



DESDE LAS PROFUNDIDADES DEL KERNEL: CÓMO CREAR UN ROOTKIT INVISIBLE EN WINDOWS

Descifrando el desarrollo de un Rootkit para Windows 11

[in/vazquez-vazquez-alejandro]

ViCONgal 2025, Galicia





WHOAMI

- **FRIKI** (**F**anático de **R**evolucionar **I**nternamente **K**ernels e **I**nicios del sistema)
- Pastor de ovejas desde los 8 años
- Me gusta el pulpo, de ahí los Rootkits
- Docente en Máster de Análisis de Malware



[[in/vazquez-vazquez-alejandro](https://twitter.com/vazquez-vazquez-alejandro)]



TABLE OF CONTENTS

Rootkit Development /Rooted[®]CON

- Concepts
 - Kernel
 - Rootkit
 - Security Mechanisms
- Checkpoint
- Development Environment
 - Script
 - Kernel Mode Driver
 - Our Malicious Driver
- Demo Time
- Development
 - Communication, Keylogger
 - Hide Processes, Hide Folders
 - Network Control, Network Requests
- Infection
- B/Rootkits in the Wild
 - FudModule
 - Fire Chili
 - SPEcter
- Demo Time



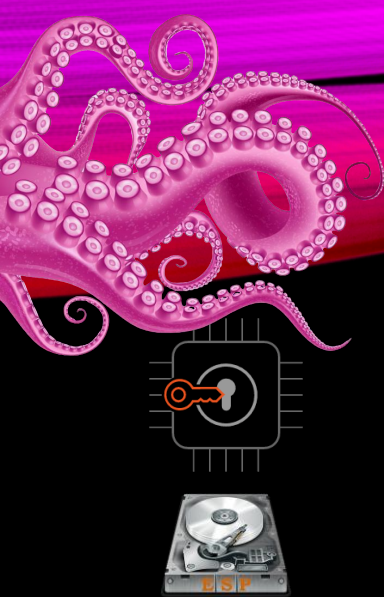
UEFI BOOTKIT DEVELOPMENT



UEFI BOOTKIT DEVELOPMENT

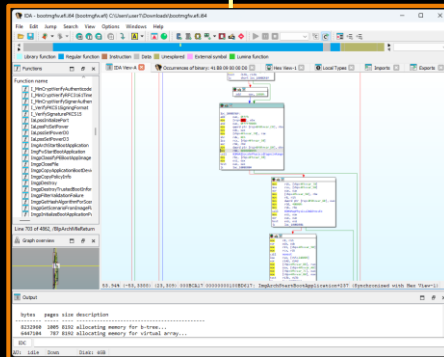
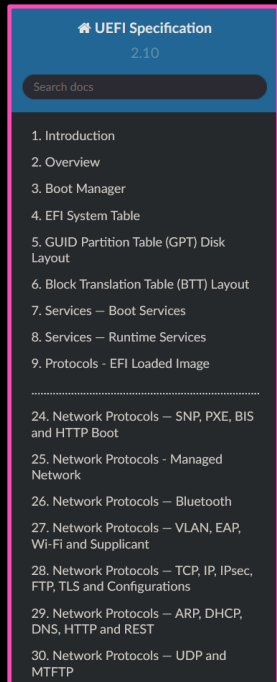
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi

UEFI Specification
2.10
Search docs
1. Introduction
2. Overview
3. Boot Manager
4. EFI System Table
5. GUID Partition Table (GPT) Disk Layout
6. Block Translation Table (BTT) Layout
7. Services — Boot Services
8. Services — Runtime Services
9. Protocols - EFI Loaded Image
24. Network Protocols — SNP, PXE, BIS and HTTP Boot
25. Network Protocols - Managed Network
26. Network Protocols — Bluetooth
27. Network Protocols — VLAN, EAP, Wi-Fi and Supplicant
28. Network Protocols — TCP, IP, IPsec, FTP, TLS and Configurations
29. Network Protocols — ARP, DHCP, DNS, HTTP and REST
30. Network Protocols — UDP and MFTP

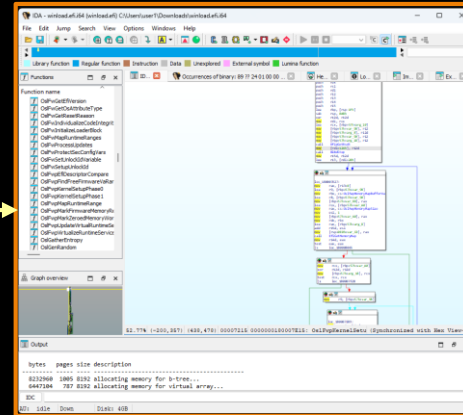


UEFI BOOTKIT DEVELOPMENT

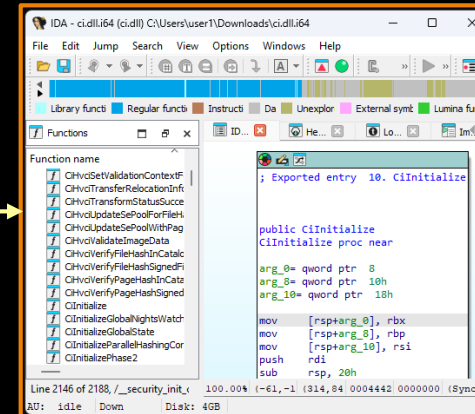
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



Windows Boot Manager
\\EFI\\Microsoft\\Boot\\
bootmgfw.efi



Windows OS Loader
%SystemRoot%\system32\
winload.efi



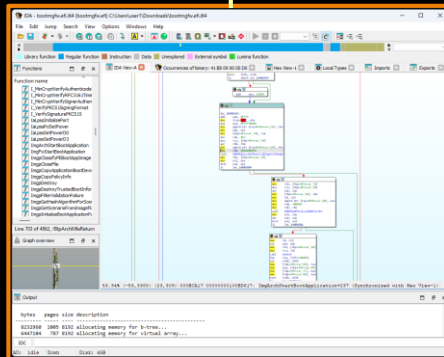
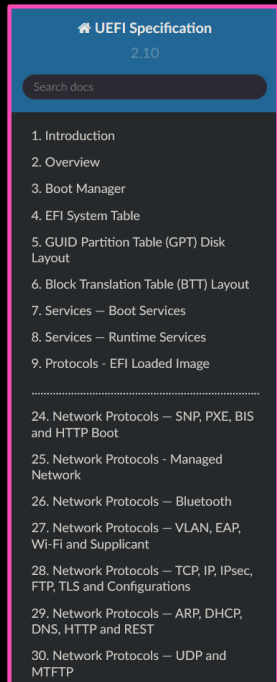
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe



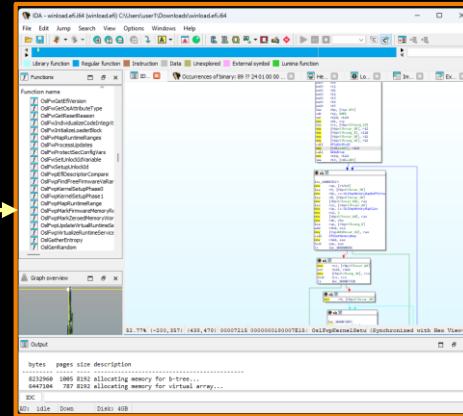
UEFI BOOTKIT DEVELOPMENT



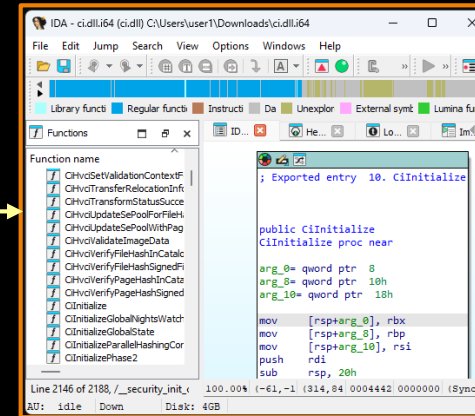
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



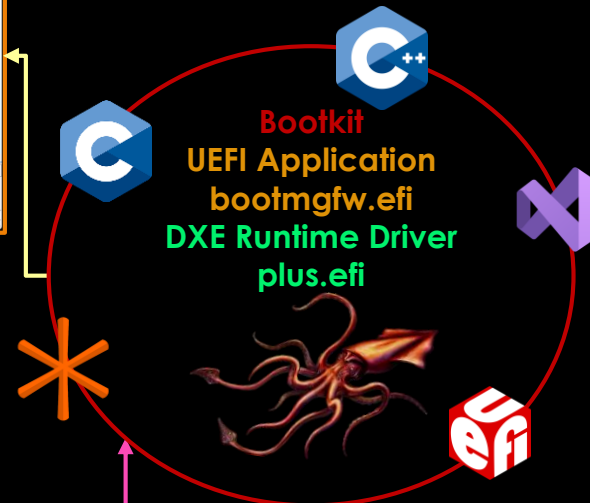
Windows Boot Manager
\\EFI\\Microsoft\\Boot\\
bootmgfw.efi



Windows OS Loader
%SystemRoot%\system32\
winload.efi



Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe

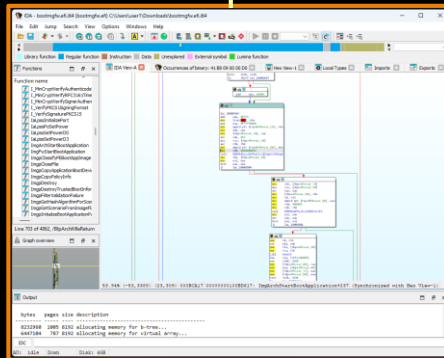
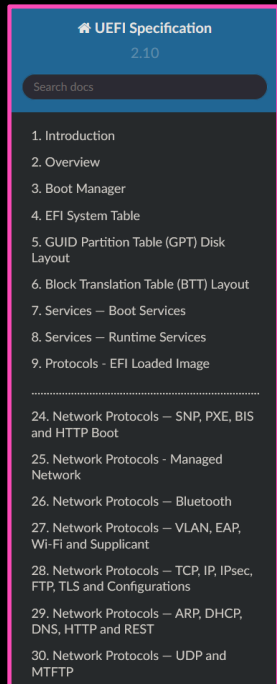




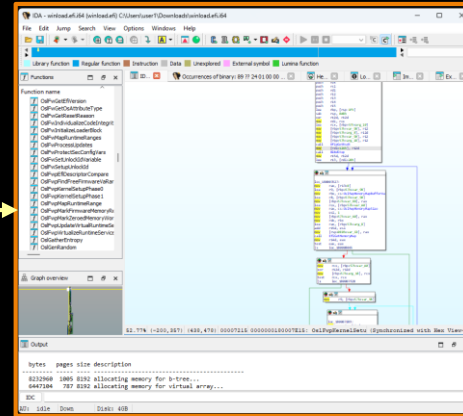
UEFI BOOTKIT DEVELOPMENT



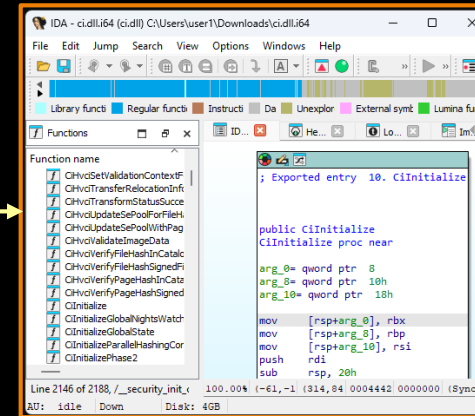
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



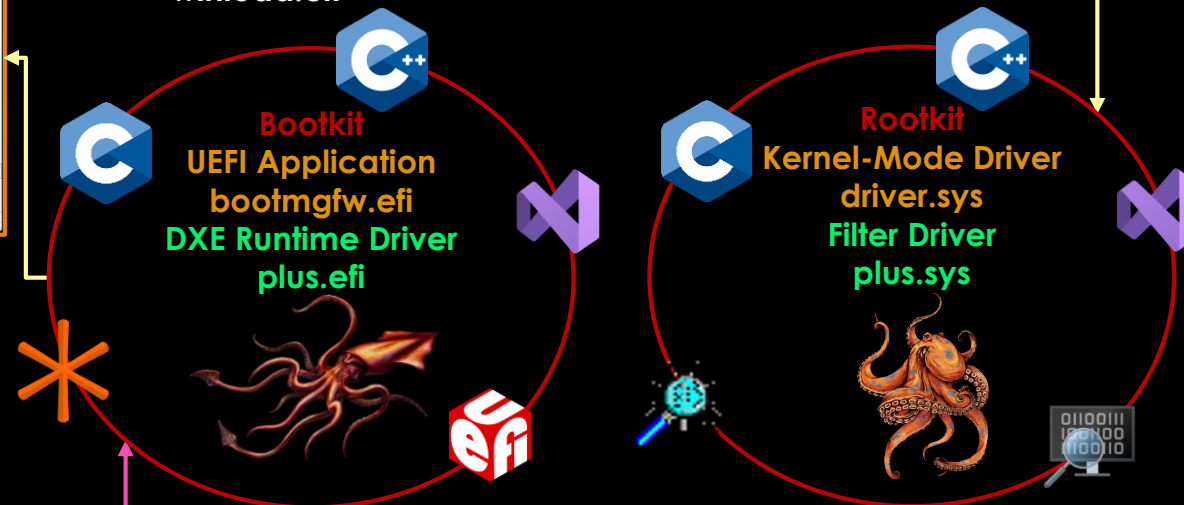
Windows Boot Manager
\\EFI\\Microsoft\\Boot\\
bootmgfw.efi



Windows OS Loader
%SystemRoot%\system32\\
winload.efi



Windows NT OS Kernel
%SystemRoot%\system32\\
ntoskrnl.exe



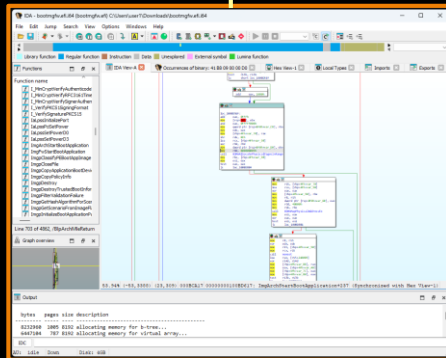


UEFI BOOTKIT DEVELOPMENT

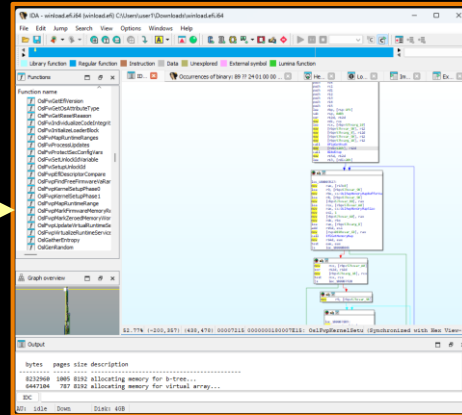
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



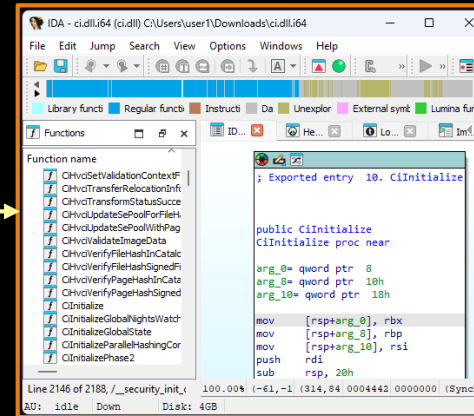
UEFI Specification
2.10
Search docs
1. Introduction
2. Overview
3. Boot Manager
4. EFI System Table
5. GUID Partition Table (GPT) Disk Layout
6. Block Translation Table (BTT) Layout
7. Services – Boot Services
8. Services – Runtime Services
9. Protocols – EFI Loaded Image
24. Network Protocols – SNP, PXE, BIS and HTTP Boot
25. Network Protocols - Managed Network
26. Network Protocols – Bluetooth
27. Network Protocols – VLAN, EAP, Wi-Fi and Supplicant
28. Network Protocols – TCP, IP, IPsec, FTP, TLS and Configurations
29. Network Protocols – ARP, DHCP, DNS, HTTP and REST
30. Network Protocols – UDP and MTFTP



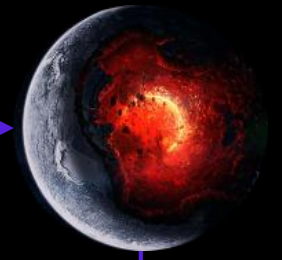
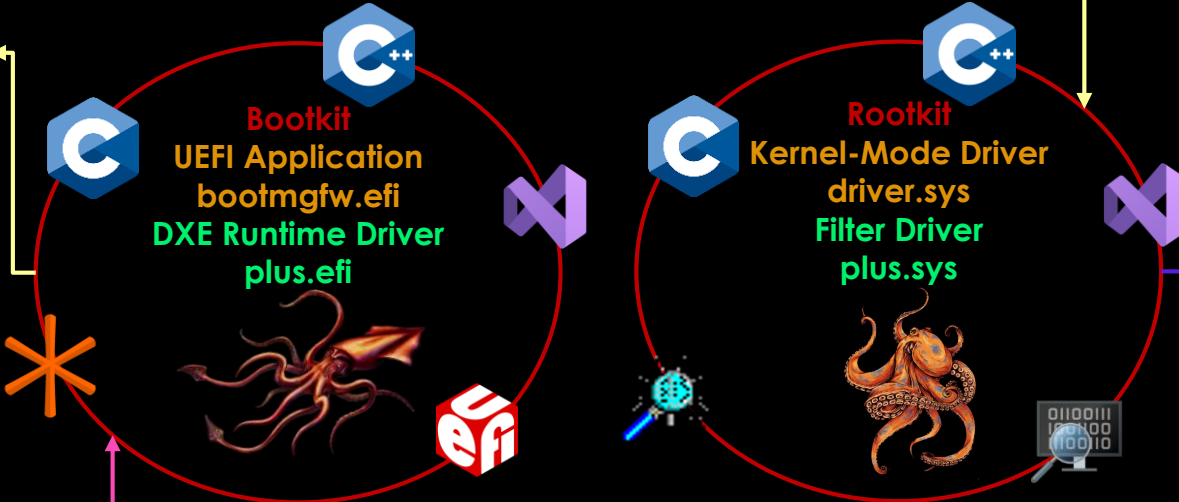
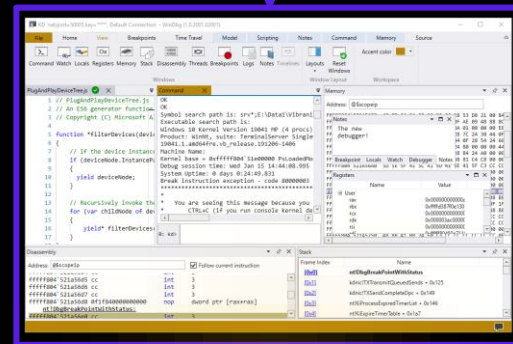
Windows Boot Manager
\\EFI\\Microsoft\\Boot\\
bootmgfw.efi



Windows OS Loader
%SystemRoot%\system32\\
winload.efi



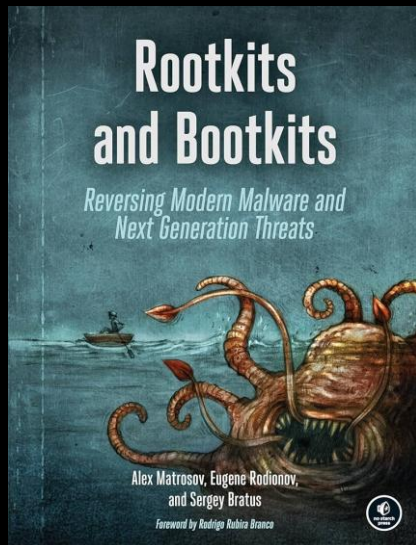
Windows NT OS Kernel
%SystemRoot%\system32\\
ntoskrnl.exe



ROOTKIT

Rootkit: Sophisticated piece of malware that can add new code to the operating system or delete and edit operating system code. Rootkits may remain in place for years because they are hard to detect, due in part to their ability to block some antivirus software and malware scanner software.

~ CrowdStrike



Kernel-Mode Driver

C/C++ - driver.sys



SECURITY MECHANISMS

[Anti-Rootkit Installation]

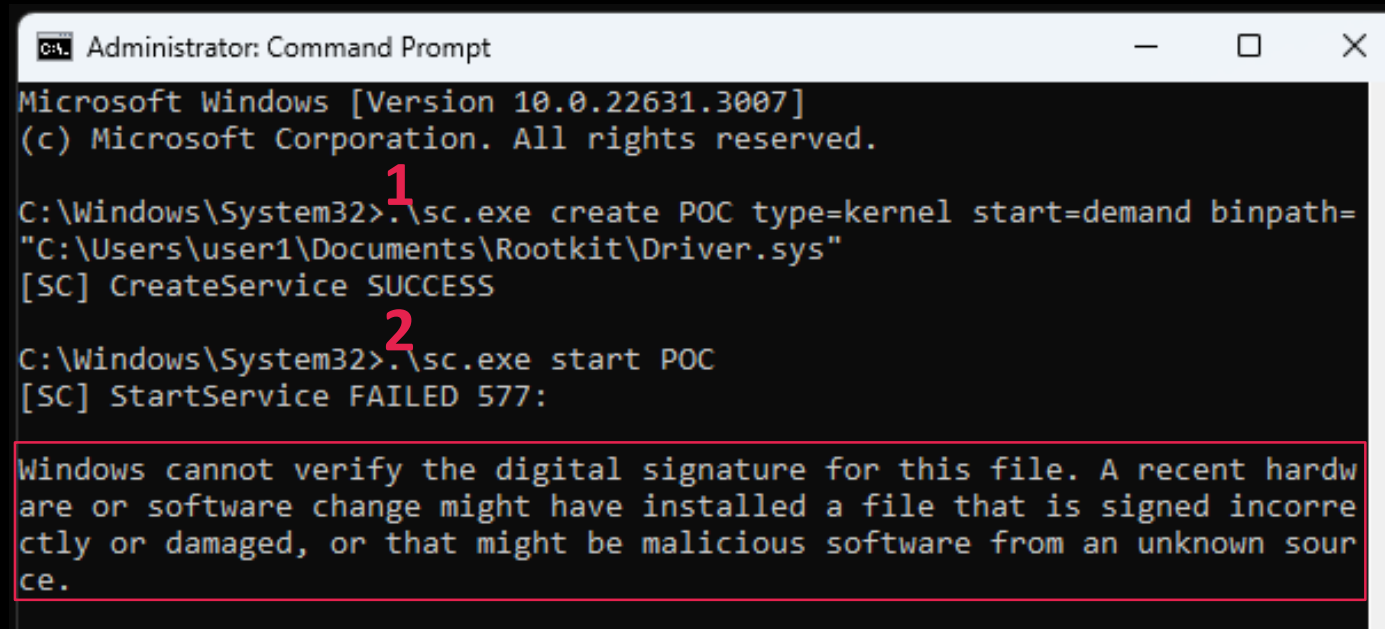
- Driver Signature Enforcement (DSE)
Windows won't run drivers not certified by Microsoft

SECURITY MECHANISMS

[Anti-Rootkit Installation]

- Driver Signature Enforcement (DSE)

Windows won't run drivers not certified by Microsoft



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>1 .\sc.exe create POC type=kernel start=demand binpath=
"C:\Users\user1\Documents\Rootkit\Driver.sys"
[SC] CreateService SUCCESS

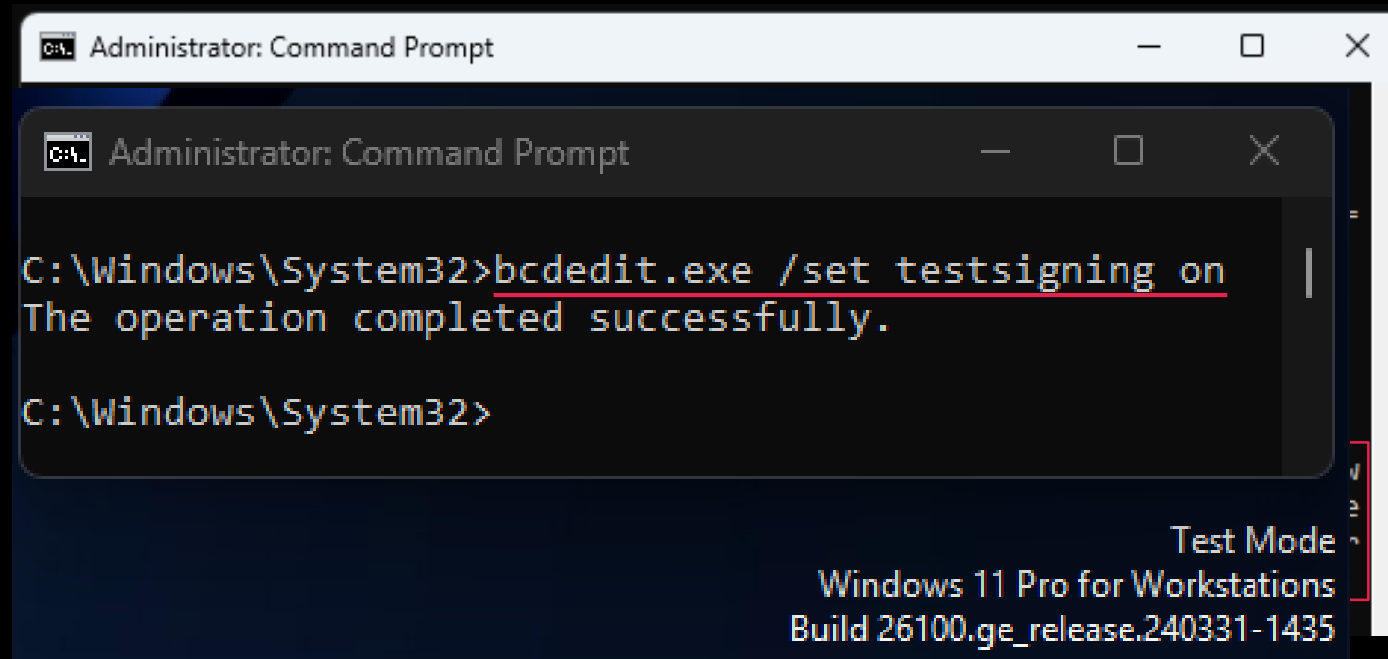
C:\Windows\System32>2 .\sc.exe start POC
[SC] StartService FAILED 577:

Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a file that is signed incorrectly or damaged, or that might be malicious software from an unknown source.
```

SECURITY MECHANISMS

[*Anti-Rootkit Installation*]

- Driver Signature Enforcement (DSE)
Windows won't run drivers not certified by Microsoft



```
Administrator: Command Prompt

Administrator: Command Prompt

C:\Windows\System32>bcdedit.exe /set testsigning on
The operation completed successfully.

C:\Windows\System32>
```

Test Mode
Windows 11 Pro for Workstations
Build 26100.ge_release.240331-1435

DEVELOPMENT ENVIRONMENT



how to develop a Windows kernel mode driver



Learn / Windows / Windows Drivers /

Tutorial: Write a Hello World Windows Driver (Kernel-Mode Driver Framework)

Article • 12/19/2024 • 9 contributors

In this article

- Prerequisites
- Create and build a driver
- Write your first driver code
- Build the driver
- Deploy the driver
- Install the driver
- Debug the driver
- Related articles

```
Windows PowerShell
=====
Overview:
- PowerShell Script for Automating Bootkits/Rootkits Development Environment Setup in Windows
Note:
- All options have been tested on the latest version of Windows 11 24H2
LinkedIn:
- https://www.linkedin.com/in/vazquez-vazquez-alejandro/
Github:
- https://github.com/TheMalwareGuardian/
=====

----- MENU -----
BOOTKITS
1a. Bootkits - Requirements -> Visual Studio 2019 Community + Git + Python + NASM + ASL
1b. Bootkits - Set Up Environment -> EDK2
1c. Bootkits - Tools -> UEFITool
1d. Bootkits - PoCs -> UEFI Applications + DXE Runtime Drivers

DEBUGGING
2a. Debugging - Requirements -> WinDbg
2b. Debugging - Set Up Environment -> Enable Debugging
2c. Debugging - Tools -> Microsoft Sysinternals Suite + Process Hacker
2d. Debugging - Scripting -> PoCs - WinDbg Classic + JavaScript + Python PYKD + WinDbg Extensions
2e. Debugging - Debugging Diagram -> Host (Debugger) + Target (Debuggee)

ROOTKITS
3a. Rootkits - Requirements -> Visual Studio 2022 Community + SDK + WDK + Visual Studio Code
3b. Rootkits - Set Up Environment -> Enable Test Mode + Disable Integrity Checks
3c. Rootkits - Tools -> OSR Driver Loader + Ghidra + IDA Free
3d. Rootkits - PoCs -> Kernel Mode Drivers & Console Applications

RESOURCES
4a. Resources - My Repositories -> A compilation of resources dedicated to bootkit and rootkit development

PROGRAM TERMINATION
Q. Exit

Choose an option: 1a
You have selected the option 'Bootkits - Requirements -> Visual Studio 2019 Community + Git + Python + NASM + ASL'
Do you want to proceed? (Press 'Y'):
```



KERNEL MODE DRIVER

```
Administrator: Command Prompt
C:\Windows\System32>sc.exe create FirstDriver type= kernel binPath= "C:\Users\TheMalwareGuardian\Documents\Development\KMDFDriver1.sys"
[SC] CreateService SUCCESS

C:\Windows\System32>sc.exe start FirstDriver

SERVICE_NAME: FirstDriver
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 4  RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0  (0x0)
        SERVICE_EXIT_CODE   : 0  (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
        PID                 : 0
        FLAGS                 :

C:\Windows\System32>sc.exe stop FirstDriver

SERVICE_NAME: FirstDriver
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 1  STOPPED
        WIN32_EXIT_CODE       : 0  (0x0)
        SERVICE_EXIT_CODE   : 0  (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

C:\Windows\System32>sc.exe delete FirstDriver
[SC] DeleteService SUCCESS

C:\Windows\System32>
```

DebugView on \\MALWARE (local)

#	Time	Debug Print
1	0.00000000	Hello World
2	0.00000220	Set DriverUnload routine
3	0.00000320	Bye
4	1.39641595	Unload routine invoked
5	1.39641953	Bye

Visual Studio Code interface showing the KMDf Driver1 project.

Driver.c

```
#include <ntddk.h>

VOID
DriverUnload(
_In_ PDRIVER_OBJECT pDriverObject
)
{
    UNREFERENCED_PARAMETER(pDriverObject);

    DbgPrint("Unload routine invoked");

    DbgPrint("Bye");
}

NTSTATUS
DriverEntry(
_In_ PDRIVER_OBJECT pDriverObject,
_In_ PUNICODE_STRING pRegistryPath
)
{
    UNREFERENCED_PARAMETER(pRegistryPath);

    DbgPrint("Hello World");

    DbgPrint("Set DriverUnload routine");
    pDriverObject->DriverUnload = DriverUnload;

    DbgPrint("Bye");

    return STATUS_SUCCESS;
}
```

Output

```
1>Driver.c
1>KMDF Driver1.vcxproj -> C:\Users\TheMalwareGuardian\source\repos\KMDF Driver1\x64\Debug\KMDFDriver1.sys
1>Done Adding Additional Store
1>Successfully signed: C:\Users\TheMalwareGuardian\source\repos\KMDF Driver1\x64\Debug\KMDFDriver1.sys
1>
1>Driver is 'Universal'.
```

Create a new project

Recent project templates

- Kernel Mode Driver, Empty (KMDF)
- Kernel Mode Driver, USB (KMDF)
- Kernel Mode Driver (KMDF)

Not finding what you're looking for? Install more tools and features

Back Next



OUR MALICIOUS DRIVER

Rootkit Development

1. User Mode - Kernel Mode Communication

ntddk.h

Toolkit

Communication

2. Direct Kernel Object Modification

ntddk.h

Hide Processes

DKOM

3. Keyboard and Mouse Filter

ntddk.h

Keylogger

Keyboard Filter

4. Windows Filtering Platform

fwpmk.h, fwpsk.h, fwpmu.h

Network Control

WFP

5. WinSock Kernel

wsk.h

Network Requests

WSK

6. File System Minifilter Driver

fltKernel.h

Hide Folders

Minifilter

THE GATEWAY



Rootkit Installation
Kernel Mode Driver ?

THE GATEWAY



Rootkit Installation Kernel Mode Driver ?

1.UEFI Bootkit

Infect a Computer's Boot process
SecureBoot
Physical Access



2. Vulnerable Kernel Driver

(BYOVD) *Not Well Known*
Bring Your Own Vulnerable Driver

Microsoft Vulnerable Driver Blocklist

Microsoft blocks drivers with security vulnerabilities from running on your device.

☐ On





B / ROOTKIS IN THE WILD

2022

MoonBounce



CosmicStrand



Fire Chili



2024

Bootkitty



Fudmodule



2021

ESpecter



FinSpy



Moriya



2023

BlackLotus



DEMO



LOADING...

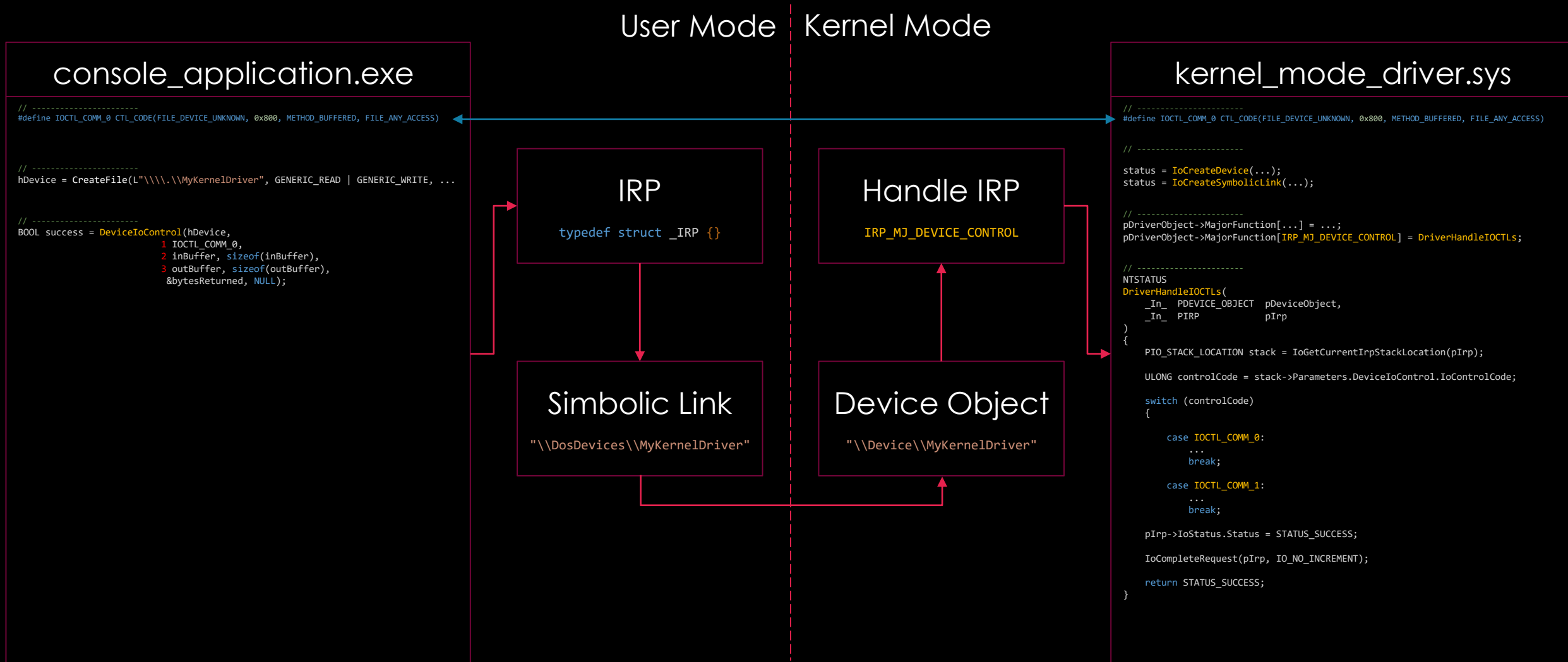


COMMUNICATION

“The bridge between user mode and kernel mode: IOCTL requests initiate communication, while IRPs manage data exchange and driver actions.”

- ✓ Via Input/Output Control Codes and Input/Output Request Packets
- ✗ Via Filter Communication Ports
- ✗ Via Network Requests
- ✗ Via Shared Memory
- ✗ Via Registry Keys
- ✗ Via Files
- ✗ Via ...

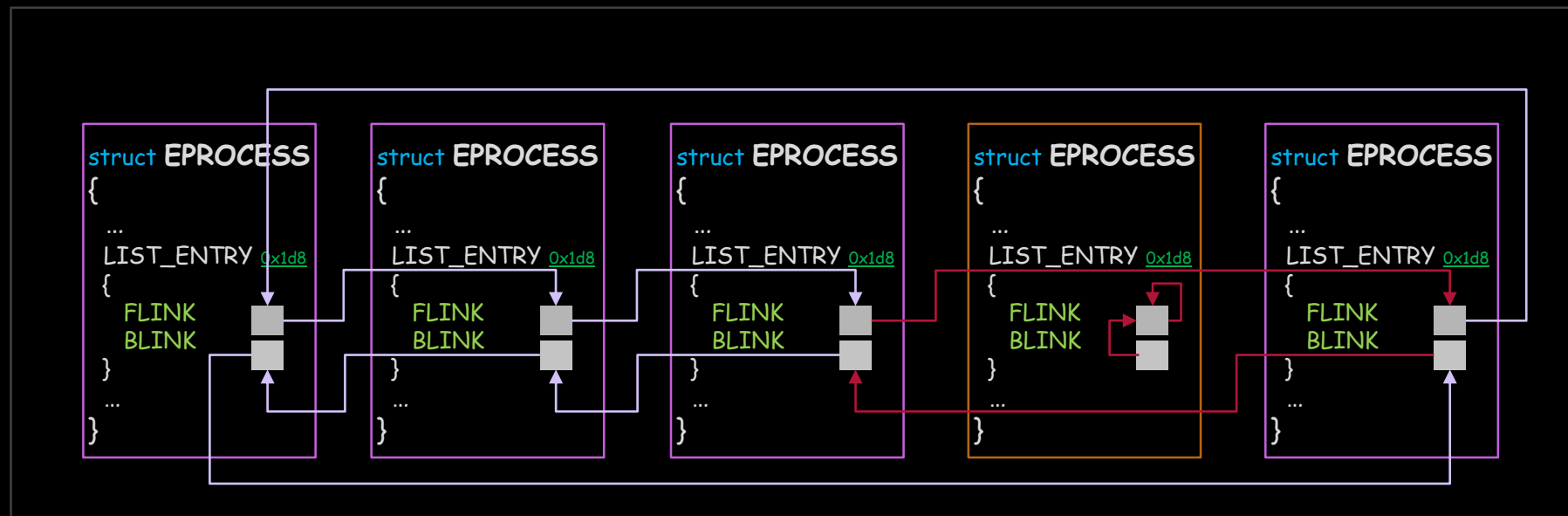
COMMUNICATION





HIDE PROCESSES

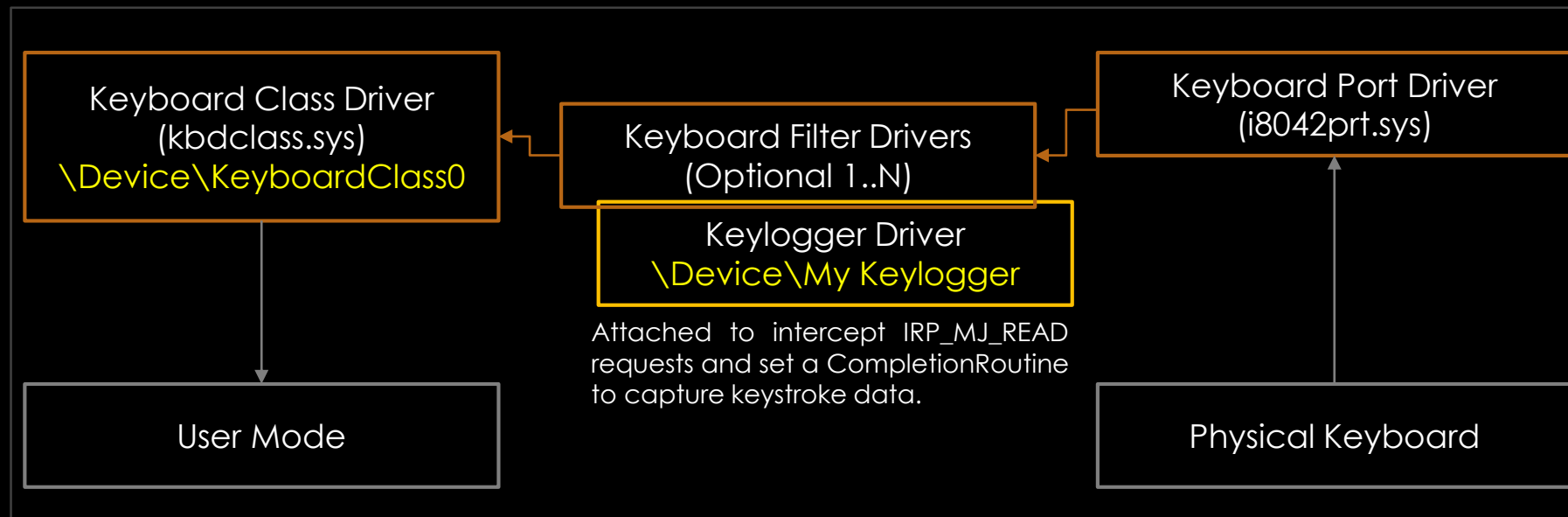
“Windows maintains a doubly linked list of active processes in (LIST_ENTRY) EPROCESS->ActiveProcessLinks. Unlink a process from the chain, and it disappears from user-mode enumeration.”





KEYLOGGER

“Keystroke interception in kernel mode: The Windows keyboard driver stack routes all keystrokes through a device object called `\Device\KeyboardClass0`. By attaching a driver to this device and registering a `CompletionRoutine`, we can capture raw keystroke data before it propagates to user-mode applications like text editors or browsers.”

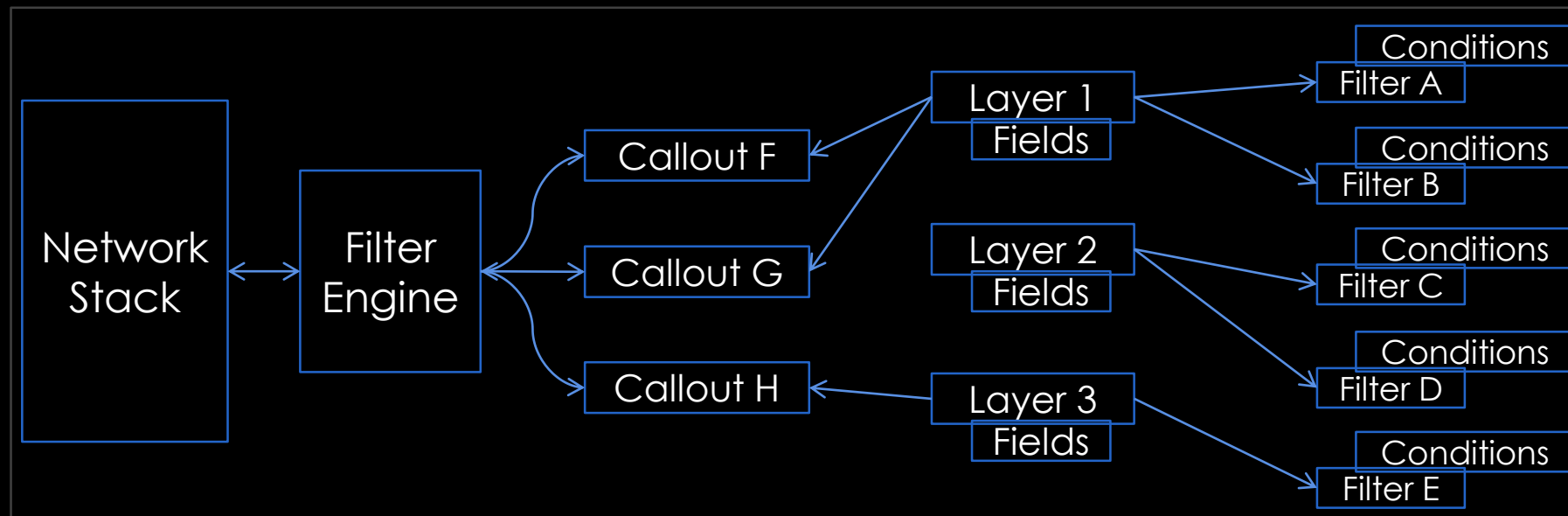




IV

NETWORK CONTROL

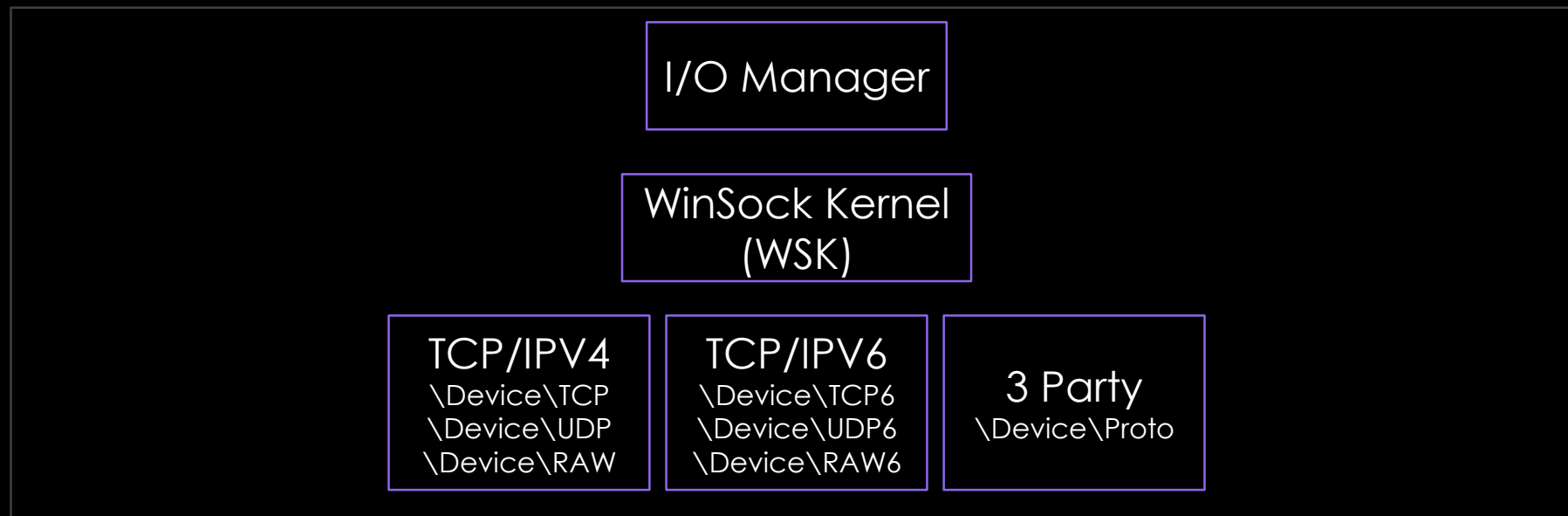
“Windows Filtering Platform (WFP) allows real-time inspection and control of network connections. By attaching filters (static rules applied at specific layers of the network stack to identify traffic based on attributes like IPs or ports) and callouts (custom drivers that execute dynamic logic on flagged traffic), it's possible to classify traffic based on metadata such as the remote IP address and the associated process. Traffic that matches specific rules can be blocked, logged, or modified, enabling comprehensive network security policies.”





NETWORK REQUESTS

“WinSock Kernel (WSK) allows kernel-mode programs to perform complex network operations, such as establishing connections, binding sockets, and transferring data. With support for asynchronous communication using IRPs, WSK enables efficient and controlled interaction with network protocols, ensuring low-latency communication and making it a robust solution for implementing kernel-level networking features.”





VI

HIDE FOLDERS

“MiniFilters attach to the file system stack to filter I/O operations.”

```
Administrator: Command Prompt

C:\Windows\System32>fltmc.exe

Filter Name                Num Instances  Altitude  Frame
-----
bindflt                    1              409800    0
UCPD                       4              385250.5  0
WdFilter                   4              328010    0
storqosflt                 0              244000    0
wcifs                     0              189900    0
CldFilt                   0              180451    0
bfs                       6              150000    0
FileCrypt                 0              141100    0
luafv                     1              135000    0
UnionFS                   0              130850    0
npsvcctrl                 1              46000     0
Wof                       2              40700     0
FileInfo                  4              40500     0

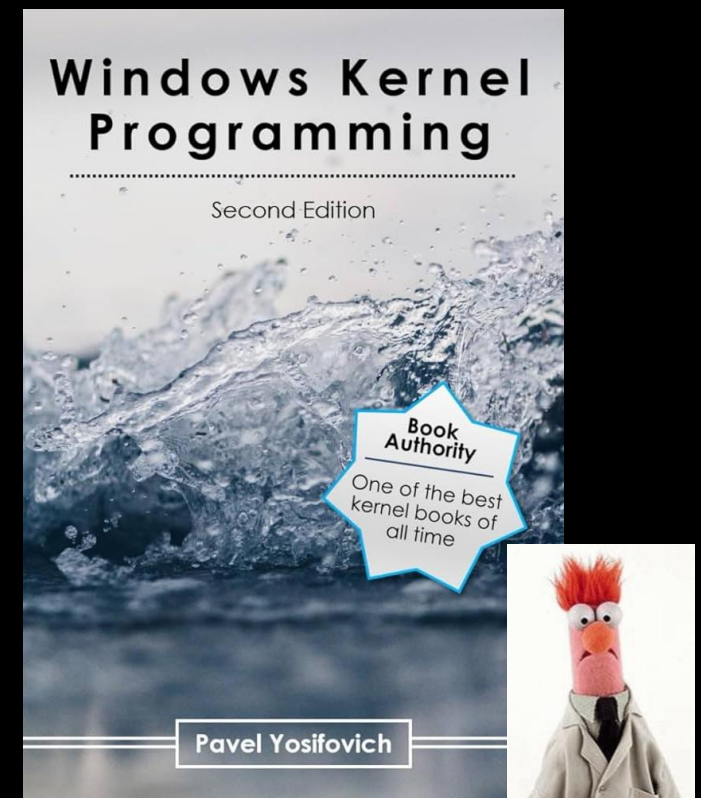
Administrator: Command Prompt

C:\Users\TheMalwareGuardian\Downloads>sc.exe create WindowsKernelMinifilter type=filesystem
start=demand binpath="C:\Users\%USERNAME%\Downloads\KMDfDriver_Minifilter.sys"
[SC] CreateService SUCCESS

C:\Users\TheMalwareGuardian\Downloads>ConsoleApp_Installation.exe
Everything is set up for service WindowsKernelMinifilter

C:\Users\TheMalwareGuardian\Downloads>fltmc load WindowsKernelMinifilter

C:\Users\TheMalwareGuardian\Downloads>
```





OUR MALICIOUS DRIVER

Rootkit Development

1. User Mode - Kernel Mode Communication

ntddk.h

Toolkit

Communication

2. Direct Kernel Object Modification

ntddk.h

Hide Processes

DKOM

3. Keyboard and Mouse Filter

ntddk.h

Keylogger

Keyboard Filter

4. Windows Filtering Platform

fwpmk.h, fwpsk.h, fwpmu.h

Network Control

WFP

5. Windows Kernel Sockets

wsk.h

Network Requests

WSK

6. File System Minifilter Driver

fltKernel.h

Hide Folders

Minifilter



REVERSE ENGINEERING / MALWARE ANALYSIS / BUG HUNTING

Module 10 – Windows Reverse Engineering

- Windows architecture (User mode and Kernel mode)
- Windows protections (DSE, KPP, VBS, CFG)
- Malware hunting with SysInternals tools
- Windows kernel opaque structures (EPROCESS, ETHREAD)
- Windows kernel debugging
- WinDbg scripting (Commands, Javascript, PyKd)
- Rootkit hooking techniques (IDT, SSDT)
- Rootkit development (Kernel Mode Drivers)
- Bootkit development (UEFI Applications)
- Bootkit analysis (ESpecter, BlackLotus)
- Kernel exploitation (Vulnerable drivers, Write-What-Where)





THANK YOU 😊

Rootkits PoCs & ViCONgal 2025 PPT:

github.com/TheMalwareGuardian/Bentico

Every resource you need to develop Rootkits:

github.com/TheMalwareGuardian/Awesome-Bootkits-Rootkits-Development

Automate Bootkits/Rootkits Development

github.com/TheMalwareGuardian/Bootkits-Rootkits-Development-Environment

Contact:

www.linkedin.com/in/vazquez-vazquez-alejandro

Agradecimientos:

