Indirector: High-Precision Branch Target Injection Attacks Exploiting the Indirect Branch Predictor

Luyi Li, Hosein Yavarzadeh, and Dean Tullsen, *UC San Diego*

https://www.usenix.org/conference/usenixsecurity24/presentation/li-luyi

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: **Indirector: High-Precision Branch Target Injection Attacks Exploiting the Indirect Branch Predictor**

Luyi Li* Hosein Yavarzadeh* Dean Tullsen

University of California San Diego

* Equal contribution joint first authors

A Artifact Appendix

A.1 Abstract

In this artifact, we describe Indirector, a collection of reverse engineering code and branch injection attacks. For reverse engineering, we provide detailed assembly benchmarks to uncover the architecture of the Branch Target Buffer (BTB) and the Indirect Branch Predictor (IBP) on modern Intel CPUs. Additionally, we analyze the microarchitectural impacts of Intel Spectre v2 mitigation techniques. For attack implementation, we present the proof-of-concept of iBranch Locator, a tool that can accurately locate any indirect branch within the IBP. Leveraging this tool, we demonstrate two high-precision injection attacks that target the IBP and BTB respectively. Furthermore, we offer a method of exploiting the IBP and BTB to break Address Space Layout Randomization (ASLR).

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Indirector requires the installation of a custom kernel module to enable reading different performance counters on Intel CPUs. No additional changes to the kernel code are necessary. Disabling secure boot in the BIOS/UEFI setup menu is also required. If the module is not installed successfully, the assembly benchmark may cause a segmentation fault. Please exercise caution.

A.2.2 How to access

The source code of Indirector is available in this Github repository: https://github.com/owenlly/Indirector_Artifact/tree/1926f7284486827b96fc5c1493b867d95c3145ed

A.2.3 Hardware dependencies

In our repository, the assembly benchmarks are primarily designed for Golden Cove and Raptor Cove, the P-core microarchitectures of 12th and 13th Gen Intel® Core™ CPUs. The exact models used in our paper are:

- i9-12900 running Ubuntu 22.04.4 LTS with Linux kernel 5.10.209
- i7-13700K running Ubuntu 20.04.6 LTS with Linux kernel 5.15.89
- i9-13900KS running Ubuntu 22.04.4 LTS with Linux kernel 6.5.0-35-generic

All the benchmarks should be compatible with the P-core of any 12th or 13th Gen Intel® Core™ CPU.

A.2.4 Software dependencies

Our assembly benchmark is implemented based on "Test programs for measuring clock cycles and performance monitoring" from Agner Fog: http://www.agner.org/optimize. We have included all the required files in the repository. Each benchmark is written in x86 assembly and compiled by the NASM assembler (version 2.14.02).

A.2.5 Benchmarks

None

A.3 Set-up

A.3.1 Installation

We have included scripts to install all the required packages. For more stable results, we also provide scripts for setting the CPU to performance mode and turning off the hardware prefetcher. Enabling SMT is required for the core-ID test, and we include a script to ensure SMT is enabled.

A.3.2 Basic Test

In the template test, one direct branch is iterated 1000 times and the entire test is repeated 5 times. If everything is set up correctly, the output will be in the *results.out* file.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Indirector is able to reveal that the BTB on Intel CPUs records metadata including branch type and core-ID. **(E1, E2)** prove this.

(C2): Indirector is able to reverse engineer the IBP on Intel CPUs, including its input, associativity, TAGE-like structure, and the precise functions governing index and tag hashing. **(E3, E4, E5, E6, E7)** prove this.

(C3): Indirector is able to reveal the indirect branch prediction policy under different BTB and IBP states. **(E8)** proves this.

(C4): Indirector is able to reverse engineer how Intel Spectre v2 defenses, including IBRS, STIBP, and IBPB, impact the states of the BTB and IBP. **(E9)** proves this.

(C5): iBranch Locator is capable of finding entry aliasing with any indirect branch within the IBP without requiring prior history information about the branch. **(E10, E11)** prove this.

(C6): Using iBranch Locator, Indirector introduces two high-precision injection attacks targeting the IBP and BTB respectively, and one efficient method to break ASLR. **(E12, E13, E14)** prove this.

A.4.2 Experiments

Preparation: Finish the initialization in Section A.3.1.

(E1): [1 human-minute + 1 compute-minute]: Test if the BTB has a branch type check for indirect branch prediction.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_1_1_BTB_branch_type* for example usage and expected results under different configurations.
Results: Only aliased indirect branches can mislead the victim indirect branch for speculative execution.

(E2): [1 human-minute + 1 compute-minute]: Test if the BTB entry is tagged with core-ID.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_1_2_BTB_core-ID* for example usage and expected results under different configurations.
Results: There are 11 available ways when both aliased branches are on the same SMT core and 10 available ways when they are on different SMT cores.

(E3): [1 human-minute + 1 compute-minute]: Test the PC address input for the IBP and reproduce Figure 4.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_2_1_IBP_pc_input* for reproducing Figure 4.
Results: IBP uses PC[15:6] as input.

(E4): [1 human-minute + 1 compute-minute]: Test if the IBP has a set-associative structure and reproduce Figure 5.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_2_2_IBP_associativity_lite* for reproducing Figure 5.
Results: The misprediction rate rises when the number of indirect branches exceeds 6.

(E5): [1 human-minute + 60 compute-minutes]: Test if the IBP has a TAGE-like structure and reproduce Figure 6.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_2_2_IBP_ associativity_complete* for reproducing Figure 6.
Results: IBP has 3 2-way set-associative tables, each indexed by a different length of branch history.

(E6): [1 human-minute + 1 compute-minute]: Verify the hash functions for the IBP index and tag.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_2_4_IBP_hash* for verifying Figure 7 and 8.
Results: Injection is successful when using the results of this aliasing search tool.

(E7): [1 human-minute + 1 compute-minute]: Test if the IBP entry is tagged with core-ID.
Execution: Please refer to the *README* file under *Sec3_Reverse_Engineering/3_2_6_IBP_core-ID*.
Results: Injection fails when aliased indirect branches are on different SMT cores.

(E8): [2 human-minutes + 2 compute-minute]: Test BTB and IBP interactions in Table 2.
Execution: Please refer to the *README* files under *Sec3_Reverse_Engineering/3_3_1_BTB_hit* and *Sec3_Reverse_Engineering/3_3_2_BTB_miss*.
Results: The results should align with Table 2.

(E9): [2 human-minutes + 2 compute-minute]: Test the impacts of Intel Spectre v2 defenses on the BTB and IBP in Table 3.
Execution: Please refer to the *README* files under *Sec4_Intel_Defense/4_1_BTB* and *Sec4_Intel _Defense/4_2_IBP*.
Results: The results should align with Table 3.

(E10): [1 human-minute + 2 compute-minutes]: Test the iBranch Index Locator and verify Figure 11.
Execution: Please refer to the *README* file under *Sec5_iBranch_Locator/5_1_index_locator_poc*.
Results: There are misprediction spikes when scanning the IBP set #184 and #384.

(E11): [1 human-minute + 2 compute-minutes]: Test the iBranch Tag Locator and verify Figure 12.
Execution: Please refer to the *README* file under *Sec5_iBranch_Locator/5_2_tag_locator_poc*.
Results: There are misprediction spikes when the tag value are 536 and 855.

(E12): [1 human-minute + 1 compute-minute]: Test cross-process IBP injection attack.
Execution: Please refer to the *README* file under *Sec6_Injection_Attack/6_1_IBP_injection_poc*.
Results: The secret is extracted successfully.

(E13): [2 human-minute + 2 compute-minutes]: Test cross-process BTB injection attack and verify Figure 13.
Execution: Please refer to the *README* file under *Sec6_Injection_Attack/6_2_BTB_injection_poc* and *Sec6_Injection_Attack/6_2_BTB_injection_plot*.
Results: The secret is extracted successfully only under IBP eviction and BTB injection.

(E14): [1 human-minute + 30 compute-minutes]: Test the method of breaking ASLR exploiting the BTB and IBP.
Execution: Please refer to the *README* file under *Sec6_Injection_Attack/6_3_ASLR_poc*.
Results: The randomized bits are successfully broken, enabling BTB injection.

A.5 Notes on Reusability

Our reverse engineering methodology can be applied to most Intel flagship processors from *Skylake* to *Raptor Lake (P-core + E-core)*, given their consistent TAGE-like structure. However, certain features, such as the PHR update hash function, BTB/IBP mapping function, and BTB/IBP associativity, may differ among microarchitectures. Users will need to make relative modifications to the code accordingly.

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/usenixsec2024/`.