BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

Improving Detection and Protection Against Anti-Sandbox Techniques

Bc. Jakub Kratochvíl*

Abstract
Automated sandboxes are essential for scaling malware analysis, but sophisticated samples increasingly employ anti-sandbox techniques that detect virtualized environments and suppress malicious behavior. This thesis proposes the design and implementation of three modules for the CAPEv2 sandbox environment: a post-analysis detector for evasive early exits, BIOS-level hardening of the analysis virtual machine against firmware fingerprinting, and an automated direct system call tracer that captures behavior that bypasses user-mode API hooks. Development is supported by 82 simulation samples across 30 anti-sandbox techniques, used to assess existing signature coverage and guide detection work. The resulting 40 CAPEv2 detection signatures with evasive exit detection support, the direct system call tracer, and the hardened BIOS are deployed internally in partnership with Gen Digital Inc.

*xkrato67@vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

[Motivation] Malware authors actively develop anti-sandbox techniques to detect analysis environments and suppress malicious behavior, rendering analysis results incomplete or misleading [1]. Closing gaps in sandbox resilience and enhancing anti-sandbox detection capabilities is not purely for academic research but to address existing limitations, which is why this thesis is conducted in partnership with Gen Digital Inc.

[Problem definition] The goal of this work is to address four concrete limitations identified in the current internal analysis environment, which is built on customized sandboxes. Specifically, the areas of improvement are: (i) low detection signature coverage for anti-sandbox techniques in CAPEv2, (ii) gaps in virtual machine hardening that leave virtualization indicators exposed in firmware tables, (iii) the absence of a post-analysis mechanism for detecting evasive exits following anti-sandbox checks, and (iv) an architectural gap by which direct system calls bypass CAPEv2's user-mode analysis entirely.

[Existing solutions] Automated malware analysis is handled by open-source sandboxes such as CAPEv2, or by commercial services such as Joe Sandbox, ANY RUN, and Hybrid Analysis [1]. CAPEv2's open, customizable design makes it the natural foundation for the internal analysis environment, which this work seeks to improve and extend.

[Our solution] We extend CAPEv2 with three complementary modules that, together, address the identified weaknesses. The first is a post-analysis module that detects evasive early exits and emits a dedicated verdict into the signature results. The second hardens the virtual machine at the firmware level by modifying and obfuscating string literals within SeaBIOS so that common VM fingerprinting checks fail. The third introduces a direct system call tracer that records kernel requests that bypass API hooks, fully orchestrated by custom CAPEv2 components that deploy the tracer, retrieve its output from the guest, and post-process the results on the host. Alongside the modules, we develop a set of simulation samples that cover specific anti-sandbox techniques, used to evaluate coverage gaps in existing sandboxes.

[Contributions] The work produces 34 new and 16 revised CAPEv2 signatures with supported evasive exit detection. Consequently, a direct system call tracer with standalone tests, and a patched BIOS to harden the analysis virtual machine. It also provides a reusable benchmark of 82 simulation samples across 30 anti-sandbox techniques (in C++ and PowerShell), used to evaluate signature coverage in sandbox environments.

2. Poster Commentary

2.1 Areas of Improvement

Before diving into the modules, this panel summarizes the four limitations this thesis targets in Gen Digital's analysis environment: weak signature coverage for anti-sandbox techniques, virtualization indicators exposed in firmware tables, no way to flag samples that exit early after an anti-sandbox check, and no visibility into system calls issued directly, bypassing CAPEv2's user-mode hooks. The next four panels address these one by one.

2.2 Direct System Call Tracer using Event Tracing for Windows (ETW)

The diagram on the left explains what the tracer is looking for. A normal system call travels through several Windows libraries before entering kernel mode in `ntoskrnl.exe` – the user-mode portion of the stack contains both DLLs. A direct system call skips that transition layer entirely: the `syscall` instruction is issued straight from the sample's own code, leaving a gap where `ntdll.dll` should be. The detection area highlights exactly this – the first user-mode frames before the kernel boundary – and if none of them resolve inside `ntdll.dll` or `win32u.dll`, the call bypasses the standard transition and is flagged.

The diagram on the right shows how the tracer is deployed and how data flows through it. Inside the analysis VM, a CAPEv2 auxiliary module launches the tracer binary before the sample runs. The tracer subscribes to the kernel ETW logger through the KrabsETW library, using the SysCallEnter, StackWalk, and lastly Process providers. It then receives a call stack for every system call the sample makes. When the sample starts, the tracer resolves its PID by process name and begins recording. ETW captures the stacks and process events in kernel mode and delivers them to the tracer in user mode, where the detection logic from the left diagram is applied. Each flagged event is written to a structured JSON log with the resolved system call name, the full stack trace, and process life-cycle data. When the sample's process tree terminates, the auxiliary uploads the JSON to the host, and a host-side processing module parses it and integrates the events into the final CAPEv2 report alongside the rest of the analysis artifacts.

2.3 Firmware Table Hardening in SeaBIOS

The before-and-after figures on the left show the same firmware table region, unpatched and patched. On the unpatched side, the strings "BOCHS" and "BXPC" sit in plain text – any malware reading ACPI tables recognizes the VM instantly. On the patched side, those strings are replaced with realistic OEM-style values. SeaBIOS performs this rewrite at boot, walking the tables QEMU hands it and overwriting the identifying entries. The same pass scrubs legacy BIOS strings and leftover debug messages from the binary itself, and XOR-encodes strings that cannot be changed, such as the "KVMKVMKVM" CPUID signature, so they are not recoverable from either the binary or raw memory.

2.4 Post-Analysis Evasive Exit Detection Module

The timeline shows what the detection actually measures. During CAPEv2's signature evaluation, the module looks at the API call that triggered an anti-sandbox signature and compares its timestamp against the sample's last recorded API call. A small delta indicates the sample terminated shortly after the check, suggesting an evasive exit and causing the module to emit its own verdict into the signature results. The threshold for what counts as "small" can be set per signature, or is derived dynamically from the total analysis time when left unset.

2.5 Anti-Sandbox Simulation Samples and CAPEv2 Signatures

The flow on this panel summarizes how the simulation corpus drove the detection work. I implemented 82 samples covering 30 anti-sandbox techniques in C++ and PowerShell, then ran them through existing sandboxes to see which techniques were actually detected. The coverage gaps from that evaluation guided the next phase: 16 existing CAPEv2 signatures were revised, and 34 new ones were written. The final signatures were tested with an internal validation tool and deployed into Gen Digital's analysis pipeline.

3. Conclusions

The three modules together extend CAPEv2 along dimensions that matter in practice – firmware, syscalls, and post-analysis verdicts – and are already deployed for testing in Gen Digital's pipeline. I am most proud of the direct system call tracer: building syscall visibility from user space, without a kernel driver, with great performance, was the hardest and most interesting part of the work for me. Future work would extend the tracer to indirect system calls and injection-based techniques, and broaden signature coverage as new anti-sandbox methods emerge.

Acknowledgements

I would like to very much thank my supervisor, Ing. Daniel Snášel, and my consultants, Bc. Michal Bandzi and Ing. Miroslav Drbal, for their help.

References

[1] Rami Sihwail, Khairuddin Omar, and Khairul Akram Zainol Ariffin. A survey on malware analysis techniques. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1662–1671, September 2018.