(M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels

Ruiyi Zhang, *CISPA Helmholtz Center for Information Security;*
Taehyun Kim, *Independent;* Daniel Weber and Michael Schwarz,
CISPA Helmholtz Center for Information Security

https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-ruiyi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: (M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels

Ruiyi Zhang
CISPA Helmholtz Center for Information Security

Taehyun Kim
Independent

Daniel Weber
CISPA Helmholtz Center for Information Security

Michael Schwarz
CISPA Helmholtz Center for Information Security

A Abstract

As discussed in the paper, we reverse engineer undocumented properties of the `monitor-` and `mwait-` instruction family that help convert microarchitectural into architectural states. In three case studies, we show the versatility of our primitive. First, with Spectral, we present a way of enabling transient-execution attacks to leak bits architecturally with up to 200 kbit/s without requiring any timer. Second, we show traditional side-channel attacks without relying on a timer. Finally, we demonstrate that when augmented with a coarse-grained timer, we can also mount interrupt-timing attacks, allowing us to, e.g., detect which website a user opens. This artifact contains the description of several experiments and proof-of-concepts for the paper.

A.1 Description & Requirements

A.1.1 Security, privacy, and ethical concerns

In our experiments, we need to modify page table entries via the PTEditor library[1].

A.1.2 How to access

The source code for this paper is available on GitHub: `https://github.com/cispa/mwait/tree/ae`.

A.1.3 Hardware dependencies

We exploit the unprivileged idle-loop optimization instructions `umonitor` and `umwait` introduced with the new Intel microarchitectures (Tremont and Alder Lake). While the reverse engineering and analysis of all `mwait-` variants are generic both on Intel and AMD processors.

[1] `https://github.com/misc0110/PTEditor`

A.1.4 Software dependencies

We recommend Ubuntu 18.04 or 20.04 and all our experiments are tested on Ubuntu 20.04 LTS (Linux kernel 5.4).

A.1.5 Benchmarks

None.

A.2 Set-up

The individual proof-of-concept implementations are self-contained and come with a Makefile and an individual description that explains how to build, run and interpret the proof-of-concept. In order to run all the proof-of-concepts, the following prerequisites need to be fulfilled:

A.2.1 Installation

- Build tools (gcc, make)
- Intel latest CPUs (Tremont and Alder Lake)
- PTEditor
- Stress

A.2.2 Basic Test

The folder `Intel-umwait` contains the basic experiment to check whether `umonitor` and `umwait` work on the current tested CPUs.

A.3 Evaluation workflow

A.3.1 Major Claims

(C1): We exploit the unprivileged idle-loop optimization instructions `umonitor` and `umwait` introduced with the new Intel microarchitectures (Tremont and Alder Lake). Although not documented, these instructions provide architectural feedback about the transient usage of a specified memory region.

(C2): We experimentally confirmed that the Intel's undocumented timed `mwait` feature can be enabled by setting bit 31 in MSR (`0xe2`). We further reverse engineered the feature and found that bit 1 of the `ECX` register of the mwait instruction indicates that the timeout feature is used. The maximum waiting time is an implicit 64-bit timestamp-counter value stored in the `EDX:EBX` register pair.

(C3): With Spectral, we present a way of enabling transient-execution attacks to leak bits architecturally with up to 200/ without requiring any architectural timer.

(C4): We show traditional side-channel attacks without relying on an architectural timer.

(C5): We demonstrate that when augmented with a coarse-grained timer, we can also mount interrupt-timing attacks, allowing us to, detect which website a user opens.

A.3.2 Experiments

(E1): Intel-umwait
Preparation: Intel Tremont and Alder Lake CPUs
Results: Test if `umonitor/umwait` work on the current processors

(E2): trigger-tester
How to: We analyzed different wake-up triggers for all `mwait-` variants both on Intel and AMD machines, including cache coherence functions. Moreover, we analyzed the memory type of the monitored address range by modifying the page table via the library PTEditor.
Results: As shown in the Table 1-2 in the paper.

(E3): timed-mwait
How to: We reverse engineered the Intel's undocumented `timed-mwait` feature via a simple Linux kernel module.
Results: As claimed in the C2.

(E4): comparison
How to: We constructed a benchmark detecting fully asynchronous events with TWMand other conventional side-channel attacks for reference.
Results: As shown in the Figure 1-2, Table 3 in the paper.

(E5): covert-channel
How to: We created a timer-less covert channel with `umonitor` and `umwait`.
Results: As shown in the Figure 4 in the paper.

(E6): spectral
How to: We used the timer-less covert channel for spectre attacks.
Results: As shown in the Figure 5-6 in the paper.

(E7): aes-example
How to: We reproduced attacks on AES T-table implementation based on our Timer-less Timing Measurement.
Preparation: The deprecated OpenSSL 1.0.1e.
Results: As shown in the Figure 3,7 in the paper.

(E8): website-fingerprinting
How to: We detected network interrupts while opening a website.
Results: As shown in the Figure 8 in the paper.

A.4 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.