JOT
JOURNAL OF
OBJECT TECHNOLOGY

HyperEvade: Countering Anti-Debugging Techniques and Enhancing Transparency in Nested Virtualization using HyperDbg

Björn Ruytenberg and Mohammad Sina Karvandi
Vrije Universiteit Amsterdam, The Netherlands

ABSTRACT Modern malware increasingly employs sophisticated anti-debugging and anti-virtualization techniques to evade analysis, particularly targeting artifacts left by virtualization and nested virtualization environments such as VMware Workstation, Hyper-V, and KVM. HyperDbg, an open-source hypervisor-level debugger, introduces advanced mechanisms to mitigate both its own hypervisor footprints and those of the underlying nested virtualization stack. In this paper, we demonstrate the capabilities of adding a transparency layer on top of the HyperDbg debugger to detect, mitigate, and bypass common and advanced anti-debugging methods leveraged against such environments. We refer to it as the *HyperEvade* project. Although achieving complete transparency remains infeasible, it significantly raises the bar for malware attempting to detect analysis environments, making evasion substantially more difficult. We further highlight the critical importance of these techniques in practical malware analysis workflows, particularly in scenarios involving snapshot restoration for analyzing and debugging internal malware behavior. By reducing observable artifacts, HyperEvade enhances the reliability of snapshot-based analysis and debugging, allowing researchers to stealthily investigate and understand the inner workings of evasive malware without premature detection or execution of anti-analysis payloads.

KEYWORDS Anti-Debugging, Anti-Virtualization, Nested Virtualization, Debugging Malware, Binary Analysis.

Introduction

Dynamic malware analysis and debugging are critical steps in understanding the behavior of sophisticated threats. Unlike static analysis, which can be hindered by heavy obfuscation, packing, or encryption, dynamic analysis enables researchers to observe real-time execution, track control flow, and extract hidden payloads. While static analysis tools like IDA Pro, Ghidra, Binary Ninja, and Radare are invaluable for reverse engineering, they become significantly less effective when dealing with packed or heavily obfuscated malware. In such cases, a debugger becomes the primary tool for malware analysts to unpack, deobfuscate, or automate the analysis of the malware's behavior.

Since regular user-mode or kernel-mode debuggers can leave detectable artifacts, hypervisor-based debuggers are often preferred for stealthier analysis. Hypervisor-level debuggers not only minimize their footprint but also provide system-level access, allowing observation of behaviors that occur even above the operating system kernel by employing hypervisor-level capabilities.

Hypervisor-based debuggers, among their benefits, come with certain artifacts that expose the presence of the hypervisor. This problem becomes even more serious in nested virtualization environments, where clues left by both the main hypervisor and the nested one can be used by malware to detect that it is being watched. Small differences in CPU behavior, hardware details, timing measurements, or system information can reveal that the malware is running inside a debugger or a virtual machine. Once detected, malware might shut itself down early or

JOT reference format:
Björn Ruytenberg and Mohammad Sina Karvandi. *HyperEvade: Countering Anti-Debugging Techniques and Enhancing Transparency in Nested Virtualization using HyperDbg.* Journal of Object Technology. Vol. 25, No. 1, 2026.
http://dx.doi.org/10.5381/jot.2026.25.1.a8

trigger fake behavior to mislead the analyst.

To address these issues, we present a transparency layer over the HyperDbg (Karvandi et al. 2022) debugger specifically designed to enhance transparency in nested virtualization environments. By not using OS-level debugging APIs, HyperDbg minimizes detectable artifacts at multiple layers, enabling analysts to conduct stealthy, fine-grained debugging sessions even against highly evasive malware. By mitigating both traditional anti-debugging techniques and emerging nested virtualization detection methods, HyperDbg significantly improves the reliability and effectiveness of dynamic malware analysis workflows.

Mitigation Methods

VM Detection Techniques and Mitigations: Malware can detect virtualized environments using a broad range of heuristics. The transparency layer offers or facilitates mitigations against multiple categories of these techniques:

CPU and Hypervisor Detection: Many detection methods query CPU features to find virtualization artifacts. These include checking the hypervisor vendor ID (via `CPUID` instructions), looking for anomalies in CPU brand strings, hypervisor flags, and reserved CPUID leaves. Specific signatures associated with platforms like KVM or Intel's KGT branch can also be probed. Mitigations typically involve masking or spoofing CPUID (using nested VM-exits) to hide hypervisor presence.

Timing Analysis: Virtual machines often introduce timing inconsistencies due to emulation overheads or event handling delays. Malware may measure execution timings to infer virtualization. The timing resources include TSC (Time Stamp Counter) or system-wide resources like external devices, HPET (High Precision Event Timer), Performance Counters, APIC (Advanced Programmable Interrupt Controller) Timers, Windows APIs, or timing thread approaches (Schwarz et al. 2017).

Mitigations involve introducing random jitter or normalizing timing outputs to appear more like bare-metal systems by employing VM-exits for TSC (`RDTSC/RDTSCP`), PMC (`RDPMC`), Different MSR registers using MSR Bitmap, monitoring for system-calls querying timers, virtualizing APIC, or monitoring for PIO/MMIO ranges of external devices. Timing-based detection methods are less common in modern malware, because Windows 11 enforces Virtualization-Based Security (VBS) by default. As a result, malware must assume it is running inside a hypervisor to avoid triggering false positives.

Windows-Specific Detection: Malware may search for VM indicators specific to Windows environments, such as `MSSMBIOS` registry keys, loaded DLLs, known registry values, or system mutexes.

Mitigations involve monitoring system-calls that query the information related to VM and spoofing registry/file entries.

CPU and Hardware Analysis: By examining hardware characteristics, malware can identify virtualized environments. Checks include inspecting processor and core counts, temperature sensors, VHD boot status, thread numbers, driver names, and hypervisor memory pools.

Mitigations involve adjusting VM configurations to emulate realistic hardware profiles.

Network Analysis: VM network adapters often use MAC addresses tied to virtualization vendors. Malware can detect these known prefixes or look for VirtualBox network drivers. Mitigations include randomizing MAC addresses and renaming network drivers.

Hardware Information: VMs may have default or invalid chassis, device, and firmware information, such as missing thermal sensors, suspicious PCI bridge names, or generic BIOS details. Malware can also scan GPU capabilities and strings to detect virtualization. Mitigations include modifying DMI tables, spoofing firmware signatures, and emulating realistic hardware characteristics (e.g., intercepting I/O ports querying PCIe to spoof vendor ID).

Instruction Set Analysis: Certain CPU instructions, such as SIDT, SGDT, SLDT, or querying system registers can behave differently in virtual environments. Malware can use these differences to detect VMs. Some techniques rely on specific features, such as VMware I/O port backdoor. Mitigations involve emulating native CPU instruction behavior and hiding special I/O ports by intercepting VM-exits related to `IN` and `OUT` instructions.

Filesystem and Storage Analysis: Malware may search for files, drivers, or directories unique to VM platforms like QEMU, KVM, or VirtualBox. Moreover, VMs often use default or small-sized disks. Malware can check disk sizes, disk serial numbers, or memory allocation patterns.

Mitigations requires intercepting system-calls and spoofing filesystem traces.

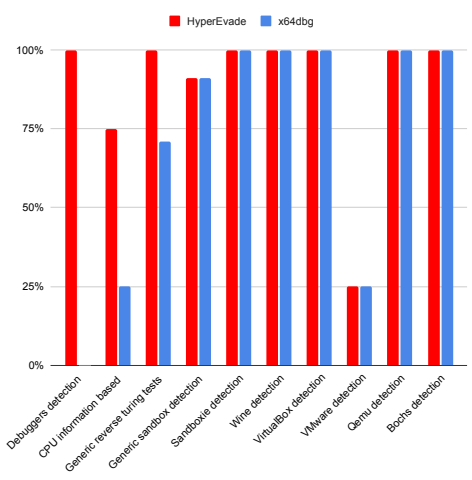
Process Analysis: Running processes related to VM services, such as VMware Tools or QEMU guests, can reveal a virtualized environment. Mitigations involve masking or terminating these processes (e.g., through system-call monitoring).

Specialized Techniques: More advanced checks include analyzing CPU thread counts (e.g., detecting odd thread numbers), probing memory regions for VM signatures, or exploiting low-level system features like the OSXSAVE instruction. Mitigations require deep modifications to the hypervisor, and CPU emulation layers to mimic bare-metal behavior accurately.

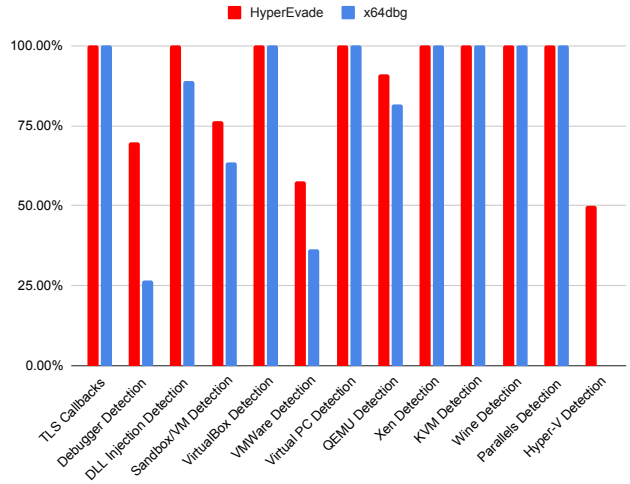
Evaluation

To evaluate the effectiveness of HyperEvade, we tested it against two widely used anti-analysis benchmark projects: *Pafish* and *Al-khaser*. These projects implement a comprehensive range of anti-debugging and anti-virtualization techniques designed to detect the presence of reverse-engineering tools, debuggers, and hypervisors.

Figure 1a presents the comparison between x64dbg (Ogilvie 2025) (with ScyllaHide (Developers 2025)) and HyperDbg (H. Developers 2025) (with HyperEvade) on the Pafish (Ortega 2025) test suite. As shown, HyperEvade consistently bypassed a larger set of detection mechanisms. While x64dbg managed to evade certain generic sandbox detection and debugging checks, its coverage was limited in CPU information–based detection and virtualization-related detection. In contrast, HyperEvade achieved close to complete coverage across nearly all categories, including debugger detection, sandbox detec-



(a) Pafish Benchmark Coverage.



(b) Al-Khaser Benchmark Coverage.

Figure 1 Comparison of HyperEvade coverage against x64dbg on Pafish (left) and Al-khaser (right) benchmarks.

tion, Wine, VirtualBox, QEMU, and Bochs detection. This demonstrates HyperEvade's ability to provide a more resilient debugging environment against typical anti-virtualization defenses.

Similarly, Figure 1b shows the results obtained using the Al-khaser (Faouzi 2025) project. Al-khaser includes an even broader and more sophisticated set of detection techniques compared to Pafish. Again, HyperEvade outperformed x64dbg by handling a greater number of detection checks successfully. This highlights the robustness and adaptability of HyperEvade across diverse anti-analysis strategies.

It is important to note, however, that achieving 100% evasion across all detection techniques cannot be guaranteed. The landscape of anti-debugging and anti-virtualization methods continues to evolve, and new techniques may still succeed in identifying the presence of a hypervisor or debugger. Therefore, HyperEvade should be considered as a strong step toward resilient stealth debugging, but not as a complete solution.

Availability

HyperEvade is open-source and publicly available[1] to foster security research.

Conclusion

Our work on top of HyperDbg demonstrates that substantial increases in transparency are achievable even in complex nested virtualization environments. Although perfect invisibility remains unattainable due to the fundamental limitations of software-based virtualization, HyperDbg significantly raises the bar for malware attempting to evade analysis. Its modular and open-source design enables continuous improvement and adaptation to emerging anti-virtualization techniques, making it a valuable tool for security researchers engaged in stealthy malware analysis, debugging, and reverse engineering.

[1] https://url.hyperdbg.org/hyperevade

Acknowledgments

We would like to thank the anonymous reviewers for their feedback. This work was supported by NWO through project "INTERSECT", and by the European Union's Horizon Europe programme under grant agreement No. 101120962 ("Rescale").

References

Developers. (2025). *Advanced usermode anti-anti-debugger.* https://github.com/x64dbg/ScyllaHide.

Developers, H. (2025). *State-of-the-art native debugging tools.* https://github.com/HyperDbg/HyperDbg.

Faouzi, A. (2025). *Public malware techniques used in the wild: Virtual machine, emulation, debuggers, sandbox detection.* https://github.com/ayoubfaouzi/al-khaser.

Karvandi, M. S., Gholamrezaei, M., Khalaj Monfared, S., Meghdadizanjani, S., Abbassi, B., Amini, A., … Schwarz, M. (2022). Hyperdbg: Reinventing hardware-assisted debugging. In *Proceedings of the 2022 acm sigsac conference on computer and communications security* (pp. 1709–1723).

Ogilvie, D. (2025). *An open-source user mode debugger for windows. optimized for reverse engineering and malware analysis.* https://github.com/x64dbg/x64dbg.

Ortega, A. (2025). *Pafish is a testing tool that uses different techniques to detect virtual machines and malware analysis environments in the same way that malware families do.* https://github.com/a0rtega/pafish.

Schwarz, M., Weiser, S., Gruss, D., Maurice, C., & Mangard, S. (2017). Malware guard extension: Using sgx to conceal cache attacks. In *Detection of intrusions and malware, and vulnerability assessment: 14th international conference, dimva 2017, bonn, germany, july 6-7, 2017, proceedings 14* (pp. 3–24).