Unveiling BYOVD Threats: Malware's Use and Abuse of Kernel Drivers

Andrea Monzani
Simone Aonzo

Antonio Parata
Davide Balzarotti

Andrea Oliveri
Andrea Lanzi

UNIVERSITÀ
DEGLI STUDI
DI MILANO

EURECOM
Sophia Antipolis

NDSS 2026

Motivation

In the **BYOVD (Bring Your Own Vulnerable Driver)** technique attackers use legitimate, signed drivers containing exploitable flaws to achieve:

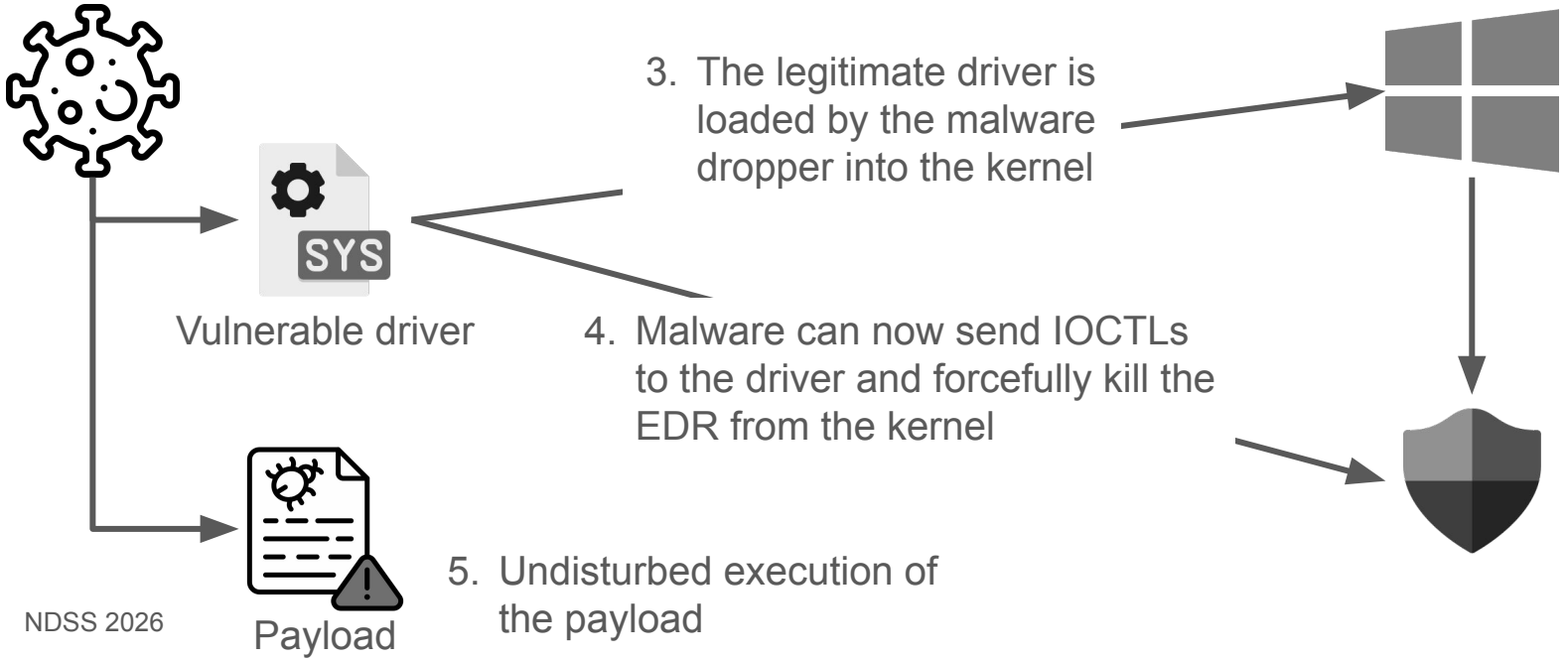
Kernel-level privilege escalation
EDR/Defender termination
Arbitrary memory access
Unsigned driver loading

BYOVD attacks have been **detected also in state-sponsored espionage and ransomware operations** (e.g., Qilin, 2025).

BYOVD Attacks

2. Malware dropper needs to disrupt EDR functionalities before deploying the final payload

1. Windows protects the running EDR from user-mode attacks by launching it as a Protected Process



3. The legitimate driver is loaded by the malware dropper into the kernel

4. Malware can now send IOCTLs to the driver and forcefully kill the EDR from the kernel

5. Undisturbed execution of the payload

Research Problem Statement

Existing sandboxes: user-mode visibility only, blind to kernel-space abuse.

Can we monitor BYOVD techniques in a sandbox?

Research challenges

> **Kernel** interactions span **multiple abstraction layers**
> Current systems (e.g., HookScout) fail against **obfuscation & dynamic resolution**
> Need **visibility across user ↔ kernel boundaries**

Contributions

1. **Build a taxonomy** to outline the typical stages of **BYOVD** attacks and the APIs commonly leveraged. Focus on **observable suspicious behaviors**

2. Design a **virtualization-based sandbox for kernel tracing**
   DRAKVUF extension to observe kernel-level behaviors.

3. **Build and evaluate real-world malware datasets** to detect both known and novel cases. Identified 48 suspicious drivers → **7 new vulnerable drivers disclosed** to Microsoft & vendors.

Taxonomy of Observable Behaviors (1° contribution)

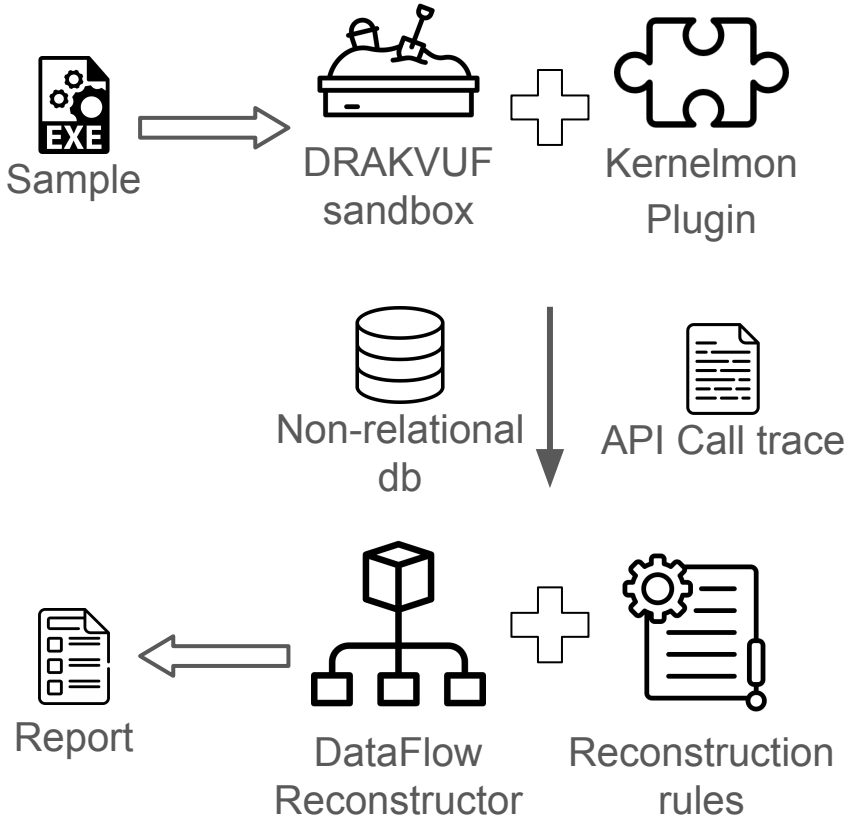
ID	Behavior	Description	Example API
B1	Protected Process Termination	Kill EDRs	*ZwTerminateProcess*
B2	Privileged Handle Leaks	Privileged kernel-calls opening user-mode handles	*ZwOpenProcess*
B3	Unsafe Pool Allocation	Allocation of W+X kernel memory	*ExAllocatePoolWithTag*
B4	Dynamic Code Execution	Execution of W+X kernel memory	__
B5	MismatchedMemoryMapping	Memory pages remapping	*MmMapIoSpace*
B6	Code Integrity Tampering	Disable DSE	*g_CiOptions*
B7	Remote Handle Closure	Cross-Process Handle Attacks	*ObCloseHandle*
B8	Arbitrary R/W (excluded) **	__	__

Sandbox Architecture (2° contribution)

Xen hypervisor + DRAKVUF sandbox.
Custom plugin: kernelmon.

Hooks on observable behaviors:

➔ Driver load/unload kernel routines (*MmLoadSystemImageEx*)
➔ IOCTL Handlers / Driver-registered callbacks
➔ Kernel function/structures

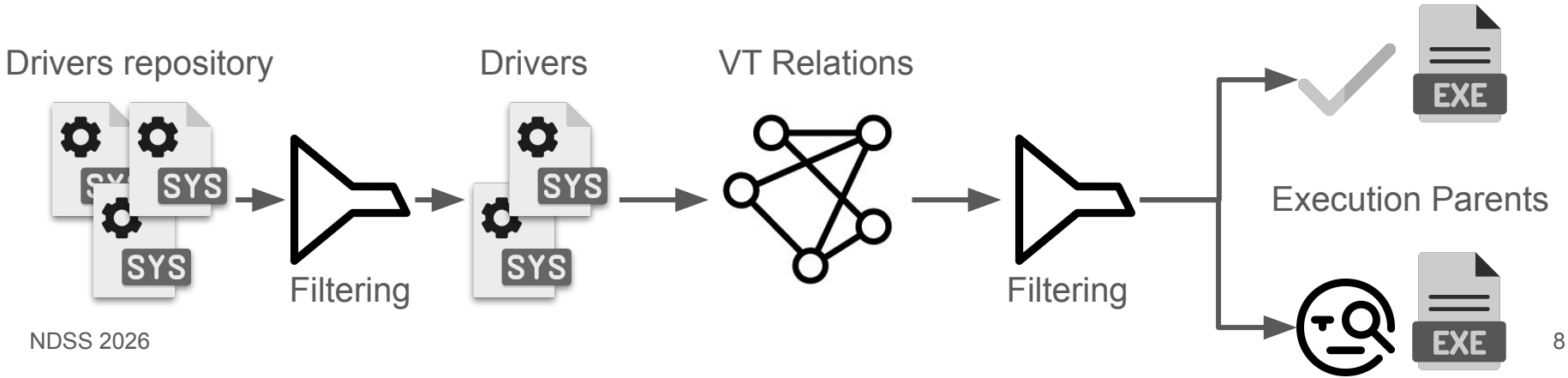


Dataset construction - (3° contribution)

We **gather driver samples** observed in the wild for each dataset

LolDrivers for a ground-truth-like dataset
VT Query of a list of **Abusable Imports** for a test dataset

We **remove drivers that are unsupported on modern Windows systems**, i.e., 32-bit drivers, not digitally signed (or with an invalid signature).



Dataset

Dataset	Source	Drivers	Executables		Final Drivers
			VT score < 6	VT score > 49	
KVD	LOLDrivers Microsoft Block List	917 (392 distinct names)	1948	1047	162 (distinct names)
PVD	VirusTotal (YARA Abusable Imports)	5589 (2000 distinct names)	3582	2202	611 (distinct names)

Total: 8,779 samples, 773 distinct drivers potentially observable by the sandbox.

Analysis results

Dataset	Driver Groups	Loaded	With an observed behavior	Without an observed behavior
KVD	162	106 (65.43%)	44 (41.51%)	62 (58.49%)
$KVD_{VT<6}$	139	91 (65.47%)	28 (30.77%)	63 (69.23%)
$KVD_{VT>49}$	95	64 (67.37%)	28 (43.75%)	36 (56.25%)
PVD	611	118 (19.31%)	48 (40.68%)	70 (59.32%)
$PVD_{VT<6}$	524	97 (18.51%)	38 (39.18%)	59 (60.82%)
$PVD_{VT>49}$	318	56 (17.61%)	19 (33.93%)	37 (66.07%)

Key Observed Behaviors

Chain	KVD Observed Samples	PVD Observed Samples
ProtectedServiceTermination	45	26
PrivilegedUserHandleFromKernelLeak	56	16
CodeIntegrityTampering	2	5
UnsafePoolAllocation → DynamicCodeExecution	30	3

False Positives and False Negatives

No classical false positives (excluding those attributable to implementation bugs):

- **Alerts** are tied to an **actual execution trace, not a heuristic guess**.
- **Interpretation still requires human intervention**

Sources of False Negatives

- **Behavior not covered** / unknown exploitation techniques.
- Implementation issues (bugs in hooks or reconstruction logic).
- **Untriggered code paths during execution** (time limits, missing stimuli, evasive behavior).

Case Studies

7 new vulnerable drivers disclosed to Microsoft & vendors

kavservice.sys - Malicious Driver

IOCTL 0x222000 kills EDR processes.
Calls ZwOpenProcess → ZwTerminateProcess.

termdd.sys - Old Microsoft driver

Arbitrary write to ci!g_CiOptions → disables Code Integrity.
Still loadable under Windows 10/11.

probmon.sys - Minifilter driver

Misuse via *FilterSendMessage* to terminate protected processes.
Assigned CVE-2024-26506 (MITRE, February 2024).

Future Works

Combine static and dynamic analysis to achieve a comprehensive view of BYOVD attacks
Integrating both results helps **overcome** missing stimuli and execution parent **limitations**

Automatic exploit generation system for driver vulnerabilities
Structural similarities among driver flaws create opportunities for automation
Explore both rule-based and LLM-based approaches

Conclusions

Built a taxonomy focused on observable suspicious behaviors to shine a light on BYOVD attacks and **built real-world malware datasets**

Sandbox successfully reconstructed multi-stage behavior chains, consistent with real BYOVD exploitation workflows.

Initial step toward closing the sandbox visibility gap and delivering insights into BYOVD attacks.

Plugin code, datasets and results can be found at https://zenodo.org/records/17047559

Question time.
Thank you very much

Discussion

Scalability barrier: can we **trace all memory writes dynamically**?

- Extend monitored memory regions
- Trace write instruction inside driver code

Sandbox evasion: a small fraction of detection techniques identify hypervisor

- Static analysis can help overcome this issue

VBS impact: some **vulnerable drivers still load under HVCI**

- Simulate VBS-enabled configurations for deeper insight