Manager works with drivers to build stacks of devices through the enumeration process. The role of Function Drivers, Bus Drivers, and Filter Drivers is discussed. PDOs, FDOs, and Filter Device Objects are defined. The relationship of the PnP process to the process of loading "legacy" style drivers is discussed.

### 4. Installing Legacy Drivers on Windows

In this section we discuss how to create installation control files for "legacy" style, software-only drivers. Also included is a brief discussion of PnP (WDM) driver installation using INF files.

### 5. Building and Debugging

This section describes how WDM drivers are built using the WDK build environment as well as the basics of how to setup and use the Windows kernel mode debugger, WinDbg.

**Lab:** Setting Up the Debugger  
Building Drivers

### 6. Interrupt Request Levels and Deferred Procedure Calls

Windows synchronizes kernel mode activity by using a set of Interrupt Request Levels (IRQLs). This section covers how IRQLs are used to achieve synchronization within the OS. Also the processing that occurs at these IRQLs - including Deferred Procedure Calls (DPCs) and dispatching - are discussed.

### 7. Passing Requests to Other Drivers

IRPs, I/O Stack Locations, and how requests are sent from driver to driver using IoCallDriver are all discussed. Synchronous and asynchronous IRP completion. A brief discussion of completion routines is also included.

### 8. I/O Function Codes and Buffer Methods

A specific discussion of I/O function codes (major and minor), as well as defining custom Device Control requests (IOCTLs). An detailed discussion of the way that Windows passes buffers from user-mode applications to kernel-mode, including the security implications of each method.

## 9. DriverEntry -- Legacy vs. PnP

This section describes how software only drivers are initialized, and includes a walk-through and discussion of the actions taken in a typical DriverEntry function. The discussion is rounded-out with a description of the initialization functions used by PnP drivers (Add Device and Start Device).

## 10. Dispatching and Completing Requests

This module describes Dispatch Routines including I/O request validation, as well as the basics of request queuing and completion.

## 11. Serialization: Wait Locks and Spin Locks

A discussion of the various mechanisms used in Windows kernel-mode programming to perform synchronization and serialization. The different types of spin locks, mutexes, and other locks are discussed, along with guidelines for when each might be appropriate for use.

**Lab:** Handling DriverEntry and Dispatching Requests  
Queuing, Buffering, Async I/O  
Driver Communication

## 12. Tools for Driver Quality

In this section, we discuss tools that are available in the Windows Driver Kit to test and validate drivers. Windows Driver Verifier, OACR/PREfast, and Static Driver Verifier are all discussed, along with the strengths and weaknesses of each tool and recommendations for the best use of each.

## 13. Boot, Crash Dump, and Hibernation Processes

A discussion of how Windows starts, including transitions from 16-bit mode. Also, a discussion of the crash dump and hibernation processes (which are remarkably similar on Windows). Version-specific differences are also discussed.

## 14. Windows System Services

How system service calls are implemented in Windows, both in new versions of the OS and historically. A description of a selection of NtXxxx and ZwXxx functions, and how their use in user-mode and kernel mode differs.

## 15. Processes, Threads and Dispatching

In this section, we discuss the role of the Process Manager, including how processes and threads are instantiated. The major data structures (ETHREAD, EPROCESS) are described. The native system services for creating processes and threads are also briefly discussed. How dispatching (scheduling) is performed in Windows.

## 16. Other Kernel Extensions

An extended discussion of methods, other than those in the I/O domain, of extending the Windows operating system using software only drivers. This includes a discussion of Object Manager, Process Manager, and Registry callbacks, including the significant differences in the availability of these methods among various Windows versions.

**Lab:** Continue previous labs  
Kernel Extensions

## **17. Cleanup, Close, and Cancel**

Cleanup, close and request cancellation are compared and contrasted. When one might need to implement support for each in a typical WDM driver is discussed. Guidelines for supporting request cancellation, including Cancel Safe Queues and in-progress request handling are presented.

## **18. Windows Virtual Memory**

A discussion of Windows virtual memory subsystem, including portions of the Memory Manager and Cache Manager. Page tables. Paging and page fault handling.

## **19. The Details of I/O Completion**

In this section, we describe the details of how Windows completes I/O requests, as well as the correct handling of different types of I/O completion requirements within drivers. This includes a discussion of how the I/O Manager manages maintains synchronous request handling when requested by an application, even when a driver performs asynchronous I/O processing. Guidelines for proper driver implementation are developed and discussed.

**Lab:** Continue previous lab sessions  
Cancel