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)

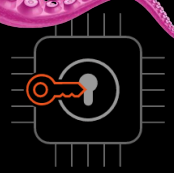
- Pastor de ovejas desde los 8 años
- Me gusta el pulpo, de ahí los Rootkits
- Docente en Máster de Análisis de Malware

[in/vazquez-vazquez-alejandro]

TABLE OF CONTENTS

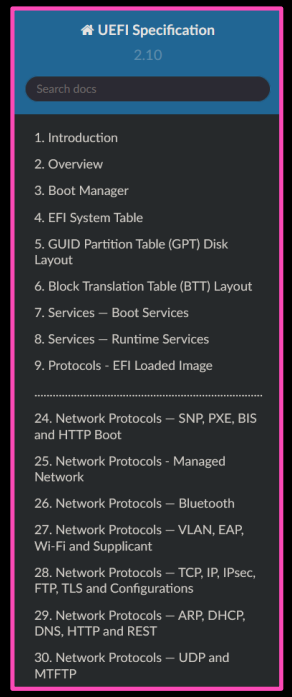
Rootkit Development /RootedCON

UEFI BOOTKIT DEVELOPMENT



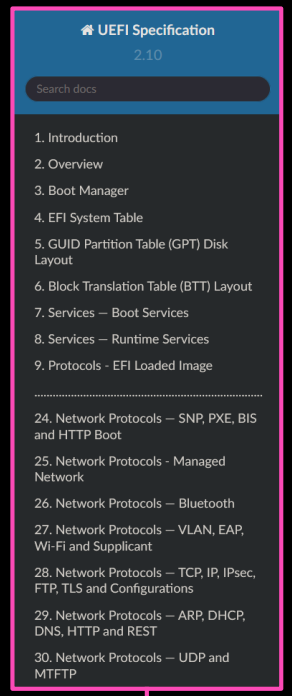
UEFI BOOTKIT DEVELOPMENT

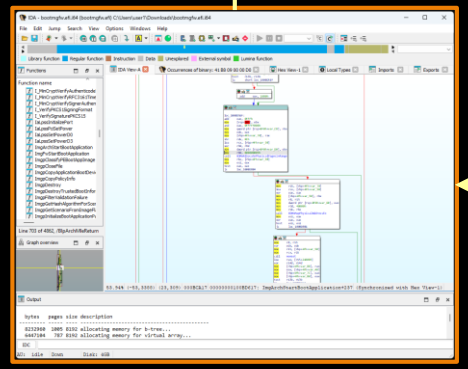
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



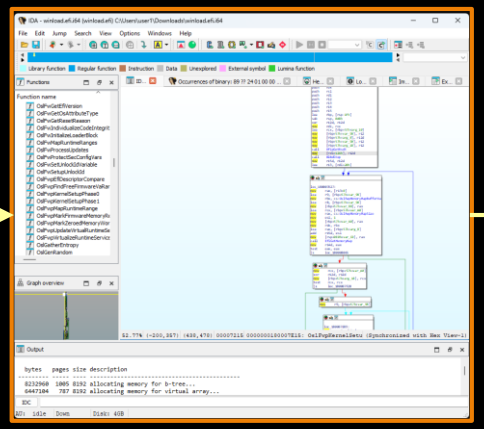
UEFI BOOTKIT DEVELOPMENT

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

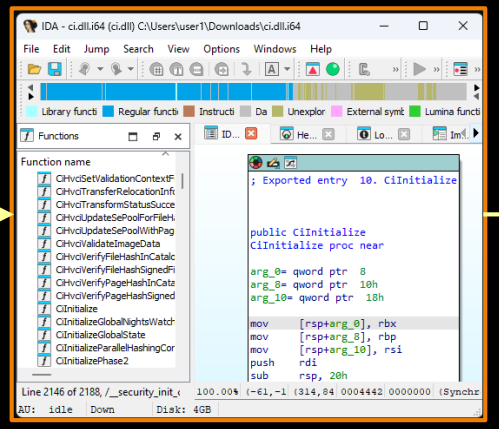




Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi



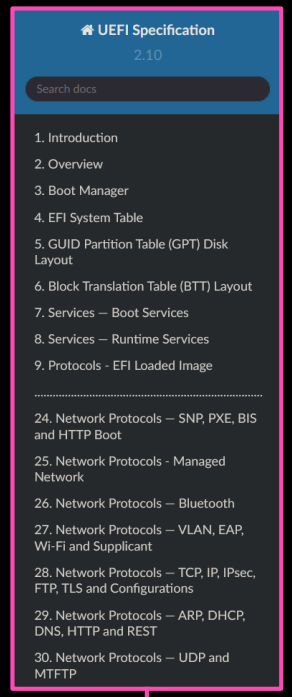
Windows OS Loader
%SystemRoot%\system32\
winload.efi

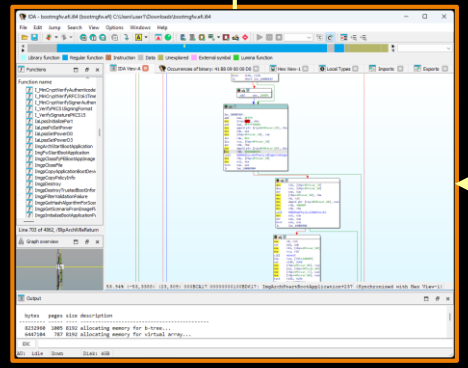


Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe

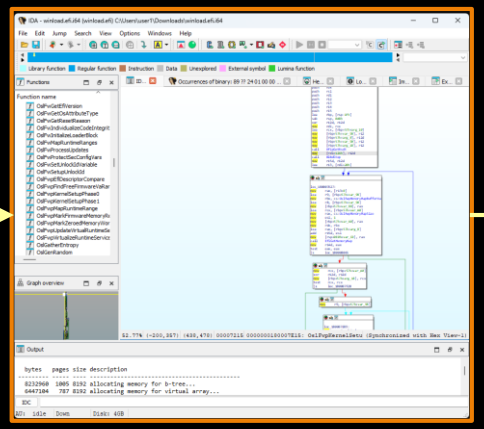
UEFI BOOTKIT DEVELOPMENT

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

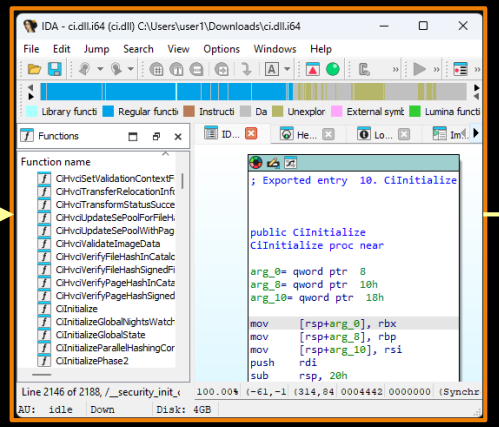




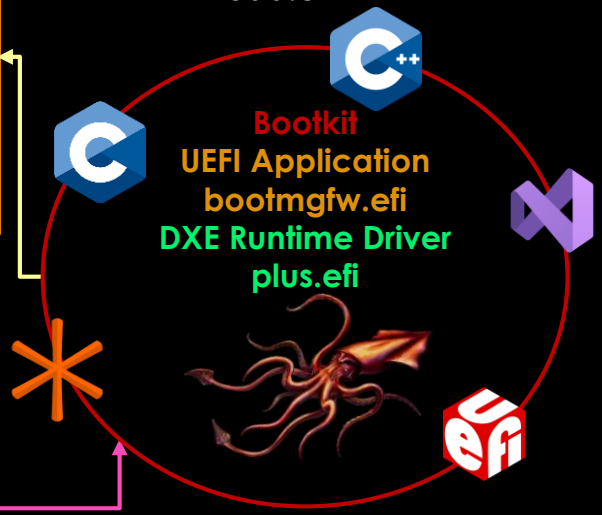
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi



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

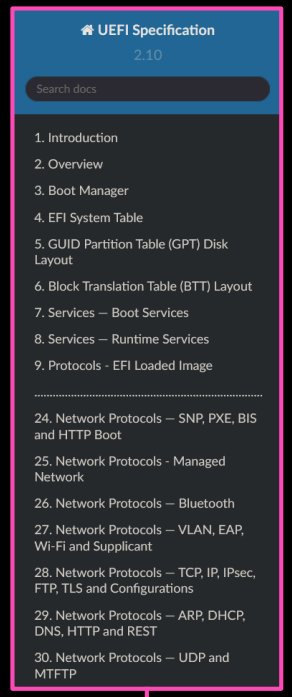


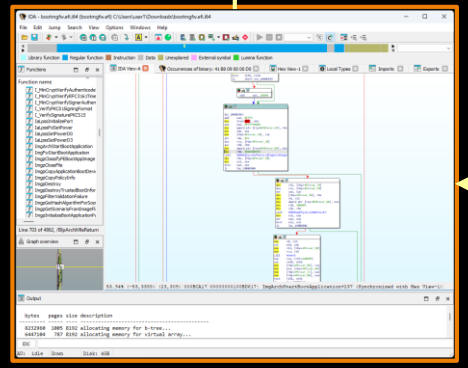
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe



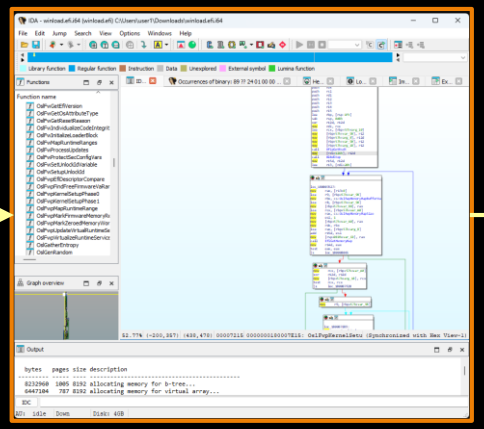
UEFI BOOTKIT DEVELOPMENT

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

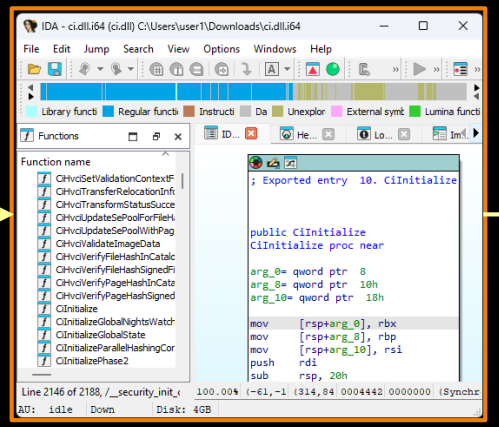




Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi



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

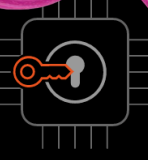


Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe

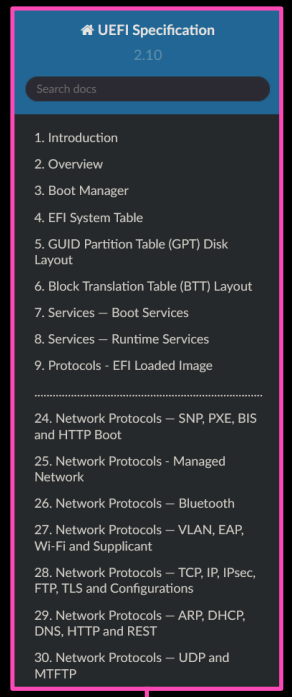
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
plus.efi

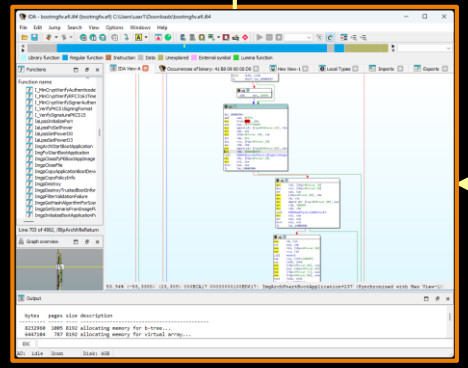
Rootkit
Kernel-Mode Driver
driver.sys
Filter Driver
plus.sys

UEFI BOOTKIT DEVELOPMENT

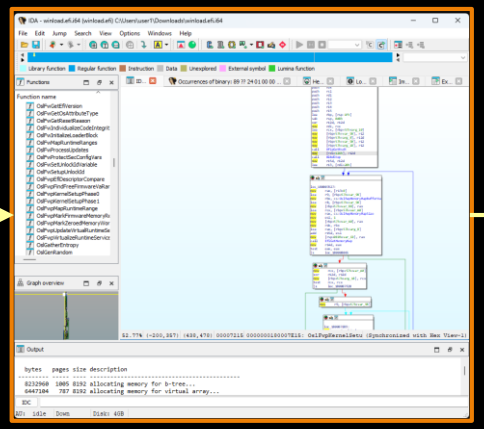


Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi

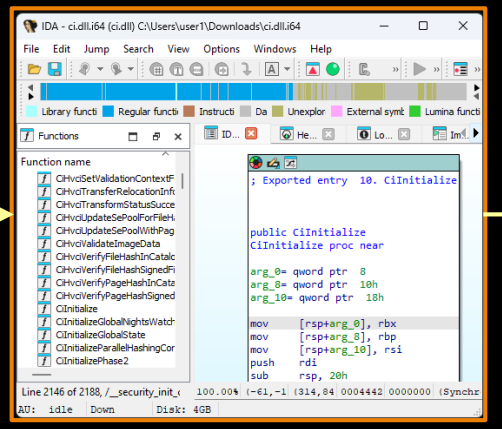




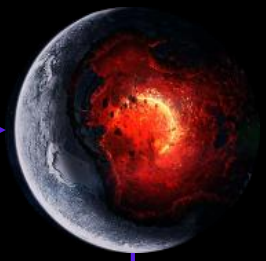
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi

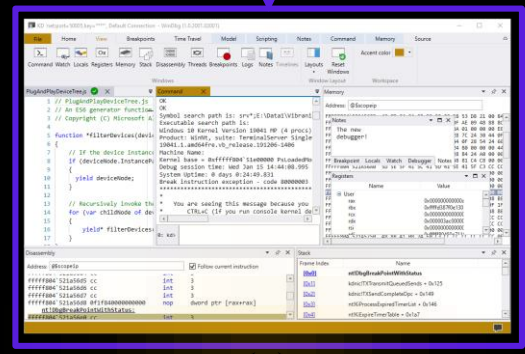


Windows OS Loader
%SystemRoot%\system32\
winload.efi



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





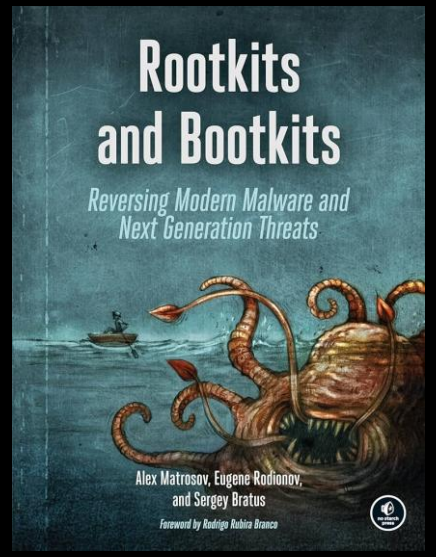
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
plus.efi

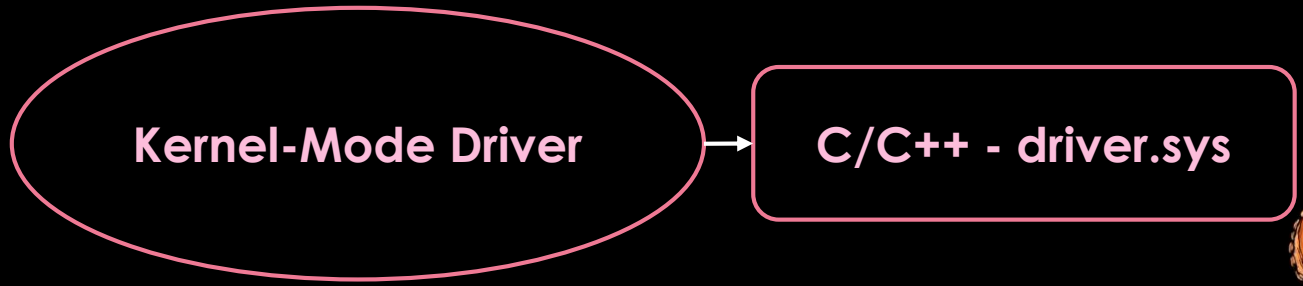
Rootkit
Kernel-Mode Driver
driver.sys
Filter Driver
plus.sys

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





SECURITY MECHANISMS

[*Anti-Rootkit Installation*]

- Driver Signature Enforcement (DSE)

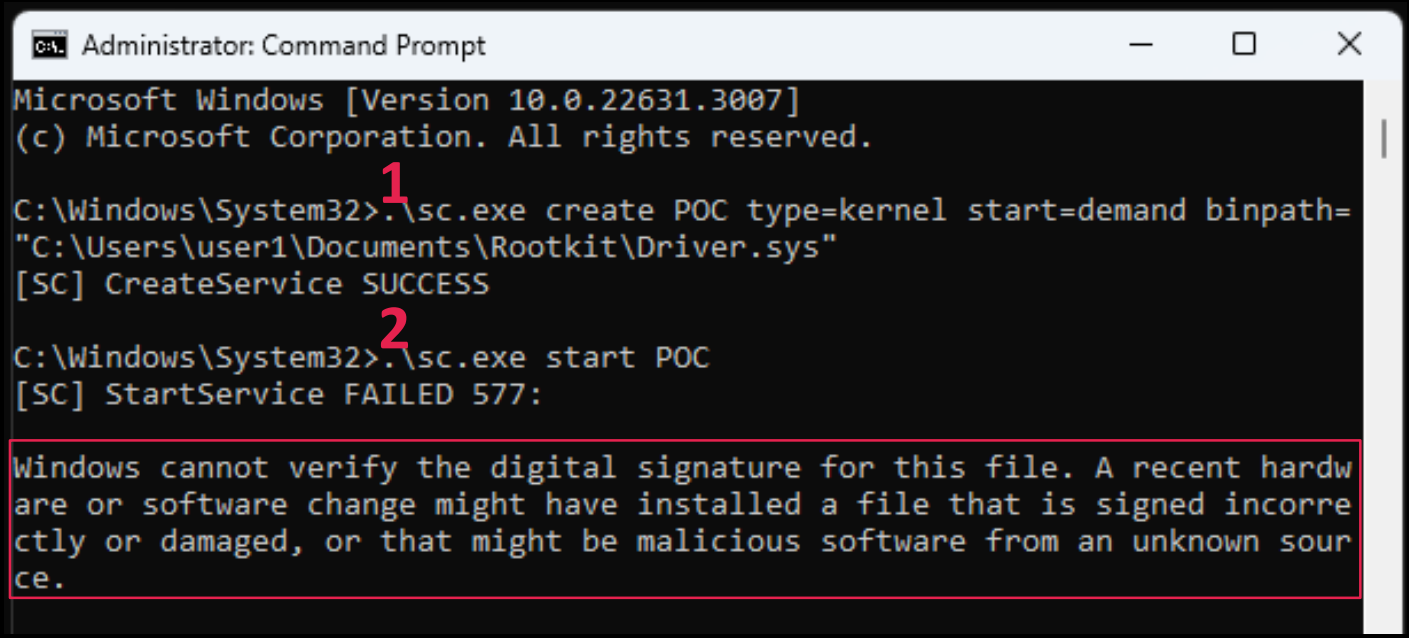
  Windows won't run drivers not certified by Microsoft

SECURITY MECHANISMS

[*Anti-Rootkit Installation*]

- Driver Signature Enforcement (DSE)

  Windows won't run drivers not certified by Microsoft

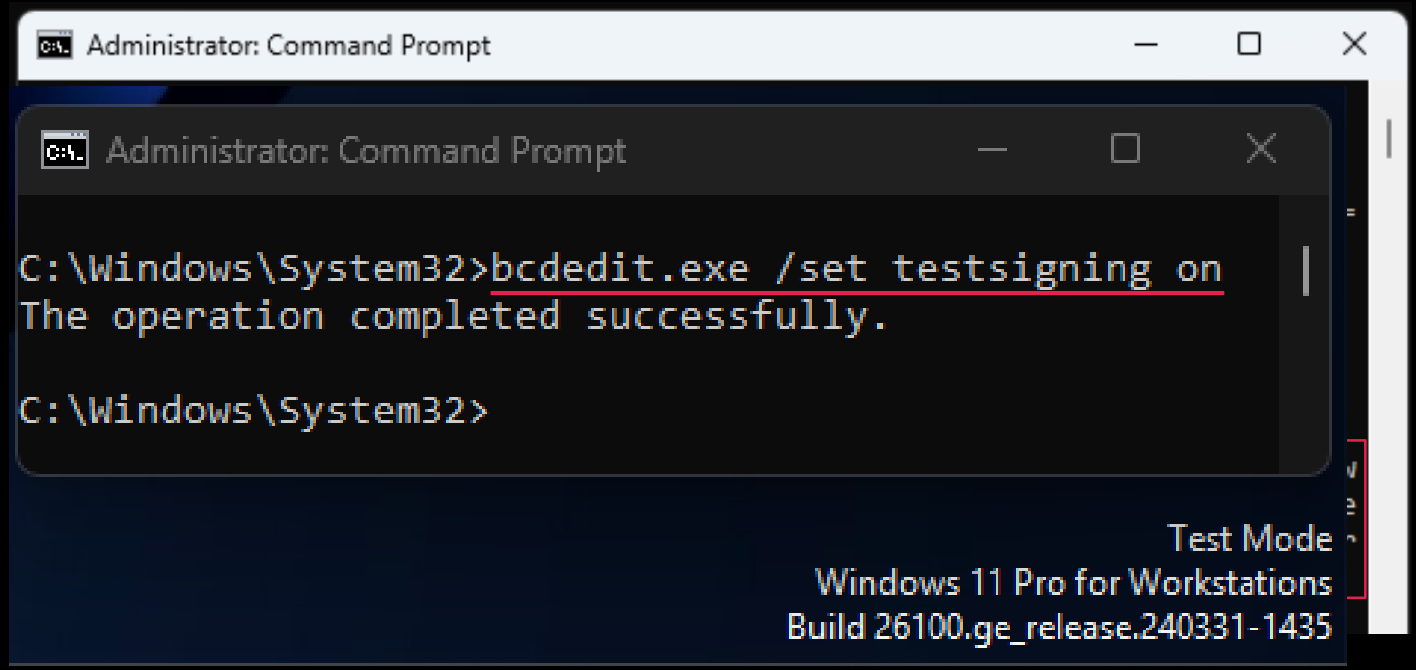


SECURITY MECHANISMS

[*Anti-Rootkit Installation*]

- Driver Signature Enforcement (DSE)

  Windows won't run drivers not certified by Microsoft



DEVELOPMENT ENVIRONMENT

Google

how to develop a Windows kernel mode driver

Learn / Windows / Windows Drivers /

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

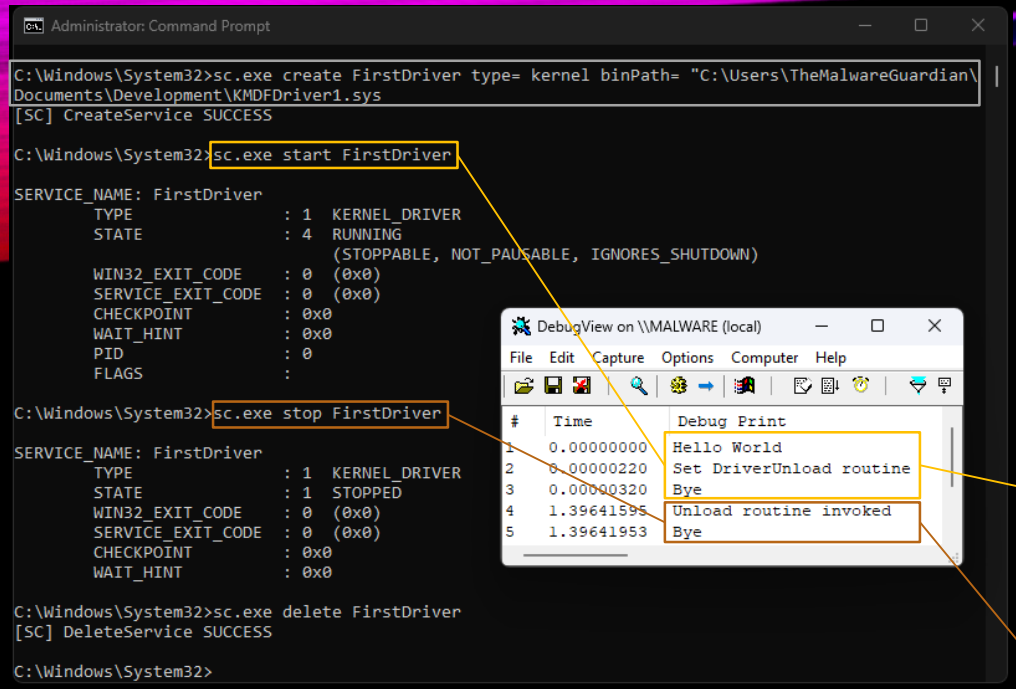
Article • 12/19/2024 • 9 contributors

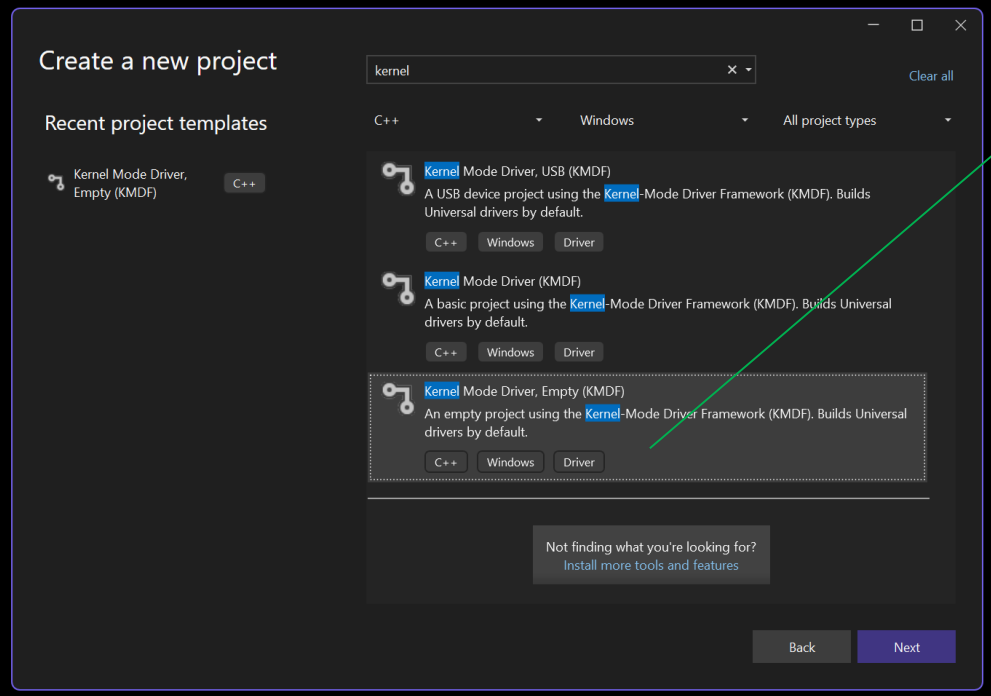
In this article

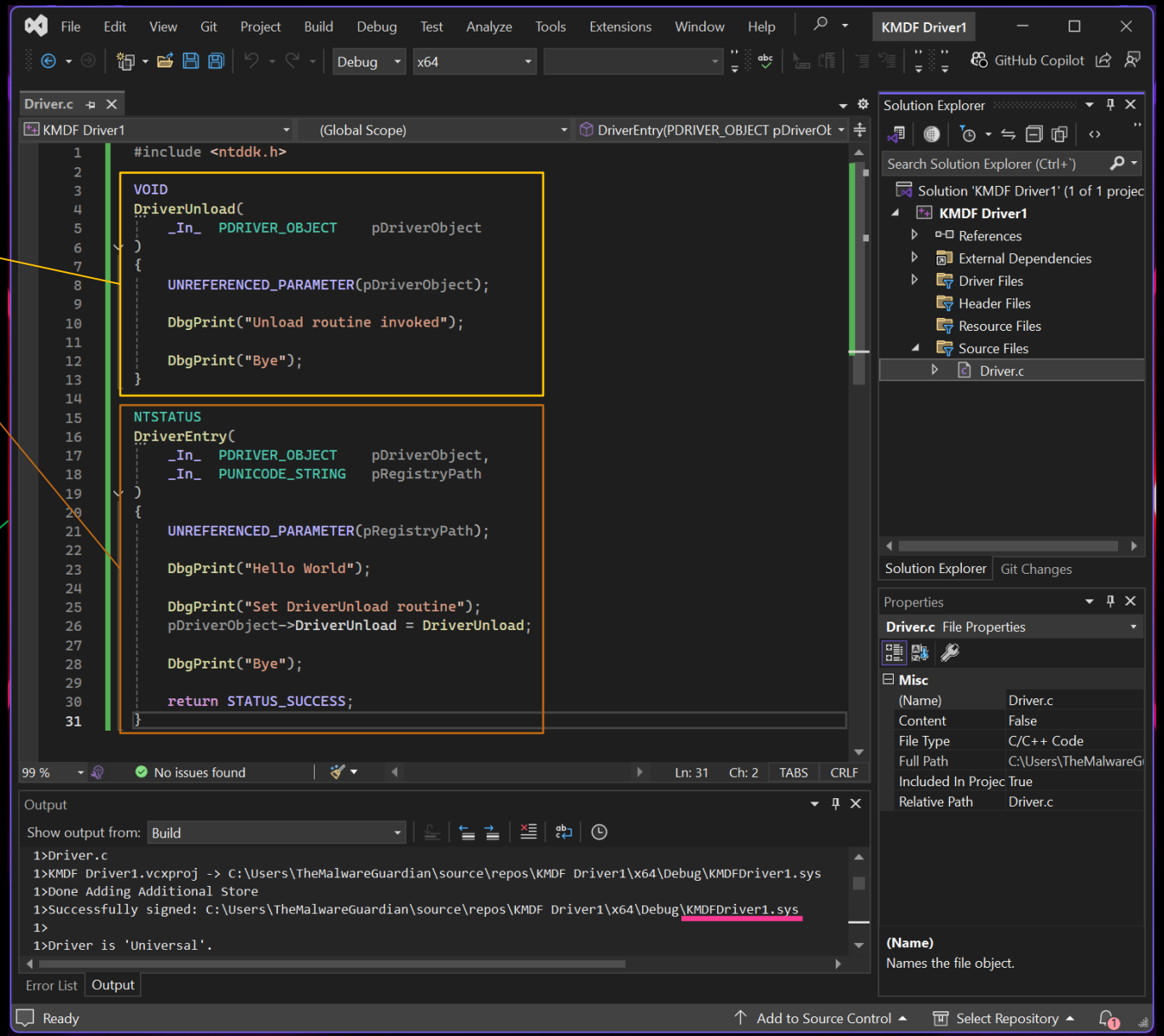
Windows PowerShell

```
================================================================================
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 (Debugee)

 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 + Ghydra + 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







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


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

User Mode | Kernel Mode

console_application.exe

```
// ----------------------
#define IOCTL_COMM_0 CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)


// ----------------------
hDevice = CreateFile(L"\\\\.\\MyKernelDriver", GENERIC_READ | GENERIC_WRITE, ...


// ----------------------
BOOL success = DeviceIoControl(hDevice,
                        1 IOCTL_COMM_0,
                        2 inBuffer, sizeof(inBuffer),
                        3 outBuffer, sizeof(outBuffer),
                         &bytesReturned, NULL);
```

IRP

```
typedef struct _IRP {}
```

Simbolic Link

```
"\\DosDevices\\MyKernelDriver"
```

Handle IRP

```
IRP_MJ_DEVICE_CONTROL
```

Device Object

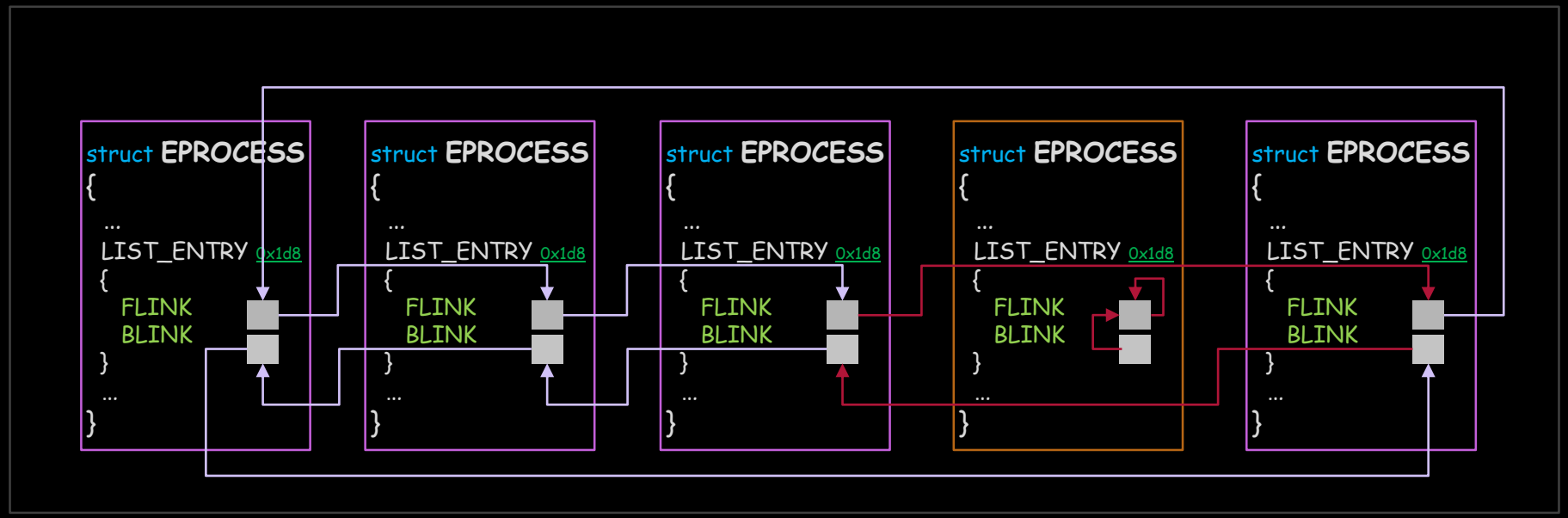
```
"\\Device\\MyKernelDriver"
```

kernel_mode_driver.sys

```
// ----------------------
#define IOCTL_COMM_0 CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)


// ----------------------

status = IoCreateDevice(...);
status = IoCreateSymbolicLink(...);


// ----------------------
pDriverObject->MajorFunction[...] = ...;
pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DriverHandleIOCTLs;


// ----------------------
NTSTATUS
DriverHandleIOCTLs(
    _In_  PDEVICE_OBJECT  pDeviceObject,
    _In_  PIRP            pIrp
)
{
    PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(pIrp);

    ULONG controlCode = stack->Parameters.DeviceIoControl.IoControlCode;

    switch (controlCode)
    {

        case IOCTL_COMM_0:
            ...
            break;

        case IOCTL_COMM_1:
            ...
            break;

    pIrp->IoStatus.Status = STATUS_SUCCESS;

    IoCompleteRequest(pIrp, IO_NO_INCREMENT);

    return STATUS_SUCCESS;
}
```

II HIDE PROCCESSES

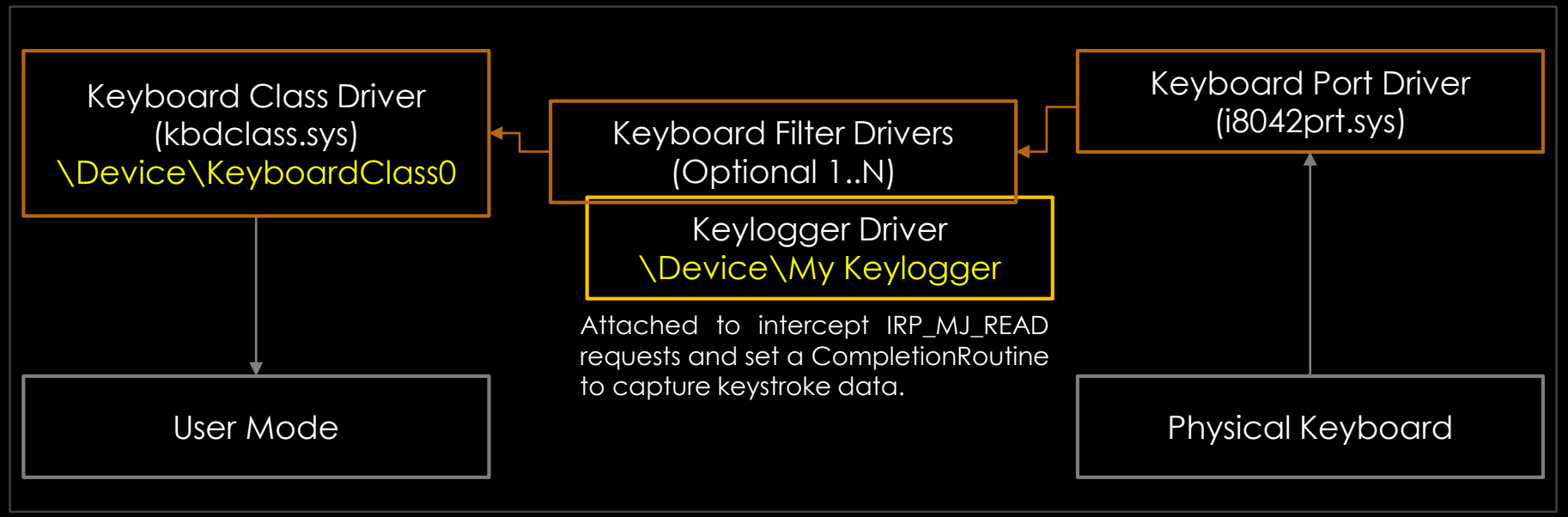
"Windows maintains a doubly linked list of active processes in (LIST_ENTRY) EPROCESS->ActiveProcessLinks. Unlink a process from the chain, and it disappears from user-mode enumeration."



III

KEYLOGGER

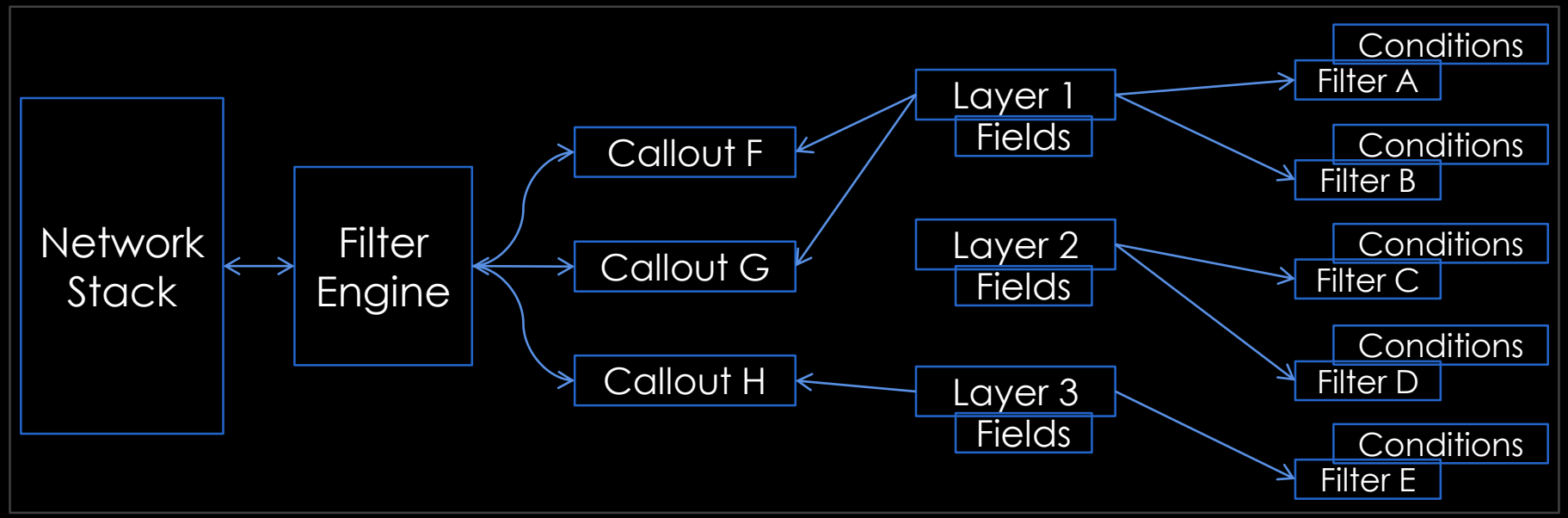
"Keystroke interception in kernel mode: The Windows keyboard driver stack routes all keystrokes through a device object called \Device\KeyboardClass0. By attaching a driver to this device and registering a CompletionRoutine, we can capture raw keystroke data before it propagates to user-mode applications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



IV

NETWORK CONTROL

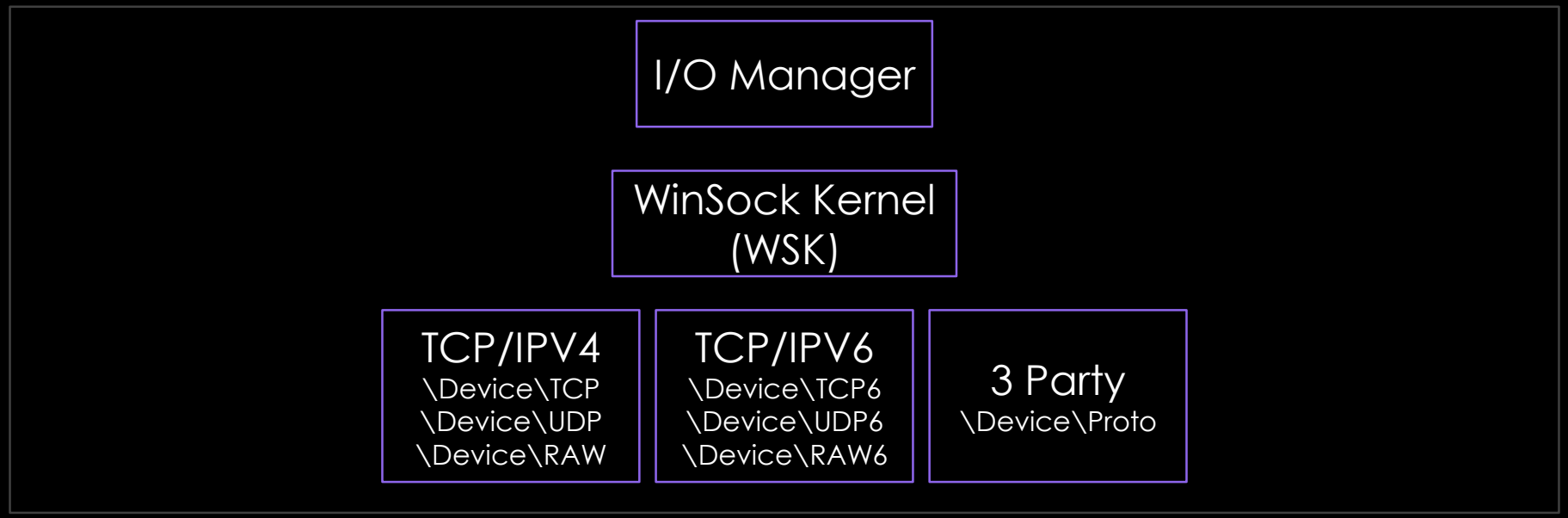
"Windows Filtering Platform (WFP) allows real-time inspection and control of network connections. By attaching filters (static rules applied at specific layers of the network stack to identify traffic based on attributes like IPs or ports) and callouts (custom drivers that execute dynamic logic on flagged 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."



V

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

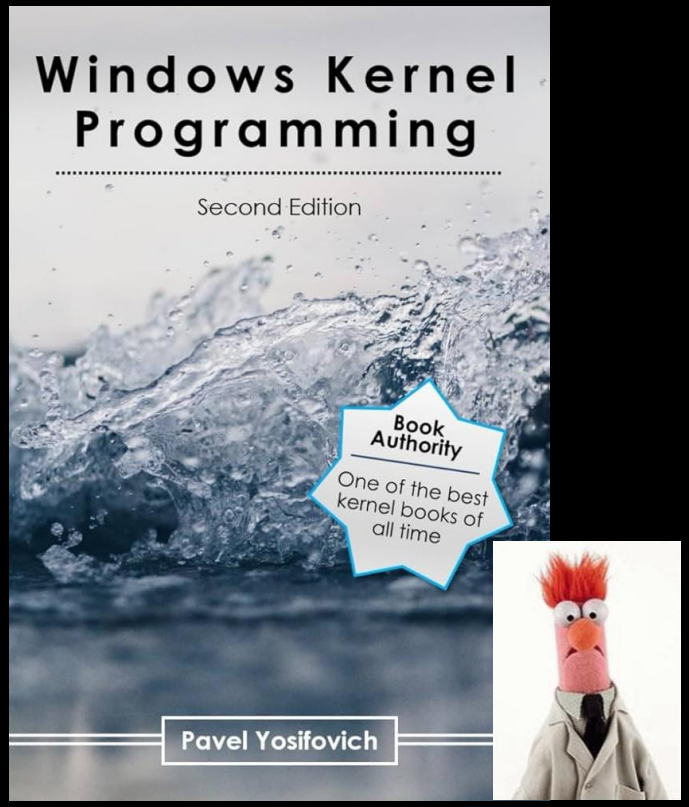
"MiniFilters attach to the file system stack to filter I/O operations."

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
CldFlt                              0       180451         0
bfs                                 6       150000         0
FileCrypt                           0       141100         0
luafv                               1       135000         0
UnionFS                             0       130850         0
npsvctrig                           1        46000         0
Wof                                 2        40700         0
FileInfo                            4        40500         0
```

Administrator: Command Prompt

```
C:\Users\TheMalwareGuardian\Downloads>sc.exe create WindowsKernelMinifilter type=filesys
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

THE GATEWAY

Rootkit Installation

Kernel Mode Driver

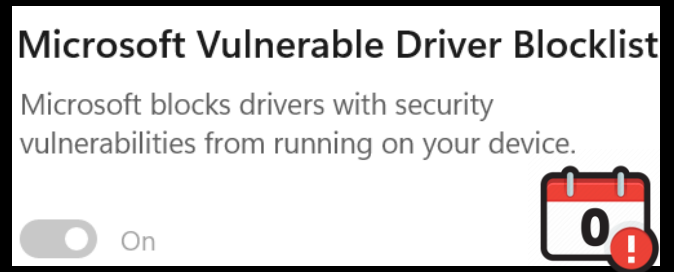
THE GATEWAY

Rootkit Installation
Kernel Mode Driver ?

1. Vulnerable Kernel Driver

(BYOVD)
Bring Your Own Vulnerable Driver

Not Well Known

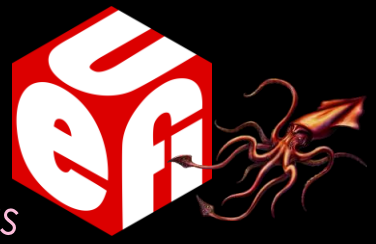


2. UEFI Bootkit

Infect a Computer's Boot process

SecureBoot

Physical Access



B / ROOTKIS IN THE WILD

2022

MoonBounce 

CosmicStrand 

Fire Chili 

2024

Bootkitty 

Fudmodule 

2021

ESPecter 

FinSpy 

Moriya 

2023

BlackLotus 

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)

mpus Internacional
BERSEGURIDAD

UNIVERSIDAD CATÓLICA SAN ANTONIO
IN LIBERTATEM VOCATI
AÑO MCMXCVI

ENIIT INNOVA IT BUSINESS SCHOOL

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: