USENIX Security '26 Artifact Appendix: Exploiting Hidden Resource Contention in Selective Speculation Defenses

Xiaoyu Cheng[1], Fei Tong[1,2,3] , Zhenyu Lei[1], Fang Jiang[1], Zhe Zhou[1], Guang Cheng[1], Trevor E. Carlson[4]

[1]School of Cyber Science and Engineering, Southeast University, China
[2]Purple Mountain Laboratories, China
[3]Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, China
[4]National University of Singapore

A Artifact Appendix

A.1 Abstract

This artifact supports our study of reservation-station (RS) contention as a hidden leakage surface in optimized selective-speculation defenses. Specifically, it demonstrates that secrets can influence RS pressure through three mechanisms that are not fully covered by existing defense assumptions: operand-dependent μop expansion in `REP`-prefixed string instructions, predicated value selection without fetch redirects, and variable-latency arithmetic or memory operations that are left to proceed under optimized policies. These mechanisms can affect frontend fetch and I-cache state during transient execution, creating measurable side channels even when defenses delay known transmit operations or prediction-resolution events.

The package includes gem5-based PoCs that bypass STT and DOLMA, real-machine measurements validating the `REP MOVSB`/`REP STOSB` contention primitive on Intel Raptor Cove P-cores, and a bit-recovery PoC. It also provides the LLVM-based gadget-searching workflow, an optional libsodium timing-channel PoC, the enhanced STT implementation, and the SPEC CPU2017 workflow used for overhead evaluation.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact contains Spectre-style PoCs and microarchitectural timing experiments. It should be run only on private research machines where the evaluator is authorized to run low-level benchmarks. It does not include a privilege-escalation payload or intentionally read third-party data. Some real-machine experiments require root-only steps such as loading `msr` or nanoBench, disabling ASLR in one shell, and pinning processes to cores; these steps should not be performed on production systems.

A.2.2 How to access

The artifact is available at https://doi.org/10.5281/zenodo.20364720. We also provide the source code for the artifact at https://github.com/Arch-security/RSbypassSP.

A.2.3 Hardware dependencies

The gem5 simulator workflows in `gem5/stt/`, `gem5/dolma/`, and `gem5/gem5-sttre-enhanced/`, as well as the LLVM gadget-searching workflow in `gadget_detect/LLVM_FIX/`, do not require a specific CPU model. A Linux x86-64 host with Docker, at least 32 GB RAM, and tens of GB of free disk space is recommended for building gem5 and running simulations.

The real-machine validation in `Real-Machine_Validation/` requires an Intel CPU with Raptor Cove P-cores; our reference systems are an Intel Core i9-13900K and an Intel i7-14650HX running Ubuntu 22.04. The optional `gadget_detect/libsodium_poc/` timing PoC is hardware sensitive; we validated it on an Intel Xeon Gold 6248R and treat it as optional supporting evidence.

A.2.4 Software dependencies

The gem5 workflows need Linux on x86-64, Docker, `git`, and `make`; the Dockerfiles install the tree-specific dependencies, including the legacy Python/SCons environments and the RISC-V toolchain used by DOLMA. The real-machine workflows need Linux on x86-64, Bash, `make`, GCC, Python 3, `matplotlib`, `numpy`, `seaborn`, and `scipy`; nanoBench counter regeneration additionally requires root, `msr`, and the nanoBench kernel module. The LLVM gadget-searching workflow needs LLVM/Clang build dependencies, CMake, GNU Make, and source checkouts of the target libraries.

SPEC CPU2017 is required only for the enhanced STT overhead workflow and is not included in the artifact.

For convenience, the main artifact folders provide dependency helpers and the corresponding `requirements.txt` files record the Python package versions used in our tested environments.

A.2.5 Benchmarks

The attack PoCs are self-contained microbenchmarks. Gadget searching builds OpenSSL, Libgcrypt, libsodium, OpenBLAS, musl, zlib, and wolfSSL. The performance-overhead experiment uses evaluator-provided SPEC CPU2017.

A.3 Set-up

A.3.1 Installation

Clone the artifact repository and enter the artifact root:

```
git clone <artifact-url> RSbypassSP
cd RSbypassSP
export ARTIFACT_ROOT=$PWD
```

The component setup helpers are:

- `./gem5/install.sh` builds all three gem5 Docker images; pass `stt`, `dolma`, or `enhanced` to build only one.
- `./Real-Machine_Validation/install.sh` installs the normal real-machine dependencies on apt-based Ubuntu systems; nanoBench still requires the root-only setup in its README.
- `./gadget_detect/install.sh` installs dependencies for gadget detection and the optional libsodium PoC.
- For enhanced STT overhead, install SPEC CPU2017 separately and set `SPEC_DIR`.

A.3.2 Basic Test

The recommended smoke test is the STT `v1rep` run, which checks the Docker image, gem5 build, PoC build, and one complete simulator execution:

```
cd $ARTIFACT_ROOT
./gem5/install.sh stt
docker run --rm -it \
-v "$ARTIFACT_ROOT":/artifact
stt-ubuntu20-py2
```

Inside the container, run:

```
export ARTIFACT_ROOT=/artifact
cd $ARTIFACT_ROOT/gem5/stt
make x86-64
cd RS_Contention_Taxonomy-overlooked
make clean
make CC=musl-gcc all
cd $ARTIFACT_ROOT/gem5/stt
make v1rep
```

A successful run prints the Futuristic STT configuration and stable bit/timing lines for the `REP MOVSB`-based PoC.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Operand-dependent μop expansion creates pre-execution RS pressure that is not covered by existing execution-phase transmit taxonomies. Our gem5 `REP MOVSB`-based PoC shows that this pressure can bypass STT-Futuristic by affecting frontend/I-cache timing before STT's issue-time delay protection can apply. This is proven by (E1), described in **Section 5 Exploiting taxonomy-overlooked μop expansion under STT (Limitation I & II)** with results illustrated in **Figure 9** of the paper.

(C2): Variable-latency memory operations that are left to proceed under the older-μop-first allocation strategy can still create secret-dependent RS pressure when amplified by dependency chains. Our gem5 PoC shows that this leakage persists under DOLMA-Conservative (M+R) despite its older-μop-first allocation strategy. This is proven by (E2), described in **Section 5 Bypassing older-μop-first with taxonomy-recognized transmit μops under DOLMA (Limitation III)** with results illustrated in **Figure 10** of the paper.

(C3): Predicated value selection can inject secrets into operands without triggering fetch redirects, which is not covered by the delay-until-resolution mechanism. Our x86 `CMOV` and RISC-V Zicond PoCs show that such selectors leak under DOLMA-Conservative (M+R), while the branch-based selector is blocked. This is proven by (E2), described in **Section 5 Implementation-dependent effectiveness of delay-until-resolution** with results illustrated in **Figure 13** of the paper.

(C4): The `REP`-based RS-contention primitive is measurable on Intel Raptor Cove P-cores and supports bit recovery. This is proven by (E3), described in **Section 4.1** and **Section 5 Contention thresholds of `REP`-prefixed string instructions on Raptor Cove** with results reported in **Table 2** and **Figures 14–16** of the paper.

(C5): RS-contention-relevant gadget patterns occur in common libraries, and the optional libsodium PoC demonstrates a real-library timing example. This is proven by (E4) and (E6), described in **Section 4.3** with results reported in **Table 4** of the paper.

(C6): Enhanced STT closes the identified gaps by covering tainted stores, operand-dependent expansion, predicate sources, and conservative taint initialization for both memory- and register-resident secrets with moderate overhead. This is evaluated by (E5), described in **Section 6** with results illustrated in **Figure 17** of the paper.

A.4.2 Experiments

(E1): STT bypass through operand-dependent micro-op expansion *[0.1 human-hours + 0.2 computer-hours].*
How to: Reproduce the `REP MOVSB` STT bypass in gem5.
Preparation: Build the STT image with `gem5/install.sh stt`; inside the container, build `build/X86/gem5.opt` and `RS_Contention_Taxonomy-overlooked/`.
Execution: Run `make v1rep` from `gem5/stt/`.
Results: The output should show distinguishable bit-dependent timing under Futuristic STT, matching Figure 9.

(E2): DOLMA bypass through predicated and variable-latency transmitters *[0.2 human-hours + 0.3 computer-hours].*
How to: Reproduce (i) a variable-latency memory gadget that leaks despite DOLMA's older-μop-first allocation, and (ii) a selector comparison showing that a branch-based selector is blocked while x86 `CMOV` and RISC-V Zicond predicated selectors leak.
Preparation: Build the DOLMA image with `gem5/install.sh dolma`; inside the container, build the x86 and RISC-V gem5 targets and the PoCs under `RS_Contention_Taxonomy-recognized/`.
Execution: Run `make s1rs64`, `make s1rs64_branch`, `make s1rs64_cmov`, and `make s1rs_riscv`.
Results: The variable-time-memory PoC should leak under DOLMA-Conservative (M+R). The branch-based selector should be blocked by delay-until-resolution, while the x86 `CMOV` and RISC-V Zicond predicated selector PoCs should leak, matching Figures 10 and 13.

(E3): Contention thresholds of REP-prefixed string instructions on Raptor Cove *[0.3 human-hours + 0.5 computer-hours].*
How to: Validate the `REP MOVSB`/`REP STOSB` contention primitive and bit-recovery PoC on Raptor Cove.
Preparation: On a Raptor Cove system, run `Real-Machine_Validation/install.sh` and build `rcx_threshold_eval/` and `REPmsAttack/`.
Execution: Run `run_measuret.sh` and `analyze.py` for threshold selection, then run `analyze_chars.sh` with an evaluator-selected one-byte secret.
Results: The threshold sweep should show that larger RCX values for `REP MOVSB` and `REP STOSB` produce higher target-instruction latency after crossing a machine-dependent threshold. The attack script should recover the configured one-byte secret by majority vote. The generated plots should show distinguishable timing distributions for secret bits.

(E4): LLVM gadget search in common libraries *[0.3 human-hours + about 0.7 computer-hours].*
How to: Reproduce the static gadget-searching workflow for common x86 libraries.
Preparation: Run `gadget_detect/install.sh`; build LLVM/Clang `llvmorg-21.1.5` with `gadget_detect/LLVM_FIX/X86MIRanalyze_pass.patch`.
Execution: Compile the libraries listed in `LLVM_FIX/README.md` with `-O2 -mllvm -x86-mir-analyze -g`, then run `scripts/collect_gadget_table.sh`.
Results: The collected gadget-count table should match Table 4.

(E5): Enhanced STT SPEC2017 overhead *[0.6 human-hours + 50.3 computer-hours].*
How to: Reproduce the enhanced STT performance-overhead workflow.
Preparation: Build the enhanced image with `gem5/install.sh enhanced` or use `install_dependencies_ubuntu22.sh`; build `build/X86/gem5.opt`, install SPEC CPU2017, and set `SPEC_DIR`.
Execution: Use `run_spec17.sh` for a smoke run or `run_spec17suit.sh` for the paper-style configurations.
Results: Compute normalized IPC from `m5out/stats.txt`; the overhead trend should match Figure 17.

(E6): Optional libsodium real-library timing PoC *[0.1 human-hours + 0.1 computer-hours].*
How to: Run the optional real-library timing PoC for libsodium Ristretto255.
Preparation: Run `gadget_detect/install.sh`, build `gadget_detect/libsodium_poc/`, enter the ASLR-disabled shell, and resolve libsodium runtime addresses.
Execution: Rebuild if addresses changed, run `./mul 0` and `./mul 1`, and plot the logs.
Results: The two prepared microarchitectural states should produce visibly different aggregate timing distributions; exact cycles are machine dependent.

A.5 Notes on Reusability

The Dockerized gem5 PoCs are the most portable reproduction path. The LLVM pass can be reused on additional x86 libraries by compiling them with `-mllvm -x86-mir-analyze`. For real-machine experiments, rerun the threshold sweep whenever the CPU, kernel, BIOS setting, frequency policy, or core placement changes.

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/.