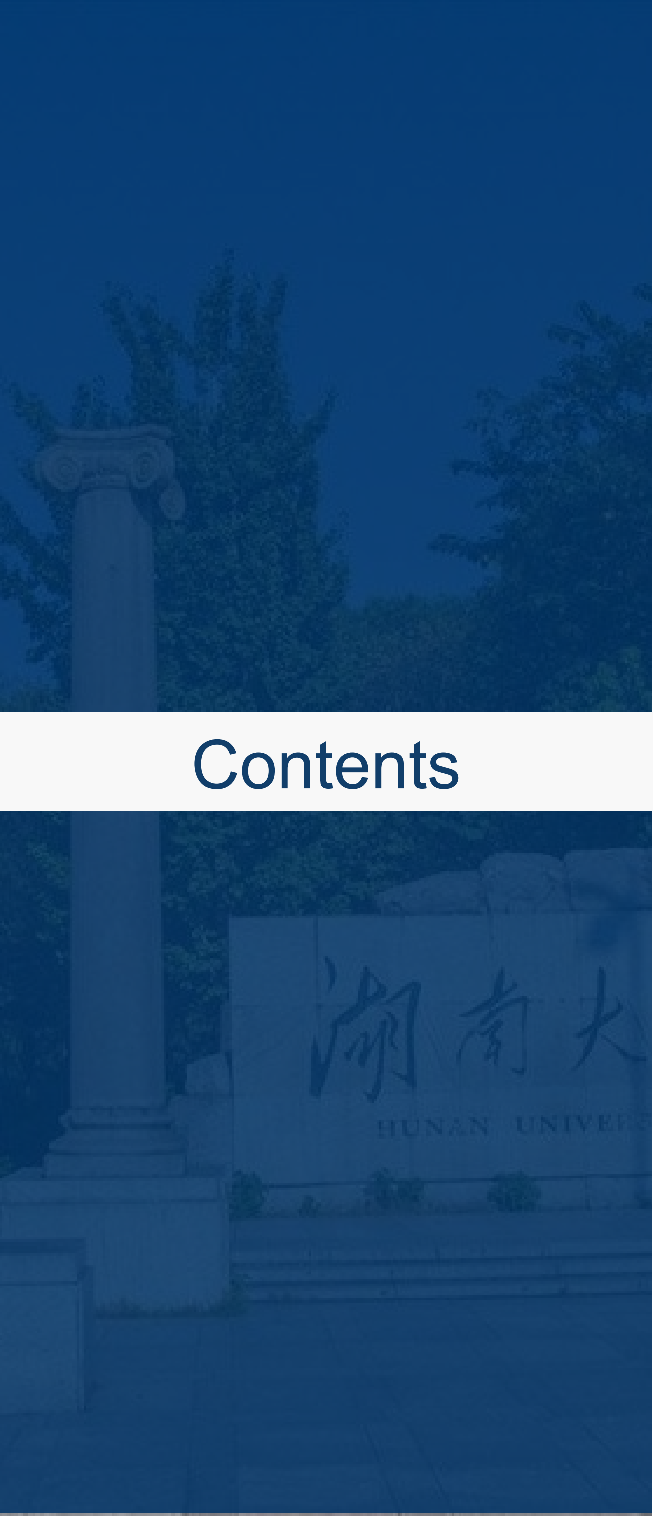
Lightweight and Secure Branch Predictors against Spectre Attacks

Congcong Chen, Chaoqun Shen, Jiliang Zhang*
College of Computer Science and Electronic Engineering
Hunan University, Changsha, China

January 17 to January 20, 2022



Contents

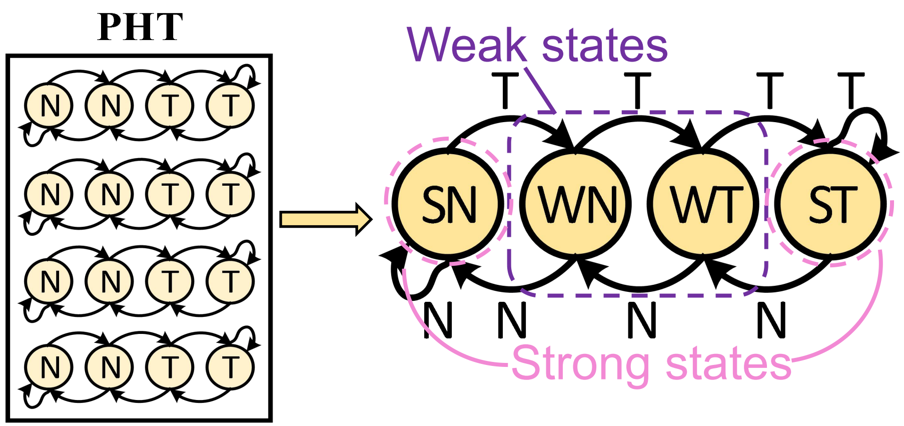
Background

A Example of Spectre Attack

```
if (x < T_size)
    y = Probe[T[x]*256];
```

A code snippet of Spectre attack

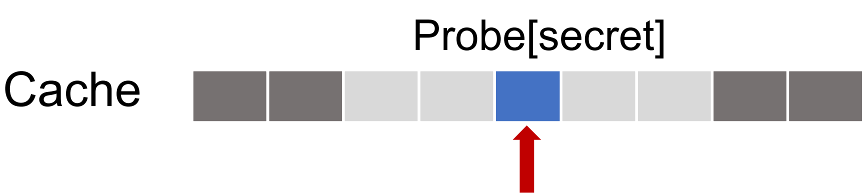
① Poisoning phase



② Execution phase

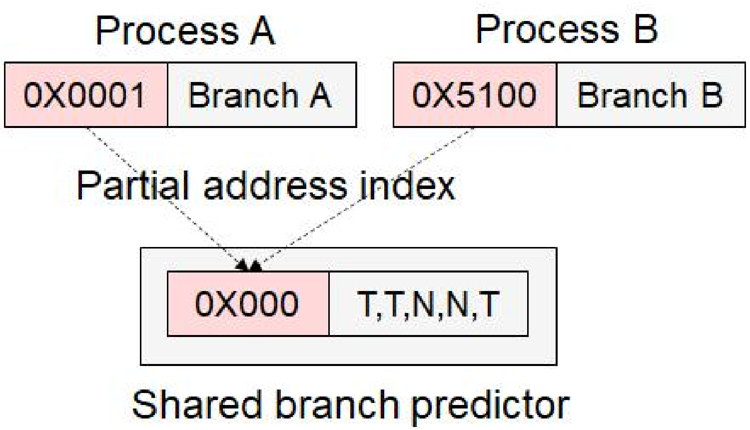
Let x = secret – T, that is, T[x] = *secret

③ Measurement phase

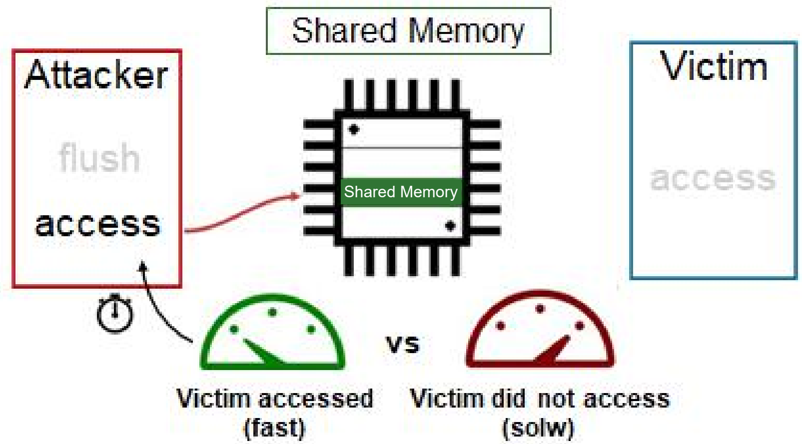


Background

How Spectre Attacks Happen:

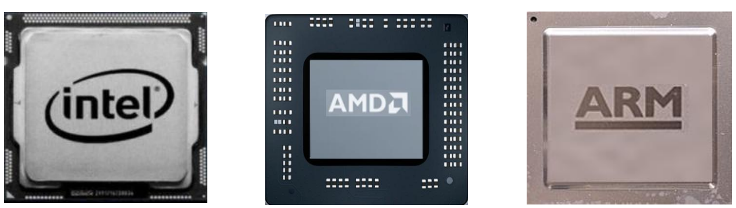


a) Branch mistraining



b) Side-channel attacks

Impact of Spectre Attacks Today



Affect billions of Intel, AMD, and ARM devices

$11.3 billion

Intel's shares fell 5.2 percent in two days, wiping $11.3 billion off its market value

Spectre Attack Types

- Spectre attack variants

Variants	Component	Primitive
Spectre-PHT (V1.0/1.1)	PHT/BTB	Bounds Check Bypass on Loads/Stores
Spectre V1.2	PTE	Read-only Protection Bypass
Spectre-BTB(V2)	BTB	Branch Target Injection
Spectre V4	SB	Speculative Store Bypass
BranchScope	PHT	Directional Branch Prediction
Spectre-RSB	RSB	Return Stack Buffer Speculative

- The root cause of all these attacks

Sharing branch predictors
among mutually distrusting processes

It’s so hard to defend against Spectre attacks!

Existing Spectre defenses

- **Eliminating or reducing the accuracy of covert channels**
  - ✓ The covert channels can be eliminated by making the execution time constant or disturbing the access time to interfere with the attacker's measurements

However, most of these methods only protect the cache hierarchy, the attackers can still exploit other covert channels.

- **Limiting transient execution to affect microarchitectural states.**
  - ✓ It makes the changes of microarchitectural state invisible to the attacker.

However, with the strict enforcement of security policies, most of them introduce unacceptable performance overhead

- **Limiting the execution of transient instructions**
  - ✓ These schemes find the gadget in the target code that may leak secrets by static or dynamic analysis. Then, fence instructions are used to serialize branches to prevent transient execution from leaking secrets.

However, they need to capture the code fragments in the binary code that can be exploited by the attacker, which produces high false negative and false positive rates

- **Isolation-based secure branch predictors (BPs)**
  - ✓ These schemes refresh BPs at context switches or use hardware to isolate BPs in different process spaces

However, these strategies either introduce a large context switching overhead or have low hardware resource utilization

Motivation

With these existing Spectre defenses,

It's difficult to balance the security and overhead

Our Works

- **Goal:** Achieve low-overhead and high-security defense on Spectre
- **Our main contributions:**

A lightweight and secure branch predictor (LS-BP) is designed

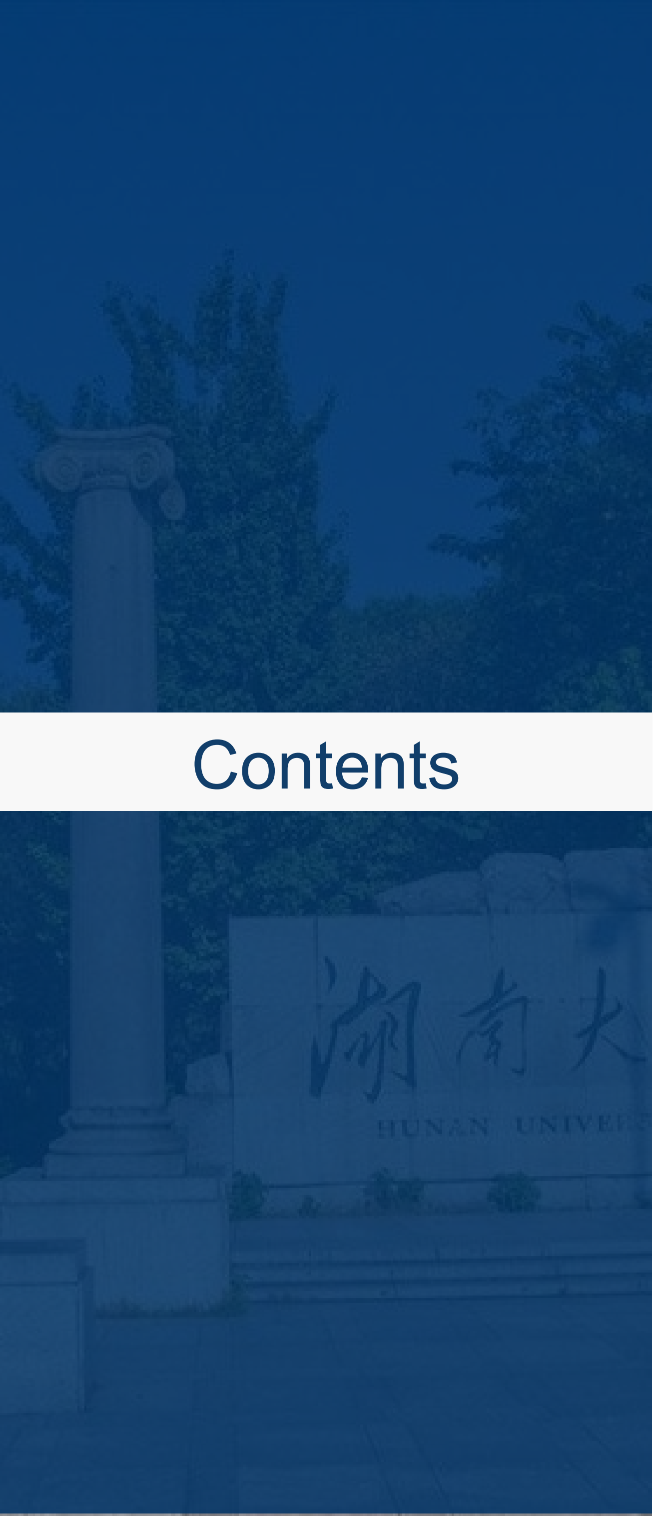
The proposed LS-BP does not restrict speculative execution to minimize performance overhead.

Detail security analysis against Spectre attacks is given

We show how the LS-BP can break the shared BP among mutually distrusting processes, thereby mitigating typical Spectre-PHT and Spectre-BTB attacks.

The proposed LS-BP is implemented and evaluated in detail

We simulate four different BPs to evaluate our design on gem5 with SPEC2006. Experimental results show that the performance overhead of LS-BP is less than 3%.



Contents

Threat Model

Suppose the attacker is co-located with the victim on a same physical core

The attacker can control over a process running on the target system with normal user privileges. We refer to this process as the spy process.

Suppose the execution of the victim's program is credible

The attacker cannot modify its program control flow or access memory directly to guarantee the confidentiality and integrity of the victim's information during the normal execution.

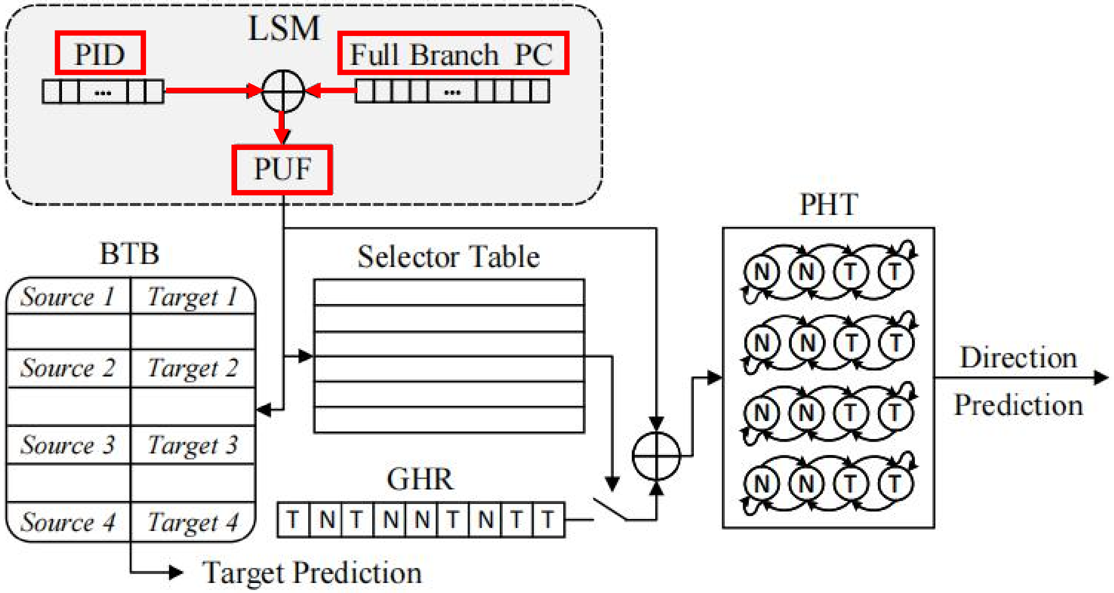
Suppose the attacker knows the gadget address that could disclose secret

The attacker can manipulate the input to mistrain the branch.

Goal: Eliminate Spectre attacks related to the branch misprediction, rather than all speculative execution attacks

The Proposed LS-BP

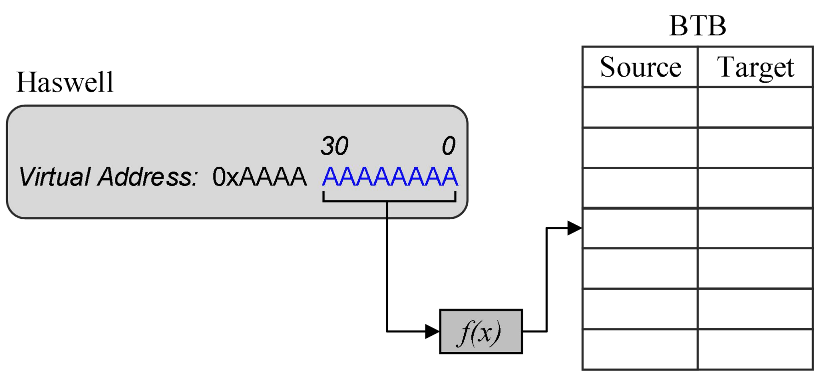
- The overall design



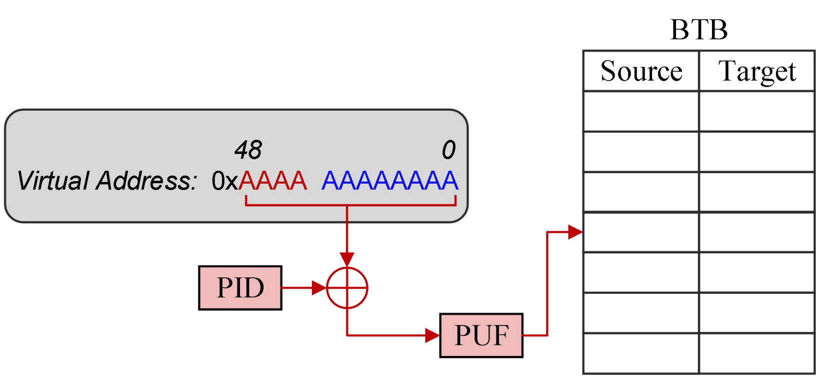
- The process identification (PID) is introduced
- The full branch source address bits are used
- PUF is used to generate a unique index without increasing the number of index bits

The Proposed LS-BP

Implementation of LS-BTB

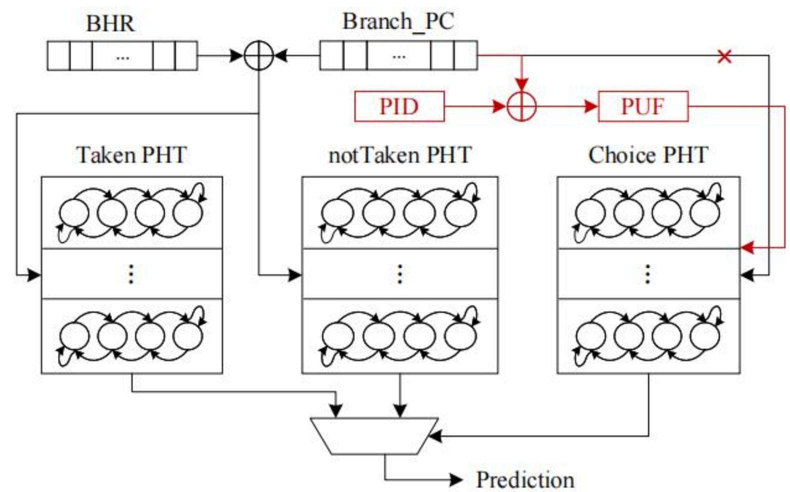


The reverse engineering of the BTB addressing

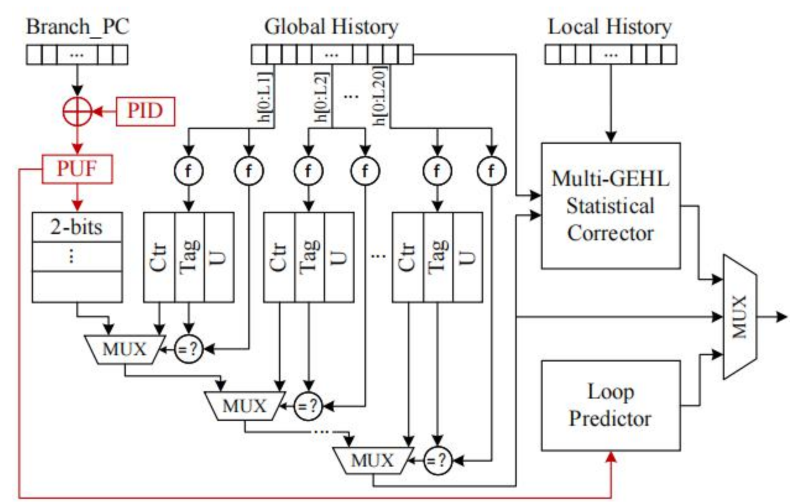


a) The LS-BTB addressing

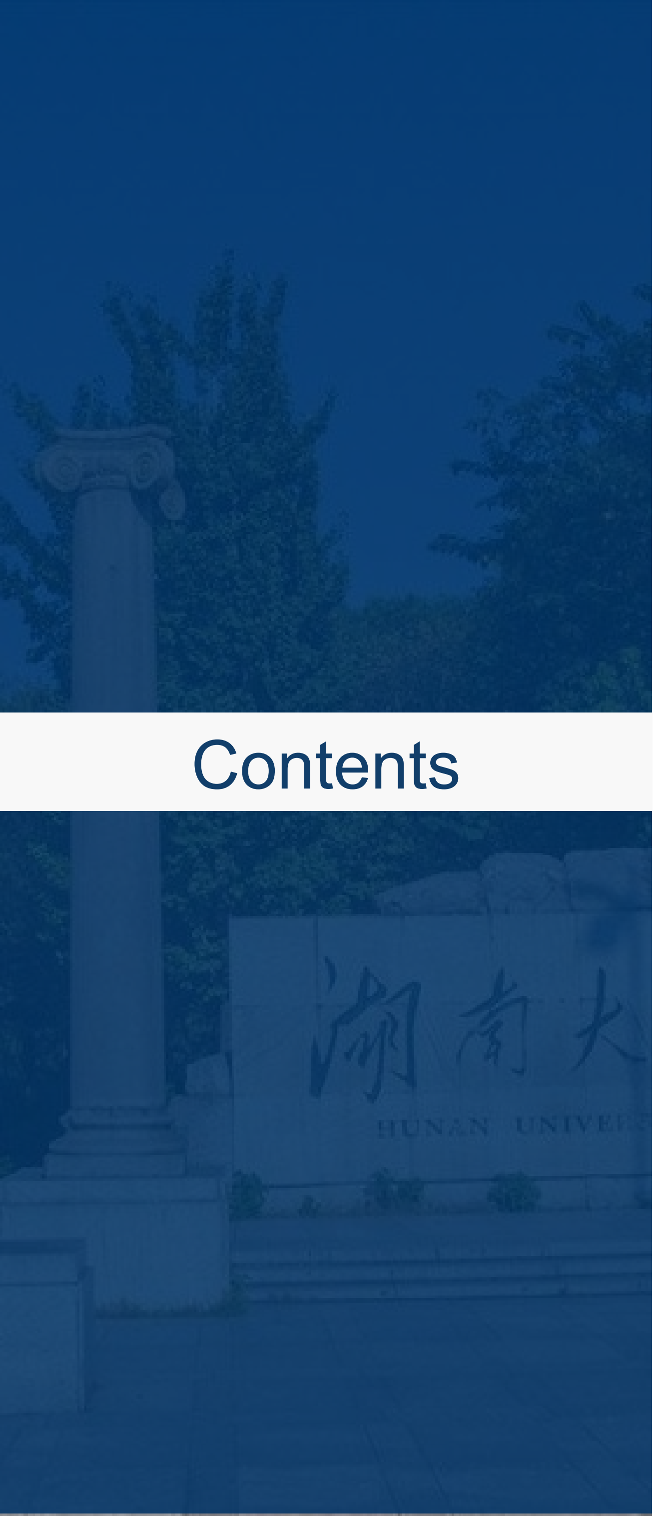
Implementation of LS-PHT



b) LS-PHT on bi-mode predictor



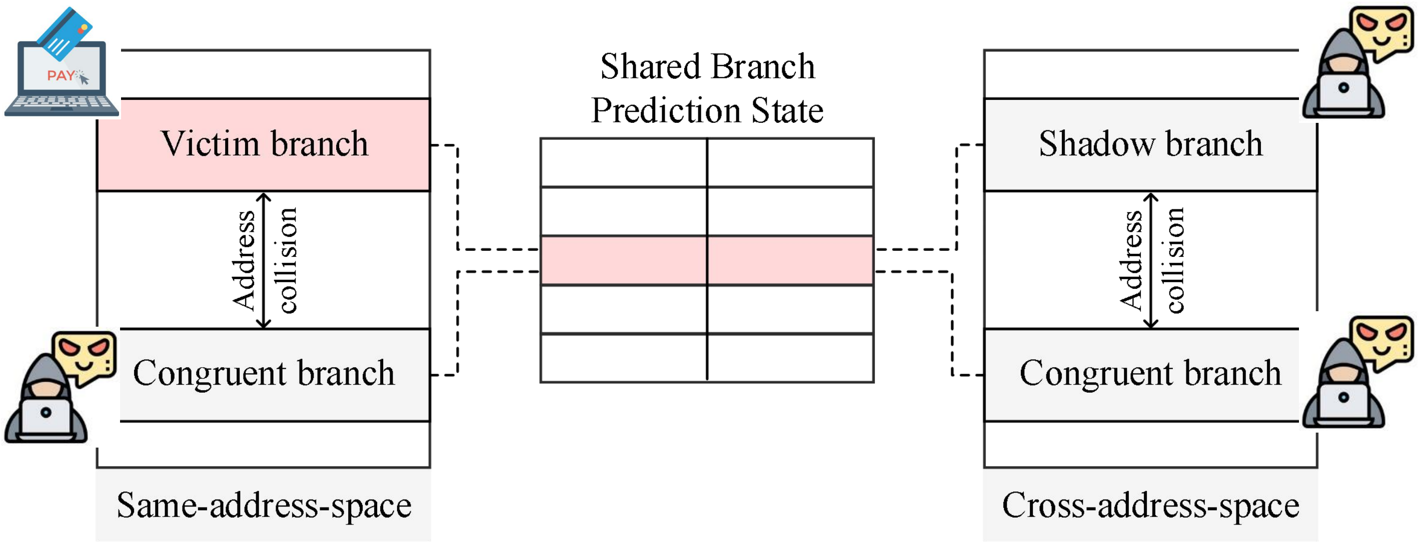
c) LS-PHT on TAGE_SC_L predictor



Contents

Security Analysis

- mistraining a branch



- **Same-address-space mistraining**. with the full virtual address bits for branch addressing, there is no congruent branch with the victim branch because all branches are located at different virtual addresses.

- **Cross-address-space mistraining**. After adding PID, a process's history in the Prime phase becomes unrecognizable in the Probe phase due to context switches

Security Analysis

- A PoC of branch mistraining

```
/* Mistraing process */
  for (int i = bound; i < bound + 100; i++)
     exploit_function (i);   // mistrian banch
  clfush (&bound);
  clfush (&exp_value);  // flush cache line
-------------------------------------------
/* Victim process */
  x = tries % bound;  // x∈[0, bound-1]
  // shared function
  viod exploit_function (int x){
  if (x < bound)
     temp &= value;
  else
     temp &= exp_value;
  }
-------------------------------------------
/*  Probe process */
  rdtscp ( );
  temp &= exp_value;  //  reload time
  rdtscp ( );
```

The experimental results show the accuracy of training BTB and PHT on a baseline without any defense is 97.2% and 98.5%, respectively. However, with LS-BP enabled, the attacker can hardly affect the victim's branch prediction (the accuracy is less than 1%).

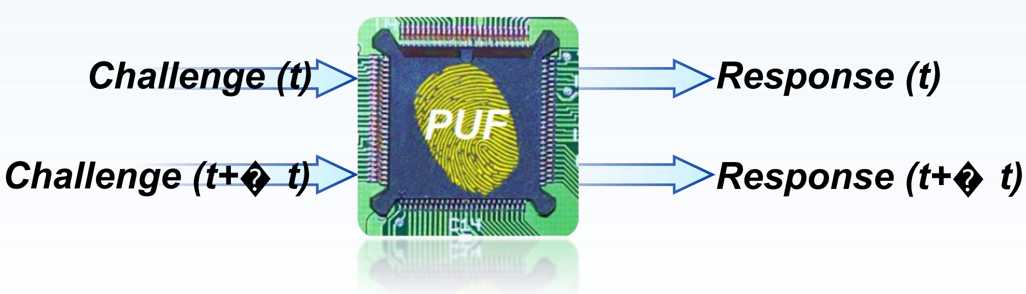
Security Analysis

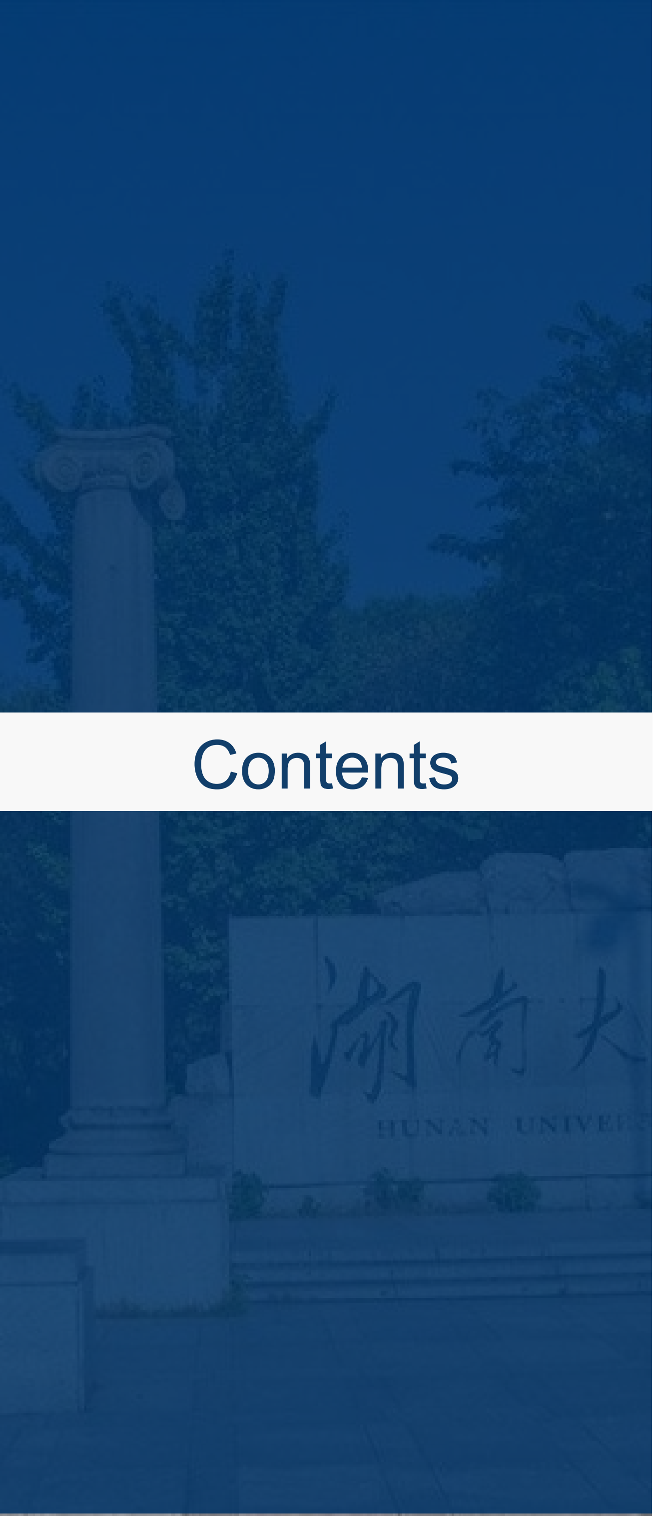
Why is PUF introduced

- Each PUF has its own unique mapping relationship that differs from the traditional mapping schemes. Even if an attacker learns the branch mapping relationship of a machine, he cannot directly use it for other machines.

- PUF is a one-way physical cryptographic function. Inputting different challenges to the PUF will generate unique and unpredictable responses.

- the PUF response is generated in real-time without storing the key, which makes our scheme higher security against memory-based attacks.





Contents

Experimental Setup

- The gem5 x86 simulator setup

TABLE II
SIMULATED PROCESSOR CORE CONFIGURATION

Parameter	Configurations
ISA	X86
Frequency	2.5GHz
CPU-type	DerivO3CPU
Pipeline	Decode-width=8, Fetch-width=8, Issue-width=8 Commit-width=8
ROB/LQ/SQ	352/127/72 entries
Issue Queue	120
BTB	4096 entries
PHT	bi-mode: 8192/8192 entries for global/choice predictor Tournament: 2048/8192/8192 entries for local/global/choice predictor TAGE: 32KB TAGE_SC_L: 66.6KB
TLB	64 entries
L1 ICache	32KB, 4-way, 64B line
L1 DCache	64KB, 4-way, 64B line
L2 Cache	512KB, 16-way, 64B line
L3 Cache	4MB, 32-way, 64B line

- Four different PHT predictors: including bi-mode, Tournament, TAGE, and TAGE SC L

- We randomly selected 12 combinations from the SPEC2006 benchmark suite

TABLE III
BENCHMARK COMBINATIONS USED IN OUR EVALUATION

Mix	Components	Mix	Components
gcc+cal	gcc, calculix	mil+pov	milc, povray
bzi+sop	bzip2, soplex	nam+les	namd, leslie3d
hmm+lbm	hmmer, lbm	gob+h26	gobmk, h264ref
gro+lbm	gromacs, lbm	mcf+sje	mcf, sjeng
sop+hmm	soplex,hmmer	sje+gcc	sjeng, gcc
mcf+per	mcf, perlbench	cal+nam	calculix, namd

Experimental Results

- The performance of LS-BTB

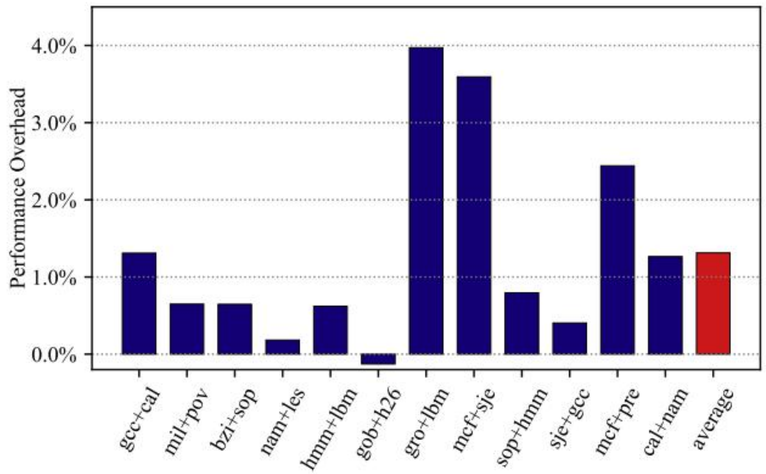


Fig. 4. The performance overhead of LS-BTB

- The performance of LS-PHT

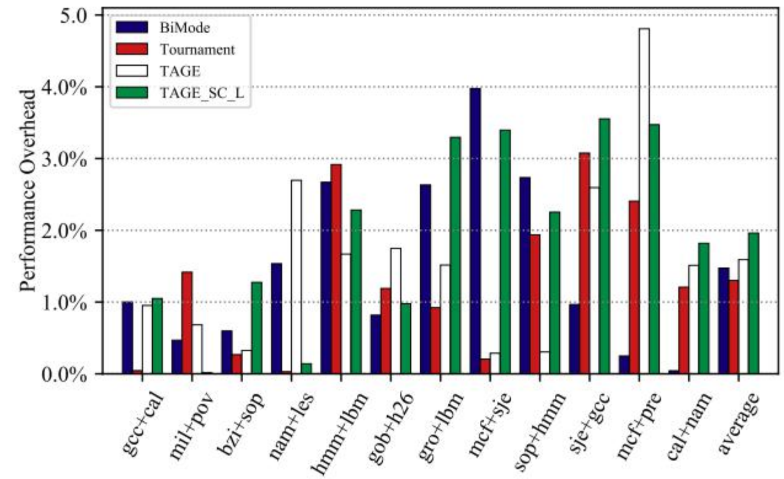


Fig. 7. The performance overhead of LS-PHT

- Combination of LS-PHT and LS-BTB

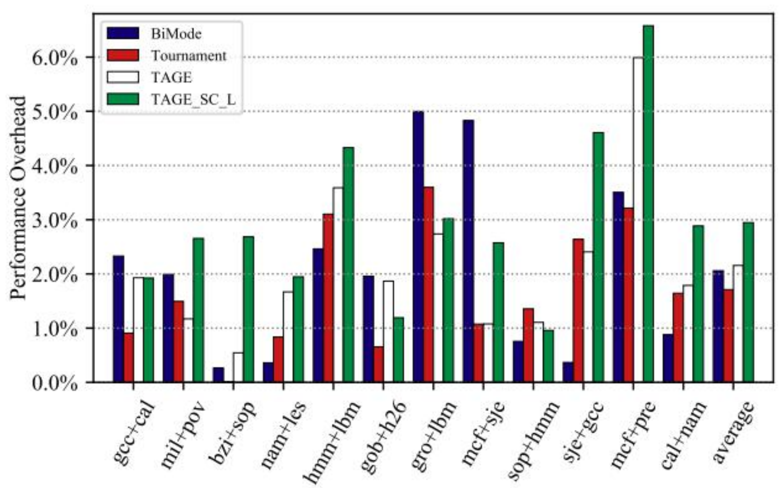


Fig. 8. The performance overhead of LS-BP

- Comparison

In context switching scenarios, the performance overhead of LS-BP (1.71% ~ 2.95%) is generally lower for same PHT predictors in [18] (2.3% ~ 4.8%).

[18] L. T. Zhao, P. N. Li, R. Hou, et al. “A Lightweight Isolation Mechanism for Secure Branch Predictors,” DAC2021.

Summary

- we design a lightweight and secure branch prediction (LS-BP) to provide secure isolation for branch prediction state of same-address-space and cross-address-space.
- The proposed LS-BP can effectively mitigate Spectre attacks based on BTB and PHT
- Experimental results on four branch predictors show that LS-BP brings less than 3% performance overhead. We believe that such a secure branch predictor will be integrated into future processors.

Thank you!
Q&A

chencongcong@hnu.edu.cn