USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Branch Privilege Injection: Compromising Spectre v2 Hardware Mitigations by Exploiting Branch Predictor Race Conditions

Sandro Rüegge, Johannes Wikner, and Kaveh Razavi, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity25/presentation/ruegge

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Branch Privilege Injection: Compromising Spectre v2 Hardware Mitigations by Exploiting Branch Predictor Race Conditions

Sandro Rüegge
ETH Zurich

Johannes Wikner
ETH Zurich

Kaveh Razavi
ETH Zurich

A Artifact Appendix

A.1 Abstract

This artifact appendix provides instructions to reproduce the results in *Branch Privilege Injection: Compromising Spectre v2 Hardware Mitigations by Exploiting Branch Predictor Race Conditions*. Our paper introduces Branch Predictor Race Condition (BPRC), an event-misordering effect in branch predictors. We uncover multiple instances of this effect on recent Intel processors and investigate the cause. To demonstrate the impact of these vulnerabilities, we built an end-to-end attack on Ubuntu 24.04, running a recent Linux kernel with default mitigations.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Some of the experiments require loading special kernel modules. *kmod_ap* allows any user process to execute arbitrary code with supervisor privileges. *kmod_spec_ctrl* provides persistent control over the speculation control register. These modules compromise the system's security but can be manually removed using `sudo rmmod ap` and `sudo rmmod spec_ctrl`.

Furthermore, the provided ansible scripts modify the system configuration (i.e. GRUB, initramfs) without any additional interaction. While the scripts make an effort to return the system to it's previous state, **please exercise extreme caution when using the ansible scripts** to ensure they do not operate outside of your expectations. Additionally, the ansible scripts will reboot the target system when necessary which makes localhost unsuited to run these scripts against.

A.2.2 How to access

The artifacts are available on zenodo at https://doi.org/10.5281/zenodo.14636810 and on GitHub at https://github.com/comsec-group/bprc (once the embargo ends on May, 13th). We provide the artifact evaluators with a ssh key in the hotcrp submission to access the git repository privately during the embargo.

A.2.3 Hardware dependencies

In our paper, we evaluated a wide range of recent microarchitectures. The exact microarchitectures used in our paper are:

- Core i7-14700K (Raptor Cove, Gracemont)
- Core i7-13700K (Raptor Cove, Gracemont)
- Core i7-12700K (Golden Cove, Gracemont)
- Core i7-11700K (Cypress Cove)
- Core i7-10700K (Skylake)
- Core i9-9900K (Skylake)
- Ryzen 9 9900X (Zen 5)
- Ryzen 7 7700X (Zen 4)
- Google Tensor (Cortex-X1)
- Google Tensor (Cortex-A76)

The microarchitectural experiments focus mostly on the Intel processors with the exception of the initial asynchronous branch predictor experiment. The attack requires Ubuntu 24.04 with kernel package version 6.8.0-47-generic. While we developed the attack specifically for a Core i7-13700K, other recent Intel processors should work as well, albeit potentially less reliably.

A.2.4 Software dependencies

We use Ubuntu 24.04, 22.04, or 20.04 to compile and run the experiments on our test machines. Our compilation scripts expect Ubuntu packages `build-essential`, `msr-tools`, `clang`, `git`, `libtirpc-dev`, and the kernel headers (`linux-headers-generic`) to be present. The python analysis scripts require `python3` with `matplotlib`.

A.2.5 Benchmarks

Our mitigation performance evaluation employs UnixBench [1] and lmbench [2].

[1] https://github.com/kdlucas/byte-unixbench
[2] https://github.com/intel/lmbench

A.3 Set-up

A.3.1 Installation

Download the artifacts using one of the links in Section A.2.2. Install the dependencies as per the *README.md* file. On Ubuntu 24.04 you can use the following commands:

```
sudo apt install python3 python3-pip \
   build-essential git clang \
   linux-headers-$(uname -r) msr-tools \
   libtirpc-dev
pip install matplotlib
```

A.3.2 Basic Test

There is a special experiment to help you verify your setup in `experiments/exp-test-setup`. Please follow the `README.md` in the same directory to run the setup test.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Branch predictor updates on recent Intel processors can occur after a measurable delay. This is proven by the experiment (E1) whose results are reported in Section 5.1 and Figure 3.

(C2): As a result, branch predictions can be injected from user to kernel ((E2), Table 2), from guest to host ((E3), Section 5.3) and across IBPB ((E4), Section 5.3).

(C3): We can use these vulnerabilities to build an end-to-end attack to leak arbitrary kernel memory and '/etc/shadow' on an Intel Raptor Lake system with Ubuntu 24.04 ((E10), Section 7).

A.4.2 Experiments

(E1): [exp-btb-delay] [5 human-minutes + 2 compute-hours]: Experiment to determine IP-based branch predictor (BTB) insertion delay (C1).
Execution: Please follow the instructions located at `experiments/exp-btb-delay/README.md`
Results: On Intel processors, we observe a high number of mispredictions when the delay (number of NOPS) is low and vice versa

(E2): [exp-leak-supervisor] [5 human-minutes + 5 compute-minutes]: Experiment to determine if $BPRC_{U\rightarrow K}$ is possible on recent Intel processors (C2).
Execution: Please follow the instructions located at `experiments/exp-leak-supervisor/README.md`
Results: We observe a success percentage above the noise level on at least two of the three tested instruction on recent Intel processors

(E3): [exp-leak-hypervisor] [5 human-minutes + 5 compute-minutes]: Experiment to determine if $BPRC_{G\rightarrow H}$ is possible on recent Intel processors (C2).
Execution: Please follow the instructions located at `experiments/exp-leak-hypervisor/README.md`
Results: We observe a success percentage above the noise level on at least two of the three tested instruction on recent Intel processors

(E4): [exp-leak-ibpb] [5 human-minutes + 5 compute-minutes]: Experiment to determine if $BPRC_{IBPB}$ is possible on recent Intel processors (C2).
Execution: Please follow the instructions located at `experiments/exp-leak-ibpb/README.md`
Results: We observe a success percentage above the noise level on at least two of the three tested instruction on recent Intel processors

(E5): [exp-ibp-insertion] [5 human-minutes + 5 compute-minutes]: Experiment to determine if a prediction is inserted into the BTB and IBP the first time it is seen.
Execution: Please follow the instructions located at `experiments/exp-ibp-insertion/README.md`
Results: We ran this experiment on the Intel performance (P-Cores), Cypress Cove and Skylake cores. For 'Random History' we expect many hits on the BTB (>90%) and very few on the IBP (<1%). For 'Matching History' we expect few hits on the BTB (<10%) and many hits on the IBP (>90%).

(E6): [exp-leak-supervisor-discern] [5 human-minutes + 5 compute-minutes]: Experiment to determine if $BPRC_{U\rightarrow K}$ is caused by the BTB or IBP on Intel.
Execution: Please use instructions at `experiments/exp-leak-supervisor-discern/README.md`
Results: We expect to see hits on the BTB above the noise level while there are no hits above noise level on the IBP.

(E7): [exp-syscall-split] [5 human-minutes + 3 compute-hours]: Experiment to determine the effect of timing on BPI.
Execution: Please follow the instructions located at `experiments/exp-syscall-split/README.md`
Results: We expect to see the number of hits in the kernel decrease with increasing number of NOPs while the number of hits in user increases with increasing number of NOPs.

(E8): [exp-leak-rounds] [5 human-minutes + 5 compute-minutes]: Test reliability of BPI when repeatedly attacking the same victim branch without modifying history with and without `BHI_DIS_S`.
Execution: Please follow the instructions located at `experiments/exp-leak-rounds/README.md`
Results: With only eIBRS we expect a decreasing success rate over successive repetitions. With `BHI_DIS_S` enabled, we expect the success rate to increase over the first two repetitions and then stay high (>90%).

(E9): [exp-bhi-dis-s] [5 human-minutes + 5 compute-minutes]: Experiment to determine how `BHI_DIS_S` is implemented on recent Intel processors.

Execution: Please follow the instructions located at `experiments/exp-bhi-dis-s/README.md`
Results: We expect a low number of mispredictions when there is no mitigation active (<10%) and a high number of mispredictions when `BHI_DIS_S` is active.

(E10): [exp-end2end] [5 human-minutes + 10 compute-minutes]: End-to-end attack against Ubuntu 24.04 kernel version 6.8.0-47-generic on an Intel Raptor Lake processor (C3).
Execution: Please follow the instructions located at `experiments/exp-end2end/README.md`
Results: We expect the attack to work successfully in a high number of cases (>90%) with a leak rate of over 5KiB/s in the median.

(E11): [exp-benchmark-mitigations] [5 human-minutes + 24 compute-hours]: Measure mitigation performance overhead.
Execution: Please use instructions at `experiments/exp-benchmark-mitigations/README.md`
Results: We observed a slowdown for all mitigations. The microcode performs best, `IPRED_DIS_S` second and retpoline third.

A.5 Notes on Reusability

Our experiments were developed to work well on the evaluated x86 processors but the methodologies can be applied to other processors and microarchitectures as well. Furthermore, the kernel modules and library code provided in `experiments/uarch-research-fw` are useful to hardware reverse engineering in general.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.