USENIX Security '25 Artifact Appendix: Exploiting Inaccurate Branch History in Side-Channel Attacks

Yuhui Zhu[1,2] and Alessandro Biondi[1]

[1]*Scuola Superiore Sant'Anna*
[2]*Scuola IMT Alti Studi Lucca*

A Artifact Appendix

A.1 Abstract

Modern out-of-order CPUs heavily rely on speculative execution for performance optimization, with branch prediction serving as a cornerstone to minimize pipeline stalls and maximize efficiency. When shared branch prediction resources lack proper isolation and sanitization methods, they can introduce security vulnerabilities that expose sensitive data across different software contexts.

This artifact evaluates the behavior of two underdocumented features of the Branch Predictor Unit: *Bias-Free Branch Prediction* and *Branch History Speculation*. These discoveries expose previously unknown cross-privilege attack surfaces for Branch History Injection (BHI).

Based on these findings, we present three novel attack primitives: two Spectre attacks, namely **Spectre-BSE** and **Spectre-BHS**, and a cross-privilege control flow side-channel attack called **BiasScope**. This artifact evaluates the presence of these primitives using user-mode intra-process proof-of-concepts, then evaluates their capability for mounting cross-privilege attacks using custom syscall handlers. Finally, we demonstrate the **Chimera** snippet using eBPF to achieve end-to-end exploitation.

A.2 Description & Requirements

This artifact contains proof-of-concept (PoC) code demonstrating the vulnerabilities discovered in the paper. The project is organized into several submodules, each addressing different attack scenarios:

- **`intra-ctx/`:** Intra-process PoCs demonstrating the relevant microarchitectural behaviors and primitives to perturb and exploit their side effects. This module covers BHB/PHT mistraining (Section 3.3), Spectre-BSE (Section 5.4), Spectre-BHS (Section 6.2), and Chimera snippets (Section 7).

- **`cross-ctx/`:** Cross-context PoCs showcasing how t

  Intra-context demos. The `intra-ctx/` module consists of multiple sub-modules (implemented by `intra-ctx/tests/*.c`) which demonstrate the exploited primitives, the corresponding microarchitectural behaviors, and basic attack flows described in the paper using intra-mode attack processes.

  Cross-context demos. hese primitives can manipulate branch prediction in kernel mode or another process. This module covers BiasScope (Section 5.3), Spectre-BSE (Section 5.4), and Spectre-BHS (Section 6.2).

- **`chimera-ebpf/`:** End-to-end Chimera attack (Section 7) implemented as an eBPF program, demonstrating practical kernel memory leakage.

Please refer to the `README.md` files in the root directory and each subdirectory for details of all these submodules.

A.2.1 Security, privacy, and ethical concerns

While the artifact does not collect, transmit, or store any personal data and poses minimal privacy risks, evaluators should be aware of several important risks. The cross-context demonstrations require patching and recompiling the Linux kernel on ARM machines, which may affect system stability and integrity, so evaluators should use dedicated test machines rather than production systems. Additionally, cross-privilege BiasScope evaluation requires disabling Spectre mitigations via `mitigations=off`, which temporarily reduces the system's protection against speculative execution attacks and should be reverted after evaluation. After evaluation, the modified kernel and disabled mitigations need to be carefully reset to restore the original security posture.

A.2.2 How to access

The artifact source code is publicly available on Zenodo at https://zenodo.org/records/15612187. The archived repository includes complete documentation with

detailed setup and execution instructions. Reviewers may evaluate these artifacts on evaluator-owned machines, provided their processors match the vulnerable models documented in our paper. Please follow the instructions provided in all `README.md` files.

A.2.3 Hardware dependencies

This artifact requires specific processor architectures to demonstrate the novel branch prediction vulnerabilities described in our paper. To fully reproduce the findings, evaluators will need access to one or more of the following processor families listed in Table 1.

A.2.4 Software dependencies

To compile the intra-context module of the artifact, `make` and `gcc` are required. For cross-compilation targeting ARM machines, you will need the `aarch64-linux-gnu-gcc` toolchain installed on your system.

Compiling or cross-compiling the patched kernel requires the standard kernel build toolchain, including appropriate cross-compilation tools for target architectures.

The artifact is compatible with any standard Linux distribution and does not require specific distribution versions.

For intra-context demonstrations in the `intra-ctx/` directory, reviewers can simply cross-compile the PoCs for their target architecture and execute them with provided parameters in standard userspace without kernel modifications.

Cross-context demonstrations in the `cross-ctx/` directory require more advanced setup including applying provided kernel patches to inject victim code into kernel space, recompiling and installing a modified kernel. Additionally, cross-privilege BiasScope demonstrations require disabling Spectre mitigations via bootloader (`mitigations=off`).

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Compiling Intra-Context PoCs

The intra-context demonstrations in `intra-ctx/` require no special setup in the evaluation environment. These self-contained programs can be compiled and executed in standard userspace. To build a test, first clean any existing builds:

```
make clean
```

For native compilation on the target architecture (e.g., building on an x86 machine for itself), use:

```
make TEST=<test_name> ARCH=amd64
```

For cross-compilation, specify the appropriate toolchain using `CROSS_COMPILE=`:

```
make TEST=<test_name> ARCH=aarch64 CROSS_COMPILE=
    aarch64-linux-gnu-
```

Multiple intra-context demonstrations are available in the `intra-ctx/tests/` directory, with executables generated as `build/main`:

- `pht-idx`: BTB/PHT mistraining experiments (Sec. 3.3)
- `spec-bse`: Spectre-BSE attack demonstration (Sec. 5.4)
- `spec-bhs`: Spectre-BHS attack demonstration (Sec. 6.2)
- `chimera`: Spectre-BHS attack demonstration using Chimera gadget (Sec. 7)

Some modules support additional debugging flags that can be passed during compilation by appending `FLAGS="FLAG1 FLAG2 ..."` to the make command:

```
make TEST=spec-bhs ARCH=aarch64 CROSS_COMPILE=
    aarch64-linux-gnu- FLAGS="DBG_ARCH_BH
    DBG_JMP_LATENCY"
```

A.3.2 Compiling Cross-Context User-Space Programs

To compile individual cross-context demonstrations in `cross-ctx/`, use the following command:

```
make $DEMO_NAME FLAGS="$FLAGS" TARGET="$TARGET"
```

The compilation requires the following parameters:

- `DEMO_NAME`: Specifies which demonstration to compile. Available options include:
  - `scope-reader`, `scope-writer`: Complementary user-space programs for the BiasScope attack that demonstrate reading from and writing to the Branch Status Table (BST) to establish a covert side channel.
  - `spec-bse-demo-el1`: Cross-privilege demonstration of the Spectre-BSE attack, exploiting Branch Status Eviction vulnerabilities.
  - `spec-bhs-demo-el1`: Cross-privilege demonstration of the Spectre-BHS attack, leveraging Branch History Speculation mechanisms.
- `TARGET`: Defines the target platform for compilation, selecting appropriate platform-specific parameters from `target.h`. Currently supported targets are `imx8` (i.MX8QuadMax MEK) and `rpi5` (Raspberry Pi 5).
- `FLAGS`: Optional compilation flags that enable specific debugging features or attack variants. Platform-specific flags are detailed in the respective demonstration sections below.

μarch	Attack Compatibility	Validated SoC
ARM Processors		
Cortex-A72 (<= r0p3)	BiasScope, Spectre-BSE	NXP i.MX8QuadMax MEK
Cortex-A76	Spectre-BHS, Chimera	Raspberry Pi 5
Cortex-A78/A78AE	Spectre-BHS, Chimera	NVIDIA Jetson AGX Orin
x86 Processors		
AMD Zen4	Spectre-BHS (only 'intra-ctx/'), Chimera	Ryzen 7 7840U
Intel Gracemont	Spectre-BHS (only 'intra-ctx/'), Chimera	N100
Intel Redwood Cove/Crestmont	Spectre-BHS (only 'intra-ctx/'), Chimera	Core Ultra 7 155H

Table 1: Availability of demos on tested hardware platforms.

A.3.3 Setting Up Custom Syscall Handlers

Cross-context demonstrations require custom syscall handlers on ARM processors to facilitate kernel-space attack components. You must patch the kernel using the code provided in `cross-ctx/kernel/` to establish these kernel-space components.

1. Download the Linux kernel source code for your target platform:
   - For Cortex-A76 (Raspberry Pi 5): Use the Linux kernel `rpi-6.6.y` branch from the Raspberry Pi kernel repository (https://github.com/raspberrypi/linux/tree/rpi-6.6.y).
   - For Cortex-A72 (i.MX8QuadMax MEK): Use the kernel included in the Yocto project distributed by NXP, version `imx-5.15.71-2.2.2` (https://github.com/nxp-imx/imx-manifest/blob/imx-linux-kirkstone/imx-5.15.71-2.2.2.xml).
2. Copy the custom syscall handler files from `custom_syscall/` to the kernel source code directory.
3. Apply the appropriate patch to `include/uapi/asm-generic/unistd.h` to register the syscall numbers:
   - For Raspberry Pi 5: Apply `rpi-6.6.y.diff`
   - For i.MX8: Apply `lf-5.15.71-2.2.2.diff`
4. Compile and install the modified kernel with the custom syscall handlers:
   - For Raspberry Pi 5: Follow the Raspberry Pi kernel documentation (https://www.raspberrypi.com/documentation/computers/linux_kernel.html)
   - For i.MX8: Follow the NXP community guidance (https://community.nxp.com/t5/i-MX-Processors-Knowledge-Base/i-MX-Yocto-Project-How-can-I-quickly-modify-the-kernel-and-test/ta-p/1129551)

A.3.4 Chimera eBPF PoC

To build the Chimera eBPF module in `chimera-ebpf/`, use one of the following commands based on your compilation target and platform:

```
make CROSS_COMPILE=aarch64-linux-gnu- ARCH=aarch64
make ARCH=amd64
```

A.3.5 Installation

This artifact does not require any installation process and is ready to run immediately after compilation. For installing the patched kernel required by cross-context demonstrations, please refer to the platform-specific documentation provided by your SoC or board manufacturer, as detailed in the kernel setup instructions above.

A.3.6 Basic Test

Run the intra-context `pht-idx` demonstration to test whether you can successfully manipulate branch prediction as described in Section 3.3 in the paper. Please follow the instructions in `intra-ctx/README.md` for guidance and expected results.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): On Cortex-A72, the presence of a Branch Status Table yields two potential side-channel attack primitives, Spectre-BSE and BiasScope. This is demonstrated by experiments (E1) and (E3).

(C2): On other processors listed in our paper, we can manipulate the Branch History Speculation mechanism to influence history-based branch prediction and induce unintended mis-speculation through Spectre-BHS primitice. This is demonstrated by experiments (E2), (E3), and (E4).

A.4.2 Experiments

(E1): Passing dummy secrets through BiasScope side channel (1 human-minute + 1 compute-hour + 10GB disk):
How to: Patch the kernel, install the patched kernel, and build the corresponding cross-context demos `scope-reader.c` and `scope-writer.c` in `cross-ctx/`.
Preparation and execution: Follow the instructions in `cross-ctx/README.md` within the code package to build the demo with kernel-user or user-user flavor. Note that to demonstrate the kernel-user flavor of this primitive, you must patch the kernel. Compiling the patched kernel may take longer time and more disk space.
Results: By running the built demo program with parameters described in `cross-ctx/README.md`, you should be able to see the dummy secret being decoded by the `scope-reader.c` program.

(E2): Inducing mis-speculation using Spectre-BSE and Spectre-BHS in the same user process (1 human-minute + 1 compute-minute + 1MB disk):
How to: Build the intra-context demos in `intra-ctx/`.
Preparation and execution: Follow the instructions in `intra-ctx/README.md` within the code package to build your selected demo.
Results: By running the built demo program with parameters described in `intra-ctx/README.md`, you should be able to see microarchitectural marks indicating a successful Spectre-BSE or Spectre-BHS attack.

(E3): Inducing kernel-space mis-speculation using Spectre-BSE and Spectre-BHS from user-space (1 human-minute + 1 compute-hour + 10GB disk):
How to: Patch the kernel, install the patched kernel, and build the cross-context demos in `cross-ctx/`.
Preparation and execution: Follow the instructions in `cross-ctx/README.md` within the code package to build your selected demo. Note that compiling the patched kernel may take longer time and more disk space.
Results: By running the built demo program with parameters described in `cross-ctx/README.md`, you should be able to see microarchitectural marks indicating a successful cross-privilege Spectre-BSE or Spectre-BHS attack.

(E4): Dumping kernel memory with Chimera eBPF PoC (1 human-minute + 1 compute-minute + 1MB disk):
How to: Build the demo in `chimera-ebpf/`.
Preparation and execution: Follow the instructions in `chimera-ebpf/README.md` within the code package to build your selected demo. Note that you may also need to have a known byte sequence in a known kernel address to evaluate the accuracy of the disclosure primitive.
Results: By running the built demo program with parameters described in `chimera-ebpf/README.md`, you should be able to dump a piece of kernel memory at a given address and given length using the side-channel primitive.

A.5 Notes on Reusability

The submodule `intra-ctx/` provides a versatile framework to monitor and evaluate speculative execution. Researchers interested in similar work may implement their own test modules to induce desired mis-speculation using novel techniques. We plan to share updated versions of this code in our git repository in the future.

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/.