Advanced Windows Exploitation (OSEE)

- 5 Days
- Lecture and Hands-on Labs

Course Overview

The OSEE is the most difficult exploit development certification you can earn. We recommend completing the 300-level certifications before registering for this course. Students who complete EXP-401 and pass the exam will earn the Offensive Security Exploitation Expert (OSEE) certification. The OSEE exam assesses not only the course content, but also the ability to think laterally and adapt to new challenges. The virtual lab environment has a limited number of target systems. The software within contains specific, unknown vulnerabilities. Students have 72 hours to develop and document exploits. The exam requires a stable, high-speed internet connection. You must submit a comprehensive penetration test report as part of the exam. It should contain indepth notes and screenshots detailing the steps taken and the exploit methods used.

Who Should Attend

- Cyber Network Professional
- Sr. Pentester / Red Team Operator
- Information Security Penetration Tester
- Threat Hunter Cybersecurity
- Cybersecurity Exploit Developer

What You'll Learn

- Bypass and evasion of user mode security mitigations such as DEP, ASLR, CFG, ACG, and CET.
- Advanced heap manipulations to obtain code execution along with guest-to-host and sandbox escapes.
- Disarming WDEG mitigations and creating version independence for weaponization.
- 64-Bit Windows Kernel Driver reverse engineering and vulnerability discovery.
- Bypass of kernel mode security mitigations such as kASLR, NX, SMEP, SMAP, kCFG, and HVCI.

Outline

2. Custom Shellcode Creation

2.1 64-bit Architecture

- 2.1.1 64-bit Memory Enhancements
- 2.1.2 Calling Conventions
- 2.1.3 Win32 APIs

2.2 Writing Exploit Code

- 2.2.1 Position-Independent-Code
- 2.2.2 Visual Studio

2.3 Shellcode Framework Creation

- 2.3.1 Finding KERNEL32.DLL Base Address: PEB Method
- 2.3.2 Resolving Symbols: Export Directory Table Method
- 2.3.3 Fetching Function's VMA

2.4 Reverse Shell

- 2.4.1 Create a Connection
- 2.4.2 Launch the Shell

2.5 Wrapping Up

3. VMware Workstation Guest-To-Host Escape

3.1 Vulnerability Classes

3.2 Data Execution Prevention (DEP)

- 3.2.1 DEP Theory
- 3.2.2 Ret2Lib Attacks and Their Evolution
- 3.2.3 Return Oriented Programming
- 3.2.4 Locating Gadgets: rp++

3.3 Address Space Layout Randomization

3.4 VMware Workstation Internals

- 3.4.1 VMware Backdoor RPC Guest-to-Host Communication
- 3.4.2 Backdoor_InOut
- 3.4.3 Opening a RPC Communication Channel
- 3.4.4 Sending the Command Data
- 3.4.5 Receiving the Reply
- 3.4.6 Closing the RPC Communication Channel

3.5 UaF Case Study: VMware Workstation Drag & Drop Vulnerability

3.6 The Windows Heap Memory Manager

- 3.6.1 Front-End Allocator
- 3.6.2 Back-End Allocator

3.7 Low Fragmentation Heap

- 3.7.1 LFH Architecture
- 3.7.2 LFH Logic

3.8 UaF Case Study: Triggering the Bug

3.9 UaF Case Study: A Deeper Look at the Bug

3.10 UaF Case Study: Reallocation Control

- 3.10.1 guest.upgrader_send_cmd_line_args
- 3.10.2 The NULL Byte Issue

4.4 Going for RIP

- 4.4.1 Vanilla Attack
- 4.4.2 CFG Internals

4.5 CFG Bypass

- 4.5.1 Return Address Overwrite
- 4.5.2 Intel CET
- 4.5.3 Out-of-Context Calls

4.6 Data Only Attack

- 4.6.1 Parallel DLL Loading
- 4.6.2 Injecting Fake Work
- 4.6.3 Faking the Work
- 4.6.4 Hot Patching DLLs

4.7 Arbitrary Code Guard (ACG)

- 4.7.1 ACG Theory
- 4.7.2 ACG Bypasses

4.8 Advanced Out-of-Context Calls

- 4.8.1 Faking it to Make it
- 4.8.2 Fixing the Crash

4.9 Remote Procedure Calls

- 4.9.1 RPC Theory
- 4.9.2 Is That My Structure
- 4.9.3 Analyzing the Buffers
- 4.9.4 Calling an API
- 4.9.5 Return of Mitigations

4.10 Perfecting Out-of-Context Calls

- 4.10.1 Come Back to JavaScript
- 4.10.2 Return Value Alignment
- 4.10.3 Call Me Again

4.11 Combining the Work

- 4.11.1 NOP'ing CFG
- 4.11.2 Call Arbitrary API

4.12 Browser Sandbox

- 4.12.1 Sandbox Theory Introduction
- 4.12.2 Sandbox Escape Theory
- 4.12.3 The Glue That Binds

4.13 Sandbox Escape Practice

- 4.13.1 Insecure Access
- 4.13.2 The Problem of Languages

6.7 Executing Code in Kernel-Mode

- 6.7.1 Leaking Nt and Win32k Base
- 6.7.2 NOP-ing kCFG
- 6.7.3 Hijacking a Kernel-Mode Routine

6.8 Wrapping Up

Prerequisites

- Learners should be experienced in developing windows exploits and understand how to operate a debugger.
- Familiarity with WinDBG, x86_64 assembly, IDA Pro and basic C/C++ programming is highly recommended.
- A willingness to work and put in real effort will greatly help students succeed in this security training course.