Anti-Cast
antisyphon

BRING YOUR OWN VULNERABLE DRIVER (BYOVD) ATTACKS

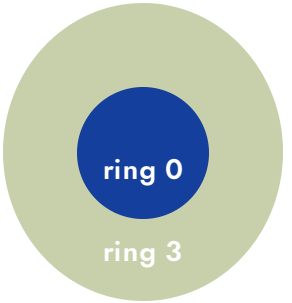
"THE WRONG BEER TO THE BARBEQUE"

ALISSA TORRES | @SIBERTOR

AGENDA

- Windows OS Driver Theory
- Windows Kernel Protections
  - Driver Signature Enforcement
  - Kernel Patch Protection
  - HyperGuard
- Recent BYOVD Adversary Campaigns
- 3 Facets of Exploited/Repurposed Drivers
  - LOLDriver
  - BYOVD
  - Legit Driver Abuse
- Detection and Blocking Strategies
  - LoLdrivers.io
  - MS Bad Drivers List
  - EDR Blocks

USER MODE VS KERNEL MODE

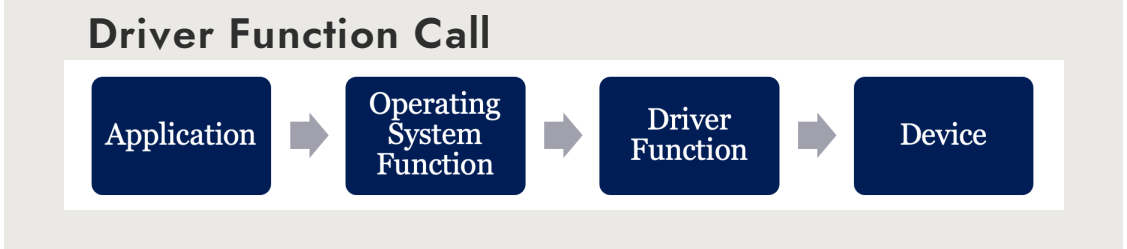


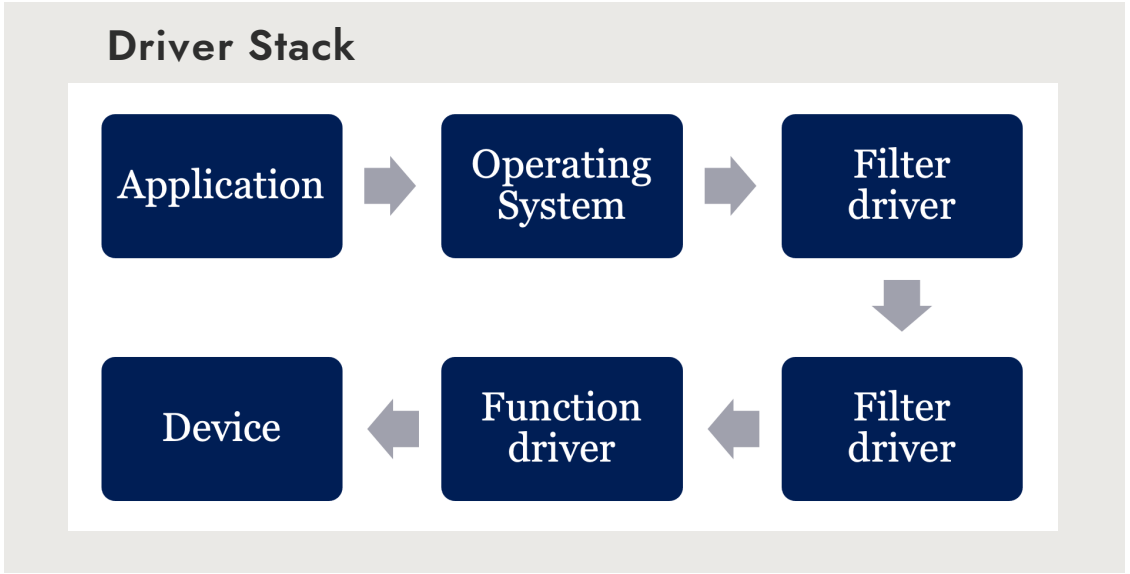
- Each process maintains its own virtual address space, running in isolation.
- User-mode processes can not directly access virtual addresses reserved for the operating systems.
- If a memory exception occurs in a user-mode application, the single process crashes.

- All code running in kernel-mode shares the same virtual address space.
- For example, a kernel-mode driver are not isolated from other drivers.
- If a memory exception occurs, the system crashes.

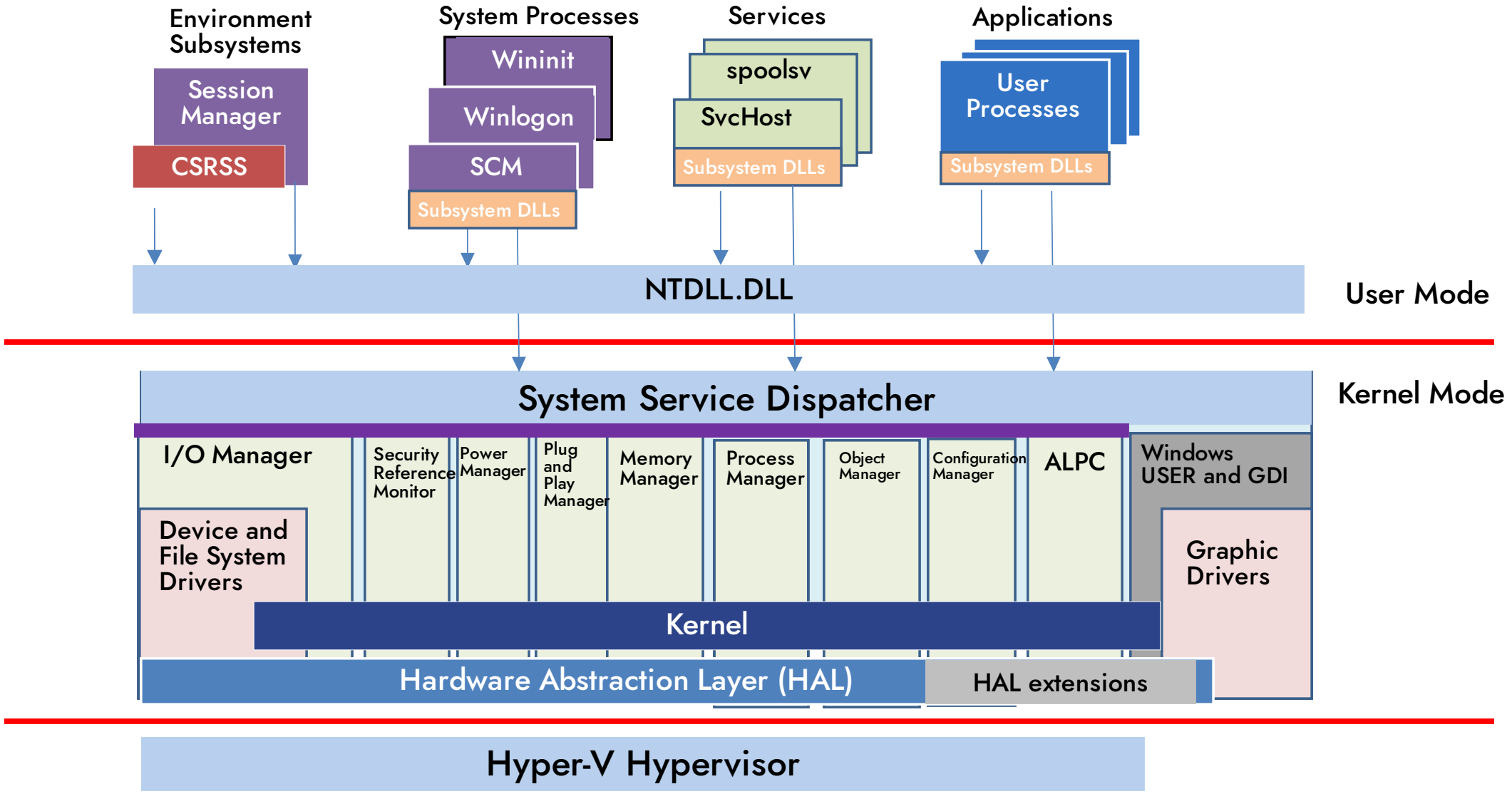
WHAT IS A DRIVER?

- A low-level software component that supports communication between the operating system and physical hardware components or software application.
- Not all drivers communicate directly with a device -> **filter** drivers.
- Other drivers communicate to kernel-mode code -> **software** drivers.





WINDOWS SYSTEM ARCHITECTURE



Sourced from Windows Internals Ed. 7

WHY DO BADDIES WANT TO ACCESS THE KERNEL?

- Executing code in kernel mode allows attackers to compromise higher privileged elements on the system.
- A malicious driver can unhook **AV/EDR callbacks** or hide exploitation/rootkit artifacts.
- Configuring a Windows service to load their malicious driver at boot, the attacker establishes persistence.

*Extend **dwell time...** and win*

BYOVD TIMELINE

- Aug 2012: **Shamoon** wiper abused **EldoS RawDisk** driver

---

- Dec 2021: **Cuba** threat group abused function in **Avast AntiRootkit** kernel driver to terminate AV & EDR solutions
- Oct 2022: **Lazarus** exploited vulnerable **Dell DBUtil hardware** driver (CVE-2021-21551)
- Oct 2022: **BlackByte** threat group exploited **MSI AfterBurner** driver (CVE-2019-16098) to unload drivers, disrupting operations.
- Nov 2022: **Backstab** abused **ProcExp.sys**
- Jan 2023: **Scattered Spider** dropped and exploited **Intel Ethernet Diagnostics** driver
- Apr 2023: **AuKill** abused **ProcExp.sys**

DRIVER SIGNATURE ENFORCEMENT

- Since 64-bit versions of Vista, Windows requires Kernel-Mode Code Signing (KMCS) of software that loads in kernel mode.
- Digital certificates are used to verify authenticity and integrity of Windows drivers.
- As of Win10, the driver signing policy requires "cross-signing", a second signature by Microsoft itself.

Attackers are left with 2 options:

1. Get their malicious driver digitally signed by a trusted Certificate Authority
2. Exploit a legitimate vulnerable driver

PATCHGUARD (KPP)

Kernel Patch Protection (KPP)

- Through numerous checks, PatchGuard identifies process list corruption, kernel image patching and critical system table manipulation.
- Upon detection, PatchGuard crashes the impacted system, making many previously common rootkit techniques ineffective.
- Upon system crash, a kernel-mode crash dump file with bugcheck code to support diagnostics.

TYPES OF DRIVER ABUSE

LoLDrivers

Vulnerable Drivers **Already Present** in the Environment that allow an adversary to "Live Off the Land"

Example: WinRing0.sys

BYOVD

Vulnerable Drivers **Dropped** by Threat Actor

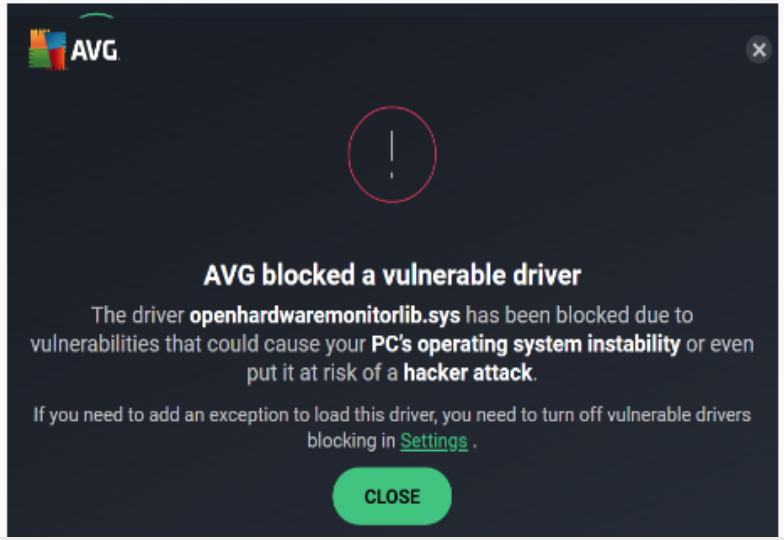
Example: ProcExp152.sys

Legit Driver Function Abuse

- SysInternals Drivers
- Endpoint Solutions Drivers

Example: Avast aswArPot.sys

EDR Alerts may not differentiate between the detected load of malicious, vulnerable or frequently abused drivers.



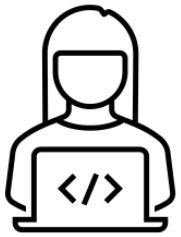
AVAST ANTI-ROOTKIT DRIVER

Cuba ransomware's use of the Avast anti-rootkit driver "aswArPot.sys"

- Initial function creates a handle to reference the recently installed Avast driver via CreateFileW API.
- Executable calls a function to find and terminate processes
  - CreateToolhelp32Snapshot -> Process32FirstW
- Executable cycles through CRC64 checksum values, each representing known AV or EDR processes.
  - Avast, AVP, McAfee, Sophos, Symantec, Kaspersky, SentinelOne, MalwareBytes, CarbonBlack
- Upon finding a match, the executable calls the Avast "process termination" function, passing the handle and matching process ID.
- The aswArPot.sys Avast driver interprets the IOCTL code as a signal to terminate a given process.

AUKILL (APR 2023)

- Defense evasion tool observed in multiple incidents recently, similar to Backstab (NOV 2022) & MalVirt (JAN 2023).
  - Used by attacker to disable EDR processes and services
  - With Admin privs, AuKill code is configured to load as a service.
- AuKill drops the SysInternals Process Explorer driver **PROCEXP152.SYS**
  - Writes outdated version 16.32 to C:\Windows\System32\Drivers path.
  - The Process Explorer driver can receive the IO control code IOCTL_CLOSE_HANDLE from user-mode applications.



Attacker gains initial access to host via phish

Attacker deprecates host security posture with BYOVD

Having neutralized protections, the attacker then encrypts or establishes backdoor.

AUKILL (APR 2023)

AuKill uses the Process Explorer driver to disable EDR processes prior to deploying a backdoor or ransomware (Medusa Locker, LockBit)

Step One. AuKill terminates EDR processes by sending IOCTL_CLOSE_HANDLE to ProcExp152.sys.

Step Two. AuKill forcibly kills processes via TerminateProcess.

Step Three. AuKill disables EDR specific services and continues to monitor to ensure they are not running.

Step Four. AuKill then unloads drivers by calling NTUnloadDriver and deletes associated service registry key.

TECH DEEP DIVE

MOST COMMON DRIVER VULNERABILITY TYPES

1 Function Calls from MSRs

example: AFD.sys

Overwriting Model Specific Register - AFD.sys vulnerability overwrote one byte in HAL dispatch table, pointed to by an MSR. Redirection to User-Mode where attacker code is located.

2 Unprotected IOCTL Requests

example: FudModule by Lazarus

3 Plug-and-Play Vulnerabilities

example: Print Nightmare

Offers adversaries the ability to load or unload code from untrusted user mode and is generally exploited to achieve privilege escalation.

4 Firmware Update Vulnerabilities

example: Reolink RLC-410W CVE-2022-21134

Firmware update drivers are built to write to onboard hardware instructions. Most commonly, adversaries exploit the lack of proper access control in these drivers. Commonly found in printer firmware and IoT devices.

5 UEFI & Boot Loader Vulnerabilities

example:

Errors in code that allow malware to bypass or disable secure boot to run unendorsed code.

SCATTERED SPIDER BYOVD (JAN 2023)

- eCrime adversary best known for credential phishing and social engineering attacks.
- CrowdStrike observed this actor dropping a vulnerable Intel Ethernet Diagnostics driver **iqvw32.sys** & **iqvw64.sys** (CVE-2015-2291) in the victim environment to bypass MS Defender for Endpoint, PAN Cortex EDR and SentinelOne.
- **CVE-2015-2291:** Prior to v.1.3.1.0, the Intel Ethernet diagnostics driver allows local users to cause a denial of service or possibly **execute arbitrary code** with kernel privileges via a crafted (a) 0x80862013, (b) 0x8086200B, (c) 0x8086200F, or (d) 0x80862007 IOCTL call.

WINRING0 DRIVER (APR 2023)

- Mining campaigns for Monero initially exploiting a fake Google Chrome update error screen to distribute Monero mining malware, implementing BYOVD.
- The malware used the BYOVD technique to abuse a bug in **WinRing0x64.sys**, also known as OpenHardwareMonitor.sys, to gain system privilege access.
- OpenHardwareMonitor.sys is used to access low level information from your hardware, such as CPU load, fan speeds, temperature sensors, and more.
- **CVE-2020-14979**:This driver creates a device object on the system which all users can access due to a NULL DACL, as shown in WinObj below. This means that any user of the system, can issue requests (IOCTLs) to this driver.

https://cyware.com/news/fake-chrome-updates-used-for-malware-distribution-4135643b

HYPERVISOR PROTECTED CODE INTEGRITY (HVCI)

- Also known as Memory Integrity
- A component of Virtualization-Based Security (VBS) that prevents users with elevated privileges from being able to read and write to kernel memory.
- VBS uses the hypervisor to create an isolated virtual environment for the root of OS trust - assumes the kernel can be compromised.

LOLDRIVERS.IO

Living Off the Land Drivers Project

- Knowledge base of Windows drivers used by adversaries to bypass security controls and carry out attacks
- Community-driven open-source project
  - Curated list of vulnerable and malicious drivers list accessible via API
  - Pre-built Sysmon Config & WDAC Policy
  - Sigma, Yara, ClamAV and SysMon Detection Rules
  - Detailed knowledge base of malicious/vulnerable drivers with downloadable samples

MICROSOFT BAD DRIVERS LIST

The vulnerable driver blocklist is designed to help harden systems against third party-developed drivers across the Windows ecosystem with any of the following attributes:

- Known security vulnerabilities that can be exploited by attackers to elevate privileges in the Windows kernel
- Malicious behaviors (malware) or certificates used to sign malware
- Behaviors that aren't malicious but circumvent the Windows Security Model and can be exploited by attackers to elevate privileges in the Windows kernel

AS OF WIN11 2022 UPDATE, THE BLOCKLIST IS ENABLED BY DEFAULT.

The Blocklist is also enabled when Memory Integrity (HVCI), Smart App Control or S Mode is active.

UPDATED 1-2 TIMES A YEAR.

FUTURE UPDATES WILL INCLUDE AN OPTION TO DISABLE THIS FEATURE.

https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/microsoft-recommended-driver-block-rules

MICROSOFT BAD DRIVERS LIST

In OCT 2022, Microsoft was criticized for not doing enough to protect systems from malicious driver loading.

- Updates to MS Malicious Driver list lapsed for a period of 2.5 years.

```
<FileRules>
  <Allow ID="ID_ALLOW_ALL_1" FriendlyName="" FileName="*" />
  <Allow ID="ID_ALLOW_ALL_2" FriendlyName="" FileName="*" />
  <Deny ID="ID_DENY_AGENT64_SHA1" FriendlyName="Agent64\05f052_4045ae_694848_8cb62c_b1d962 Has
  <Deny ID="ID_DENY_AGENT64_SHA256" FriendlyName="Agent64\05f052_4045ae_694848_8cb62c_b1d962 H
  <Deny ID="ID_DENY_AGENT64_SHA1_PAGE" FriendlyName="Agent64\05f052_4045ae_694848_8cb62c_b1d96
  <Deny ID="ID_DENY_AGENT64_SHA256_PAGE" FriendlyName="Agent64\05f052_4045ae_694848_8cb62c_b1d
  <Deny ID="ID_DENY_ASIO_32_SHA1" FriendlyName="ASIO32.sys Hash Sha1" Hash="D569D4BAB86E70EFBC
  <Deny ID="ID_DENY_ASIO_32_SHA256" FriendlyName="ASIO32.sys Hash Sha256" Hash="80599708CE61EC
  <Deny ID="ID_DENY_ASIO_32_SHA1_PAGE" FriendlyName="ASIO32.sys Hash Page Sha1" Hash="80FA962B
  <Deny ID="ID_DENY_ASIO_32_SHA256_PAGE" FriendlyName="ASIO32.sys Hash Page Sha256" Hash="9091
  <Deny ID="ID_DENY_ASIO_32_SHA1_1" FriendlyName="ASIO32.sys Hash Sha1" Hash="5A7DD0DA0AEE0BDE
  <Deny ID="ID_DENY_ASIO_32_SHA256_1" FriendlyName="ASIO32.sys Hash Sha256" Hash="92EDD48DFAC0
  <Deny ID="ID_DENY_ASIO_32_SHA1_PAGE_1" FriendlyName="ASIO32.sys Hash Page Sha1" Hash="1ACC7A
  <Deny ID="ID_DENY_ASIO_32_SHA256_PAGE_1" FriendlyName="ASIO32.sys Hash Page Sha256" Hash="F8
  <Deny ID="ID_DENY_ASIO_32_SHA1_2" FriendlyName="ASIO32.sys Hash Sha1" Hash="55AB7E27412ECA43
  <Deny ID="ID_DENY_ASIO_32_SHA256_2" FriendlyName="ASIO32.sys Hash Sha256" Hash="C1B41D6B9144
  <Deny ID="ID_DENY_ASIO_32_SHA1_PAGE_2" FriendlyName="ASIO32.sys Hash Page Sha1" Hash="1E7C24
  <Deny ID="ID_DENY_ASIO_32_SHA256_PAGE_2" FriendlyName="ASIO32.sys Hash Page Sha256" Hash="10
  <Deny ID="ID_DENY_ASIO_64_SHA1" FriendlyName="ASIO64.sys Hash Sha1" Hash="E5C090903A20744BA3
  <Deny ID="ID_DENY_ASIO_64_SHA256" FriendlyName="ASIO64.sys Hash Sha256" Hash="9DCFD796E244D0
  <Deny ID="ID_DENY_ASIO_64_SHA1_PAGE" FriendlyName="ASIO64.sys Hash Page Sha1" Hash="CA5FF4EB
  <Deny ID="ID_DENY_ASIO_64_SHA256_PAGE" FriendlyName="ASIO64.sys Hash Page Sha256" Hash="D884
  <Deny ID="ID_DENY_ASIO_64_SHA1_1" FriendlyName="ASIO64.sys Hash Sha1" Hash="C92148D0666F2235
```

https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/microsoft-recommended-driver-block-rules

DETECTION & BLOCKING STRATEGIES

- Baseline your Environment's Active Loading Drivers
- Alert and/or Block known vulnerable and abused drivers

```
sc.exe create awSP_ArPot2 binPath= C:\Windows\Temp\asmWrPot.sys type= kernel
```

- Use a Windows Defender Application Control Policy (WDAC) to block list of drivers.
- Enable Attack Surface Reduction (ASR) rule "Block abuse of exploited vulnerable signed drivers" to prevent an application from writing a vulnerable signed driver to disk.

REFERENCES

- Learn.Microsoft: Microsoft Recommended Driver Block Rules

  https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/microsoft-recommended-driver-block-rules

- I solemnly swear my driver is up to no good: Hunting for Attestation Signed Malware - Mandiant blog

  https://www.mandiant.com/resources/blog/hunting-attestation-signed-malware

- “These are the Drivers You are Looking For”

  https://www.splunk.com/en_us/blog/security/these-are-the-drivers-you-are-looking-for-detect-and-prevent-malicious-drivers.html

- Learn.Microsoft: What is a Driver?

  https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/what-is-a-driver-