

INFECTING THE BOOT TO OWN THE KERNEL: BOOTKITS AND ROOTKITS DEVELOPMENT

Demistifying UEFI Bootkits and Kernel-Mode Rootkits Development

[in/vazquez-vazquez-alejandro]

[in/mariasanjose]

DEF CON 33, Las Vegas 2025



WHOAREWE



GEEK (**G**entleman **E**xploring u**E**FI and OS **K**ernel)



GEEK (**G**irl **E**xploring **E**xploits and **K**ernel threats)

OCTOPUS LOVERS



Sensitive Content

This presentation contains age-restricted materials including malware and explicit hooking techniques. By entering, you affirm that you are at least 18 years of age and you consent to viewing “hacker” stuff.

Let me In
This is real malware

No
I prefer to stay with the humans in the Villages



BOOTKITS & ROOTKITS DEMYSTIFIED

- Bootkit: Malicious program designed to load as early as possible in the boot process, in order to control all stages of the operating system start up, modifying system code and drivers before security components are loaded.

  ~ Kaspersky

- Rootkit: Sophisticated piece of malware that can add new code to the operating system or delete and edit operating system kernel code.

  ~ Crowdstrike

BOOTKITS & ROOTKITS DEMYSTIFIED

- Bootkit: Malicious program designed to load as early as possible in the boot process, in order to control all stages of the operating system start up, modifying system code and drivers before security components are loaded.

  ~ Kaspersky

UEFI Application ⇢ **C/C++ - boot.efi**



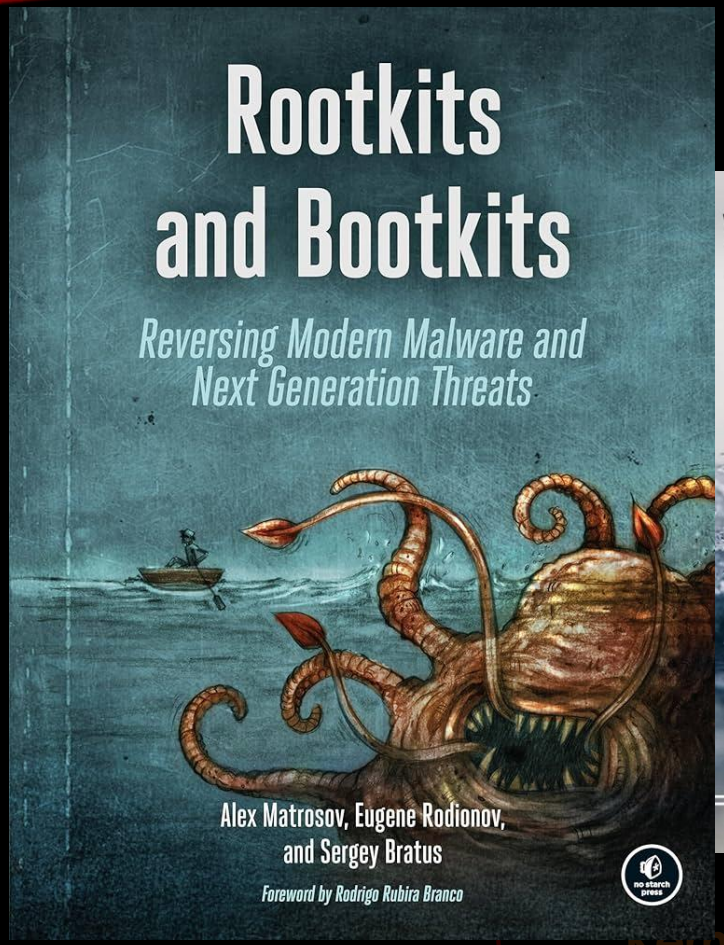
- Rootkit: Sophisticated piece of malware that can add new code to the operating system or delete and edit operating system code.

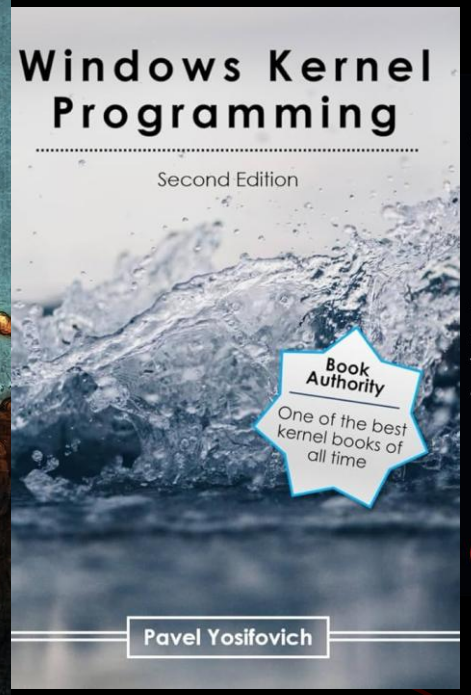
  ~ Crowdstrike

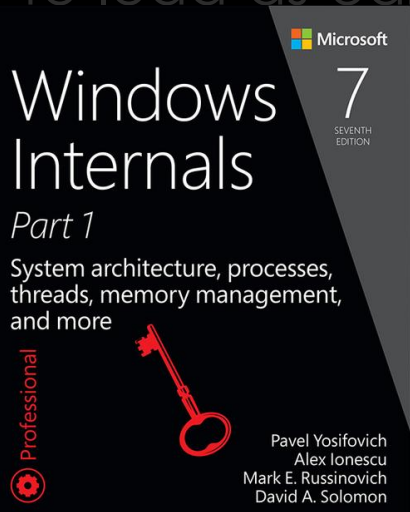
Kernel-Mode Driver ⇢ **C/C++ - driver.sys**

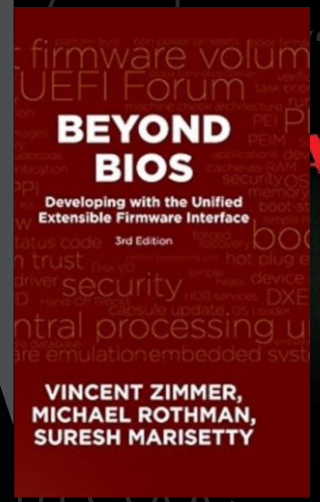


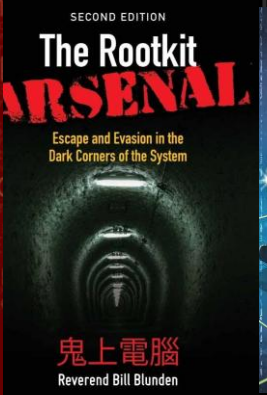
BOOTKITS & ROOTKITS DEMYSTIFIED

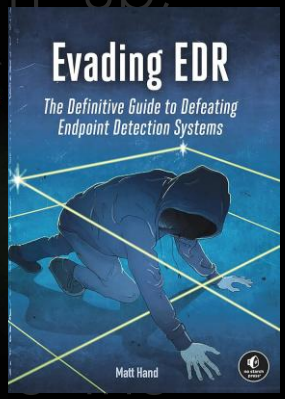












to load as early as possible in the boot

Kernel-Mode Driver

C/C++ - driver.sys



SECURITY MECHANISMS TO BYPASS

* [*Anti-Bootkit Installation*] - Prevent installation of malicious UEFI applications
* [*Anti-Rootkit Installation*] - Prevent installation of malicious Kernel-mode drivers

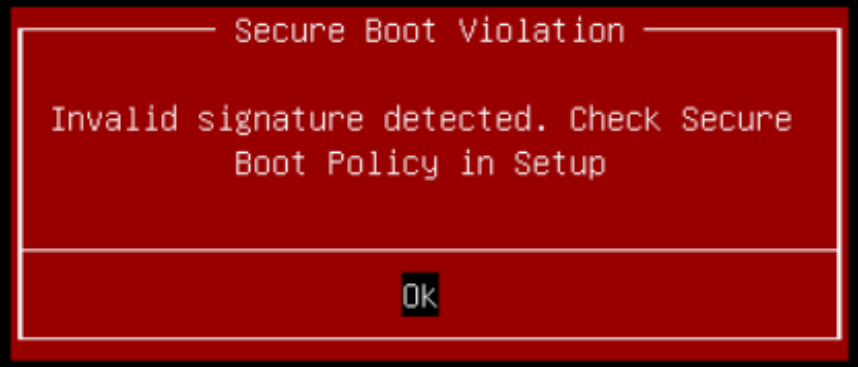
SECURITY MECHANISMS TO BYPASS

* [*Anti-Bootkit Installation*] - Prevent installation of malicious UEFI applications

- SecureBoot

  Only software trusted by the Original Manufacturer

  Firmware checks the signature of UEFI firmware drivers, EFI applications and SO



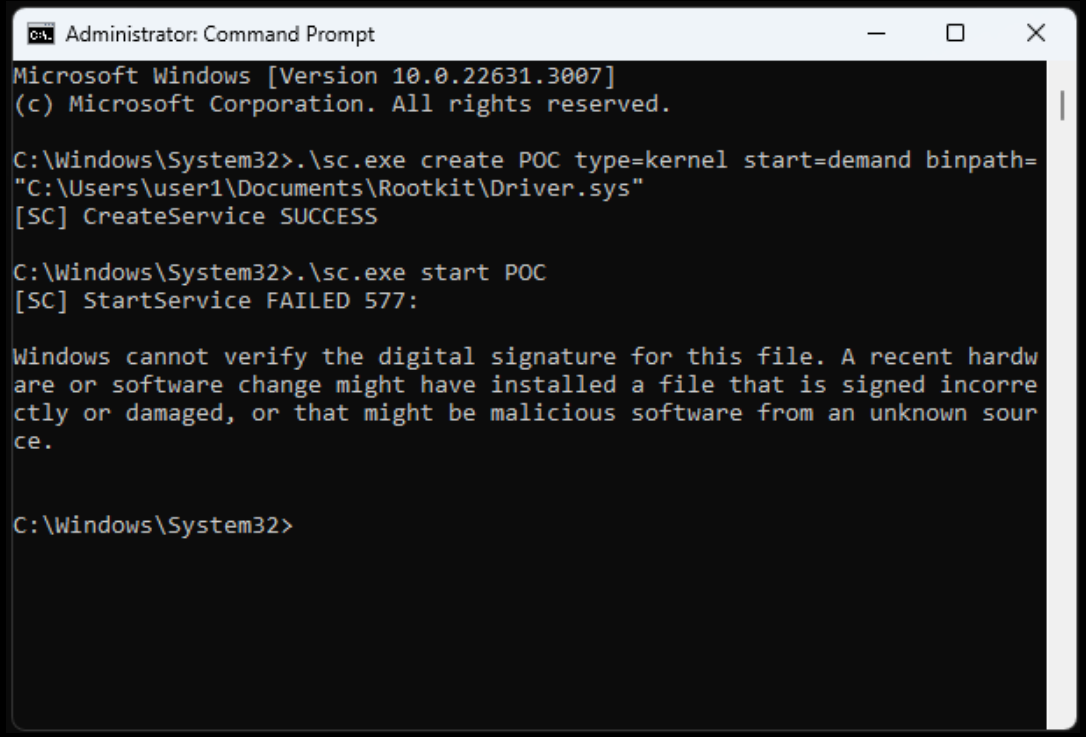
CVE-2025-33043 (SMRAM write)
CVE-2024-8105 (PKfail)
CVE-2024-7344 (Custom PE loader)
CVE-2023-40238 (LogoFAIL Image parsing)
CVE-2022-21894 (Baton Drop)

SECURITY MECHANISMS TO BYPASS

* [*Anti-Rootkit Installation*] - Prevent installation of malicious Kernel-mode drivers

- Driver Signature Enforcement (DSE)

  Windows won't run drivers not certified by Microsoft



Bring Your Own Vulnerable Driver (BYOVD)

RTCore64.sys
Viragt64.sys
dbutil_2_3. sys
loldrivers

Bootkit/Rootkit Development

CHECKPOINT

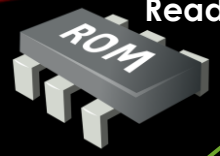
- Bootkit = Malicious UEFI Application
- Rootkit = Malicious Kernel Mode Driver
- Protection Mechanisms
  - SecureBoot
  - DSE
  - PatchGuard
  - Vulnerable driver blocklist
  - …..



UNDERSTANDING THE BOOT PROCESS

Power ON

Read Instructions



POST



UEFI Firmware



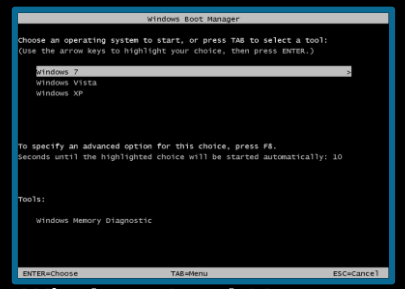
Boot Information



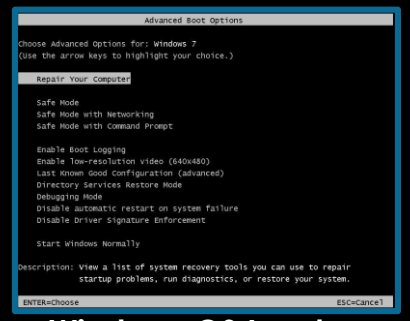
EFI System Partition



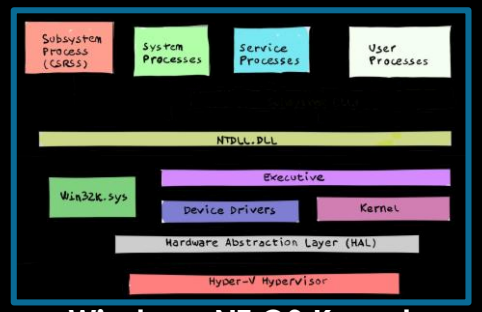
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



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



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



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

MANIPULATING THE BOOT PROCESS

Power ON

Read Instructions

POST

UEFI Firmware

Boot Information

EFI System Partition

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

```
// Find function pattern
Status = FindPattern(TargetFunctionSignature, 0xCC, sizeof(TargetFunctionSignature), (UINT8*)ImageBase...

// Find function start
OriginalFunctionPointer = (VOID*)FindStartAddress(ImageBase, NtHeaders, Found);

// Hook bootmgfw.efi!Function, winload.efi!Function
VOID* HookAddress = (VOID*)&MyHookHandlerFunction;

// Backup original function bytes
CopyMem(BackupOriginalBytes, OriginalFunctionPointer, sizeof(HookTemplate));

// Place hook trampoline at the start of the original function
CopyMem(OriginalFunctionPointer, HookTemplate, sizeof(HookTemplate));

// Write hook handler address into trampoline template
CopyMem((UINT8*) OriginalFunctionPointer + HookTemplateAddressOffset, (UINTN*)&HookAddress, sizeof(UINTN));
```

Power ON

Read Instructions

POST

Boot Information

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

EFI System Partition

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

MANIPULATING THE BOOT PROCESS

MANIPULATING THE BOOT PROCESS

Power ON

Read Instructions

POST

UEFI Firmware

Boot Information

EFI System Partition

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

boot ➔ bootmgfw.efi

- ExitBootServices();
- gBS->LoadImage();

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

bootmgfw.efi ➔ winload.efi

- ImgArchStartBootApplication();
- ArchPx64TransferTo64BitApplicationAsm();

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

winload.efi ➔ ntoskrnl.exe

- OslArchTransferToKernel();
- OslFwpKernelSetupPhase1();
- BlImgAllocateImageBuffer();

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

ntoskrnl.exe ➔ user

- CmGetSystemDriverList();
- SeCodeIntegrityQueryInformation();

Bootkit
UEFI Application
bootmgfw.efi

MANIPULATING THE BOOT PROCESS

Power ON

Read Instructions

POST

UEFI Firmware

Boot Information

EFI System Partition

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

Power ON

Read Instructions

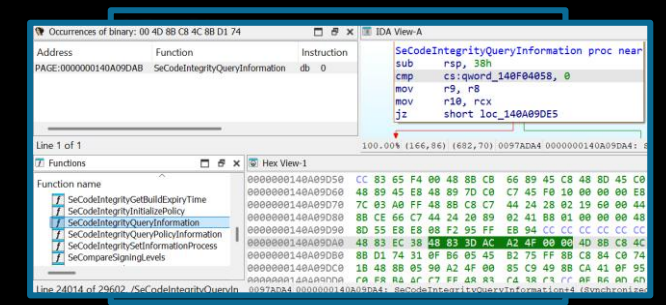
POST

UEFI Firmware

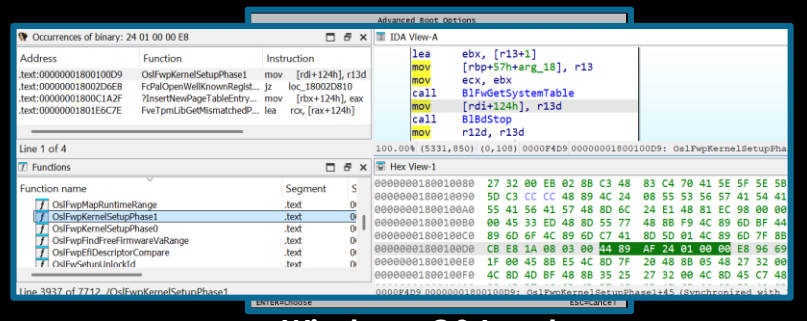
Boot Information

EFI System Partition

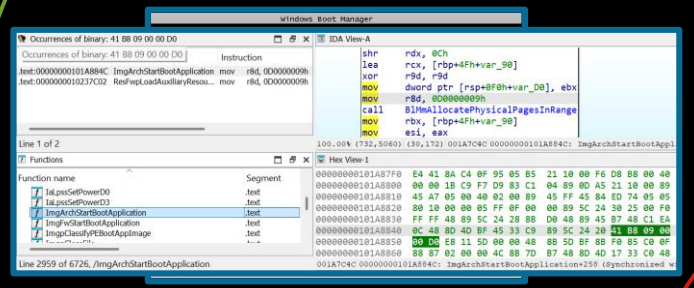
Boot order
Boot0001 = /EFI/Microsoft/boot/bootmgfw.efi
Boot0002 = /EFI/Ubuntu/shimx64.efi
Boot000x = /EFI/Vendor/bootx64.efi



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



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



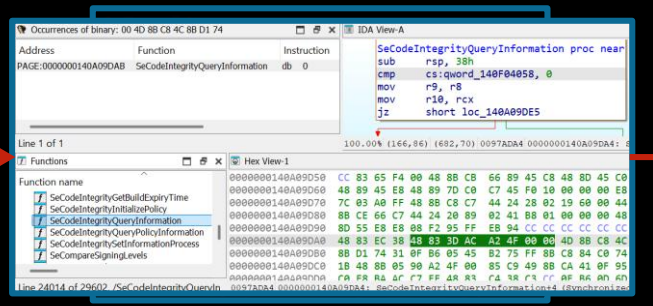
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi

Bootkit
UEFI Application
bootmgfw.efi

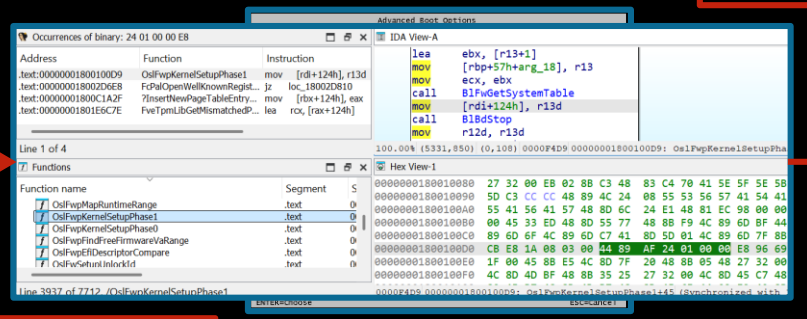
Rootkit
Kernel-Mode Driver
kmdf.sys

DEMO

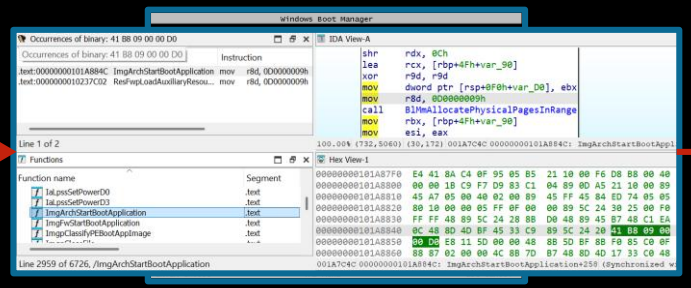




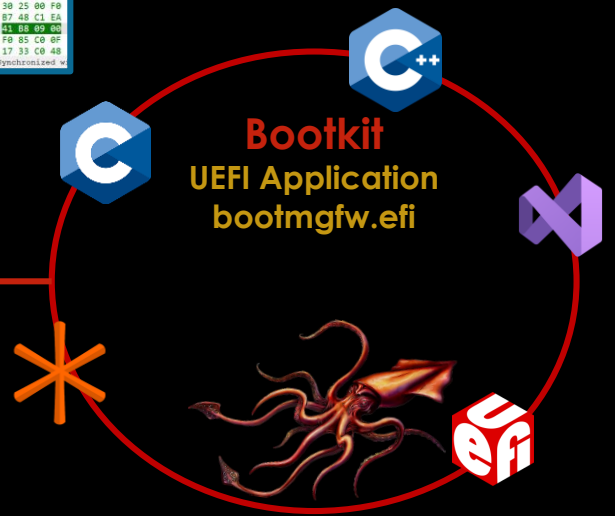
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe

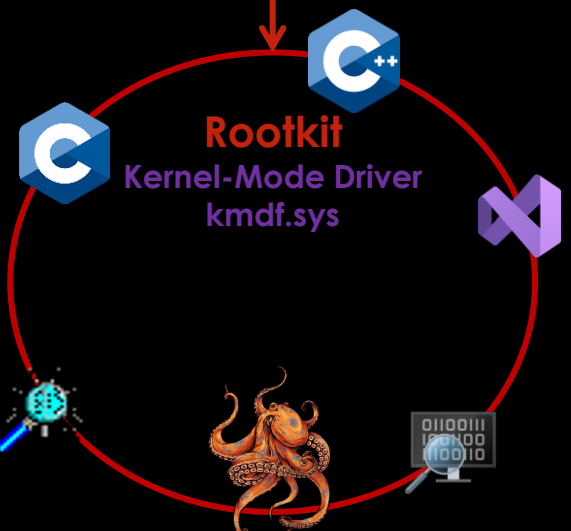


Windows OS Loader
%SystemRoot%\system32\
winload.efi

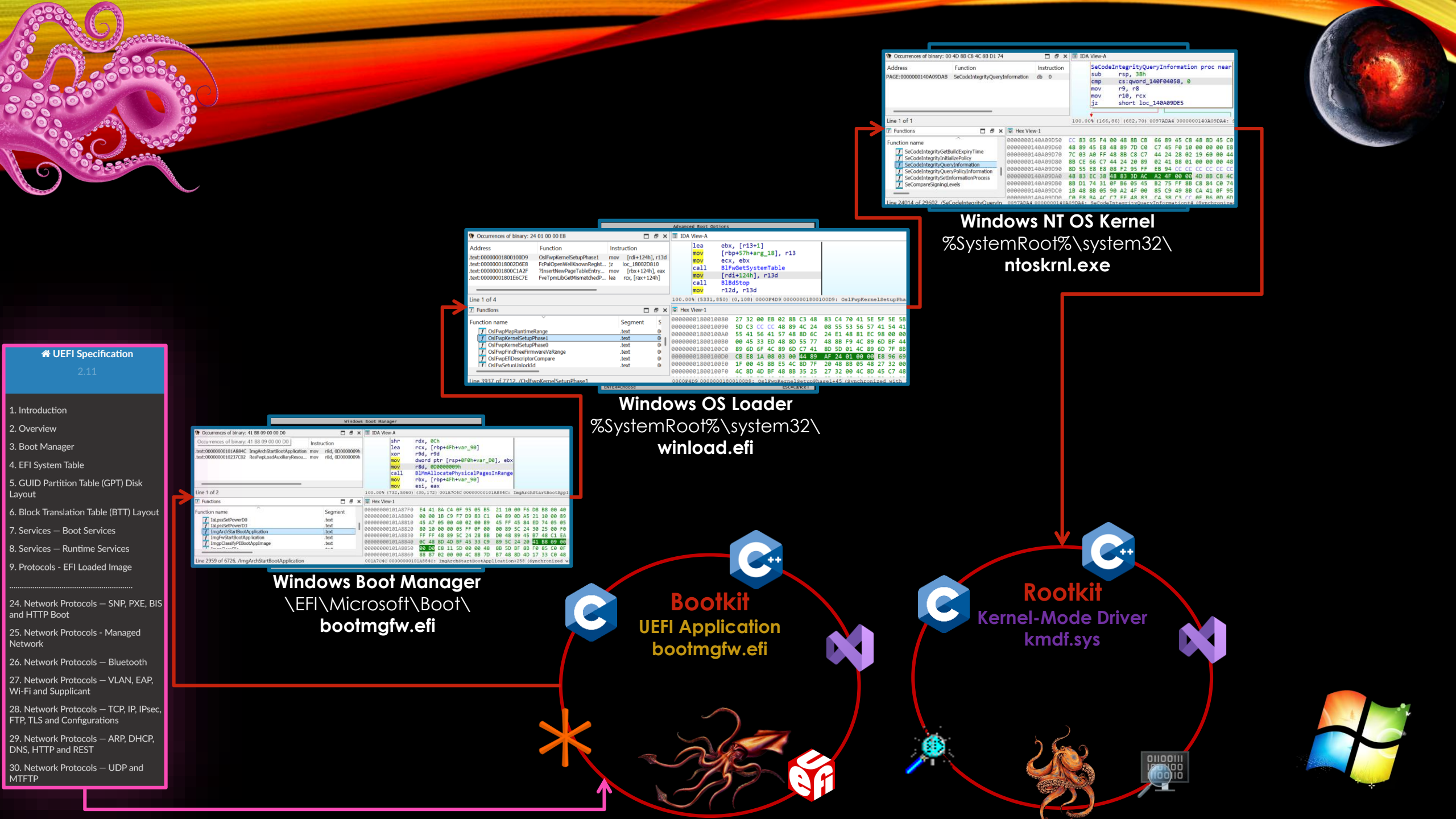


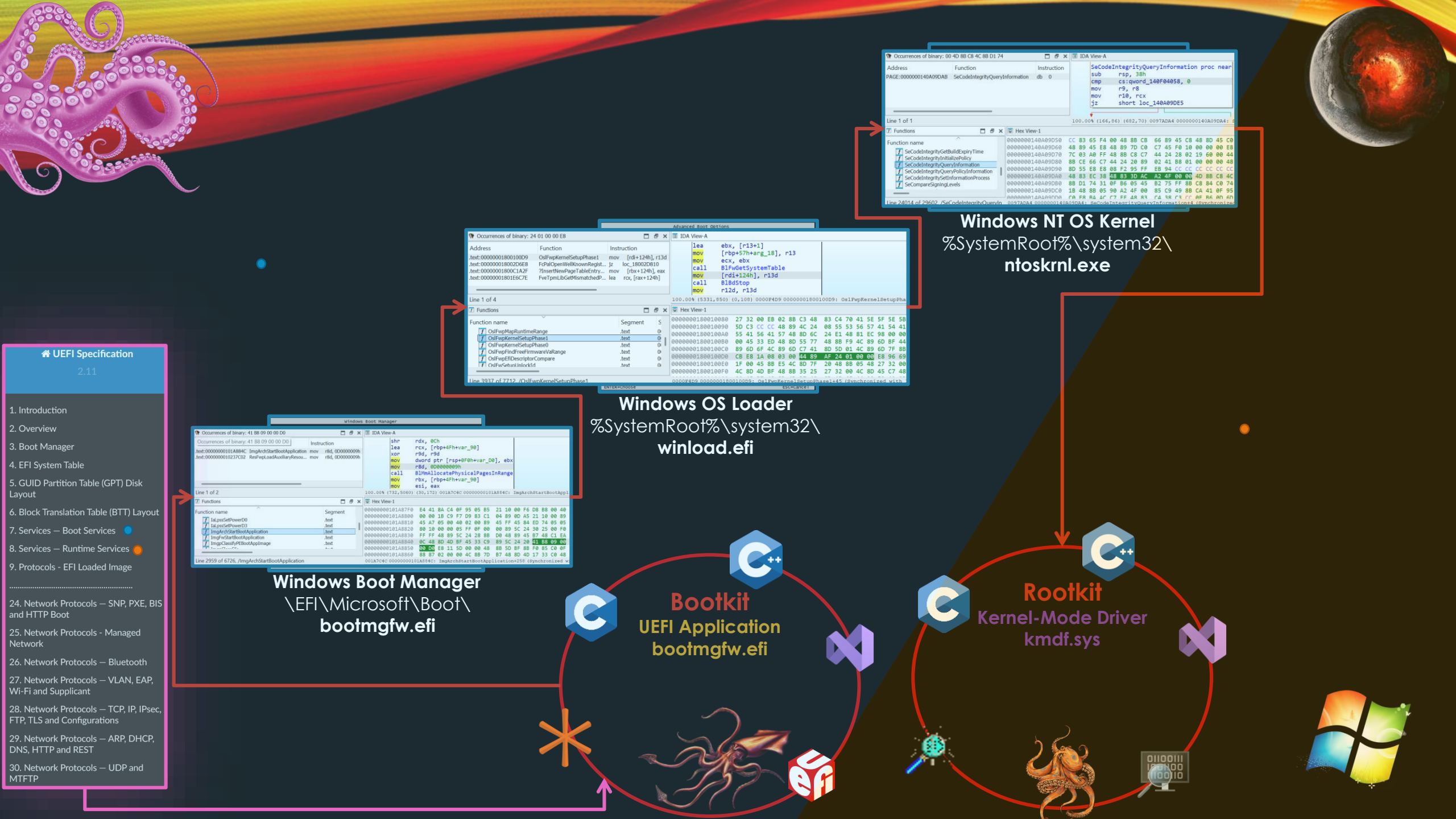
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi

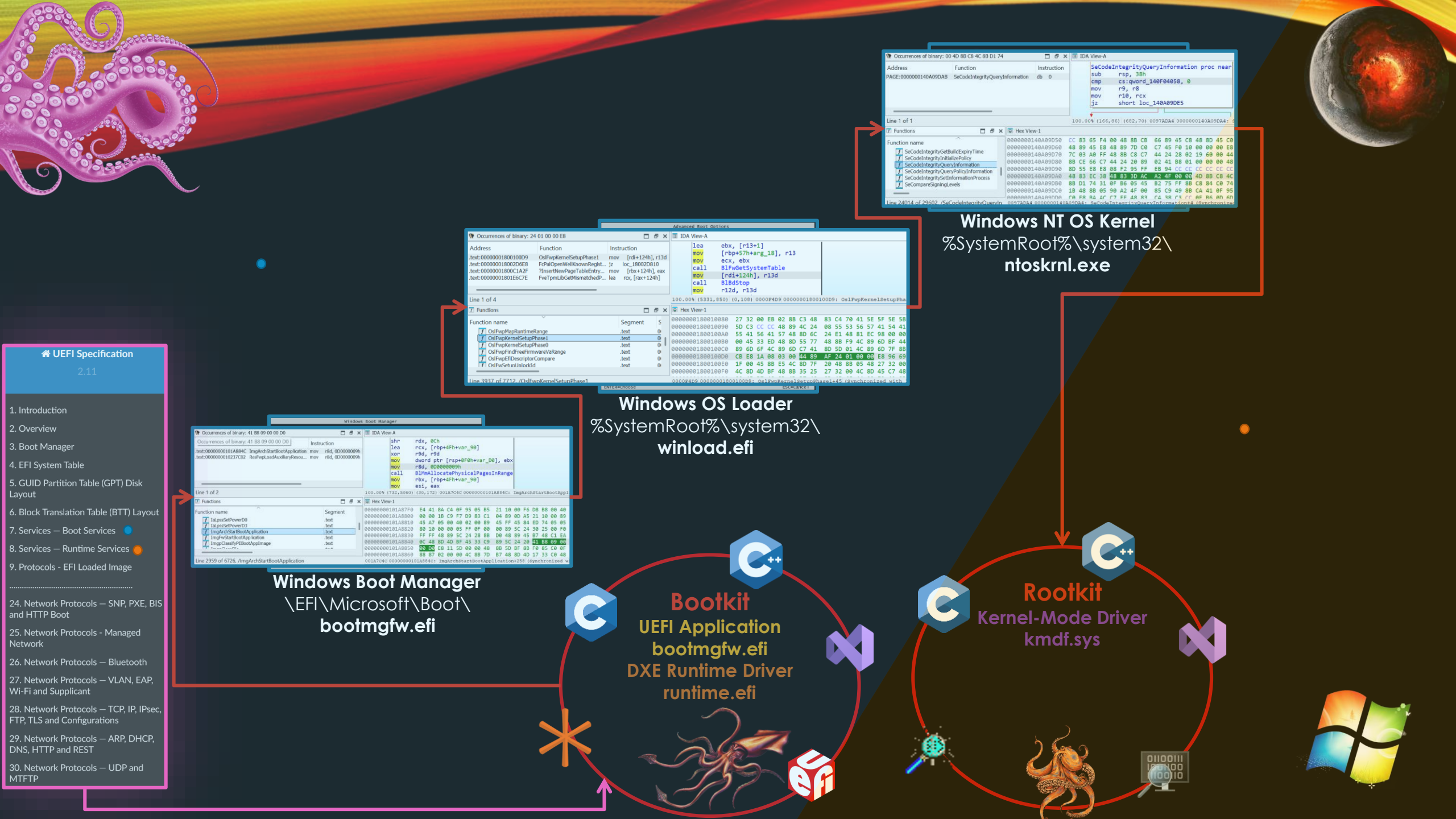






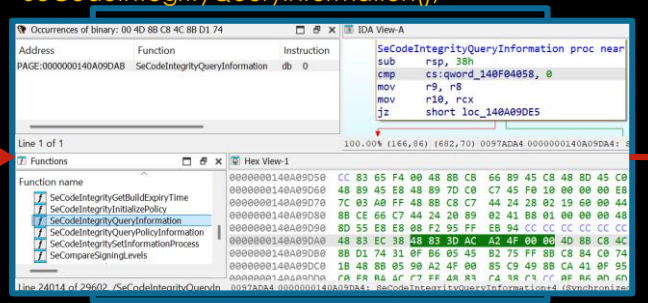

UEFI Specification
2.11
1. Introduction
2. Overview
3. Boot Manager
4. EFI System Table
5. GUID Partition Table (GPT) Disk Layout
6. Block Translation Table (BTT) Layout
7. Services — Boot Services
8. Services — Runtime Services
9. Protocols - EFI Loaded Image
.....
24. Network Protocols — SNP, PXE, 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
Bootkit
UEFI Application
bootmgfw.efi
Rootkit
Kernel-Mode Driver
kmdf.sys


UEFI Specification
2.11
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
Bootkit
UEFI Application
bootmgfw.efi
Rootkit
Kernel-Mode Driver
kmdf.sys


UEFI Specification
2.11
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
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
runtime.efi
Rootkit
Kernel-Mode Driver
kmdf.sys

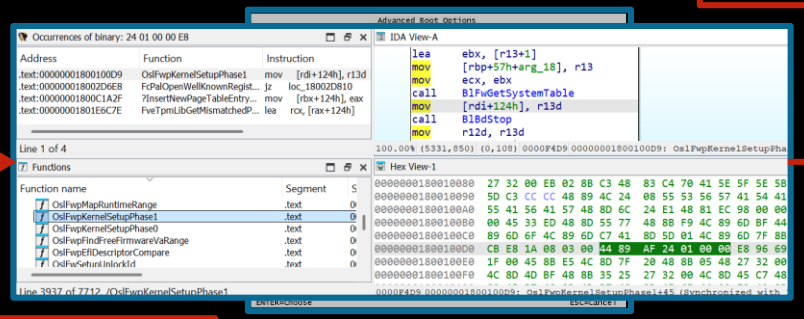


CmGetSystemDriverList();
SeCodeIntegrityQueryInformation();



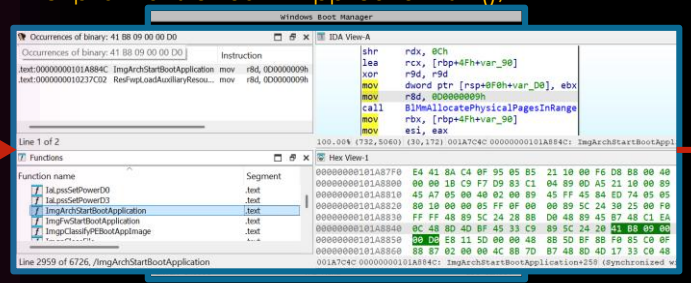
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe

OslArchTransferToKernel();
OslFwpKernelSetupPhase1();
BlImgAllocateImageBuffer();



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

ImgArchStartBootApplication();
Archpx64TransferTo64BitApplicationAsm();



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

UEFI Specification
2.11

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

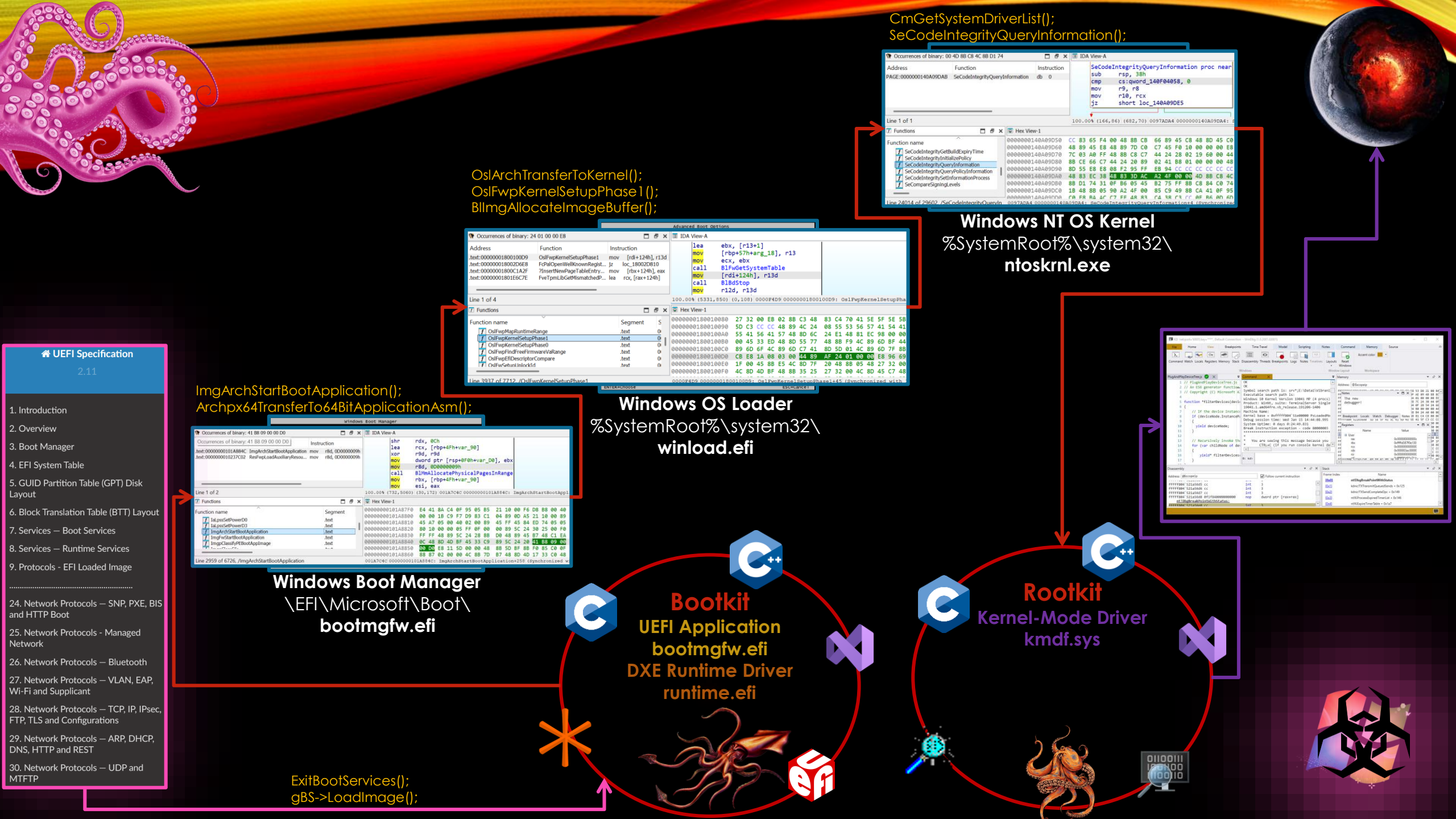
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
runtime.efi

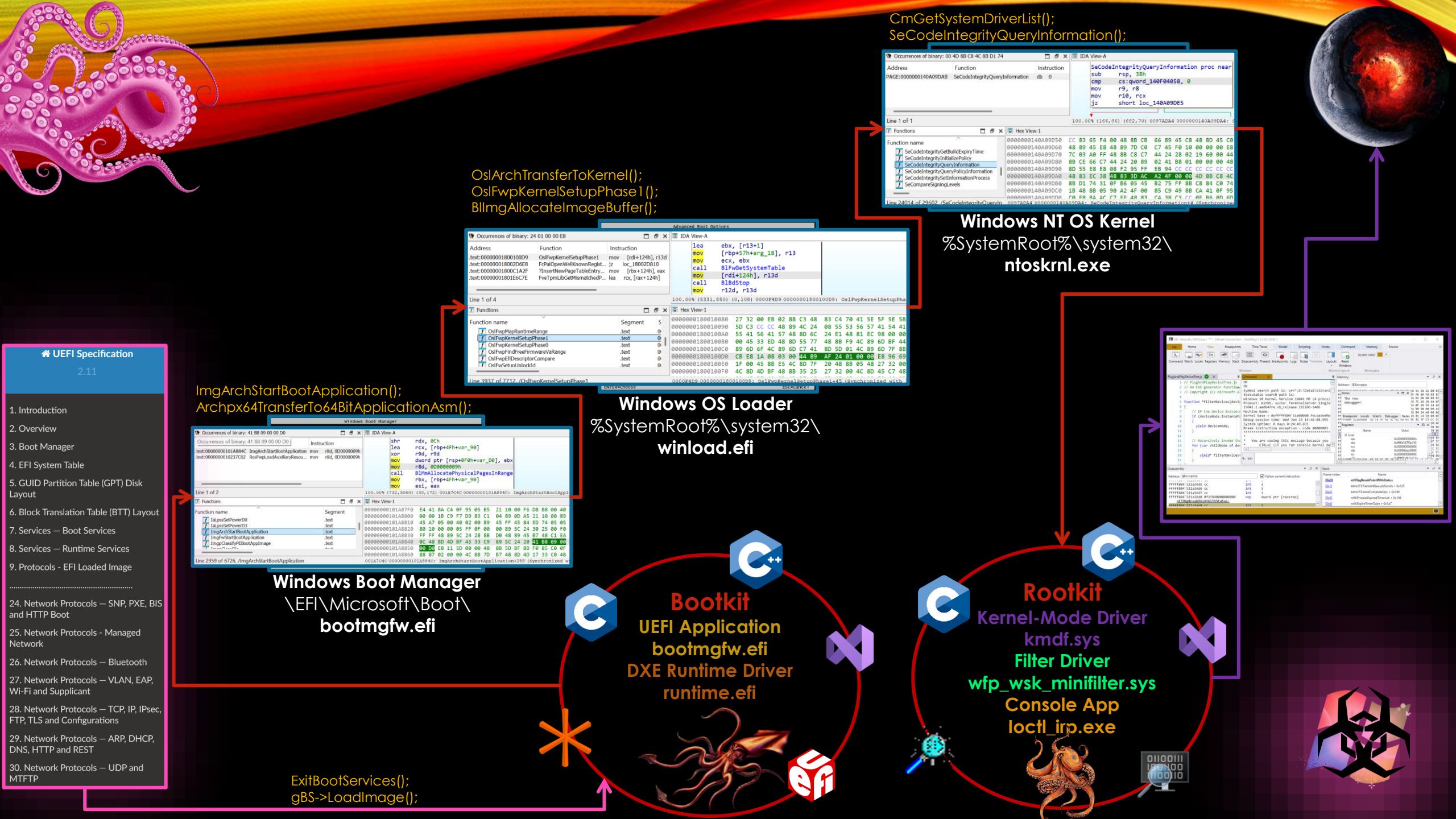
Rootkit
Kernel-Mode Driver
kmdf.sys

ExitBootServices();
gBS->LoadImage();

DEMO




UEFI Specification
2.11
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
ImgArchStartBootApplication();
ArchpxTransferTo64BitApplicationAsm();
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi
OslArchTransferToKernel();
OslFwpKernelSetupPhase1();
BlImgAllocateImageBuffer();
Windows OS Loader
%SystemRoot%\system32\
winload.efi
CmGetSystemDriverList();
SeCodeIntegrityQueryInformation();
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
runtime.efi
Rootkit
Kernel-Mode Driver
kmdf.sys
ExitBootServices();
gBS->LoadImage();


UEFI Specification
2.11
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
ImgArchStartBootApplication();
ArchpxTransferTo64BitApplicationAsm();
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi
OslArchTransferToKernel();
OslFwpKernelSetupPhase1();
BlImgAllocateImageBuffer();
Windows OS Loader
%SystemRoot%\system32\
winload.efi
CmGetSystemDriverList();
SeCodeIntegrityQueryInformation();
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
runtime.efi
Rootkit
Kernel-Mode Driver
kmdf.sys
Filter Driver
wfp_wsk_minifilter.sys
Console App
Ioctl_irp.exe
ExitBootServices();
gBS->LoadImage();

Rootkit Development



1. User Mode - Kernel Mode Communication ntddk.h		Toolkit Communication
2. Direct Kernel Object Manipulation 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

DEMO





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,
                               * IOCTL_COMM_0,
                               * inBuffer, sizeof(inBuffer),
                               * 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

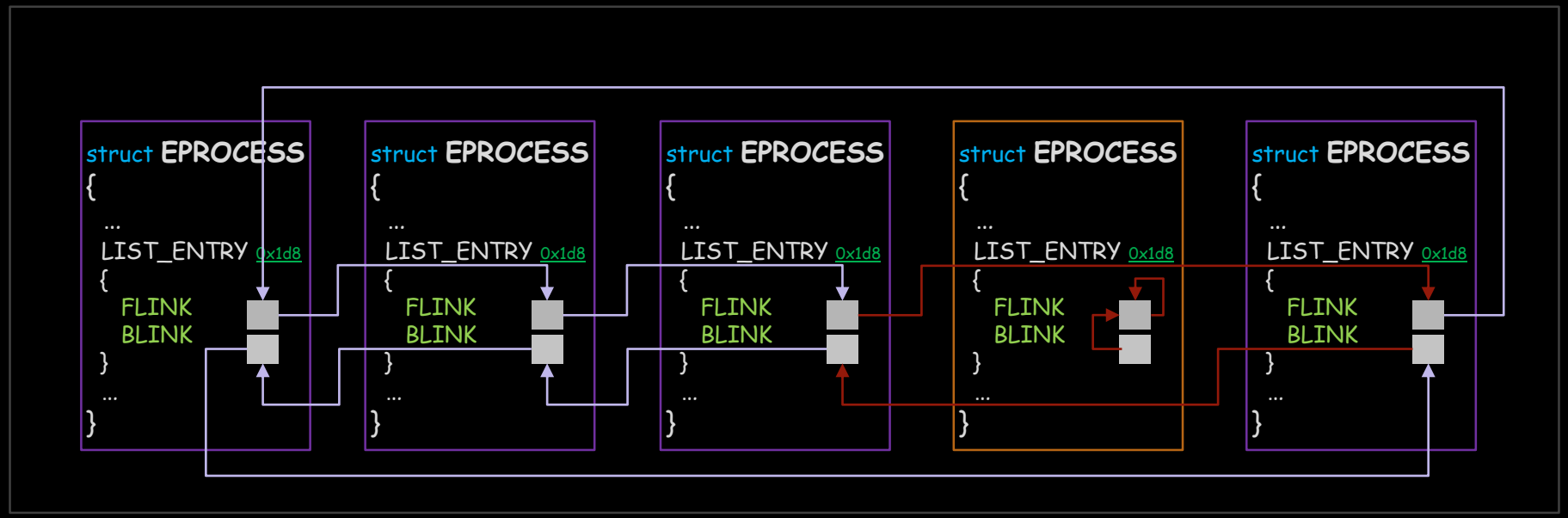
```
// ----------------------
#define IOCTL_COMM_0 CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)


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

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."

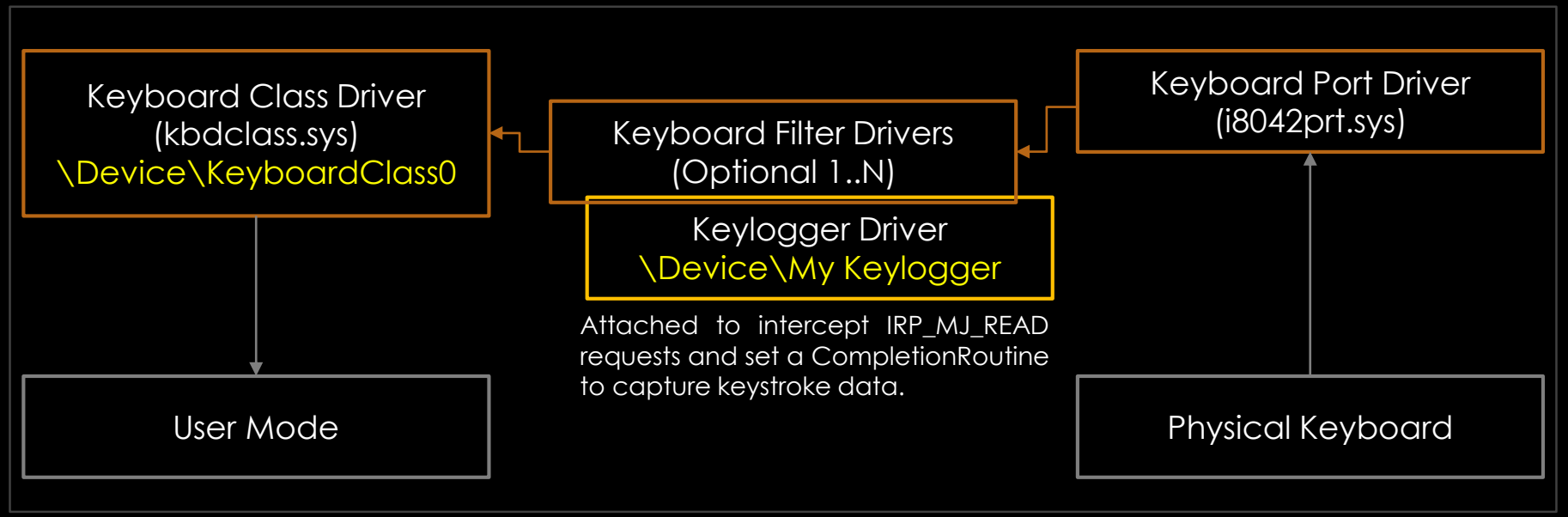




III

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 (a callback executed after an IRP has been processed by lower drivers, allowing access to data before it reaches the next stage), we can capture raw keystroke data before it propagates to user-mode applications like text editors or browsers."

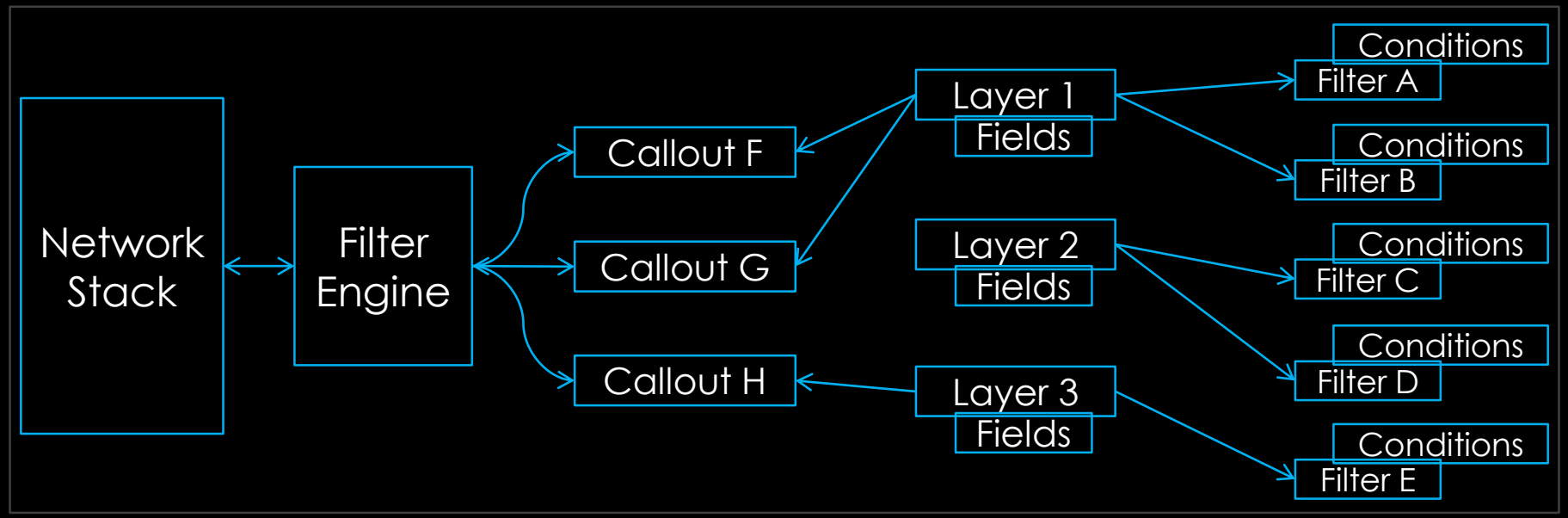




IV

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."

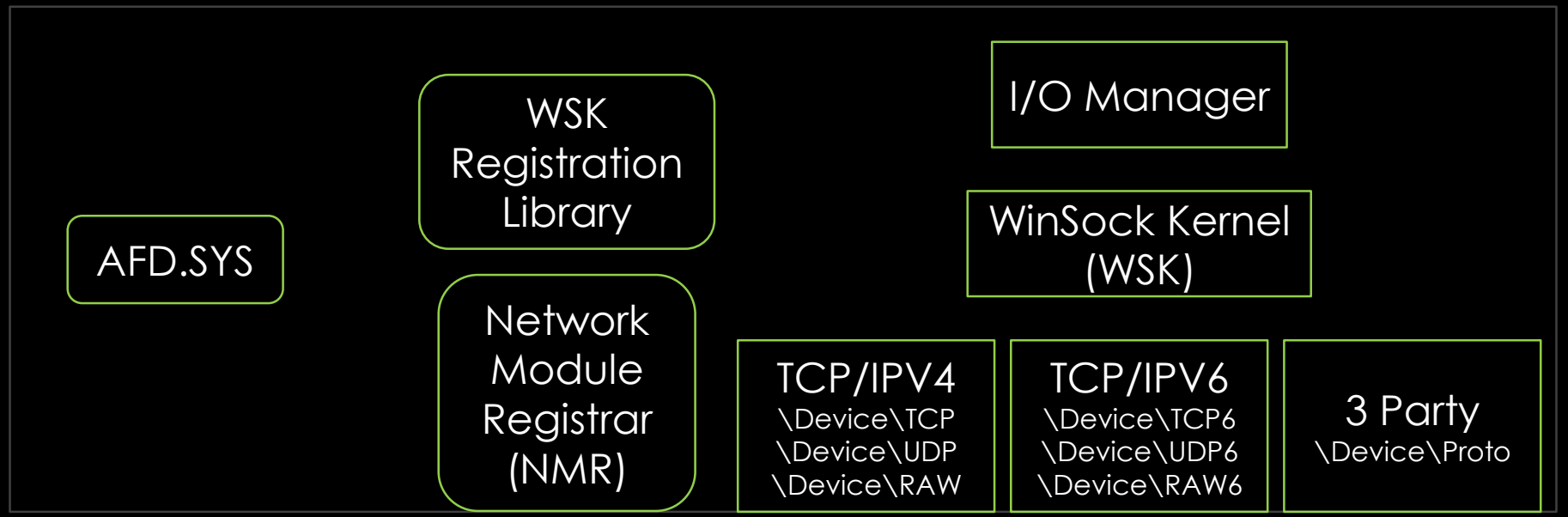




V

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."

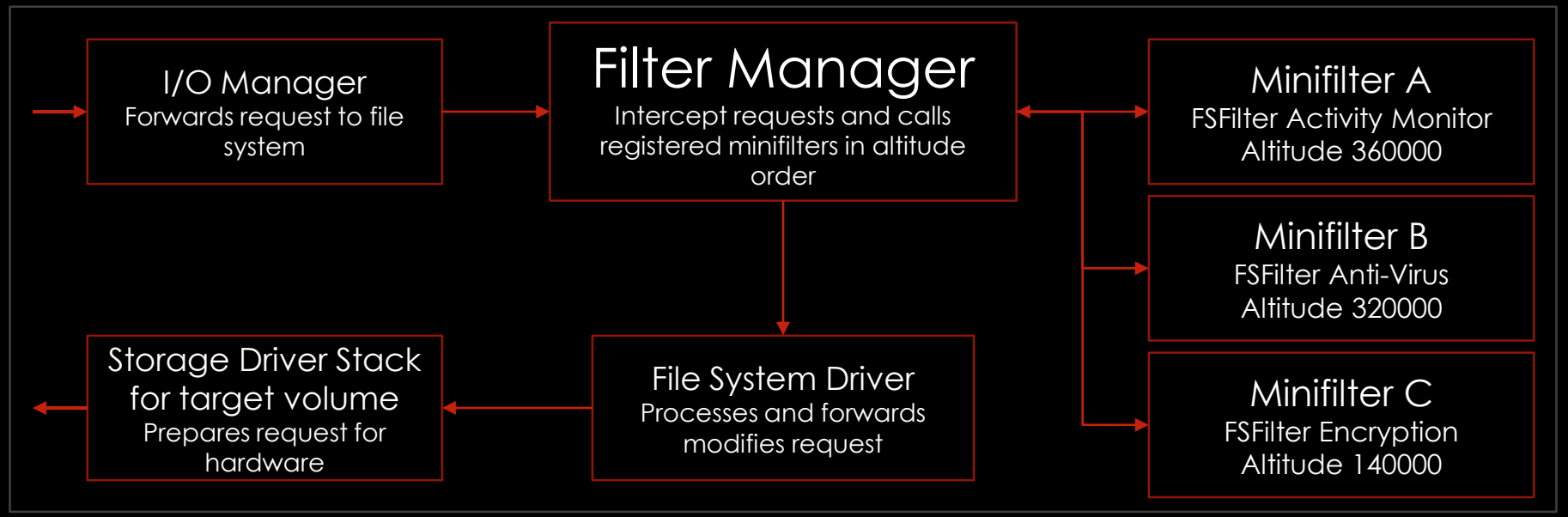


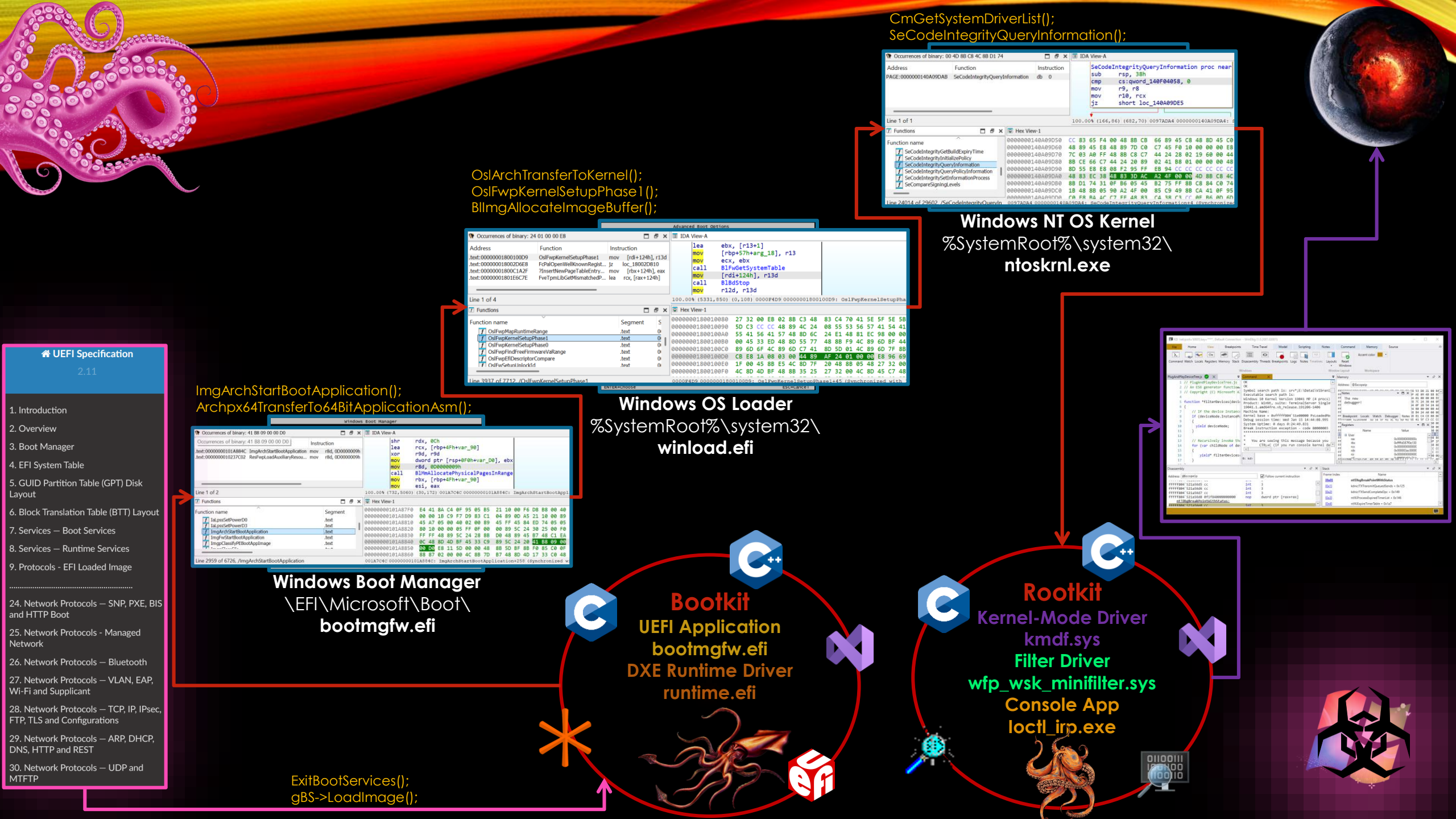


VI

HIDE FOLDERS

"MiniFilters attach to the file system stack to filter I/O operations. Using a PreOperation callback (triggered before the file system processes a request), access to files or directories can be explicitly denied by returning STATUS_ACCESS_DENIED or FLT_PREOP_COMPLETE. In the PostOperation callback (triggered after the request finishes), the DirectoryBuffer - which temporarily holds the directory listing - can be modified to remove specific entries, effectively making files and folders invisible to user-mode applications like File Explorer."




UEFI Specification
2.11
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
ImgArchStartBootApplication();
ArchpxTransferTo64BitApplicationAsm();
Windows Boot Manager
\EFI\Microsoft\Boot\
bootmgfw.efi
OslArchTransferToKernel();
OslFwpKernelSetupPhase1();
BlImgAllocateImageBuffer();
Windows OS Loader
%SystemRoot%\system32\
winload.efi
CmGetSystemDriverList();
SeCodeIntegrityQueryInformation();
Windows NT OS Kernel
%SystemRoot%\system32\
ntoskrnl.exe
Bootkit
UEFI Application
bootmgfw.efi
DXE Runtime Driver
runtime.efi
Rootkit
Kernel-Mode Driver
kmdf.sys
Filter Driver
wfp_wsk_minifilter.sys
Console App
Ioctl_irp.exe
ExitBootServices();
gBS->LoadImage();

DEMO



FROM UEFI TO KERNEL








THANK YOU 🐾



♥ In loving memory of Shira
01-01-2013 / 04-07-2025

UEFI Bootkit

- github.com/TheMalwareGuardian/**Abyss**

Kernel-Mode Rootkit

- github.com/TheMalwareGuardian/**Benthic**

Every resource you need to develop Bootkits/Rootkits

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

Automate Bootkits/Rootkits Development Environment

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

Contact:

- https://www.linkedin.com/in/**vazquez-vazquez-alejandro**
- https://www.linkedin.com/in/**mariasanjose**