CheckMate: Automated Synthesis of Hardware Exploits and Security Litmus Tests

Naorin Hossain nhossain@princeton.edu
Caroline Trippel ctrippel@princeton.edu
Margaret Martonosi mrm@princeton.edu
Princeton University

- Widely implemented microarchitectural features such as speculative execution have been found to be exploitable using well-known side-channel attacks resulting in new exploits such as Spectre and Meltdown
- Our automated approach uses microarchitectural happens-before analysis to determine whether a given microarchitectural design is vulnerable to a given security exploit class
- If a vulnerability is found, our tool outputs synthesized code for a possible attack

2018: The Year of the Hardware Security Exploit

Starting with Meltdown and Spectre, a recent wave of exploits leverage the effects of speculative execution on non-architectural state to make sensitive information available to software via some well-known side-channel attack. What is novel and surprising about these attacks is their ability to create practical working exploits out of a variety of widely-implemented μarch features. We present CheckMate, an approach & automated tool for determining μarch susceptibility to formally specified security exploit classes, and for synthesizing proof-of-concept exploit code when applicable.

Well-known cache side-channel attack: FLUSH+RELOAD

+

Newly-exploited implementation feature: Speculative Cache Pollution

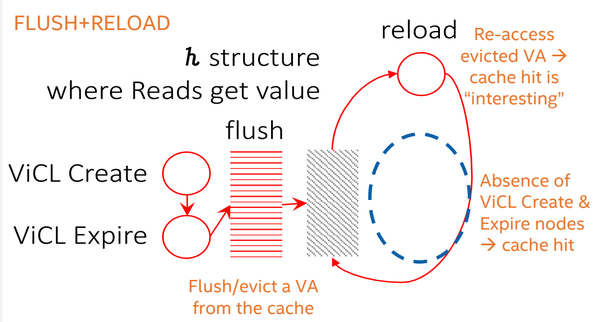
=

New exploits

Augmenting μhb Graphs for Security Verification

Exploit Patterns

- Add μhb patterns to our modeling framework (i.e., μhb sub-graphs)
- Exploit pattern = μarch execution patterns indicative of some exploit class
- Abstract yet still expressive



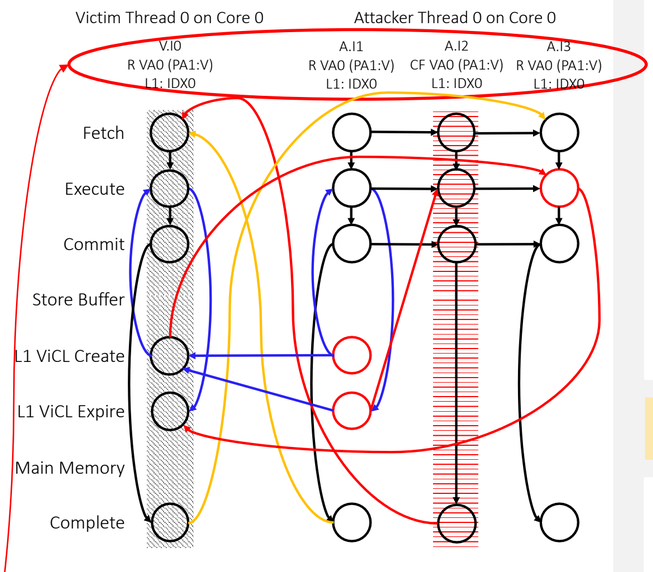
Security Litmus Tests

- Compact program that can induce exploit execution pattern
- Abstracted for efficient analysis; path to full exploit is clear

VA to PA Address Mapping: VA0 (PA1:V) VA to Cache Index Mapping: VA0:IDX0	
Victim T0 on C0	Attacker T0 on C0
(i0) R [VA0] → r1	(i1) R [VA0] → r2
	(i2) CLFLUSH [VA0] ← Flush
	(i3) R [VA0] → r2 ← Reload

Exploit Program Execution

- μhb graph that features an exploit pattern of interest

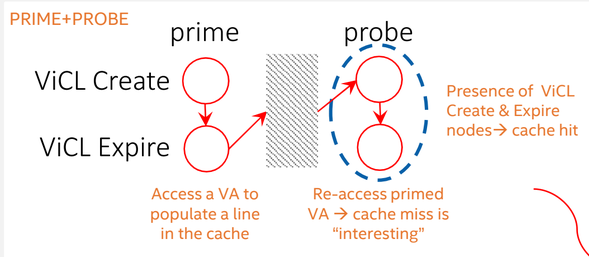


Relational Model Finding (RMF) for Security Litmus Test Synthesis

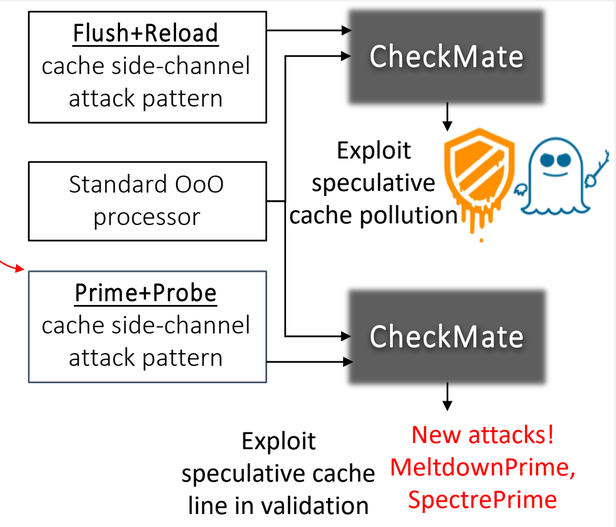
A relational model (e.g., a μhb graph) is a set of constrains on an abstract system of atoms and N-dimensional relations.

- Constraints: cycles + exploit patterns
- Abstract system: μhb graph nodes (atoms) + μhb graph edges (2D relations)

Case Study 1: Evaluating Susceptibility of a Speculative OoO Processor to Cache Side-Channel Attacks



Given this exploit pattern, CheckMate will generate programs representing all of the ways in which the input μarch (+ related OS support) could induce a miss on the probe access.

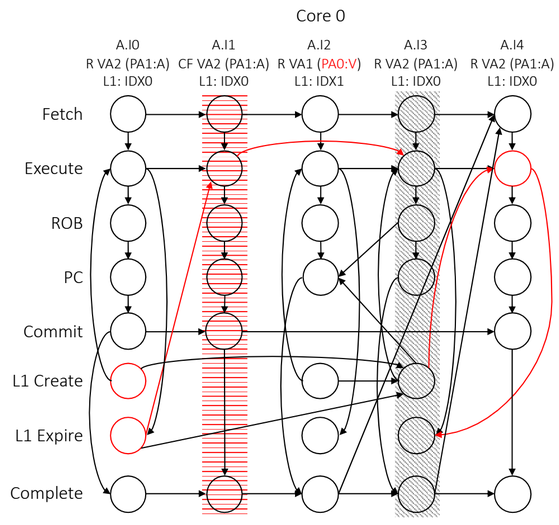


Case Study 2: Evaluating a SandyBridge Processor Model

- Expand CheckMate's capabilities and techniques with a more modern, complex microarchitectural model, resembling that of Intel Sandy Bridge
- SandyBridge model handles both virtual and physical addressing and supports microarchitectural events leading from hardware-OS interactions. Several new features have been incorporated:
  - TLBs
  - System Calls and Interrupts
  - Hardware Page Table Walks
  - Memory Access Status Bits

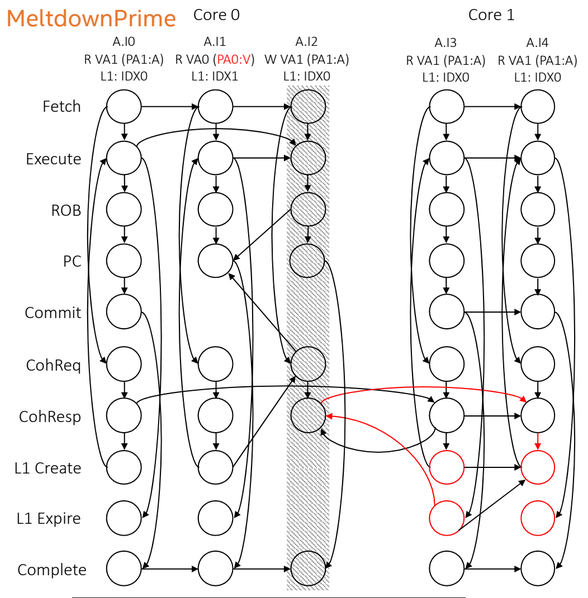
Results: Automatically Generated Exploits

Meltdown (Spectre v3)



VA to PA Mapping: VA2:(PA1:A), VA1:(PA0:V) VA to Cache Index Mapping: VA2:IDX0, VA1:IDX1
Attacker T0 on C0
CLFLUSH [VA2]→0 ← Flush
R [VA1] → r1 (SQUASH)
addr. dependency
R [f(r1)=VA2] → 0 (SQUASH)
R [VA2]→0 ← Reload

MeltdownPrime

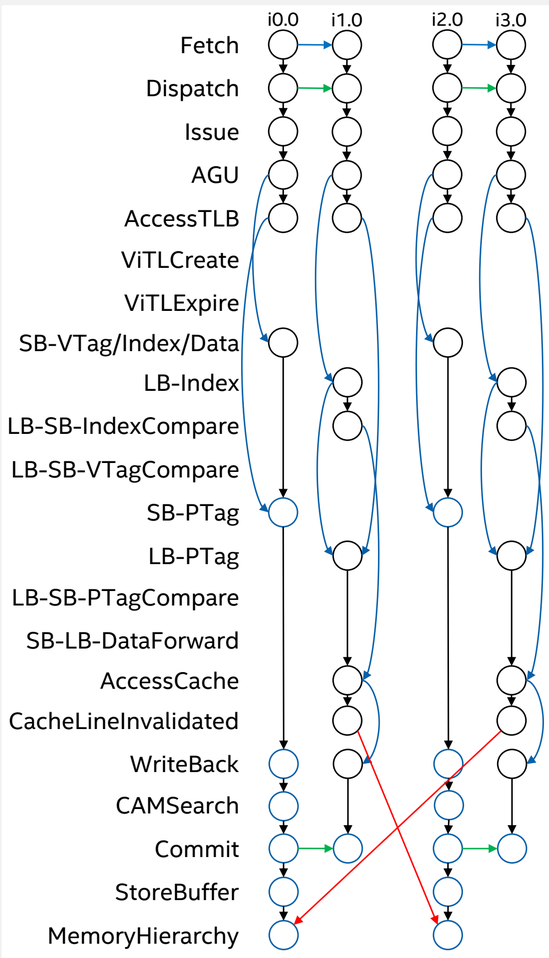


VA to PA Mapping: VA1:(PA1:A), VA0:(PA0:V) VA to Cache Index Mapping: VA1:IDX0, VA0:IDX1	
Attacker T0 on C0	Attacker T1 on C1
R [VA1]→0	R [VA1]→0 ← Prime
R [VA0] → r1 (SQUASH)	
addr. dependency	
W [f(r1)=VA1] → 0 (SQUASH)	
	R [VA1]→0 ← Probe

Testing on Real Hardware

We extended the SpectrePrime security litmus test and demonstrated SpectrePrime on a MacBook with a 2.4 GHz Intel Core i7 Processor running MacOS Sierra, Version 10.12.6.

SandyBridge Program Execution Model



sb Litmus Test

Initially: [x] = 0, [y] = 0, VA x → PA a (R/W, acc, dirty), VA y → PA b (R/W, acc, dirty),	
Core 0	**Core 1**
(i0.0) St [x/a] ← 1	(i2.0) St [y/b] ← 1
(i0.1) Ld PTE(x)	(i2.1) Ld PTE(y)
(i1.0) Ld [y/b] → r1	(i3.0) Ld [x/a] → r2
(i1.1) Ld PTE(y)	(i3.1) Ld PTE(x)
Outcome: Permitted	

Highlighted instructions are "ghost" instructions. In this litmus test, they are page table walks.

Ghost instructions are in the process of being synthesized with our models. We will soon be interleaving them with user-level instructions. They will include page table walks, dirty bit status updates, INVLPG calls, etc.

In order to use CheckMate with this model to find security exploits, our ongoing work includes adding ghost instructions to our security litmus tests.

Ongoing Work

- Extended litmus test generation (for hardware-OS interactions)
- Extend CheckMate's capabilities in the following ways:
  Branch Target Buffer **Security Enclaves (Intel SGX)** **Floating-Point Registers**
- Next steps include extending our methodology with performance modeling capabilities (edges labeled with thermal or timing info)
- Automatically synthesize new exploit patterns

Conclusions

- Demonstrated value of SMT and RMF techniques for hardware-aware analysis
- Interactive runtimes in minutes or hours: dozens of unique exploits in a lunch break
- Early-stage verification: synthesize real-world exploits on abstract representations of hardware; abstract exploit representations can synthesize a variety of exploits

Side Channel Academic Programme (Intel SCAP)