CENTER FOR
CYBER SECURITY
TRAINING

Windows Kernel Rootkits

Abstract

To achieve maximum stealth and obtain unabated access to the system, rootkits execute in kernel mode. This course focuses on the kernel interfaces (APIs), data structures and mechanisms that are exploited by rootkits to achieve their goals at every stage of their execution. Kernel security enhancements that have been progressively added from Windows 7 to the latest version of Windows are discussed along with some circumvention techniques.

This advanced course provides a comprehensive end-to-end view of the modus-operandi of rootkits by taking an in-depth look at behind the scenes working of the Windows kernel and how these mechanisms are exploited by malware through hands-on labs and real-world case studies. Kernel security enhancements that have been progressively added to Windows are discussed along with some circumvention techniques. Attendees will study key techniques used by rootkits to understand the real-world applicability of these concepts for offensive and defensive purposes.

This training is beneficial to anyone responsible for developing, detecting, analyzing, and defending against rootkits and other Windows kernel post-exploitation techniques including EPP/EDR software developers, anti-malware engineers, security researchers, red/blue/purple teamers.

A special version of this training is also available for malware, rootkit forensics analysts where the focus is not on implementing rootkit functionality but rather on investigating rootkits using tools such as WinDBG and Volatility. This analyst version does not require attendees to have a programming background and contains topics related to rootkit detection and case studies.

Learning Objectives

Students will:

- Understand vulnerabilities in the Windows kernel and device drivers.
- Be able to write and modify kernel-mode exploits.
- Understand the security enhancements that have been added to the Windows kernel over time.
- Be able to bypass some of the security mitigations in recent versions of Windows.
- Understand the post-exploitation steps performed by kernel-mode rootkits.
- Understand the techniques used by real-world rootkits.
- Understand how rootkits hide their presence in the system.
- Understand how rootkits intercept systemwide networking activity.
- Be able to identify malicious behavior and defend against rootkits.

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:

This is an advanced level course which requires attendees to be fluent in C/C++ programming, have a good knowledge of the Windows kernel internals/APIs and be able to use the kernel debugger (WinDBG) to debug Windows kernel modules.

Course Outline:

Topics

- Kernel Attacks
- Kernel Shellcoding
- Kernel Hooking and Injection
- Kernel Callbacks
- Kernel Filtering
- Kernel Networking
- Virtualization Based Security

Kernel Attacks

The objective of this section is to learn about vulnerabilities in kernel-mode drivers and how they are exploited by attackers to escalate privilege and gain code execution. It covers topics such as phases of rootkit execution, remote code execution, local code execution, hostile environment detection, kernel exploitation primitives, exploiting vulnerable drivers, determining kernel version, privilege escalation methods, internal kernel structure manipulation, and methods of controlling the kernel-mode instruction pointer. System defenses such as kernel-mode address space layout randomization (KASLR), supervisor mode execution prevention (SMEP), and kernel virtual address shadowing (KVAS) are also covered.

- Kernel attack workflow
- Types of vulnerabilities
- Environment detection
- Exploiting drivers
- Direct kernel object manipulation (DKOM)
- Privilege escalation
- Kernel execution vectors

Kernel Shellcoding

The objective of this section is to learn about developing kernel-mode shellcode using raw x64 assembler and high-level languages. It covers topics such as kernel-mode shellcode consideration and constraints, shellcoding tools (MASM/NASM/YASM), x64 assembler limitations and workarounds, developing kernel-mode shellcode in C/C++, leveraging the PE file .pdata section, shellcode injection and execution, reflective driver loading, methods for bypassing write protection in kernel-mode, calling kernel internal (un-exported) functions, accessing kernel internal global data structures, removing shellcode execution artifacts. System defenses such as driver signature enforcement (KMCS), kernel-mode DEP, and non-executable non-paged pool (NonPagedPoolNx) and are also covered.

- Kernel mode shellcode
- Shellcoding tools
- Shellcoding in C/C++
- PE exception table
- Calling non-exported functions
- Kernel Payload Loader
- Circumventing memory protection

Kernel Hooking and Injection

The objective of this section is to learn about code flow subversion techniques in the kernel and methods to inject and executed code into user-mode processes from kernel-mode. It covers topics such as function prolog/epilog hooking, trampolines, multi-processor synchronization, code caves, kernel function tables, function pointer hooking, stealth filtering through data structure redirection, user-mode code-injection, user-mode APCs, thread register context manipulation, user-mode callbacks, and hook detection methods. System defenses such as kernel control flow guard (KCFG), kernel patch protection (PatchGuard) are also covered.

- Code flow subversion methods
- Function hooking
- Function pointer hijack
- Import hooking
- Data structure hooking
- Code injection and execution

- Hook detection

Kernel Callbacks

The objective of this section is to learn about the different callback mechanisms available to kernel-mode drivers to intercept systemwide activity that is interesting from a security perspective. It covers topics such as usage of process callbacks to veto process creation, thread callbacks to detect remote thread injection, image notification callbacks to block driver loading, object manager callback to block process memory access, shutdown callbacks to implement self-protection, bug-check callbacks for anti-forensics, and power callbacks for user presence detection.

- Process callbacks
- Thread callbacks
- Image notification callbacks
- Object manager callbacks
- Shutdown notifications
- Bug-check callbacks
- Power notification callbacks

Kernel Filtering

The objective of this section is to learn about the different filtering mechanisms available to kernel-mode drivers to intercept device, file access, and registry access. It covers topics such as device stacks, filter drivers, usage of IRP filters for keylogging and disk access, filter registration, dynamic device attachment/detachment, registry filters, registry key context management, hiding registry entries, filter manager, FS mini-filters, context management, hiding directory entries, finding filter driver callbacks neutering filter callbacks.

- Filtering models
- IRP filters
- PnP hardware detection
- Stealth filtering
- Registry filters
- File system mini-filters

- Neutering filters

Kernel Networking

The objective of this section is to discuss the kernel-mode networking stack components, the interfaces to intercept networking activity in the system, and inject network data. It covers topics such as Windows networking architecture, kernel network interfaces, network packet data representation, NDIS drivers (miniport, intermediate, protocol, and filter), WFP architecture, Windows firewall, content inspection/modification of TCP streams, network layer-2 filtering, NDIS internal data structures and low-level network I/O.

- Kernel network interfaces
- Net buffer lists (NBL) and net buffers (NB)
- Windows filtering platform (WFP)
- WFP MAC layer filtering
- NDIS driver types
- NDIS lightweight filters (LWF)
- NDIS internal data structures and hooking

Virtualization Based Security

The objective of this section is to learn about virtualization-based security and its impact on rootkits. It covers topics such as Hyper-V platform requirements, VT-x/AMD-V, second-level address translation (SLAT), mode based execution control (MBEC), Hyper-V top-level functional specification (TLFS), virtual trust levels (VTL0/VTL1), normal kernel, secure kernel, secure kernel patch guard (SKPG) functionality, HVCI restrictions on kernel drivers, the effects of KCFG on code flow subversion, secure pools and the effects of KDP on DKOM.

- Hyper-V Architecture
- Virtual Trust Levels (VTL)
- Secure Kernel (SK)
- HyperGuard (SKPG)
- HyperVisor Protected Code Integrity (HVCI)
- Kernel Control Flow Graph (KCFG)
- Kernel Data Protection (KDP)