CENTER FOR CYBER SECURITY TRAINING

Windows Kernel Internals

Abstract

Kernel-mode software has unrestricted access to the system. This is why most anti-malware solutions and rootkits are implemented as Windows kernel modules. To analyze rootkits, identify indicators of compromise (IoC) and collect forensic evidence it is critical to have a good understanding of the architecture and internals of the Windows kernel. This course takes a deep dive into the internals of the Windows kernel from a security perspective with an emphasis on internal algorithms, data structures, debugger usage.

This training course focuses on security-related topics and **does not cover topics related to hardware** such as plug and play, power management, BIOS, or ACPI.

In the hands-on lab exercises, students dig into the kernel using the kernel debugger (WinDBG/KD) commands and learning how to interpret the debugger output of these commands to understand how the kernel works. Hands-on lab exercises are performed on pre-captured memory dumps and on a live VM running the latest version of Windows 10 64-bit.

Learning Objectives

Students will:

- Understand the key principles behind the design and implementation of the Windows kernel.
- Understand the major components in the Windows Kernel and the functionality they provide.
- Be able to investigate system data structures using kernel debugger and interpret the output of debugger commands.
- Be able to navigate between different data structures in the kernel using debugger commands.
- Be able to locate indicators of compromise while hunting for kernel-mode malware.

- Be able to perform forensic analysis of the Windows kernel.
- Understand how kernel-mode rootkits and commercial anti-malware solutions interact with the system

Target Audience

Anti-malware engineers, malware analysts, forensics examiners, security researchers who are responsible for detecting, analyzing, and defending against rootkits and other kernel post exploitation techniques.

Requirements

Students will need:

Hardware

- Virtualization capable CPU(s)
- Minimum 8GB of RAM (for running one guest VM)
- Minimum 40 GB free disk space
- Working USB Port
- Working Wireless LAN

Software

- Host OS Windows 10 64-bit
- Windows Enterprise WDK for Windows 10 Version 1709 (RS3)
- Debugging Tools for Windows (included in WDK)
- SysInternals Tools
- Volatility Framework
- Virtualization Software (Hyper-V, VMWare, VirtualBox)
- Guest OS Windows 10 64-bit Version 1709 (RS3)
- System Administrator access required on both host and guest OSs
- WinDBG must be setup and configured on the host to debug the guest OS
- All other software will be provided by the instructor.

Students' Knowledge Pre-Requisites:

Attendees must have a solid understanding of operating system concepts and have a working knowledge of Windows. This course does not require any programming knowledge.

Course Outline:

Topics

- Kernel Architecture
- Processes and Threads
- System Mechanisms
- Execution Contexts
- Synchronization
- Memory Management
- I/O Management
- Kernel Security Mitigations

Kernel Architecture

The objective of this section is to learn about the architecture of the Windows kernel and key kernel-mode components. It covers topics such as privilege levels, segment registers, global descriptor table (GDT), modern PC platform, NTOSKRNL component list, HAL, Win32K.sys refactoring, kernel module list, code integrity (CI), driver load notification callbacks.

- Execution rings
- Platform architecture
- Kernel-mode components
- NTOSKRNL layers
- Kernel module list
- Image notification callbacks

Processes and Threads

The objective of this section to learn about how the support provided by the kernel for user-mode code execution. It covers topics such as process resources, process and thread data structures (EPROCESS/KPROCESS, EHTREAD/KTHREAD), system processes, system idle process, minimal processes, system call dispatching, user-mode and kernel-mode stacks, different lists that processes and threads are maintained in the kernel and process/thread creation and termination callbacks.

- Processes and threads
- Process and thread data structures
- Special processes
- Process resources

- System calls
- User kernel transition
- Process lists
- Process and thread callbacks

System Mechanisms

The objective of this section to discuss the foundational building blocks of the system that kernel components rely on. It covers topics such as Zw/Nt APIs, model-specific registers, dispatching native API to NTOSKRNL.exe and Win32K.sys, 64-bit SSDT, machine frames, trap frames, .PDATA section, runtime image info structures, exception handling, KPCR, KPRCB, TEB, IRQLs, and DISPATCH_LEVEL restrictions.

- Native APIs
- Model-specific registers (MSRs)
- System service dispatching
- Trap frames
- Exception handling
- Kernel process control region (KPCR)
- Interrupt request levels (IRQL)

Execution Contexts

The objective of this section is to learn about the different mechanisms available for kernel-mode code execution. It covers topics such as kernel timers, executive timers, DPCs, user APCs, kernel APCs, special kernel APCs, process/thread suspend/resume, system worker threads, work items, executive work queues, custom driver worker threads.

- Kernel timers
- Deferred procedure calls (DPC)
- Asynchronous procedure calls (APC)
- Thread suspend/resume
- System worker threads
- Work items
- Custom driver threads

Synchronization

The objective of this section is to learn about the different synchronization primitives available in the Windows kernel. It covers topics such as dispatcher objects, thread waitlists, interlocked operations, critical regions, mutually exclusive locks vs reader-writer locks, mutexes, fast mutexes, high IRQL synchronization, spin-locks, in-stack queued spin-locks, reader-writer spin-locks, and the considerations when selecting a synchronization mechanism.

- Dispatcher objects
- Interlocked operations
- Mutexes
- Critical regions
- Executive resources
- Push-locks
- Spin-locks

Memory Management

The objective of this section is to understand how kernel memory is managed by Windows. It covers topics such as physical and virtual address translation, page table entries (PTEs), physical page management, kernel virtual address space (KVAS) layout, page table space, session space, thread kernel stacks, stack jumping, pool types, small and large pool allocations, lookaside lists, usage of MDLs for memory mapping.

- Address translation
- Kernel virtual address space
- Page frame number (PFN) database
- Session space
- Kernel stacks
- Kernel pools
- Memory Descriptor Lists (MDL)

I/O Management

The objective of this section is to understand how drivers interface with the Windows kernel. It covers topics such as driver dispatch entry points, driver objects, device objects, file objects,

symbolic links, driver types (function, bus, filter), device types (FDO, PDO, FiDO), driver layering, device attachment/detachment, IRPs, I/O stack locations, IRP processing, I/O completion routines, I/O cancellation, I/O requests filtering.

- Driver architecture
- I/O manager data structures
- Driver types
- Device object types
- IRPs and I/O stack locations (IOSLs)
- I/O processing
- Filter drivers

Kernel Security Mitigations

The objective of this section is to understand the different exploit mitigations and anti-rootkit features that have been added to the Windows kernel over the course of its lifetime. It covers topics such as kernel attack surface, GS cookies, NULL page allocation prevention, safe linking and unlinking, executable and non-executable (NX) pools, kernel ASLR, page table base randomization, driver signature enforcement, attestation signing, PatchGuard, meltdown mitigations, software SMEP, KVA shadowing.

- Kernel exploitation
- Kernel data execution prevention (KDEP)
- Kernel address layout randomization (KASLR)
- Supervisor mode execution prevention (SMEP)
- Kernel-mode code signing (KMCS)
- Kernel patch protection (PatchGuard)
- Kernel virtual address shadowing (KVAS)