ETH zürich

COMSEC

An Evaluation of Speculative Retbleed

Jean-Claude Graf

20. July 2022

Introduction

- RETBLEED is a new transient execution attack
- PF based mitigation introduces zero overhead
  - Is said to be incomplete

Research Question 1

Can be build a PF free variant of RETBLEED?

- In-depth mitigations seem to introduced large overheads

Research Question 2

What is the performance overhead of the in-depth mitigation?

Introduction

- RETBLEED is a new transient execution attack
- PF based mitigation introduces zero overhead
  - Is said to be incomplete

Research Question 1

Can be build a PF free variant of RETBLEED?

- In-depth mitigations seem to introduced large overheads

Research Question 2

What is the performance overhead of the in-depth mitigation?

Introduction

- RETBLEED is a new transient execution attack
- PF based mitigation introduces zero overhead
  - Is said to be incomplete

Research Question 1

Can be build a PF free variant of RETBLEED?

- In-depth mitigations seem to introduced large overheads

Research Question 2

What is the performance overhead of the in-depth mitigation?

Introduction

- RETBLEED is a new transient execution attack
- PF based mitigation introduces zero overhead
  - Is said to be incomplete

Research Question 1

Can be build a PF free variant of RETBLEED?

- In-depth mitigations seem to introduced large overheads

Research Question 2

What is the performance overhead of the in-depth mitigation?

Outline

Branch Prediction Unit

- Predicts the target of a branching instruction
  - If the destination takes some time to be evaluated
- Consists of multiple branch predictors

Direct/Indirect Branch Predictor

Assumes: Branches go to same location as they went before

Implemented: BTB which is indexed by PC and auxiliary structures like BHB

Return Instruction Predictor

Assumes: Function return to where they are called from

Implemented: RSB

Property: Falls back to BTB on:

- RSB underflow (CoffeeLake)
- Collision with indirect branch (Zen1/Zen2)

Branch Prediction Unit

- Predicts the target of a branching instruction
  - If the destination takes some time to be evaluated
- Consists of multiple branch predictors

Direct/Indirect Branch Predictor

Assumes: Branches go to same location as they went before

Implemented: BTB which is indexed by PC and auxiliary structures like BHB

Return Instruction Predictor

Assumes: Function return to where they are called from

Implemented: RSB

Property: Falls back to BTB on:

- RSB underflow (CoffeeLake)
- Collision with indirect branch (Zen1/Zen2)

Branch Prediction Unit

- Predicts the target of a branching instruction
  - If the destination takes some time to be evaluated
- Consists of multiple branch predictors

Direct/Indirect Branch Predictor

Assumes:	Branches go to same location as they went before
Implemented:	BTB which is indexed by PC and auxiliary structures like BHB

Return Instruction Predictor

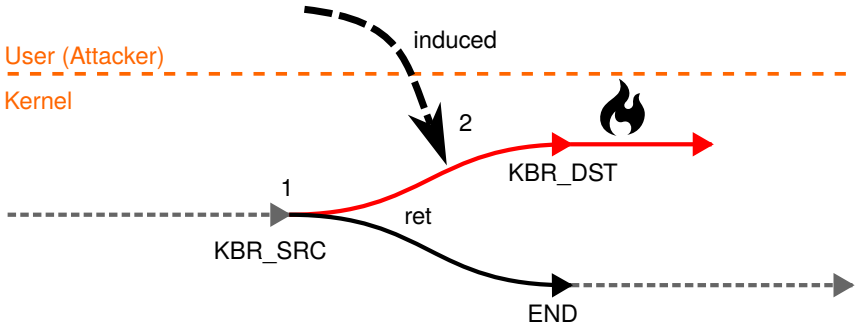
Assumes:	Function return to where they are called from
Implemented:	RSB
Property:	Falls back to BTB on: – RSB underflow (CoffeeLake) – Collision with indirect branch (Zen1/Zen2)

Retbleed

- Is a Spectre V2 like attack targeting return instruction
- Requires two primitive:
  1. RSB falls back to BTB
  2. BTI works across privilege boundaries

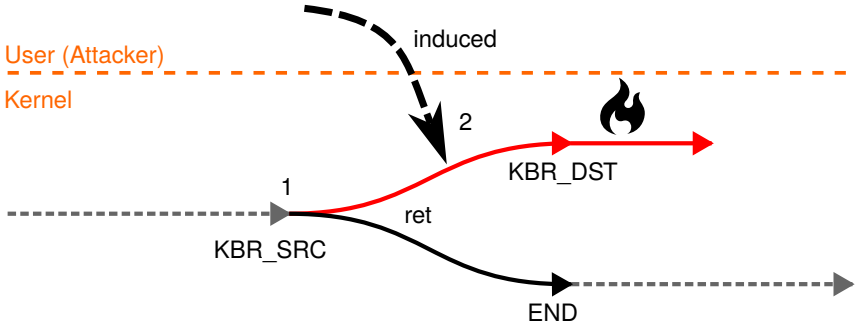
Retbleed

- Is a Spectre V2 like attack targeting return instruction
- Requires two primitive:
  1. RSB falls back to BTB
  2. BTI works across privilege boundaries



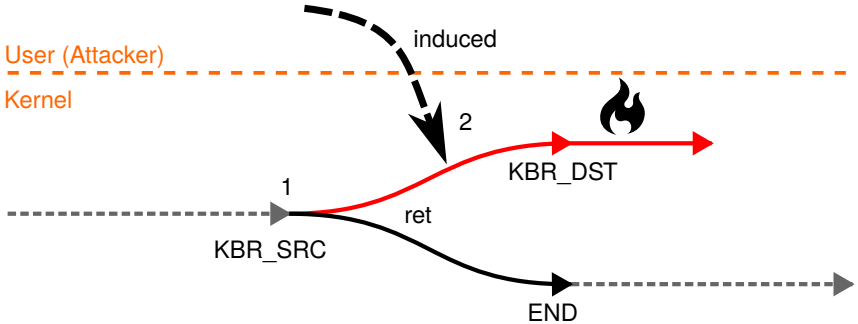
Retbleed

- Is a Spectre V2 like attack targeting return instruction
- Requires two primitive:
  1. RSB falls back to BTB
  2. BTI works across privilege boundaries



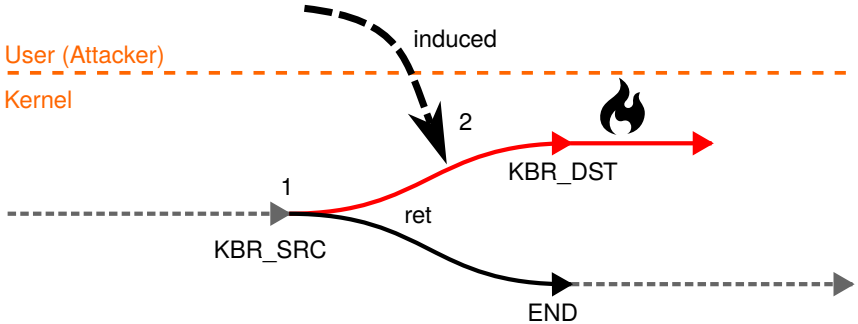
Retbleed

- Is a Spectre V2 like attack targeting return instruction
- Requires two primitive:
  1. RSB falls back to BTB
  2. BTI works across privilege boundaries



Retbleed

- Is a Spectre V2 like attack targeting return instruction
- Requires two primitive:
  1. RSB falls back to BTB
  2. BTI works across privilege boundaries



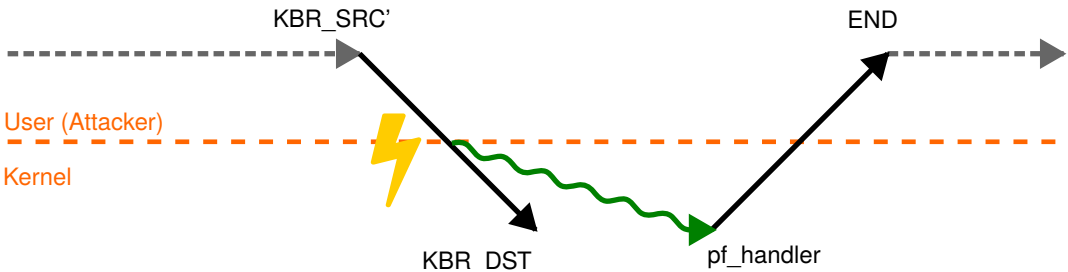
Retbleed
Why are PFs cause?

- BTI across privilege boundaries
  - Indirect jump from `KBR_SRC'` to `KBR_DST`
    - `KBR_SRC` and `KBR_SRC'` collide

Retbleed

Why are PFs cause?

- BTI across privilege boundaries
  - Indirect jump from `KBR_SRC'` to `KBR_DST`
    - `KBR_SRC` and `KBR_SRC'` collide



Outline

Goal

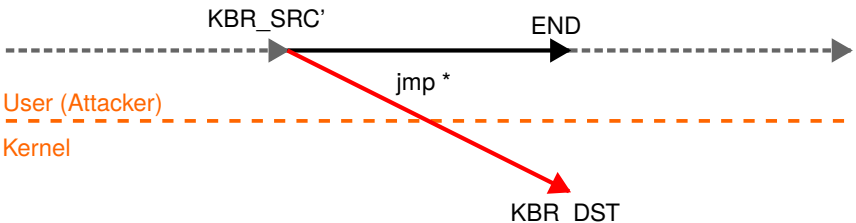
Can be build a PF free variant of RETBLEED?

The Idea

⇒ **Speculative BTI**

The Idea

⇒ **Speculative BTI**

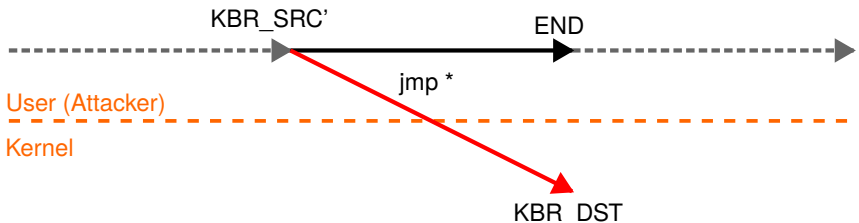


The Idea

⇒ **Speculative BTI**

Pros:

- No PF is raised
- BTI still works



The Idea

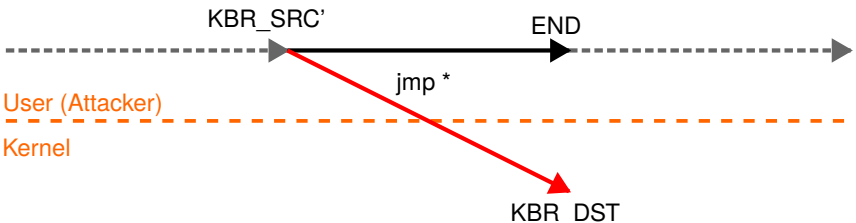
⇒ **Speculative BTI**

Pros:

- No PF is raised
- BTI still works

Cons:

- Does it actually work?



Speculative RET-BTI

RET-BTI PoC in Detail

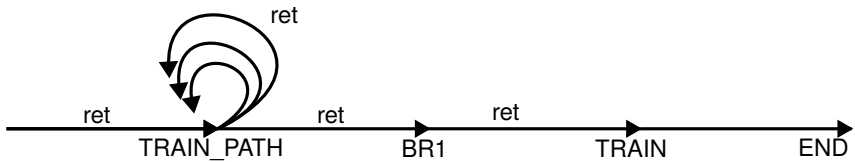
- Verify that spec BTI works in same privilege domain
- RET-BTI exploits the first required primitive

Speculative RET-BTI

RET-BTI PoC in Detail

- Verify that spec BTI works in same privilege domain
- RET-BTI exploits the first required primitive

Training Phase:

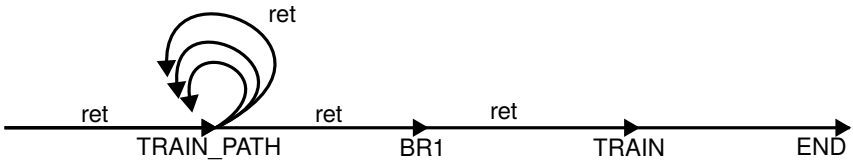


Speculative RET-BTI

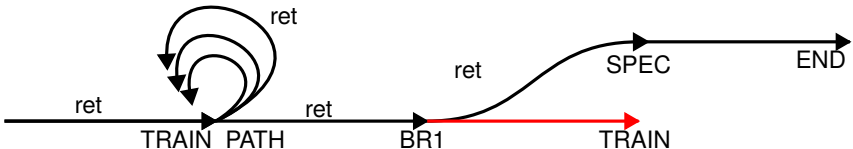
RET-BTI PoC in Detail

- Verify that spec BTI works in same privilege domain
- RET-BTI exploits the first required primitive

Training Phase:



Speculation Phase:



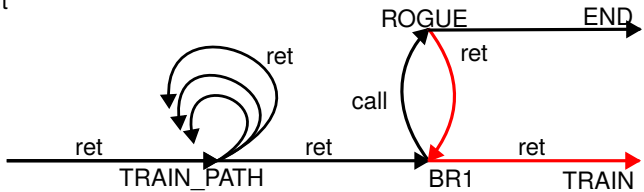
Speculative RET-BTI

- Use SpectreRSB to cause speculation
- History needs to be equivalent

Speculative RET-BTI

- Use SpectreRSB to cause speculation
- History needs to be equivalent

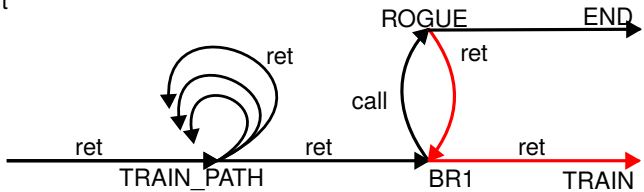
Training Phase:



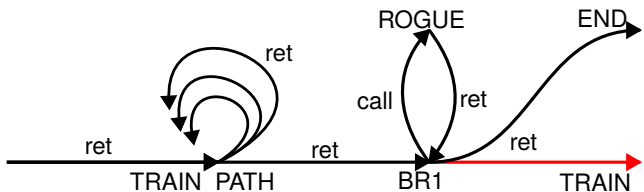
Speculative RET-BTI

- Use SpectreRSB to cause speculation
- History needs to be equivalent

Training Phase:



Speculation Phase:



Speculative RET-BTI
Results

- Speculative BTI works in same privilege domains
  - For Intel CoffeeLake and AMD Zen1, Zen1+ and Zen2

Speculative CP-BTI

CP-BTI PoC in Detail

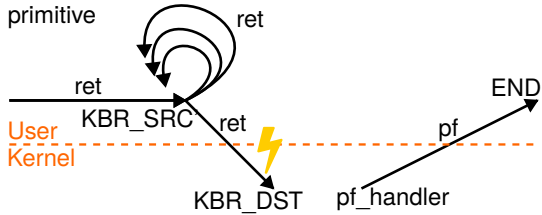
- Verify that spec BTI works across privilege boundaries
- CP-BTI exploits the second required primitive

Speculative CP-BTI

CP-BTI PoC in Detail

- Verify that spec BTI works across privilege boundaries
- CP-BTI exploits the second required primitive

Training Phase:

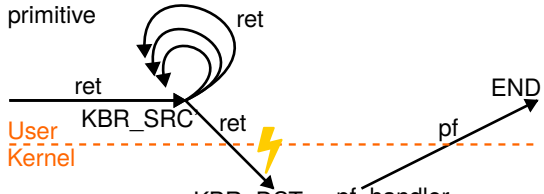


Speculative CP-BTI

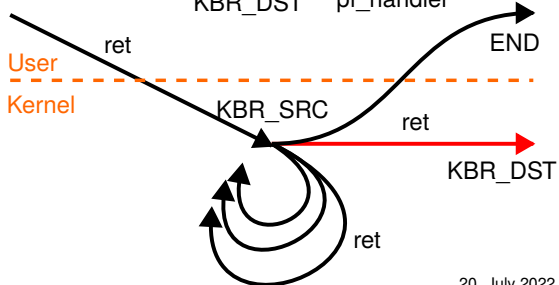
CP-BTI PoC in Detail

- Verify that spec BTI works across privilege boundaries
- CP-BTI exploits the second required primitive

Training Phase:



Speculation Phase:



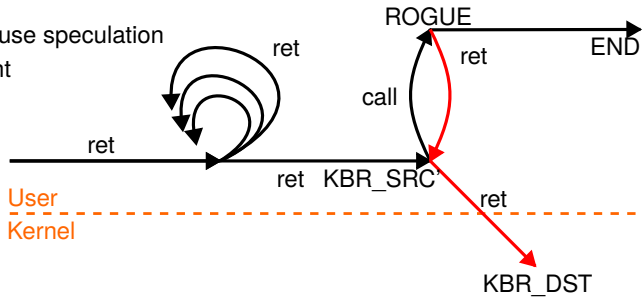
Speculative CP-BTI

- Again, use SpectreRSB to cause speculation
- History needs to be equivalent

Speculative CP-BTI

- Again, use SpectreRSB to cause speculation
- History needs to be equivalent

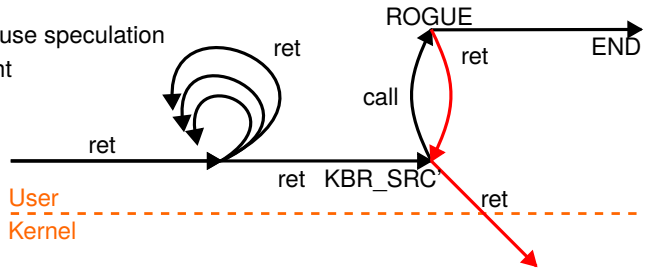
Training Phase:



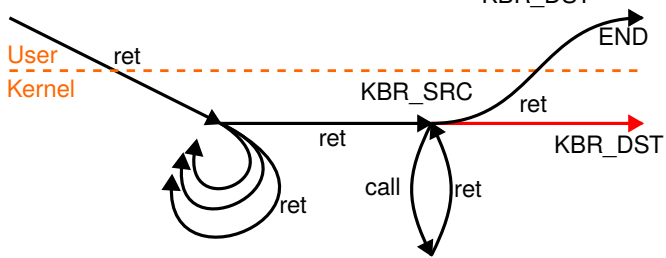
Speculative CP-BTI

- Again, use SpectreRSB to cause speculation
- History needs to be equivalent

Training Phase:



Speculation Phase:



Speculative CP-BTI
Results

- Speculative BTI works across privilege boundaries
  - Shown only for Intel CoffeLake

Summary

⇒ **It is possible to create a version of RETBLEED which does not rely on PFs!**

Outline

Goal

What is the performance overhead of the in-depth mitigation?

In-Depth Mitigation Overview

Microarch.	Single Overhead in %	Multiple Overhead in %
Coffee Lake	26.79	22.09
Zen1	13.65	5.12
Zen1(NoSmt)	12.83	36.71
Zen2	15.49	13.13

Outline

Conclusion

⇒ Spec BTI works in same and cross privilege domain

⇒ PF free RETBLEED is possible

⇒ In-depth mitigations introduce potentially huge overheads

Conclusion

⇒ Spec BTI works in same and cross privilege domain

⇒ PF free RETBLEED is possible

⇒ In-depth mitigations introduce potentially huge overheads

Conclusion

⇒ Spec BTI works in same and cross privilege domain

⇒ PF free RETBLEED is possible

⇒ In-depth mitigations introduce potentially huge overheads

Rogue Function

```
asm(
    ".align 0x80000\n\t"
    "rogue_spec_dst:\n\t"
        "callq rogue_gadg_dst\n\t"
        // Training: execute following code speculatively
        // Misspredict: execute following code architectually
        "jmp *%r9\n\t"
    "rogue_gadg_dst:\n\t"
        // If %rsi = 1: add 8 to rsp => cause speculation
        // If %rsi = 0: do othing
        "lfence\n\t"
        "movq %rsp, %rdx\n\t"
        "addq $0x8, %rdx\n\t"
        "cmp $1, %rsi\n\t"
        "cmoveq %rdx, %rsp\n\t"
        "clflush (%rsp)\n\t"
        "ret\n\t"
    "rogue_spec_dst_end:\n\t"
);
```

Speculative RET-BTI
Results

- Speculative BTI works in same privilege domains
  - For Intel CoffeeLake and AMD Zen1, Zen1+ and Zen2
- Success rate in $\%$

	CoffeeLake	Zen1	Zen2
RET-BTI	56.00	98.85	99.31
Spec RET-BTI	77.40	?	?

- Speculative RET-BTI is less stable
  - Standard Derivation is up to 10 times as large

Speculative CP-BTI
Results

- Speculative BTI works across privilege boundaries
  - Shown only for Intel CoffeLake
- Success rate in $\%$

	CP-BTI	Spec CP-BTI
CoffeeLake	27.16	89.93

- Mean standard derivation of:
  - CP-BTI: 0.13
    - Stable for N
  - Spec CP-BTI: 0.26
    - Decreases for increasing N

In-Depth Mitigation Overview

Microarch.	Single		Multiple	
	Norm. Index Score	Overhead in %	Norm. Index Score	Overhead in %
Coffee Lake	0.78869	26.79	0.81910	22.09
Zen1	0.87993	13.65	0.95128	5.12
Zen1(NoSmt)	0.88631	12.83	0.73145	36.71
Zen2	0.86586	15.49	0.88393	13.13

- Benchmark Suit: Byte-UnixBench
- Geometric Mean of the median of 10 invocations of each workload
- Overhead calculated as $\frac{\text{unpatched}}{\text{patched}} - 1$