

Advanced Windows Exploitation

Offensive Security



Copyright © 2021 Offensive Security Ltd.

All rights reserved. No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright owner, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, any broadcast for distant learning, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission from the author.

Table of Contents

- 1 Introduction
- 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
- 3.11 UaF Case Study: Fake Virtual Table
- 3.12 UaF Case Study: ROP Storage
 - 3.12.1 `unity.window.contents.start` : Locating the Function
 - 3.12.2 `unity.window.contents.start` : Arg Processing
 - 3.12.3 `unity.window.contents.chunk` : Expanded Data Storage
- 3.13 UaF Case Study: Bypassing ASLR
 - 3.13.1 Hunting for Pointers
- 3.14 UaF Case Study: Stack Pivoting
- 3.15 UaF Case Study: Defeating DEP
 - 3.15.1 `GetModuleHandle` ROP Chain
 - 3.15.2 `GetProcAddress` ROP Chain
 - 3.15.3 `WriteProcessMemory` ROP Chain
- 3.16 Restoring the Execution Flow
- 3.17 Executing Shellcode
- 3.18 Windows Defender Exploit Guard
- 3.19 Testing the WDEG Protections
 - 3.19.1 The Ghost of ASLR Returns
- 3.20 ROP Mitigations
 - 3.20.1 Disarming WDEG: Theory
 - 3.20.2 Disabling WDEG: Practice
 - 3.20.3 Defeating EAF
- 3.21 Wrapping Up
- 4 Microsoft Edge Type Confusion

- 4.1 Edge Internals
 - 4.1.1 JavaScript Engine
 - 4.1.2 Chakra Internals
 - 4.1.3 JIT and Type Confusion
- 4.2 Type Confusion Case Study
 - 4.2.1 Triggering the Vulnerability
 - 4.2.2 Root Cause Analysis
- 4.3 Exploiting Type Confusion
 - 4.3.1 Controlling the auxSlots Pointer
 - 4.3.2 Abuse AuxSlots Pointer
 - 4.3.3 Create Read and Write Primitive
- 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
- 4.14 The Great Escape
 - 4.14.1 Activation Factory
 - 4.14.2 GetTemplateContent
 - 4.14.3 What Is As?
 - 4.14.4 Loading the XML
 - 4.14.5 Allowing Scripts
 - 4.14.6 Pop That Notepad
 - 4.14.7 Getting a Shell
- 4.15 Upping The Game - Making the Exploit Version Independent
 - 4.15.1 Locating the Base
 - 4.15.2 Locating Internal Functions and Imports
 - 4.15.3 Locating Exported Functions
- 4.16 Wrapping Up
- 5 Driver Callback Overwrite
 - 5.1 The Windows Kernel
 - 5.1.1 Privilege Levels
 - 5.1.2 Interrupt Request Level (IRQL)
 - 5.2 Kernel-Mode Debugging on Windows
 - 5.2.1 Remote Kernel Debugging Over TCP/IP

- 5.2.2 Remote Kernel Debugging Over Serial Ports
- 5.2.3 Local Kernel Debugging Through VMware (VirtualKD)
- 5.3 Communicating with the Kernel
 - 5.3.1 Native System Calls
 - 5.3.2 Device Drivers
- 5.4 Windows Kernel Security Mitigations
- 5.5 Vulnerability Classes
- 5.6 Kernel-Mode Shellcode
 - 5.6.1 Token Stealing
 - 5.6.2 ACL NULL-ing / Editing
 - 5.6.3 Rookits
- 5.7 Vulnerability Overview and Exploitation
 - 5.7.1 Triggering the Vulnerability
 - 5.7.2 Controlling the Callback Context
 - 5.7.3 Redirecting Execution to Usermode
 - 5.7.4 SMEP Says Hello
 - 5.7.5 Introduction to Memory Paging and Structures
 - 5.7.6 The PML4 Self-Reference Entry
 - 5.7.7 PML4 Self-Reference Entry Randomization
- 5.8 ROP-Based Attack
 - 5.8.1 Stack Pivoting
 - 5.8.2 Kernel Read/Write Primitive
 - 5.8.3 Restoring the Execution Flow
 - 5.8.4 Leaking Virtual PTE Start
 - 5.8.5 Flipping U/S Bit
 - 5.8.6 Meltdown and KVA Shadow
 - 5.8.7 Flipping the PML4 EXB Bit
 - 5.8.8 Token Stealing
- 5.9 Version Independence
 - 5.9.1 Dynamic Gadget Location
- 5.10 Wrapping Up
- 6 Unsanitized User-mode Callback
 - 6.1 Windows Desktop Applications
 - 6.1.1 Windows Kernel Pool Memory

- 6.1.2 Creating Windows Desktop Applications
- 6.1.3 Reversing the TagWND Object
- 6.1.4 Kernel User-mode Callbacks
- 6.1.5 Leaking pWND User-Mode Objects
- 6.2 Triggering the Vulnerability
 - 6.2.1 Spraying the Desktop Heap
 - 6.2.2 Hooking the Callback
 - 6.2.3 Arbitrary WndExtra Overwrite
- 6.3 TagWND Write Primitive
 - 6.3.1 Overwrite pWND[0].cbWndExtra
 - 6.3.2 Overwrite pWND[1].WndExtra
- 6.4 TagWND Leak and Read Primitive
 - 6.4.1 Changing pWND[1].dwStyle
 - 6.4.2 Setting The TagWND[1].spmenu
 - 6.4.3 Creating a fake TagWND[1].spmenu
 - 6.4.4 GetMenuBarInfo Read Primitive
- 6.5 Privilege Escalation
 - 6.5.1 Low integrity
- 6.6 Virtualization-Based Security
 - 6.6.1 Windows Hypervisor Theory
 - 6.6.2 Windows Hypervisor Debugging
 - 6.6.3 Data Only Attack
 - 6.6.4 Restoring The Execution Flow
- 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
 - 6.7.4 Wrapping Up