USENIX Security '26 Artifact Appendix: Spectre on RISC-V Silicon: Attacks and Defenses on Commercial Out-of-Order Processors

Lukas Gerlach[1] Marton Bognar[2] Daniel Weber[1] Michael Schwarz[1] Jo Van Bulck[2]

[1]*CISPA Helmholtz Center for Information Security* [2]*DistriNet, KU Leuven*

A Artifact Appendix

A.1 Abstract

This artifact supports the Open Science scope stated in the paper. It contains the unified Spectre proof-of-concept framework, microarchitectural measurement tools, speculation-barrier tests, Smatch/CodeQL gadget-scanning scripts, cross-architecture mitigation-diffing code, branchless uBPF dispatch prototype and benchmark, optional SPEC CPU 2017 build setup, cache/timer primitives, BPF gadget-emitter source, and Linux kernel patches.

A.2 Description & Requirements

Full reproduction requires the evaluated RISC-V hardware.

A.2.1 Security, privacy, and ethical concerns

Several experiments show Spectre-style leakage, load kernel modules and change BPF settings. Run them only on isolated systems owned or explicitly authorized by the evaluator. The artifact does not process personal data or contact third-party systems.

A.2.2 How to access

Download the latest Zenodo release at `https://doi.org/10.5281/zenodo.18452278`. The relevant directories are:

- `spectre-poc-framework/`: 13 Spectre variants plus runner.
- `microarch-measurements/`: RSB-size and speculation-window tests.
- `barrier-test/`: candidate speculation-barrier tests.
- `gadet-scanning/`: Smatch and CodeQL kernel gadget scanning.
- `gadet-scanning/mitigation_diffing/`: semantic mitigation-diffing tool.
- `specbuild/`: SPEC CPU 2017 cross-build setup for mitigation benchmarks.
- `branchless-jit-dispatch/`: branchless dispatch prototype and modified uBPF.
- `cacheutils.h`, `counter-thread-timer/`: cache primitives and timer benchmark.
- `bpf-spectre-exploit/`: BPF Gadget Emitter: BPF bytecode to reliably emit the type confusion gadget and leak from a controlled kernel address.
- `kernel-patches/`: contributed RISC-V Spectre-v1 patches.

A.2.3 Hardware dependencies

The main hardware targets are T-Head Xuantie C910/C920 and SiFive P550 systems with root access. BTB experiments require BTB prediction to be enabled in firmware, which is not the case in the stock configuration of either board. P550 cache-eviction experiments require huge pages:

```
sudo sysctl -w vm.nr_hugepages=64
```

On some cores, userspace access to the cycle counter has to be enabled explicitly, otherwise the timed experiments abort with an illegal-instruction fault:

```
sudo sysctl -w kernel.perf_user_access=2
```

BPF experiments require a Linux target with BPF maps and JIT support.

A.2.4 Software dependencies

The target-board experiments (Basic Test, E1–E3, and E5) require a C/C++ toolchain, `make`, `cmake`, `bc`, Python 3, and the kernel/BPF settings listed per experiment. All experiments that are run on non RISC-V hardware (gadget scanning, cross-compiling SPEC CPU) are bundled in a Docker container that installs all the required dependencies.

A.2.5 Benchmarks

We use the `bpf-benchmark` to evaluate performance of our mitigated BPF interpreter. The proprietary SPEC CPU 2017 benchmark is optional. The `specbuild/` helper includes tooling to cross-compile the benchmarks on a fast machine and then run them on the target.

A.3 Set-up

A.3.1 Installation

Obtain the artifact from the provided Zenodo link and unpack it.

A.3.2 Basic Test

On a RISC-V target, run:

```
cd spectre-poc-framework
python3 run_all_tests.py --group baseline \
  --runs 1 --iterations 100
```

The runner should detect the platform, build the baseline binary, and print precision/recall metrics (compare to Table 2).

A.4 Evaluation workflow

A.4.1 Major Claims

(C1) C910/C920 and P550 are vulnerable to major Spectre variants; reproduced by E1.

(C2) The BPF source emits the paper's speculative type-confusion gadget and shows that the leakage can be reproduced; reproduced by E2.

(C3) Speculation barriers, RSB size, speculation windows, cache channels, and counter-thread timing are processor-dependent; reproduced by E3.

(C4) Gadget scanning and mitigation diffing reveal missing RISC-V kernel hardening; reproduced by E4.

(C5) The branchless uBPF dispatch mitigation can be built and benchmarked with bundled BPF programs; reproduced by E5.

(C6, optional) The SPEC CPU 2017 mitigation setup reproduces the reported mitigation overheads given a licensed SPEC tree, a target sysroot, and dedicated RISC-V hardware; reproduced by optional E6.

A.4.2 Experiments

(E1) Spectre variant framework [15 human-minutes + 1-2 compute hours per board].

```
cd spectre-poc-framework
python3 run_all_tests.py --group all --runs 30 \
  --csv results.csv
```

Compare the CSV to Table 2 in the paper. The test runner also outputs human-readable logs with all the information. C910/C920 should show high recall for most PHT, RSB, and STL variants. P550 should show strong PHT sa-ip, RSB, and STL results with lower cross-address recall. The BTB variants only leak once BTB prediction is enabled in firmware, so they show no leakage on stock boards.

(E2) BPF proof-of-concept [20 human-minutes].

```
cd bpf-spectre-exploit
make clean && make exploit-complete
sudo sysctl -w kernel.perf_event_paranoid=0 \
  kernel.perf_user_access=2 \
  net.core.bpf_jit_enable=1 vm.nr_hugepages=1000
sudo ./exploit-complete
```

Expected result on the C910/C920: the program leaks the kernel banner. Additionally one can inspect the emitted RISC-V bytecode from the BPF JIT by setting:

```
echo 2 | sudo tee /proc/sys/net/core/bpf_jit_enable
```

This command dumps a lot of data to the kernel log.

(E3) Microarchitectural measurements [20 human-minutes + 1 compute-hour per board].

```
cd barrier-test && make
sudo ./barrier_test && sudo ./barrier_test_rsb
cd ../microarch-measurements/rsbsize-test
make && ./test
cd ../speculation-window
make && sudo ./main > spec-window.csv
cd ../../counter-thread-timer
sudo sysctl -w kernel.perf_event_paranoid=0 \
  kernel.perf_user_access=2
make && COUNTER_CORE=1 taskset -c 0 ./bench
```

Expected results: barrier tests report low hit rates for effective barriers. The RSB-size test shows a cliff near 12 entries on C910/C920 and 16 on P550. The counter-thread timer reports approximately cycle-scale resolution on C910/C920.

(E4, local) Kernel gadget scanning, mitigation diffing, and patches [1 human-hour + 1-2 compute-hours].

```
# Build the analysis image.
cd gadet-scanning
docker build -t riscv-gadget-scan .

# Get the kernel source.
git clone --depth 1 --branch v6.6 \
  https://github.com/torvalds/linux.git ~/linux-6.6
CACHE=mitigation_diffing/.embedding_cache
mkdir -p results .hf-cache "$CACHE"

# Run Smatch.
docker run --rm \
  -v ~/linux-6.6:/kernel \
  -v "$PWD/results:/opt/gadet-scanning/results" \
  riscv-gadget-scan \
  ./run-analysis.py linux-6.6 /kernel --tool smatch

# Run CodeQL. Creates a reusable database in
# results/linux-6.6/codeql-db/.
docker run --rm \
  -v ~/linux-6.6:/kernel \
  -v "$PWD/results:/opt/gadet-scanning/results" \
  riscv-gadget-scan \
  ./run-analysis.py linux-6.6 /kernel --tool codeql

# Run mitigation diffing.
docker run --rm \
  -v ~/linux-6.6:/kernel:ro \
  -v "$PWD/.hf-cache:/root/.cache" \
  -v "$PWD/$CACHE:/opt/gadet-scanning/$CACHE" \
  riscv-gadget-scan bash -lc "cd mitigation_diffing \
    && mitigation-diffing /kernel -m mitigations.csv"

# Compare tools (Smatch vs CodeQL).
```

```
docker run --rm \
  -v "$PWD/results:/opt/gadet-scanning/results" \
  riscv-gadget-scan \
  ./compare-tools.py linux-6.6
```

Expected results: the scanner emits kernel gadget reports; `mitigation-diffing` reports missing RISC-V mitigation counterparts for the listed cross-architecture mitigation sites. This should reproduce the findings in Section 4.1, where CodeQL and Smatch do not find the RISC-V specific gadgets that our mitigation diffing approach found.

(E5) Branchless uBPF dispatch benchmark [30 human-minutes + benchmark compute time].

```
cd branchless-jit-dispatch
make && ./dispatch
cd ubpf
cmake -S . -B build_baseline -DUBPF_ENABLE_TESTS=ON \
  -DCMAKE_BUILD_TYPE=Release
cmake --build build_baseline --target ubpf_test --parallel
cmake -S . -B build_mitigated -DUBPF_ENABLE_TESTS=ON \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_C_FLAGS="-DUBPF_JIT_DISPATCH"
cmake --build build_mitigated --target ubpf_test --parallel
./benchmark.sh build_baseline/bin/ubpf_test \
  build_mitigated/bin/ubpf_test \
  ../bpf-benchmark/bpf_progs 50
```

Expected results: `dispatch` runs the branchless direct-`jal` prototype. By inspecting the binary with `objdump -d` it can be confirmed that no indirect branches are used for the dispatch. The uBPF benchmark prints CSV rows `program,baseline_ms,mitigated_ms,overhead` which should reproduce the numbers in Section 5.2.

(E6, optional) SPEC CPU 2017 mitigation setup [2 human-hour + benchmark compute time]. Run it with a licensed SPEC CPU 2017 installation and a target sysroot.

```
cd specbuild
python3 -m venv .venv
. .venv/bin/activate
python3 -m pip install --upgrade pip pyyaml
docker build -t specbuild docker/
./fetch-sysroot.py <riscv-host> \
  ./sysroots/<riscv-host>-sysroot
# Place SPEC CPU 2017 at cpu2017/spec_cpu/
# or edit spec-build.yaml.
python3 build.py -l
python3 build.py -t <target> --size test \
  --benchmarks=505.mcf_r
scp packages/<target>-*.tar.gz <riscv-host>:~
# On <riscv-host>: unpack one package and run:
#   ./run.sh 505.mcf_r --csv spec.csv
```

Expected results: `specbuild` lists configured targets and fetches a sysroot from the target machine. The sysroot allows to cross compile the SPEC benchmarks for the target machine. The LLVM mitigation profiles additionally require the compiler bundle described in `specbuild/compilers/README.md`. The SPEC benchmarks should reproduce the overheads listed in Figure 3 of the paper.

A.5 Notes on Reusability

The artifact can be reused by testing all or a subset of the spectre variants on new RISC-V hardware. This requires adapting timers and the cache side channel to the new hardware, but should be much simpler than building a proof-of-concept from scratch. Our mitigation diffing can be used to map code sites from a kernel of one architecture to another. We believe that this is a useful tool not only for Spectre mitigations, but working with cross-architecture kernel code in general. The branchless JIT dispatch prototype could be extended into a more general mitigation by implementing it as part of a compiler pass.

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/usenixsec2026/.