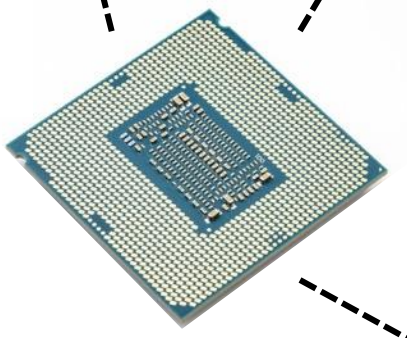
Three-Step Models for Timing Channels in Processor Caches and TLBs

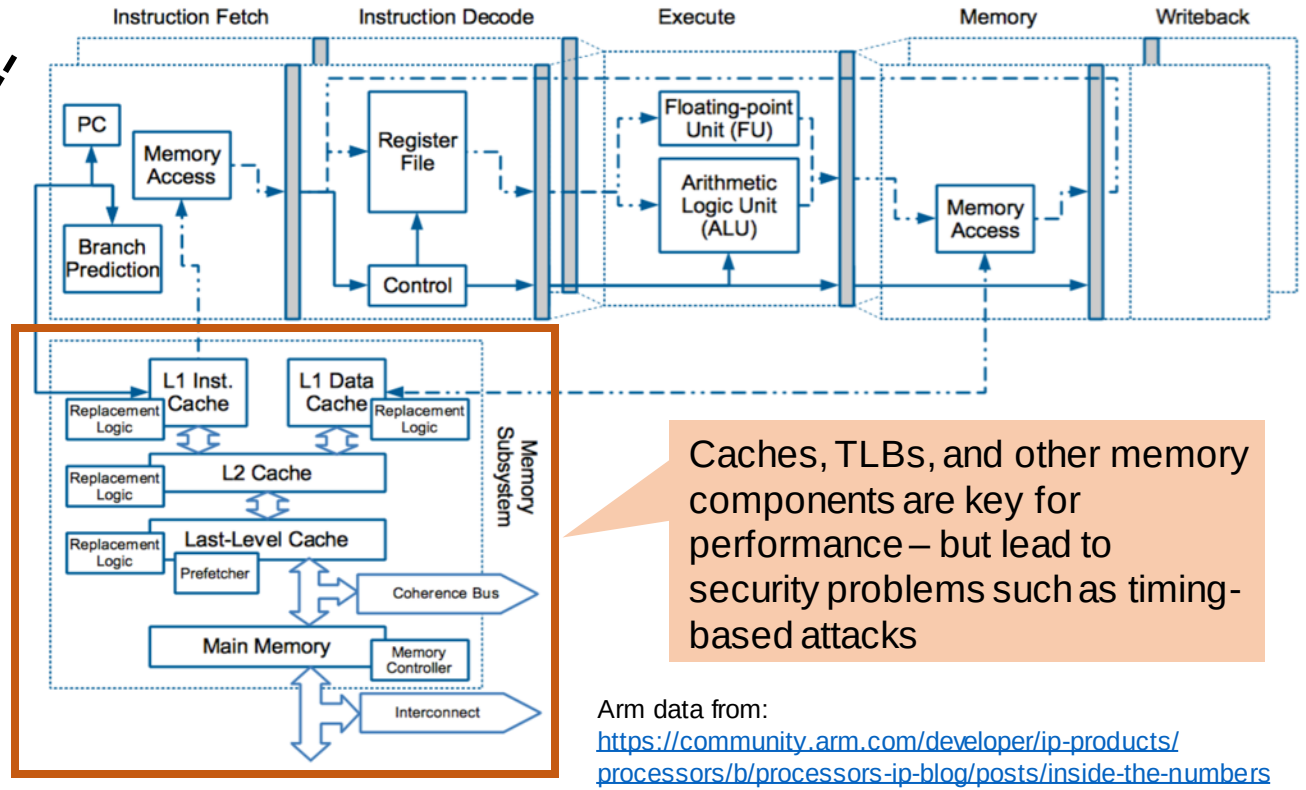
Jakub Szefer
Assistant Professor
Dept. of Electrical Engineering
Yale University

Arm Research Summit 2019 – September 16, 2019

Caches, Security, and Embedded Systems

- Embedded systems are prevalent in today's world
- Processors are key part of any embedded system
- In 2017 there were 100 billion Arm-based chips deployed





Caches, TLBs, and other memory components are key for performance – but lead to security problems such as timing-based attacks

Arm data from:
https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/inside-the-numbers-100-billion-arm-based-chips-1345571105

Recent Uses of Cache Timing Attacks

- There is renewed interest in timing attacks due to Transient Execution Attacks
- Most of them use **transient executions and** leverage **cache timing attacks**
- Variants using cache timing attacks (side or covert channels):

Variant 1:	Bounds Check Bypass (BCB)	Spectre
Variant 1.1:	Bounds Check Bypass Store (BCBS)	Spectre-NG
Variant 1.2:	Read-only protection bypass (RPB)	Spectre
Variant 2:	Branch Target Injection (BTI)	Spectre
Variant 3:	Rogue Data Cache Load (RDCL)	Meltdown
Variant 3a:	Rogue System Register Read (RSRR)	Spectre-NG
Variant 4:	Speculative Store Bypass (SSB)	Spectre-NG
(none)	LazyFP State Restore	Spectre-NG 3
Variant 5:	Return Mispredict	SpectreRSB

- NetSpectre, Foreshadow, SGXSpectre, or SGXPectre
- SpectrePrime and MeltdownPrime (both use Prime+Probe instead of original Flush+Reload cache attack)
- And more…

Classical Channels – which do not require speculative execution

Speculative Channels – which are based on speculative execution

Root cause of the both types remains the same

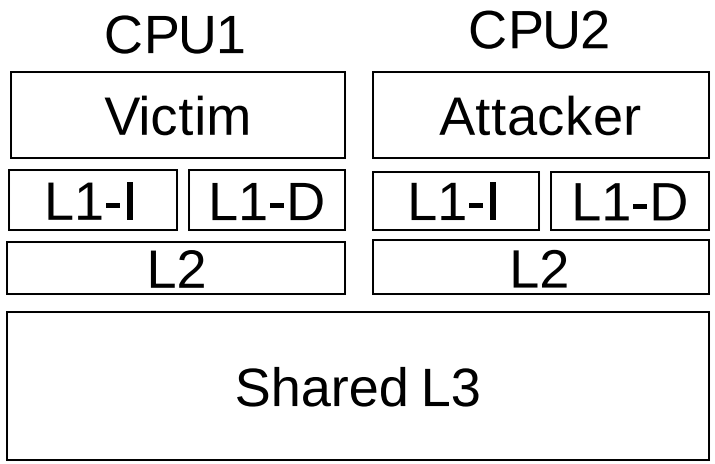
- Defending classical attacks defends speculative attacks as well, but not the other way around

Three-Step Model for Cache Timing Attacks

Timing Attack Example: Prime-Probe Attacks

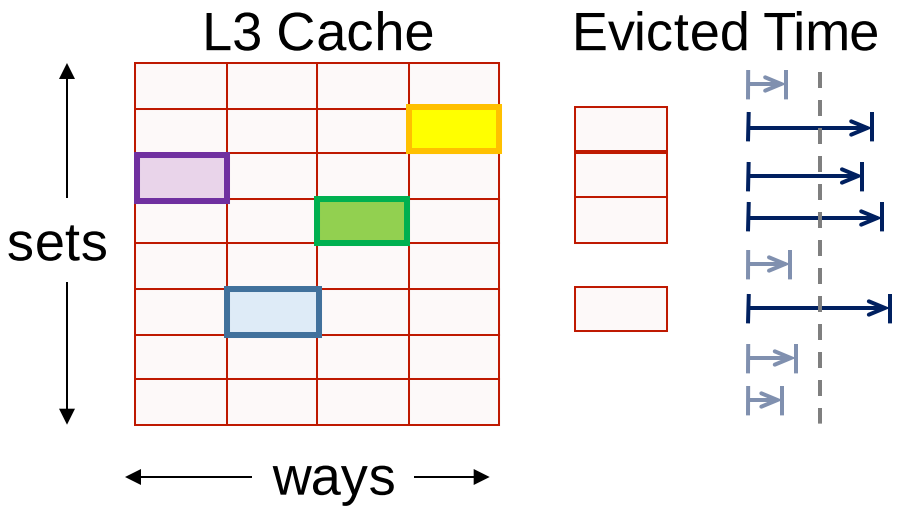
Osvik, D. A., Shamir, A., & Tromer, E, "Cache attacks and countermeasures: the case of AES". 2006.

2- Victim accesses critical data



1- Attacker primes each cache set

3- Attacker probes each cache set (measure time)

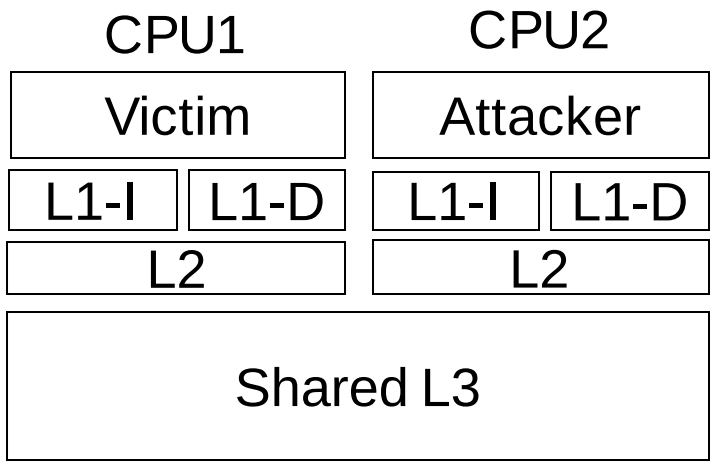


Data sharing is not needed

Timing Attack Example: Flush-Reload Attack

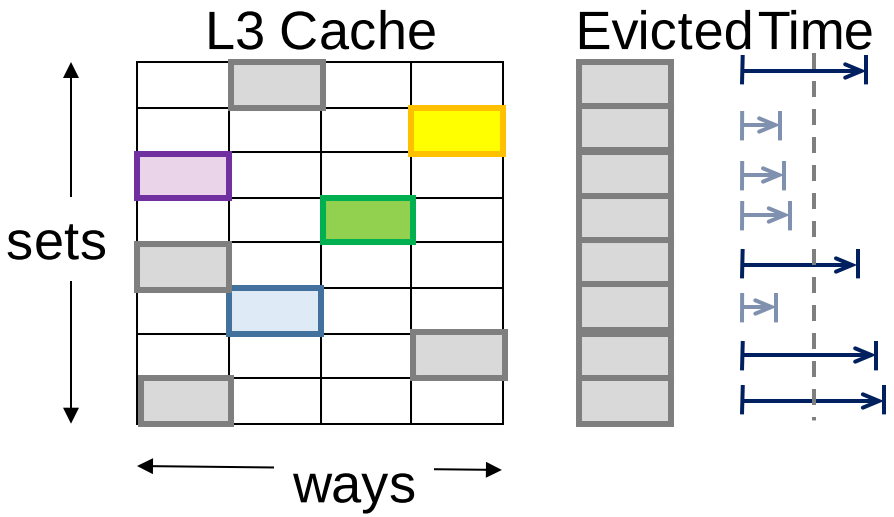
Yarom, Y., & Falkner, K. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack", 2014.

2- Victim accesses critical data



1- Attacker flushes each line in the cache

3- Attacker reloads critical data by running specific process (measure time)



Data sharing is needed

A Three-Step Model for Cache Timing Attack Modeling

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Observation:

- All the existing cache timing attacks equivalent to three memory operations → three-step model
- Cache replacement policy the same to each cache block → focus on one cache block

The Three-Step Single-Cache-Block-Access Model

$Step1$	⇝	$Step2$	⇝	$Step3$ (fast/slow)
The initial state of the cache block set by a memory operation		Memory operation alters the state of the cache		Final memory operations and timing observation (fast/slow)

- Analyzed possible states of the cache block + used cache three-step simulator and reduction rules derive all the effective vulnerabilities

- **There are 72 possible cache timing attack types**

17 States in the Model

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

There are 17 possible states for each step in the model:

V_u, A_a, V_a, A_a^{alisa}, V_a^{alias}, A_d, V_d, A^{inv}, V^{inv}, A_a^{inv}, V_a^{inv}, $A_a^{aliasinv}$, $V_a^{aliasinv}$, A_d^{inv}, V_d^{inv}, V_u^{inv}, $*$

- *V* represents that the state is a result of the victim's operation, while *A* represents that the state is a result of the attacker's operation
- *x* denotes the set of virtual memory addresses storing addresses of sensitive data
- *u* denotes the victim's secret address within *x* which is unknown to the attacker
- *a*, a^{alias} and *d* denote known memory addresses that map to the same cache line
- *d* refers to an address outside of *x*, while the others are the address within *x*.

The attacker's goal is to obtain u

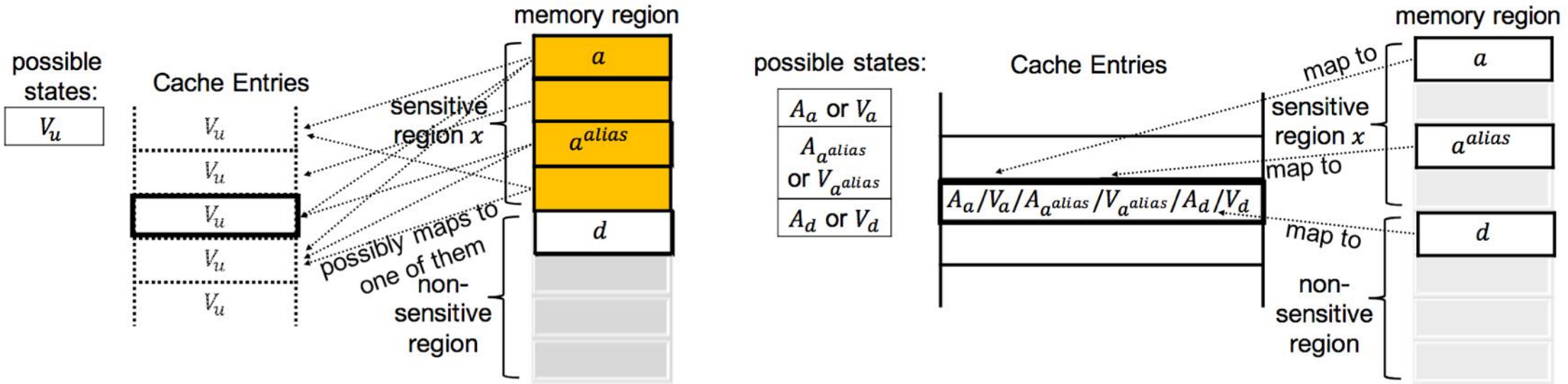
State	Description
V_u	A memory location u belonging to the victim is accessed and is placed in the cache block by the victim (V). Attacker does not know u, but u is from a set x of memory locations, a set which is known to the attacker. It may have the same index as a or a^{alias}, and thus conflict with them in the cache block. The goal of the attacker is to learn the index of the address u. The attacker does not know the address u, hence there is no A_u in the model.
A_a or V_a	The cache block contains a specific memory location a. The memory location is placed in the cache block due to a memory access by the attacker, A_a, or the victim, V_a. The attacker knows the address a, independent of whether the access was by the victim or the attacker themselves. The address a is within the range of sensitive locations x. The address a is known to the attacker.
$A_{a^{alias}}$ or $V_{a^{alias}}$	The cache block contains a memory address a^{alias}. The memory location is placed in the cache block due to a memory access by the attacker, $A_{a^{alias}}$, or the victim, $V_{a^{alias}}$. The address a^{alias} is within the range x and not the same as a, but it has the same address index and maps to the same cache block, i.e. it "aliases" to the same block. The address a^{alias} is known to the attacker.
A_d or V_d	The cache block contains a memory address d. The memory address is placed in the cache block due to a memory access by the attacker, A_d, or the victim, V_d. The address d is not within the range x. The address d is known to the attacker.
A^{inv} or V^{inv}	The cache block is now invalid. The data and its address are "removed" from the cache block by the attacker, A^{inv}, or the victim, V^{inv}, as a result of cache block being invalidated, e.g., this is a cache flush of the whole cache.
A_a^{inv} or V_a^{inv}	The cache block state can be anything except a in this cache block now. The data and its address are "removed" from the cache block by the attacker, A_a^{inv}, or the victim, V_a^{inv}. E.g., by using a flush instruction such as $clflush$ that can flush specific address, or by causing certain cache coherence protocol events that force a to be removed from the cache block. The address a is known to the attacker.
$A_{a^{alias}}^{inv}$ or $V_{a^{alias}}^{inv}$	The cache block state can be anything except a^{alias} in this cache block now. The data and its address are "removed" from the cache block by the attacker, $A_{a^{alias}}^{inv}$, or the victim, $V_{a^{alias}}^{inv}$. E.g., by using a flush instruction such as $clflush$ that can flush specific address, or by causing certain cache coherence protocol events that force a^{alias} to be removed from the cache block. The address a^{alias} is known to the attacker.
A_d^{inv} or V_d^{inv}	The cache block state can be anything except d in this cache block now. The data and its address are "removed" from the cache block by the attacker A_d^{inv} or the victim V_d^{inv}. E.g., by using a flush instruction such as $clflush$ that can flush specific address, or by causing certain cache coherence protocol events that force d to be removed from the cache block. The address d is known to the attacker.
V_u^{inv}	The cache block state can be anything except u in the cache block. The data and its address are "removed" from the cache block by the victim V_u^{inv} as a result of cache block being invalidated, e.g., by using a flush instruction such as $clflush$, or by certain cache coherence protocol events that force u to be removed from the cache block. The attacker does not know u. Therefore, the attacker is not able to trigger this invalidation and A_u^{inv} does not exist in the model.
$\star$	Any data, or no data, can be in the cache block. The attacker has no knowledge of the memory address in this cache block.

States of the Cache Block

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

There are 17 possible states for each step in the model:

V_u, A_a, V_a, A_a^{alisa}, V_a^{alias}, A_d, V_d, …

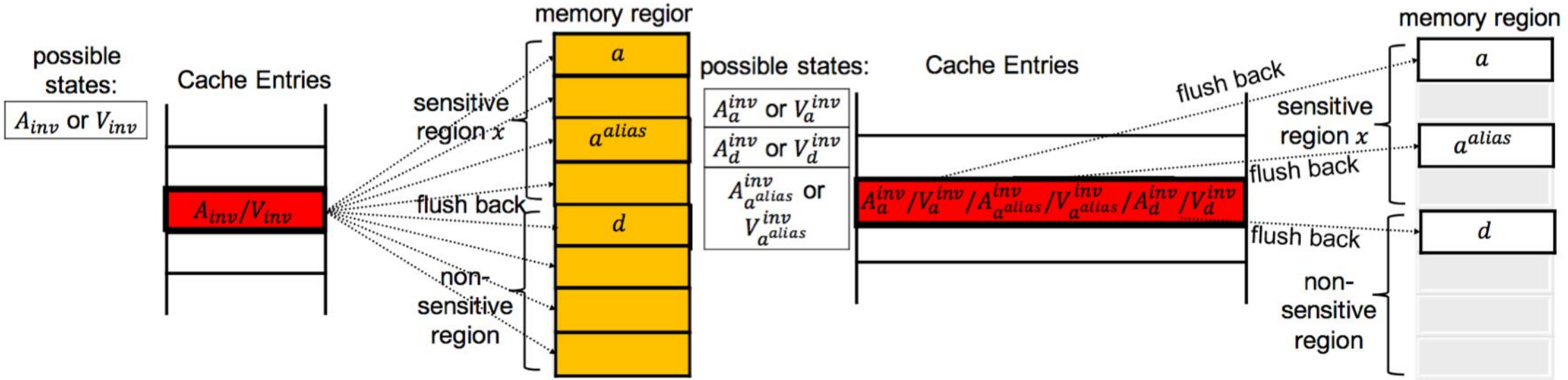


States of the Cache Block

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

There are 17 possible states for each step in the model:

V_u, A_a, V_a, A_a^{alisa}, V_a^{alias}, A_d, V_d, A^{inv}, V^{inv}, A_a^{inv}, V_a^{inv}, $A_a^{aliasinv}$, $V_a^{aliasinv}$, A_d^{inv}, V_d^{inv}, …

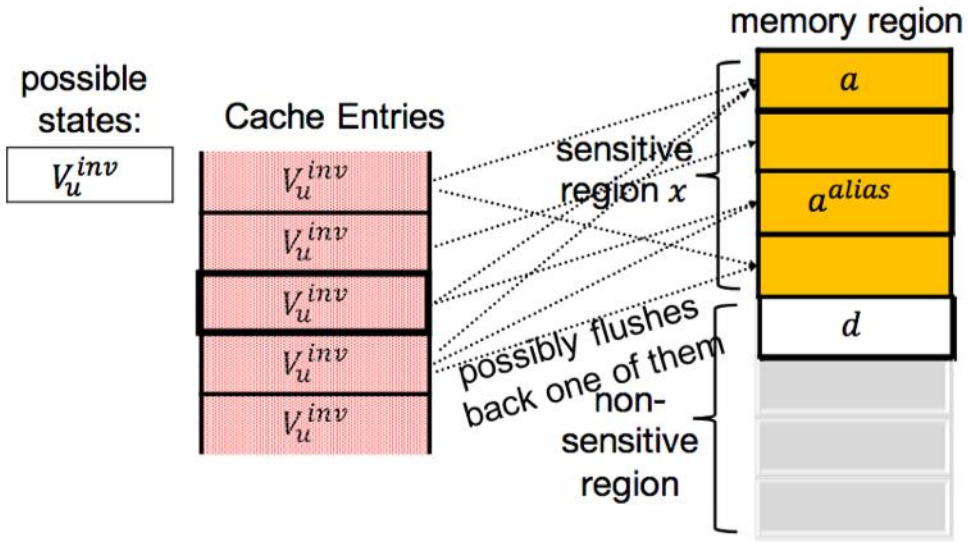


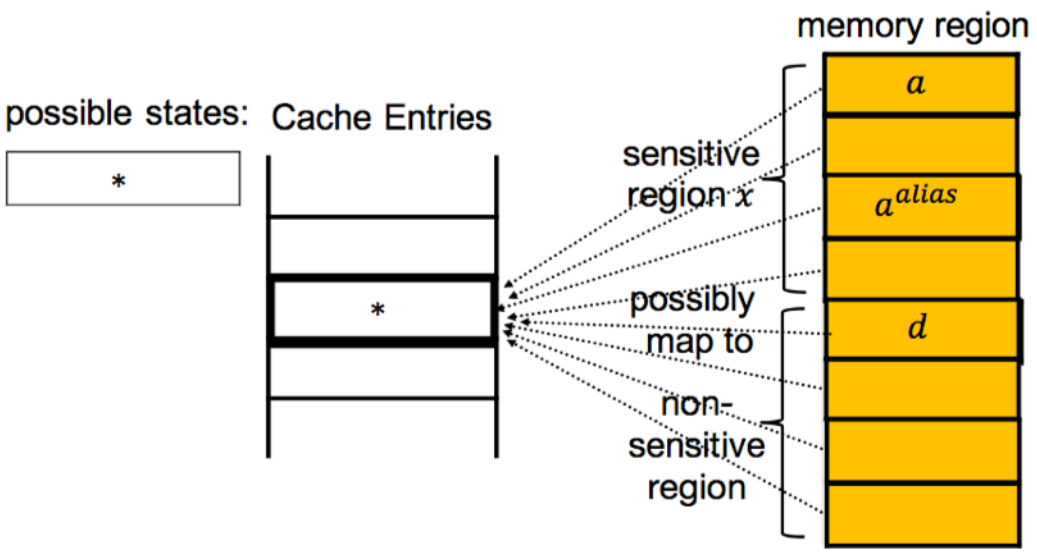
States of the Cache Block

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

There are 17 possible states for each step in the model:

V_u, A_a, V_a, A_a^{alisa}, V_a^{alias}, A_d, V_d, A^{inv}, V^{inv}, A_a^{inv}, V_a^{inv}, $A_a^{aliasinv}$, $V_a^{aliasinv}$, A_d^{inv}, V_d^{inv}, V_u^{inv}, *

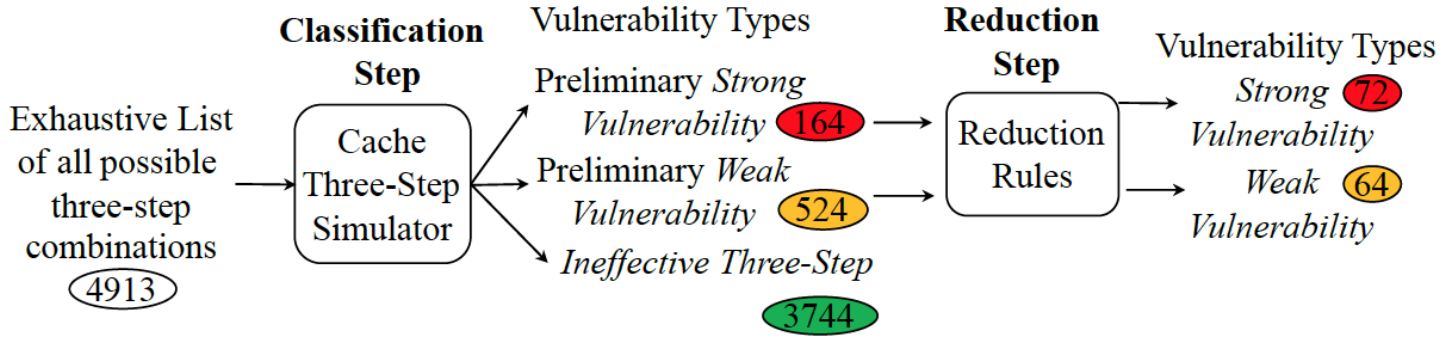


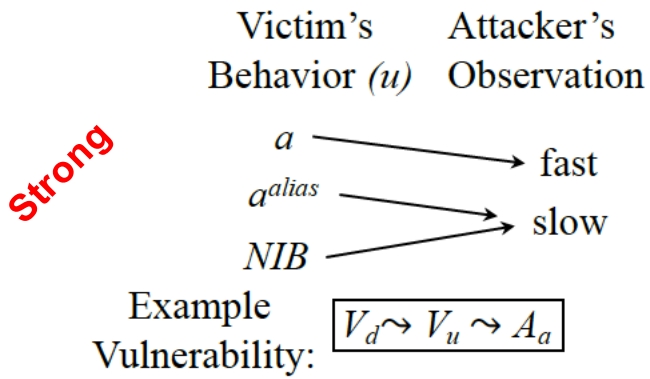


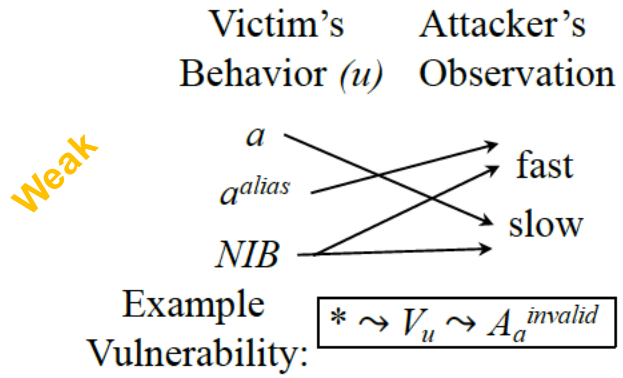
Strong and Weak Vulnerabilities

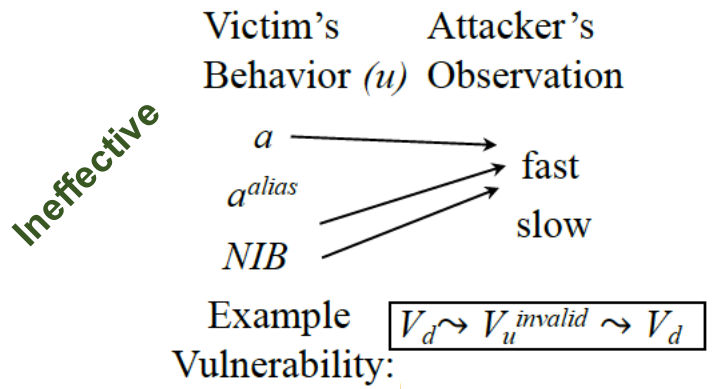
"Analysis of Secure Caches and Timing-Based Side-Channel Attacks", S. Deng, et al., 2019
"Cache Timing Side-Channel Vulnerability Checking with Computation Tree Logic", S. Deng, et al., 2018

- Exhaustively evaluate all 17 (step1) * 17 (step2) * 17 (step3) = 4913 three-step patterns
- Used cache three-step simulator and reduction rules to find all the strong effective vulnerabilities
- In total 72 strong effective vulnerabilities were derived and presented









Exhaustive List of Cache Timing Side- Channel Attacks

Deng, S., Xiong, W., Szefer, J., "Analysis of Secure Caches and Timing-Based Side-Channel Attacks", 2019

Attack Strategy	Vulnerability Type			Macro Type	Attack
	Step 1	*Step 2*	*Step 3*		
Cache Internal Collision	A^{inv}	V_u	V_a (fast)	IH	(2)
	V^{inv}	V_u	V_a (fast)	IH	(2)
	A_d	V_u	V_a (fast)	IH	(2)
	V_d	V_u	V_a (fast)	IH	(2)
	$A_{a^{alias}}$	V_u	V_a (fast)	IH	(2)
	$V_{a^{alias}}$	V_u	V_a (fast)	IH	(2)
	A_a^{inv}	V_u	V_a (fast)	IH	(2)
	V_a^{inv}	V_u	V_a (fast)	IH	(2)
Flush + Reload	A_a^{inv}	V_u	A_a (fast)	EH	(5)
	V_a^{inv}	V_u	A_a (fast)	EH	(5)
	A^{inv}	V_u	A_a (fast)	EH	(5)
	V^{inv}	V_u	A_a (fast)	EH	(5)
	A_d	V_u	A_a (fast)	EH	(5)
	V_d	V_u	A_a (fast)	EH	(5)
	$A_{a^{alias}}$	V_u	A_a (fast)	EH	(5)
	$V_{a^{alias}}$	V_u	A_a (fast)	EH	(5)
Reload + Time	V_u^{inv}	A_a	V_u (fast)	EH	new
	V_u^{inv}	V_a	V_u (fast)	IH	new
Flush + Probe	A_a	V_u^{inv}	A_a (slow)	EM	(6)
	A_a	V_u^{inv}	V_a (slow)	IM	new
	V_a	V_u^{inv}	A_a (slow)	EM	new
	V_a	V_u^{inv}	V_a (slow)	IM	new
Evict + Time	V_u	A_d	V_u (slow)	EM	(1)
	V_u	A_a	V_u (slow)	EM	(1)
Prime + Probe	A_d	V_u	A_d (slow)	EM	(4)
	A_a	V_u	A_a (slow)	EM	(4)
Bernstein's Attack	V_u	V_a	V_u (slow)	IM	(3)
	V_u	V_d	V_u (slow)	IM	(3)
	V_d	V_u	V_d (slow)	IM	(3)
	V_a	V_u	V_a (slow)	IM	(3)
Evict + Probe	V_d	V_u	A_d (slow)	EM	new
	V_a	V_u	A_a (slow)	EM	new
Prime + Time	A_d	V_u	V_d (slow)	IM	new
	A_a	V_u	V_a (slow)	IM	new
Flush + Time	V_u	A_a^{inv}	V_u (slow)	EM	new
	V_u	V_a^{inv}	V_u (slow)	IM	new

(1) Evict + Time attack [31].
(2) Cache Internal Collision attack [4].
(3) Bernstein's attack [2].
(4) Prime + Probe attack [31,33], Alias-driven attack [16].
(5) Flush + Reload attack [50,49], Evict + Reload attack [15].
(6) SpectrePrime, MeltdownPrime attack [41].

Attack Strategy	Vulnerability Type			Macro Type	Attack
	Step 1	*Step 2*	*Step 3*		
Cache Internal Collision Invalidation	A^{inv}	V_u	V_a^{inv} (slow)	IH	new
	V^{inv}	V_u	V_a^{inv} (slow)	IH	new
	A_d	V_u	V_a^{inv} (slow)	IH	new
	V_d	V_u	V_a^{inv} (slow)	IH	new
	$A_{a^{alias}}$	V_u	V_a^{inv} (slow)	IH	new
	$V_{a^{alias}}$	V_u	V_a^{inv} (slow)	IH	new
Flush + Flush	A_a^{inv}	V_u	V_a^{inv} (slow)	IH	(1)
	V_a^{inv}	V_u	V_a^{inv} (slow)	IH	(1)
	A_a^{inv}	V_u	A_a^{inv} (slow)	EH	(1)
	V_a^{inv}	V_u	A_a^{inv} (slow)	EH	(1)
Flush + Reload Invalidation	A^{inv}	V_u	A_a^{inv} (slow)	EH	new
	V^{inv}	V_u	A_a^{inv} (slow)	EH	new
	A_d	V_u	A_a^{inv} (slow)	EH	new
	V_d	V_u	A_a^{inv} (slow)	EH	new
	$A_{a^{alias}}$	V_u	A_a^{inv} (slow)	EH	new
	$V_{a^{alias}}$	V_u	A_a^{inv} (slow)	EH	new
Reload + Time Invalidation	V_u^{inv}	A_a	V_u^{inv} (slow)	EH	new
	V_u^{inv}	V_a	V_u^{inv} (slow)	IH	new
Flush + Probe Invalidation	A_a	V_u^{inv}	A_a^{inv} (fast)	EM	new
	A_a	V_u^{inv}	V_a^{inv} (fast)	IM	new
	V_a	V_u^{inv}	A_a^{inv} (fast)	EM	new
	V_a	V_u^{inv}	V_a^{inv} (fast)	IM	new
Evict + Time Invalidation	V_u	A_d	V_u^{inv} (fast)	EM	new
	V_u	A_a	V_u^{inv} (fast)	EM	new
Prime + Probe Invalidation	A_d	V_u	A_d^{inv} (fast)	EM	new
	A_a	V_u	A_a^{inv} (fast)	EM	new
Bernstein's Invalidation Attack	V_u	V_a	V_u^{inv} (fast)	IM	new
	V_u	V_d	V_u^{inv} (fast)	IM	new
	V_d	V_u	V_d^{inv} (fast)	IM	new
	V_a	V_u	V_a^{inv} (fast)	IM	new
Evict + Probe Invalidation	V_d	V_u	A_d^{inv} (fast)	EM	new
	V_a	V_u	A_a^{inv} (fast)	EM	new
Prime + Time Invalidation	A_d	V_u	V_d^{inv} (fast)	IM	new
	A_a	V_u	V_a^{inv} (fast)	IM	new
Flush + Time Invalidation	V_u	A_a^{inv}	V_u^{inv} (fast)	EM	new
	V_u	V_a^{inv}	V_u^{inv} (fast)	IM	new

(1) Flush + Flush attack [14].

New attacks not considered prior to our work

Three-Step Model for TLB Timing Attacks

A Three-Step Model for TLB Timing Attack Modeling

Deng, S., Xiong, W., Szefer, J., "Secure TLBs", ISCA 2019

Observation:

- All the existing TLB timing attacks can be modeled using three memory operations
- TLB replacement policy applies to each TLB block equally: can model only one block

The Three-Step TLB Model

$Step1$	⇝	$Step2$	⇝	$Step3$ (fast/slow)
The initial state of the cache block set by a memory operation		Memory operation alters the state of the cache		Final memory operations and timing observation (fast/slow)

- Analyzed possible states of the TLB block + used TLB three-step simulator and reduction rules derive all the effective vulnerabilities

- **There are 43 possible cache timing attack types**

TLB Timing Attacks

Our model considers 10 states for each step

A_a **or** V_a
$A_{a^{alias}}$ **or** $V_{a^{alias}}$
A_d **or** V_d
A_{inv} **or** V_{inv}
V_u
$*$

Total 10^3 = 1000 combinations

- Apply three-step simulator and reduction rules to eliminate non-vulnerabilities

We consequently found **24** vulnerabilities

- **16** new vulnerabilities
- **8** vulnerabilities map to existing attacks

Attack Strategy	Vulnerability Type			Macro Type	Attack
	$Step\ 1$	$Step\ 2$	$Step\ 3$		
TLB Prime + Probe	A_d	V_u	A_d (slow)	EM	(1)
	A_a	V_u	A_a (slow)	EM	(1)
TLB Internal Collision	A_{inv}	V_u	V_a (fast)	IH	(2)
	V_{inv}	V_u	V_a (fast)	IH	(2)
	A_d	V_u	V_a (fast)	IH	(2)
	V_d	V_u	V_a (fast)	IH	(2)
	$A_{a^{alias}}$	V_u	V_a (fast)	IH	(2)
	$V_{a^{alias}}$	V_u	V_a (fast)	IH	(2)
TLB Evict + Time	V_u	A_d	V_u (slow)	EM	**new**
	V_u	A_a	V_u (slow)	EM	**new**
TLB Evict + Probe	V_d	V_u	A_d (slow)	EM	**new**
	V_a	V_u	A_a (slow)	EM	**new**
TLB Prime + Time	A_d	V_u	V_d (slow)	IM	**new**
	A_a	V_u	V_a (slow)	IM	**new**
TLB Flush + Reload	A_{inv}	V_u	A_a (fast)	EH	**new**
	V_{inv}	V_u	A_a (fast)	EH	**new**
	A_d	V_u	A_a (fast)	EH	**new**
	V_d	V_u	A_a (fast)	EH	**new**
	$A_{a^{alias}}$	V_u	A_a (fast)	EH	**new**
	$V_{a^{alias}}$	V_u	A_a (fast)	EH	**new**
TLB version of Bernstein's Attack	V_u	V_a	V_u (slow)	IM	**new**
	V_u	V_d	V_u (slow)	IM	**new**
	V_d	V_u	V_d (slow)	IM	**new**
	V_a	V_u	V_a (slow)	IM	**new**

New attacks not considered prior to our work

(1) TLBBleed. Ben Gras et al. "Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks" USENIX Security Symposium, 2018.

(2) Double Page Fault Attack. Ralf Hund et al. "Practical timing side channel attacks against kernel space ASLR" S&P, 2013.

Towards Security Benchmarks & Summary

Towards Security Benchmarks

Deng, S., Xiong, W., Szefer, J., "A Benchmark Suite for Evaluating Caches' Vulnerability to Timing Attacks", Technical Report

- The theoretical three-step model can be used to design security benchmarks for caches (and TLBs and other cache-like structures) and create security benchmark suite

- Levels of complexity
  1. Theoretical attack analysis
  2. Benchmarks and test code
  3. Actual security attacks

The three-step model

Benchmark derived from three-step model

Overly complex for purpose of showing vulnerability

Towards design of benchmarks

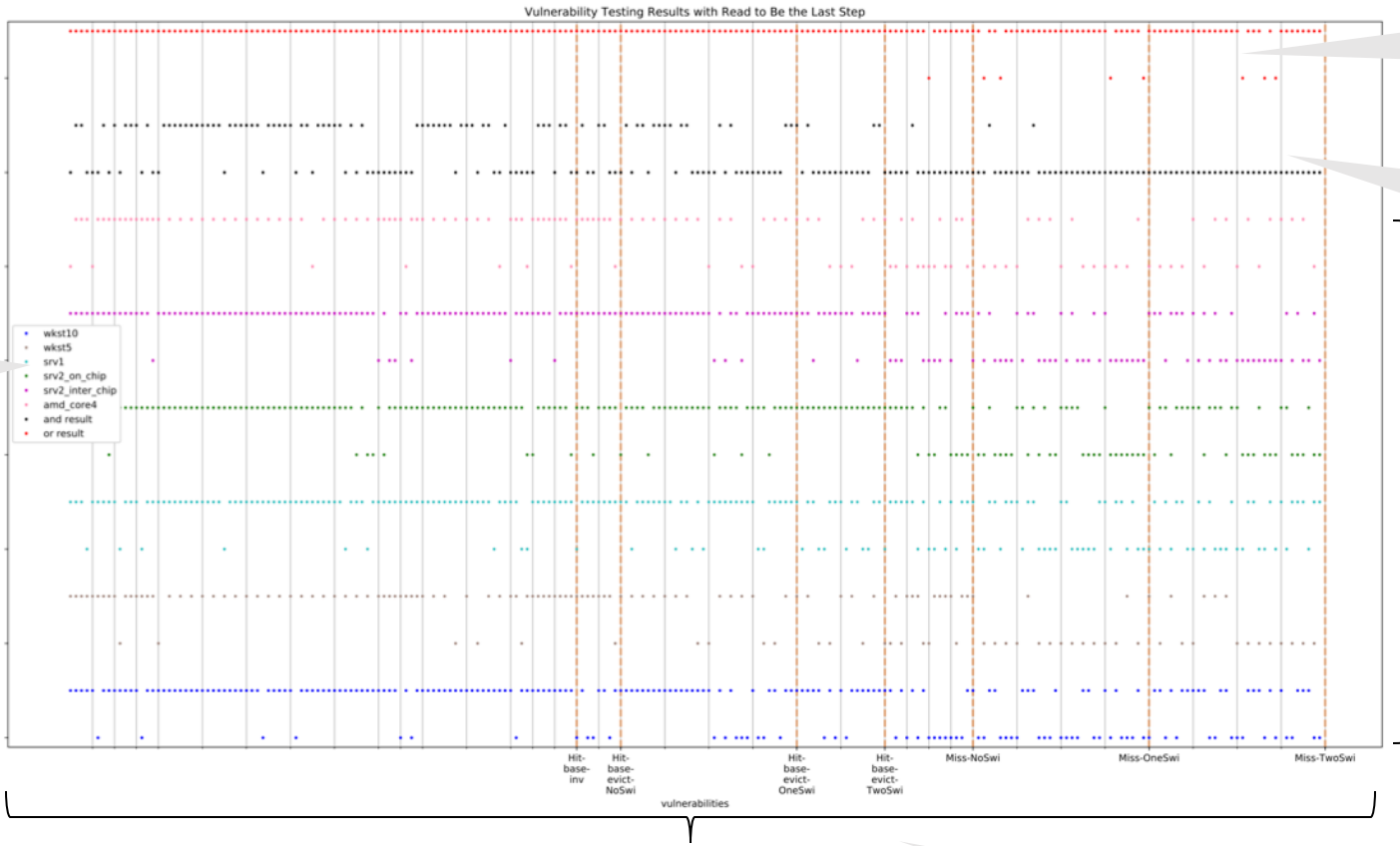
- Convert three-step patterns into code snippets
- Run code on actual hardware or simulator
- Derive timing of the last step
- Use Welch's t-test to check if distribution of timing for different values of *u* can be distinguished

- The number of three-steps that have distinguishable timing can be a security score
  - Smaller is better, means more secure cache

Preliminary Benchmarks and Results

Deng, S., Xiong, W., Szefer, J., "A Benchmark Suite for Evaluating Caches' Vulnerability to Timing Attacks", Technical Report

- On-going research in our group looks into development of open-source benchmarks that can generate the CTVS



Different CPUs tested

Tests that pass for one or more CPUs

Tests that pass for all CPUs

Tests for individual CPUs

Three-step tests: 72 * 8 or 16 variants

Summary

- Timing attacks have a long history, but the research on attacks and defenses is still a very active field
  - There is renewed interest (especially in cache) timing attacks due to Transient Execution Attacks
- Showed three-step model for timing attacks on caches
  - Total of 72 types of possible attacks, with 43 attacks not previously considered
- Showed a three-step model for timing attacks on TLBs
  - Total of 24 types of possible attacks, with 16 attacks not previously considered
- Discussed preliminary results for security benchmarks
  - Allow users and designers to tests caches
  - Can help compare different processor and cache configurations with a simple metric

References:

- Shuwen Deng, Wenjie Xiong, and Jakub Szefer, "**Secure TLBs**", in Proceedings of the International Symposium on Computer Architecture (ISCA), June 2019.
- Shuwen Deng, Wenjie Xiong, and Jakub Szefer, "**Analysis of Secure Caches using a Three-Step Model for Timing-Based Attacks**", accepted for publication, HASS Journal, 2019.

Thank You!

Related reading…

*Jakub Szefer, "**Principles of Secure Processor Architecture Design**," in Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers*, October 2018.

https://caslab.csl.yale.edu/books/