Processor Architecture Security
Part 2: Side and Covert Channels

Jakub Szefer
Assistant Professor
Dept. of Electrical Engineering
Yale University

(These slides include some prior slides by Jakub Szefer, Wenjie Xiong, and Shuwen Deng from HOST 2019 Tutorial)

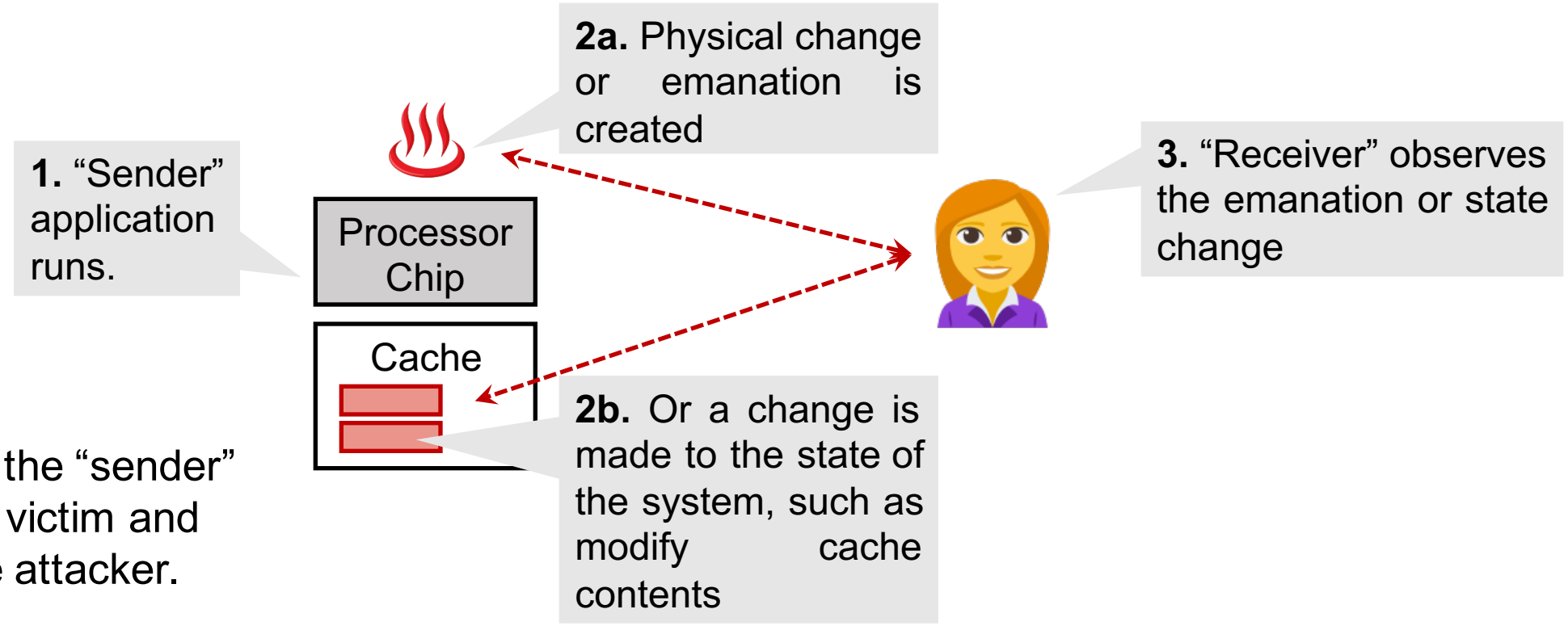
ACACES 2019 – July 14th - 20th, 2019

Slides and information available at: https://caslab.csl.yale.edu/tutorials/acaces2019/

© Jakub Szefer 2019

Side and Covert Channels in Processors

A **covert channel** is an intentional communication between a sender and a receiver via a medium not designed to be a communication channel.



In a **side channel**, the "sender" in an unsuspecting victim and the "receiver" is the attacker.

Emoji Image:
https://www.emojione.com/emoji/2668
https://www.emojione.com/emoji/1f469-1f4bc

© Jakub Szefer 2019

Side and Covert Channels

Covert Channel – a communication channel that was not intended or designed to transfer information, typically leverage unusual methods for communication of information

Side Channel – is similar to a covert channel, but the sender does not intend to communicate information to the receiver, rather sending (i.e. leaking) of information is a side effect of the implementation and the way the computer hardware or software is used**.**

- Covert channel is easier to establish, a precursor to side-channel attack
- Differentiate side channel from covert channel depending on who controls the “sender”

Means for transmitting information:

- **Timing**
- **Power**
- **Thermal emanations**
- **Electro-magnetic (EM) emanations**
- **Acoustic emanations**

© Jakub Szefer 2019

Goals of Side and Covert Channels

Goal of side or covert channels is to break the logical protections of the computer system and leak confidential or sensitive information.

- Typically attacks on confidentiality (leak data from secure to insecure)
  - All attacks fall in this category, they establish a channel to exfiltrate information
- Could be used in "reverse" to attack integrity (insecure data leaks to, and affects secure data)
  - Power, thermal, or EM fault attacks can also fall in this category
- Beyond leaking data:
  - Leak control flow or execution patterns
  - Leak memory access patterns
  - Leak hardware usage patterns
- **For timing channels, goal is to break the logical isolation of the memory protection system, e.g. leak information between two processes**

© Jakub Szefer 2019

Channels: Victim-to-Attacker and Attacker-to-Victim

Typically a channel is **from an unsuspecting victim to an attacker**:

- Goal is to extract some information from victim
- Victim does not observe any execution behavior change

Victim's operation sends information to attacker

Attacker obtains information via the side channel

A channel can also exist **from attacker to victim**:

- Attacker's behavior can "send" some information to the victim
- The information, in form of processor state for example, affects how the victim behaves unbeknownst to them

E.g. modulate branch predictor state to affect execution of the victim

Attacker modulates some information that is sent to victim

Victim's operation depends on the information sent from attacker

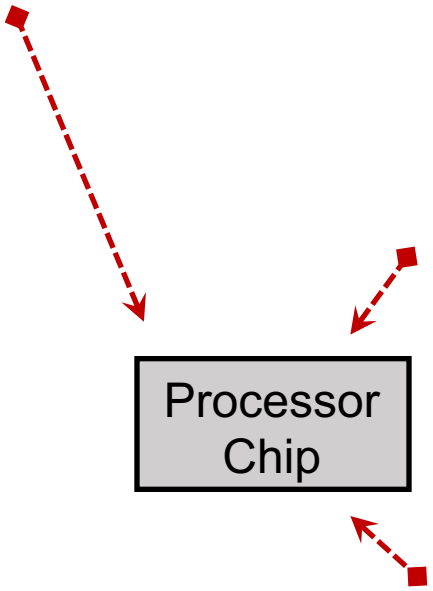
Emoji Image:
https://www.emojione.com/emoji/1f468-1f4bc
https://www.emojione.com/emoji/1f469-1f4bc

© Jakub Szefer 2019

Side and Covert Channels and Physical Proximity

Distance: infinity
(assuming network connection)

Timing channels don't require measurement equipment, only attacker can run code on victim (not even always needed, c.f. AVX-based channel used in NetSpectre) and have network connection.



Distance: small
(0m or Physical Connection)

Power channels require physical connection to measure the power consumed by the CPU (assuming there is no on-chip sensor that can be abused to get power measurements).

Distance: medium
(emanations signal range)

Thermal, **acoustic**, and **EM** emanation based channels allow for remote signal collection, but depend on strength of the transmitter and type of emanation.

© Jakub Szefer 2019

Sources of Timing Side Channels

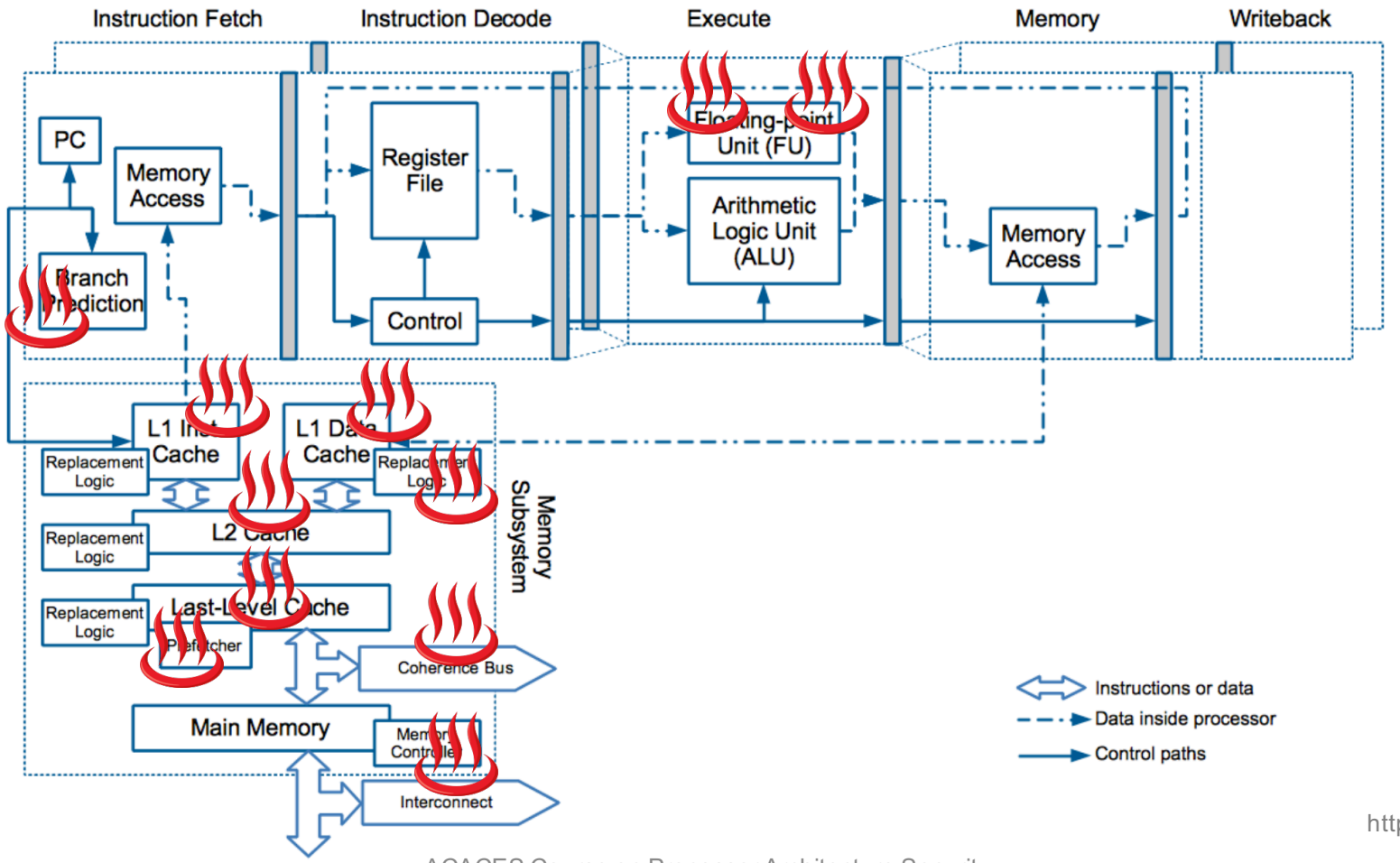
Six source of timing channels that can lead to attacks:

1. **Instruction with Different Execution Timing** – Execution of different instructions takes different amount of time (e.g. ADD vs. FPMUL)
2. **Variable Instruction Timing** – Execution of a specific instruction takes different time (e.g. AVX instructions are fast or slow when AVX is powered on and off, respectively)
3. **Functional Unit Contention** – Sharing of hardware leads to contention, whether a program can use some hardware leaks information about other programs
4. **Stateful Functional Units** – Program's behavior can affect state of the functional units, and other programs can observe the output (which depends on the state)
5. **Prediction Units** – Prediction units can be used to build timing channels, this is different from prediction units being used as part of transient attacks
6. **Memory Hierarchy** – Data caching creates fast and slow execution paths, leading to timing differences depending on whether data is in the cache or not

© Jakub Szefer 2019

Timing Channels Inside a Processor

Many components of a modern processor pipeline can contribute to timing channels.



Emoji Image:
https://www.emojione.com/emoji/2668

Instruction with Different Execution Timing

Computer architecture principles of **pipelining** and **making common case fast** drive processor designs where certain operations take more time than others – program execution timing may reveal which instruction was used.

- Multi-cycle floating point operations vs. single cycle addition
- Execution time of a piece of code depends on the types of instructions it uses, especially, between different runs of software can distinguish from timing if different instructions were executed

Constant time software implementations strive to choose instructions to try to make software run in constant time independent of any secret values

- Instructions with different execution timing are easiest to deal with
- Other sources of timing differences make it more difficult or even not possible to make software run in constant time
  - Note, ”constant time” is not always same time, just that time is independent of secret values

© Jakub Szefer 2019

Variable Instruction Timing

For a specific instruction, its timing depends on the state of the processor. Different state, or different execution history of instructions, affect timing of certain instructions:

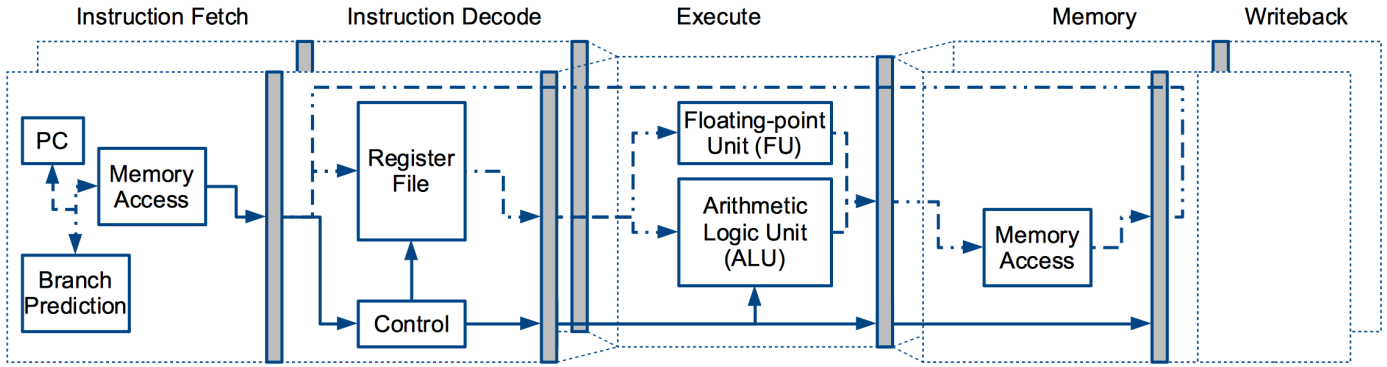
- **Memory loads and stores**: memory access hitting in the cache vs. memory access going to DRAM
- **Multimedia instructions**: whether AVX unit is powered on or not affects timing
- **Reading from special registers** such as RNG: random number generator slows down if it is used a lot and entropy drops
- **Instructions that trigger some state cleanup**, e.g. interrupt latency for SGX enclaves depends on amount of data processor has to clean up and secure before handling the interrupt

© Jakub Szefer 2019

Functional Unit Contention

Functional units within processor are re-used or shared to save on area and cost of the processor resulting in varying program execution.

- Contention for functional units causes execution time differences



Spatial or Temporal Multiplexing allows to dedicate part of the processor for exclusive use by an application

- Negative performance impact or need to duplicate hardware

© Jakub Szefer 2019

Stateful Functional Units

Many functional units inside the processor keep some history of past execution and use the information for prediction purposes.

- Execution time or other output may depend on the state of the functional unit
- If functional unit is shared, other programs can guess the state (and thus the history)
- E.g. caches, branch predator, prefetcher, etc.

Flushing state can erase the history.

- Not really supported today
- Will have negative performance impact

© Jakub Szefer 2019

Prediction Units

Prediction units can be used to build timing channels, this is different from prediction units being used as part of transient attacks.

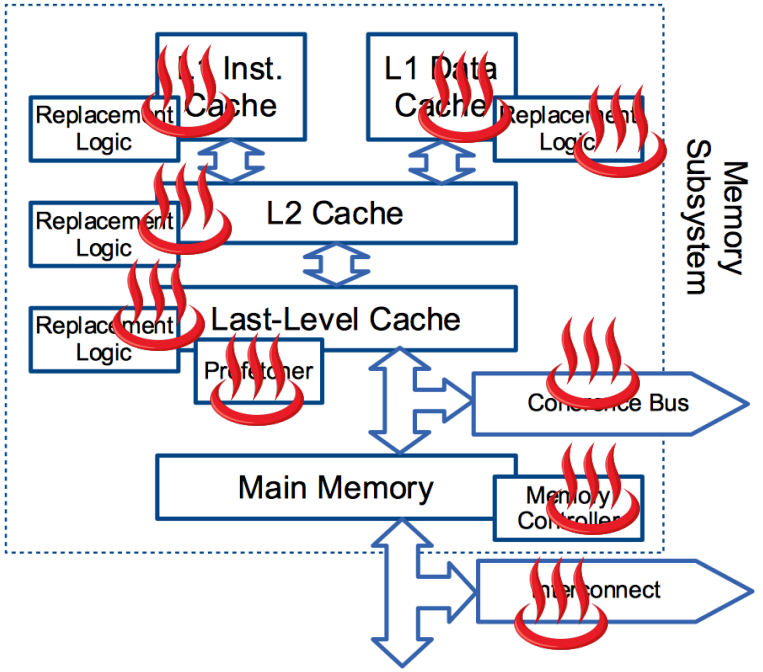
- The **prediction units make prediction based on history** of executed instructions and the processor's state
- The **prediction units are often shared** between threads running on the same core
- Victim's or **sender's execution history can affect the prediction observed by the attacker** thread, and the attacker observe the timing difference

© Jakub Szefer 2019

Memory Hierarchy

Memory hierarchy aims to improve system performance by hiding memory access latency (creating fast and slow executions paths); and most parts of the hierarchy area a shared resource.

- **Caches**
  - Inclusive caches, Non-inclusive caches, Exclusive caches
  - Different cache levels: L1, L2, LLC
- **Cache Replacement Logic**
- **Load, Store, and Other Buffers**
- **TLBs**
- **Directories**
- **Prefetches**
- **Coherence Bus and Coherence State**
- **Memory Controller and Interconnect**



Emoji Image:
https://www.emojione.com/emoji/2668

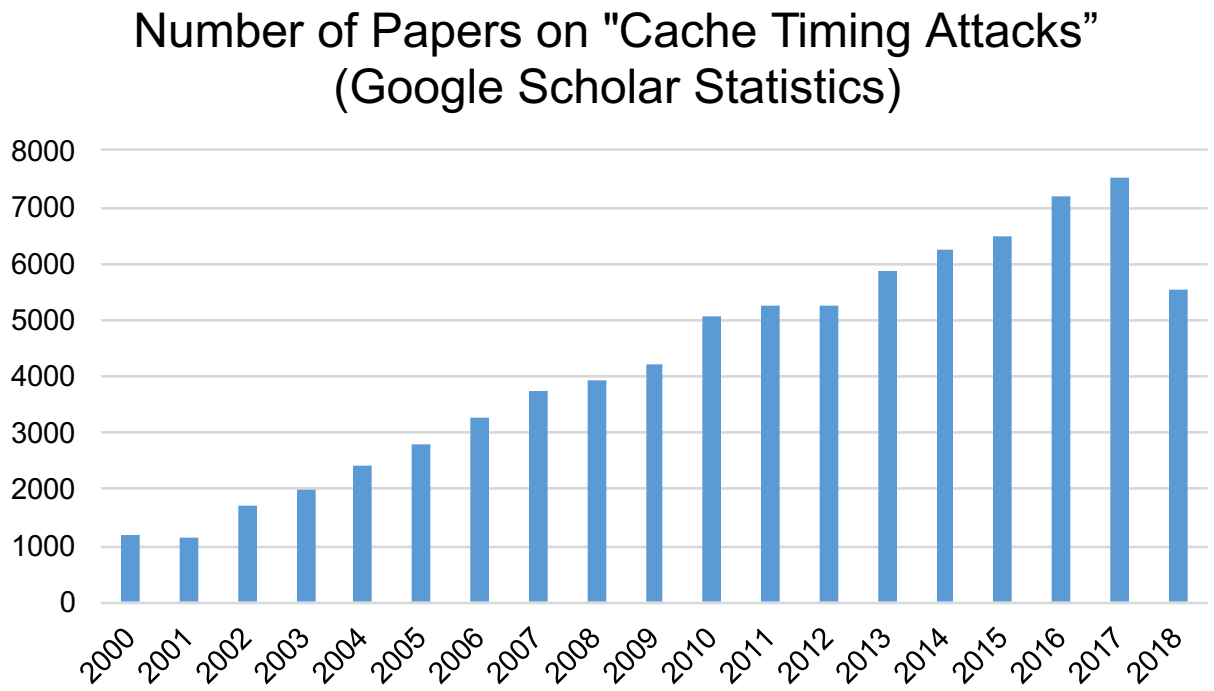
© Jakub Szefer 2019

Timing Channels in Caches

© Jakub Szefer 2019

Cache Timing Attacks Continue to Raise Concerns

- Cache timing attacks have a long history, but the research on attacks and defenses is still a very active field
- Timing attacks using caches, and other cache-like structures, often target cryptographic software
- Very difficult to write “constant time” software, so attacks are still potent
- Attacks can achieve quite high bandwidth in idealized settings, about 1Mbps or more

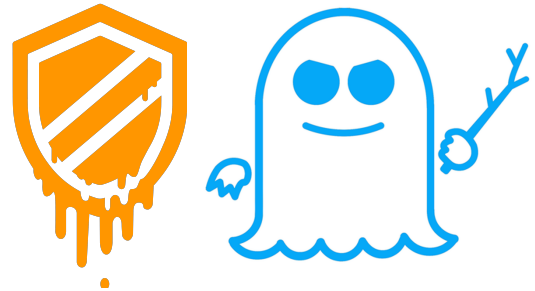


© Jakub Szefer 2019

Cache Timing Attacks Continue to Raise Concerns

- 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



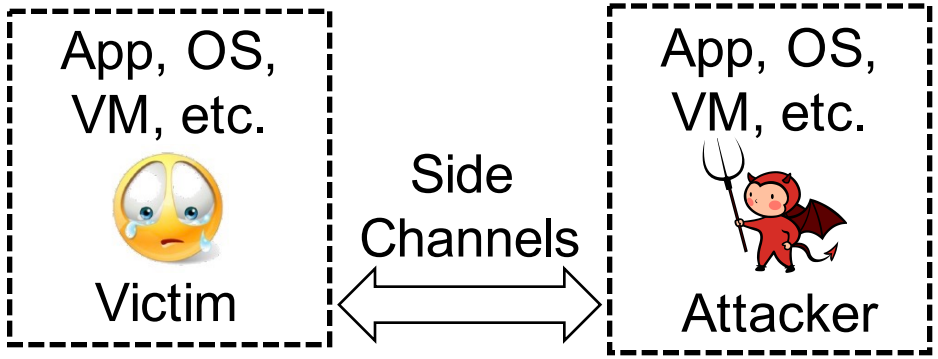
NetSpectre, Foreshadow, SGXSpectre, or SGXPectre
SpectrePrime and MeltdownPrime (both use Prime+Probe instead of original Flush+Reload cache attack)

And more…

© Jakub Szefer 2019

Cache Timing Attacks

- **Attacker and Victim**
  - Victim (holds security critical data)
  - Attacker (attempts to learn the data)



- **Attack requirement**
  - Attacker has ability to monitor timing of cache operations made by the victim or by self
  - Can control or trigger victim to do some operations using sensitive data

- **Operations having timing differences**
  - Memory accesses: load, store
  - Data invalidation: different flushes (clflush, etc.), cache coherence

- **Side-Channel vs. Covert-Channel Attack**
  - Side channel: victim is not cooperating
  - Covert channel: victim (sender) works with attacker – easier to realize and higher bandwidth

- **Many Known Attacks:** Prime+Probe, Flush+Reload, Evict+Time, or Cache Collision Attack

© Jakub Szefer 2019

Cache Eviction Sets

"Theory and Practice of Finding Eviction Sets", P. Vila, et al., arXiv 2018.

- **Eviction Set:** groups of virtual addresses that map to the same cache set

- Many micro-architectural attacks rely on the capability of an **attacker to efficiently find eviction sets** (cache timing channels, Rowhammer attacks, and transient execution attacks, which often use cache timing attacks)

- **Cache designs affecting eviction sets:**
  - Cache is divided into number of sets, each set has way number of cache blocks (also called lines)
  - Each set uses a replacement policy, LRU, PseudoLRU, FIFO, or even dynamic policies in Intel chips
  - Caches can be inclusive, exclusive, or non-inclusive
  - Virtually or physically indexed
  - Caches can be sliced, LLC is divided into slices distributed among cores, cache set to slice mapping is undocumented
  - Academic proposals for randomized caches or skewed caches don't have usual set-associative designs

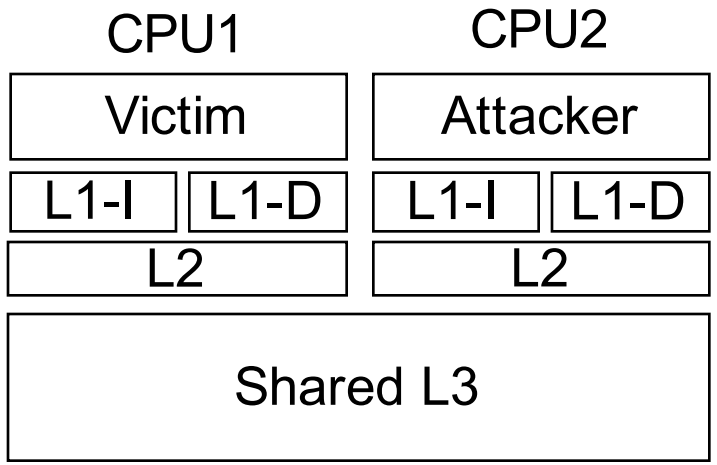
- **Tool for finding eviction sets:** https://github.com/cgvwzq/evsets
  - Have not used the tool, but want to promote development and use of such tools

© Jakub Szefer 2019

Prime-Probe Attacks

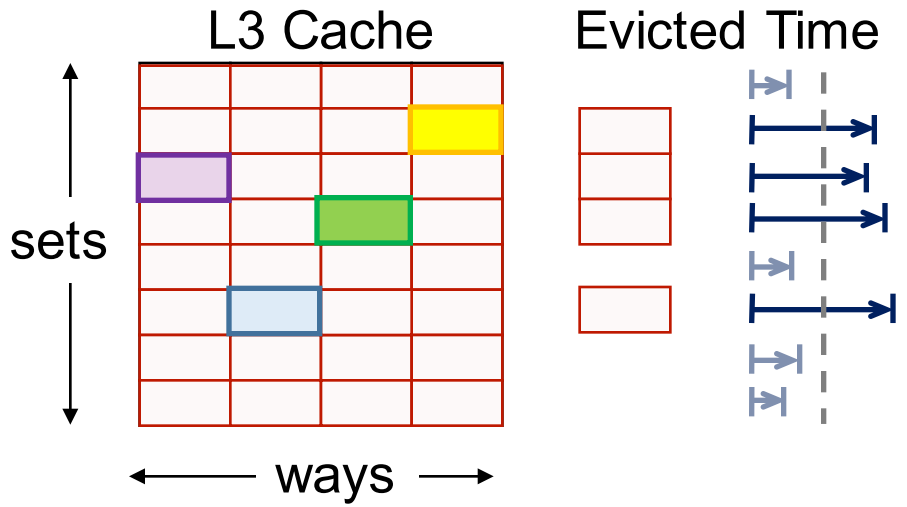
Osvik, D. A., Shamir, A., & Tromer, E, "Cache attacks and countermeasures: the case of AES". 2006.

2- Victim accesses critical data



1- Attacker primes each cache set

3- Attacker probes each cache set (measure time)



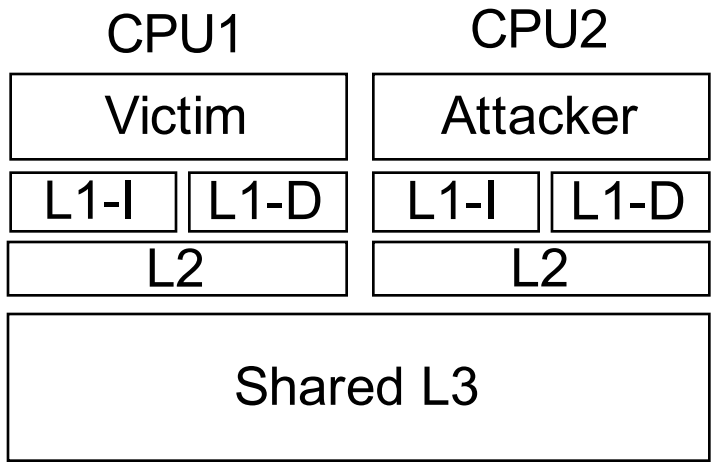
Data sharing is not needed

© Jakub Szefer 2019

Flush-Reload Attack

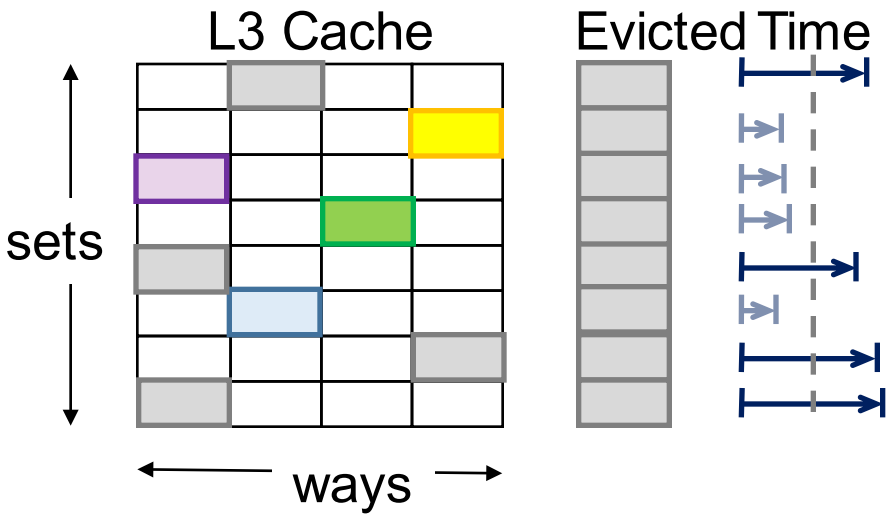
Yarom, Y., & Falkner, K. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack", 2014.

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

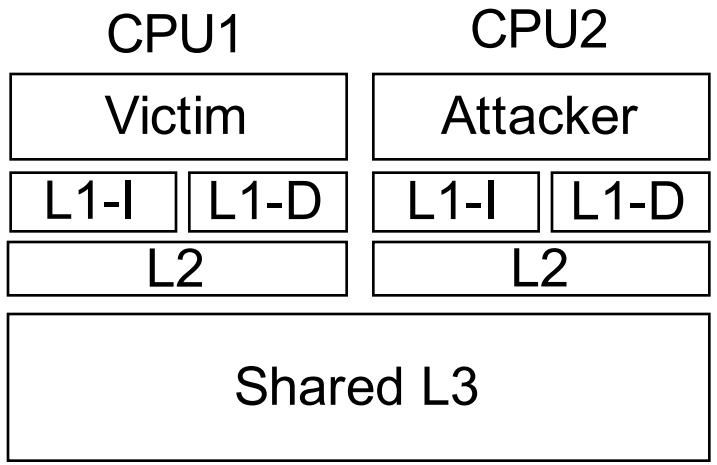
© Jakub Szefer 2019

Evict-Time Attack

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

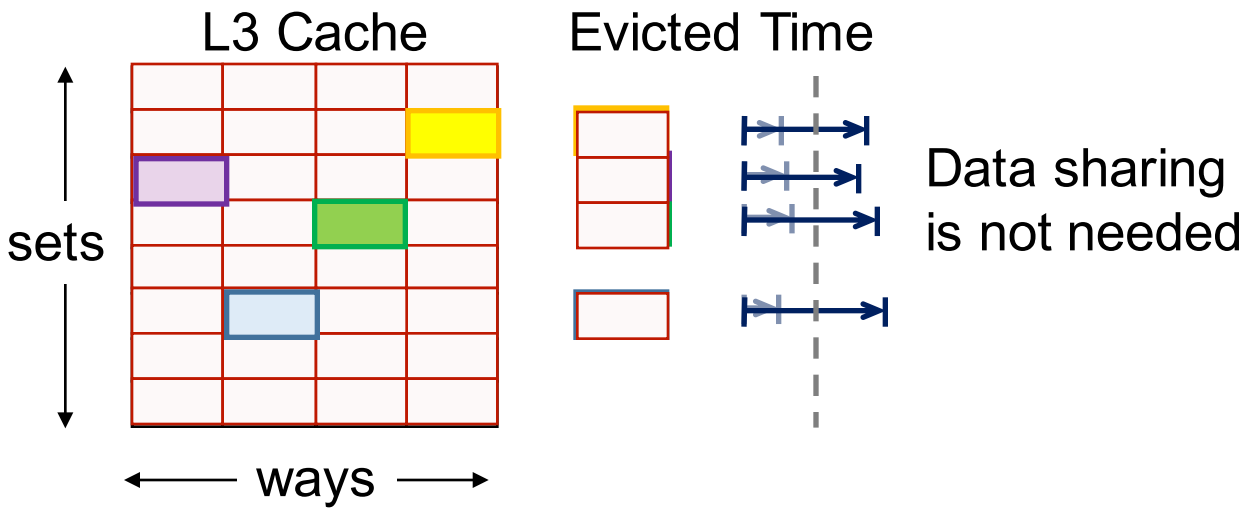
1- Victim has some critical data

3- Victim accesses critical data



2- Attacker evicts cache set and fill its own data (can evict set by set)

(Attacker measures time)



Data sharing is not needed

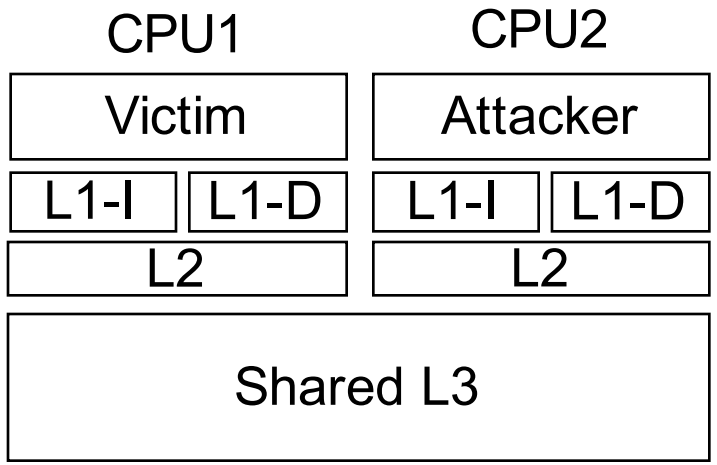
© Jakub Szefer 2019

Cache Collision Attack

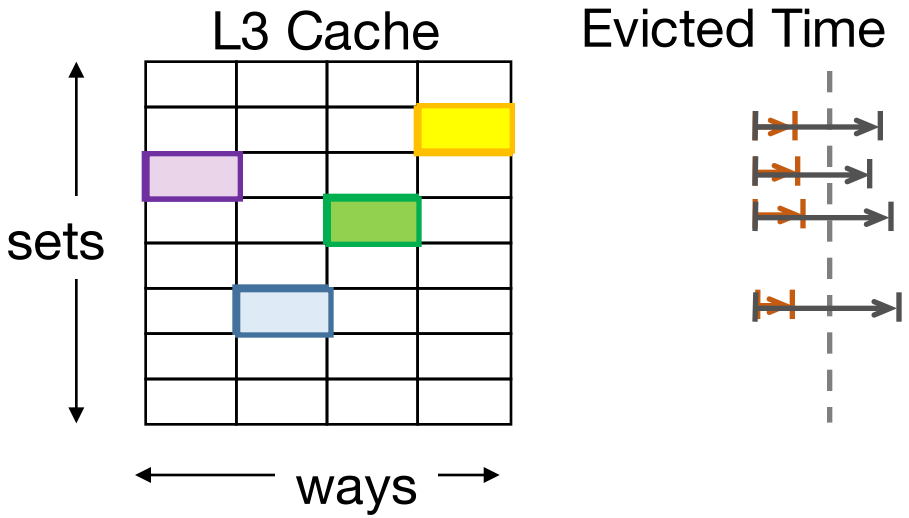
Bonneau, J., & Mironov, I. “Cache-collision timing attacks against AES”, 2006.

1- Victim has some critical data

2- Victim reuses critical data



(Attacker measures time)



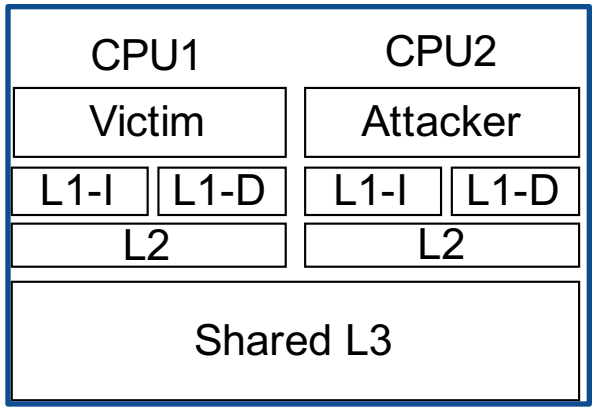
Data sharing is not needed

© Jakub Szefer 2019

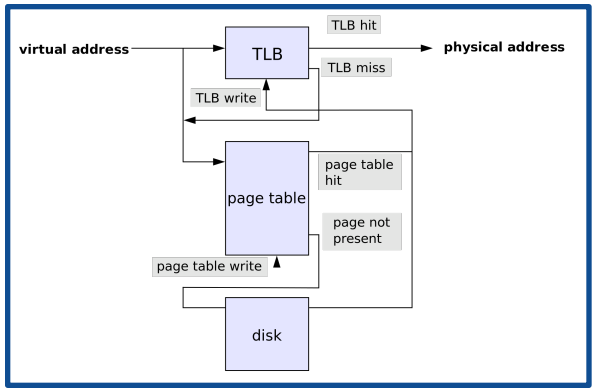
Similar Attacks on Cache-Like Structures

Timing attacks do not only leverage caches, but any cache-like structure with varying timing (due to hits or missies in the structure) can be vulnerable to timing attacks

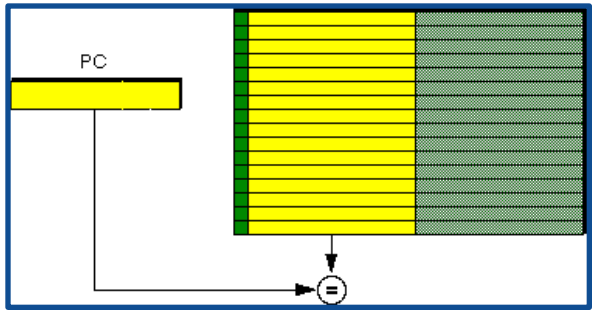
Instruction or Data Cache



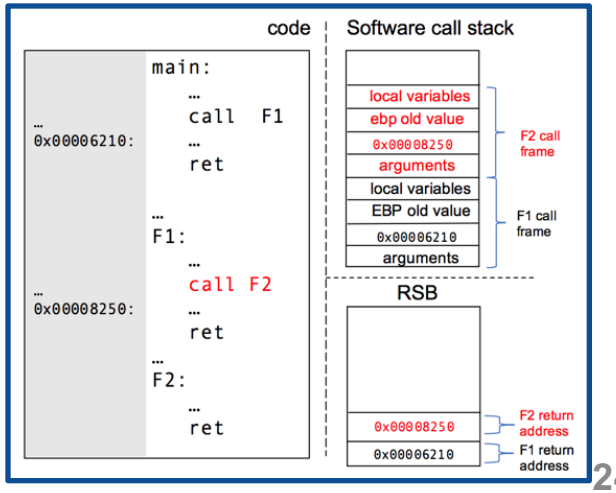
Translation Look-aside Buffer (TLB)



Branch Target Buffer (BTB)



Return Stack Buffer (RSB)



Typical attacks:
Cache: Bonneau, J., & Mironov, I, “Cache-collision timing attacks against AES”, 2006
TLB: Gras, B., Razavi, K., Bos, H., & Giuffrida, C, “Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with {TLB} Attacks”, 2018
BTB: Evtyushkin, D., Riley, R., Abu-Ghazaleh, N. C., & Ponomarev, D, “Branchscope: A new side-channel attack on directional branch predictor”, 2018
RSB: Koruyeh, E. M., Khasawneh, K. N., Song, C., & Abu-Ghazaleh, N., “Spectre returns! speculation attacks using the return stack buffer”, 2018

© Jakub Szefer 2019

All Possible Timing Attacks in Caches

© Jakub Szefer 2019

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 within 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)

$Step1$	$Step2$	$Step3$ (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

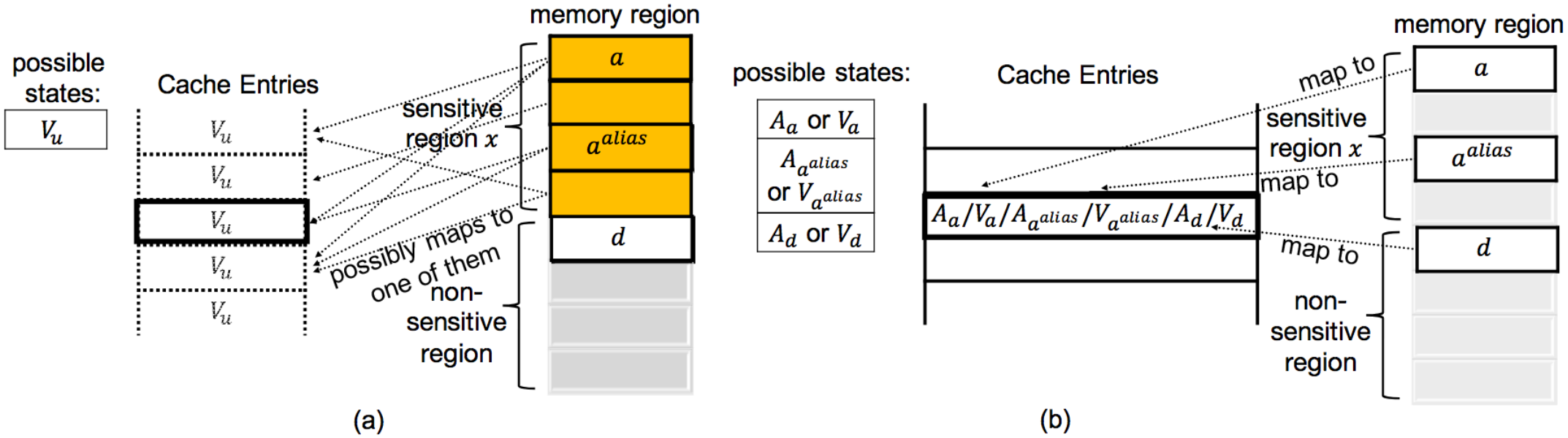
- **There are 72 possible cache timing attack types**

© Jakub Szefer 2019

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

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

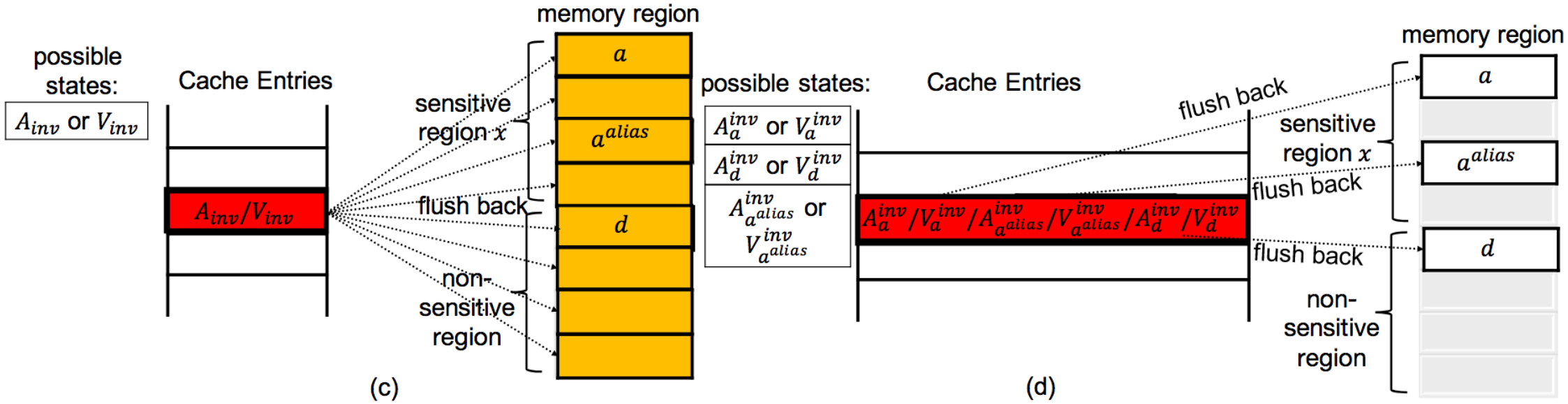


© Jakub Szefer 2019

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

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

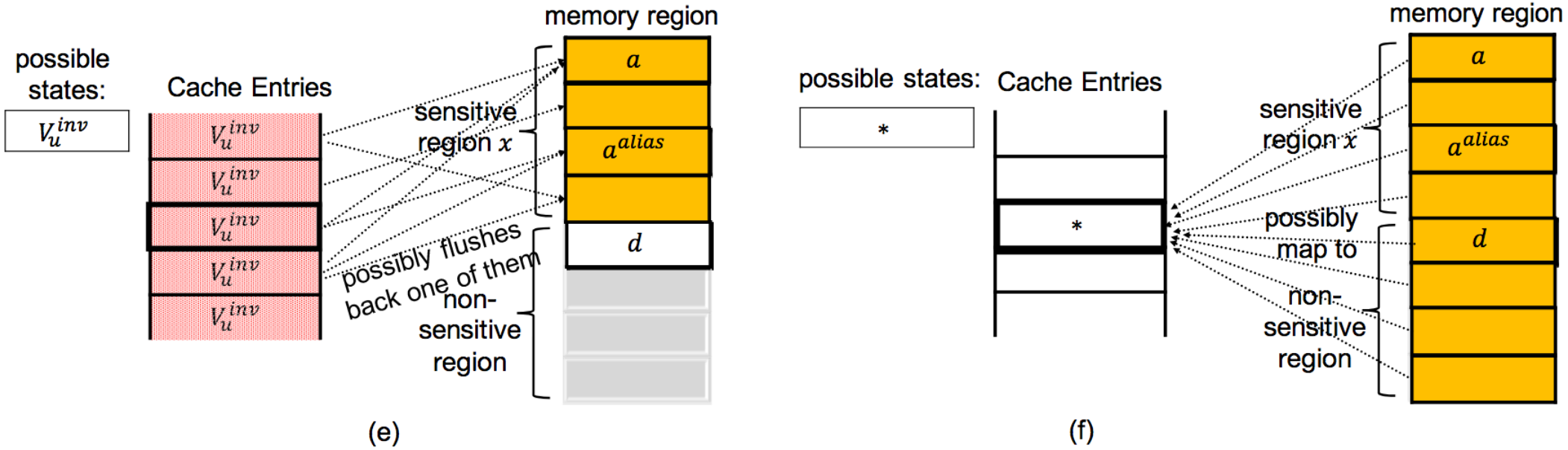


© Jakub Szefer 2019

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

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

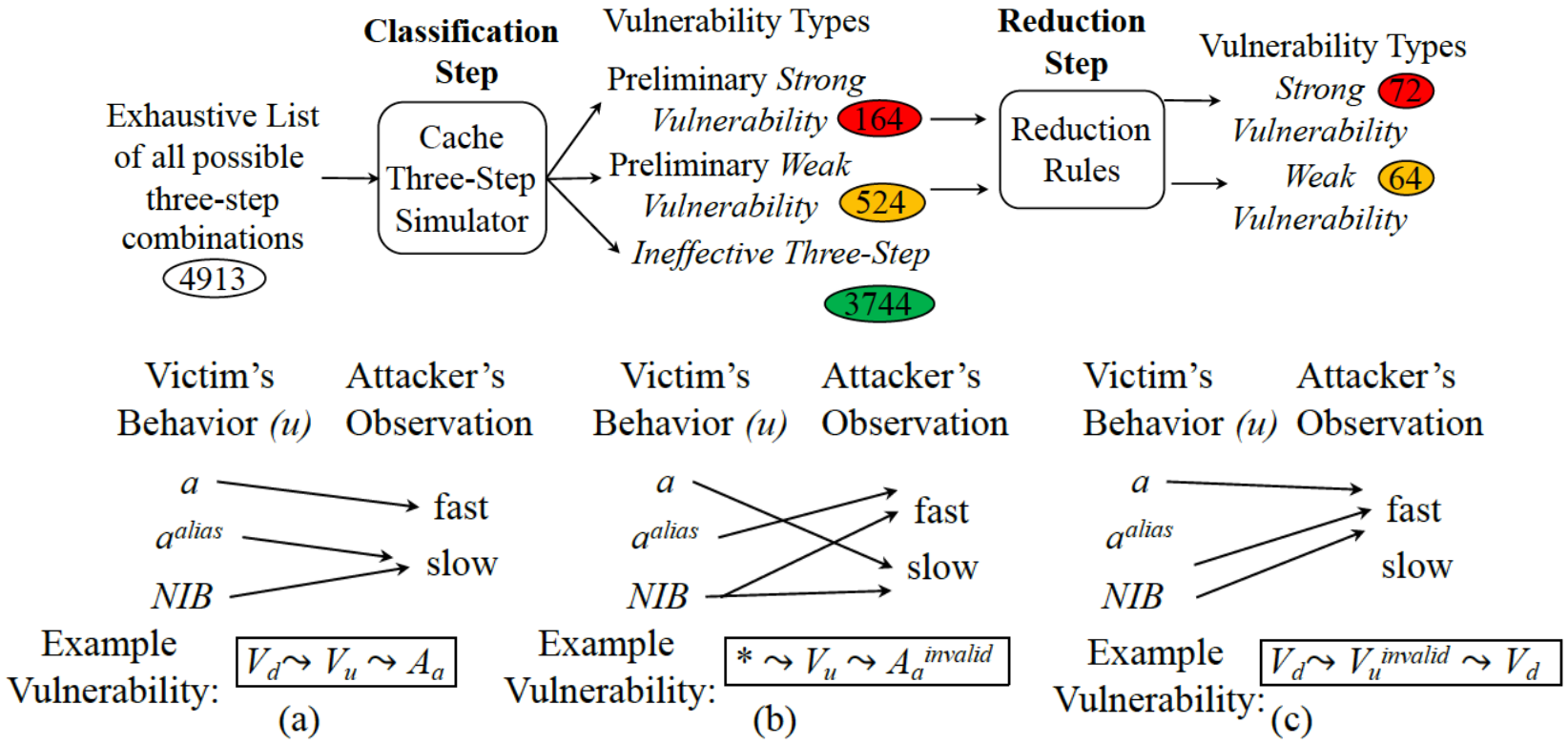


© Jakub Szefer 2019

A Three-Step Model for Cache Timing Attack Modeling

"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



© Jakub Szefer 2019

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

Cache Collision

Flush+ Reload

Evict+ Time

Prime+ Probe

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	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].

Understanding All Possible Timing Attacks

- The Prime+Probe, Flush+Reload, Evict+Time, or Cache Collision attacks are just some of the possible timing attacks
- **Defenders need to understand all possible types of attacks**, as attacker just needs to find out that works – but defenders need to protect all types of attacks
- A recent **3-step model can be used to understand timing attacks**…

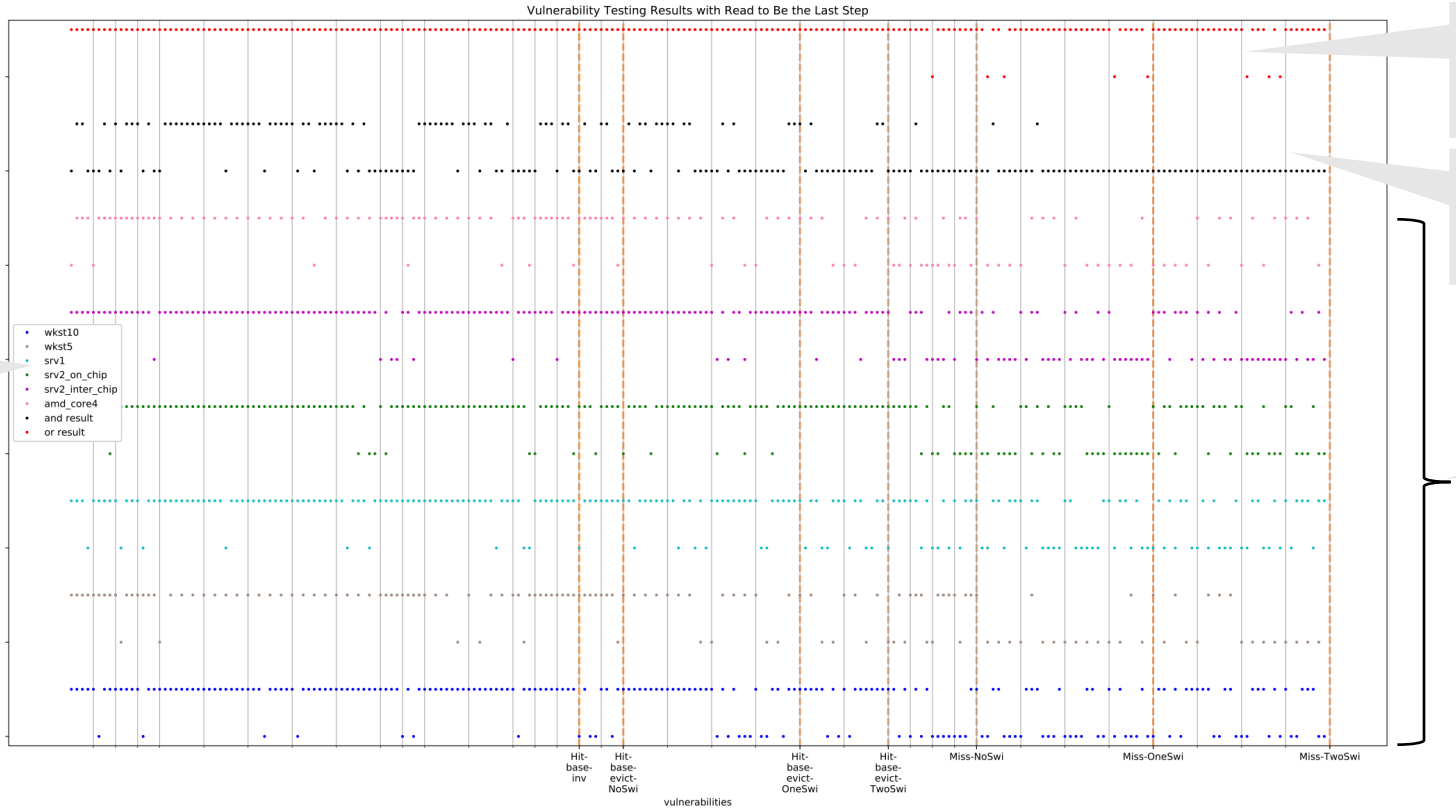
…most attacks have been known in literature under various names, but:

- **Possible new, untested attacks exist**
- Systematic approach to checking for attacks is necessary, not just for caches, but TLBs and other cache-like structures.

© Jakub Szefer 2019

Security Micro-Benchmarks for Cache Timing Attacks

- On-going research in Prof. Szefer's group looks into development of open-source benchmarks for quantifying cache timing attacks



Different CPUs tested

Tests that pass for one or more CPUs

Tests that pass for all CPUs

Tests for individual CPUs

© Jakub Szefer 2019

Beyond Classical Cache Channels

© Jakub Szefer 2019

Timing Channels due to Other Components

- **Cache Replacement Logic** – LRU states can be abused for a timing channel, especially cache hits modify the LRU state, no misses are required
- **Load, Store, and Other Buffers** – different buffers can forward data that is in-flight and not in caches, this is in addition to recent Micro-architectural Data Sampling attacks
- **TLBs** – Translation Look-aside Buffers are types of caches with similar vulnerabilities
- **Directories** – Directory used for tracking cache coherence state is a type of a cache as well
- **Prefetches** – Prefetchers leverage memory access history to eagerly fetch data and can create timing channels
- **Coherence Bus and Coherence State** – different coherence state of a cache line may affect timing, such as flushing or upgrading state
- **Memory Controller and Interconnect** – memory and interconnect are shared resources vulnerable to contention channels

© Jakub Szefer 2019

Classical vs. Speculative Side-Channels

Side channels can now be classified into two categories:

- **Classical** – which do not require speculative execution
- **Speculative** – which are based on speculative execution

Difference is victim is not fully in control of instructions they execute (i.e. some instructions are executed speculatively)

Root cause of the attacks remains the same

State of functional unit is modified by victim and it can be observed by the attacker via timing changes

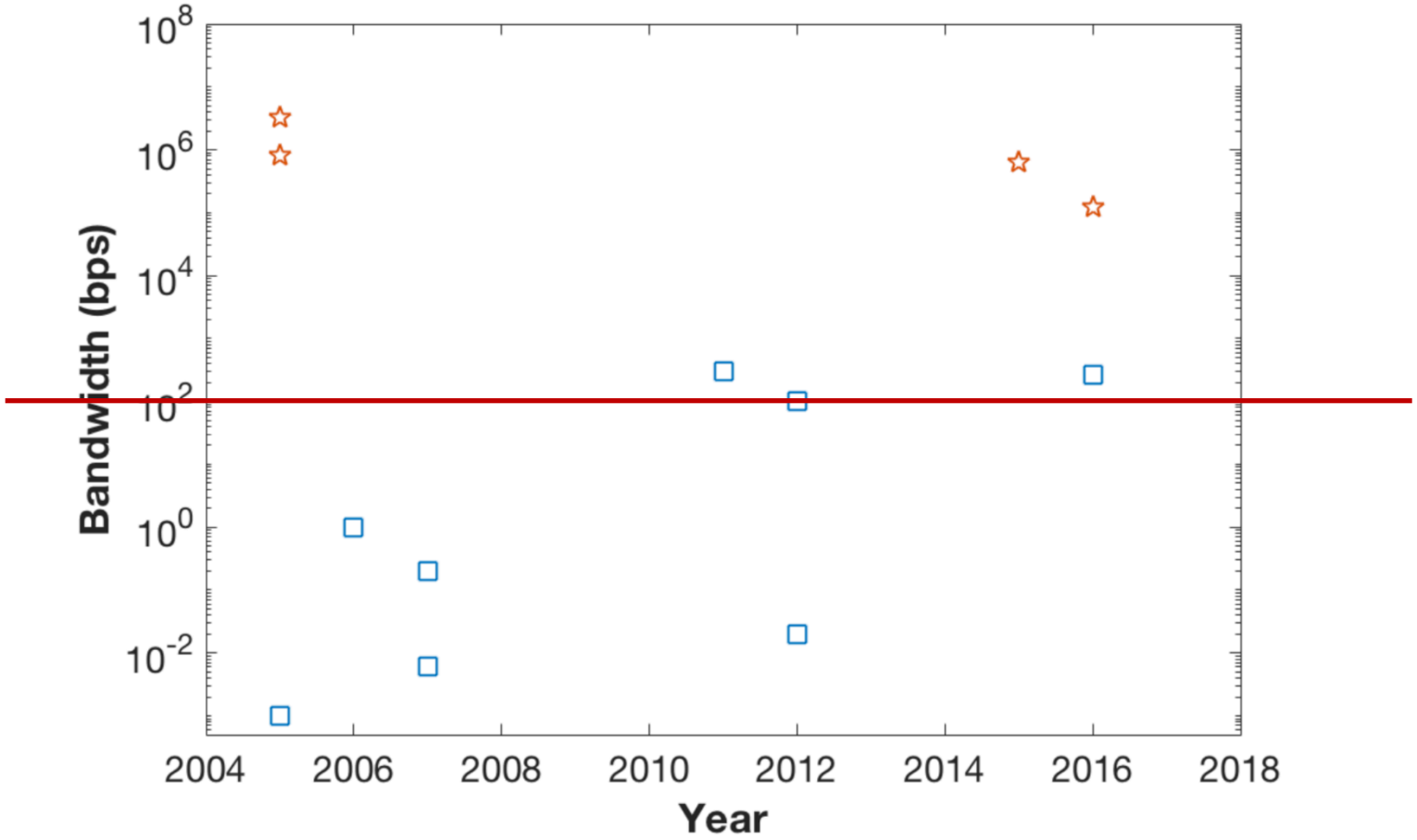
Defending classical attacks defends speculative attacks as well, but not the other way around

Focusing only on speculative attacks does not mean classical attacks are prevented, e.g. defenses for cache-based attacks

© Jakub Szefer 2019

Timing Channel Bandwidths

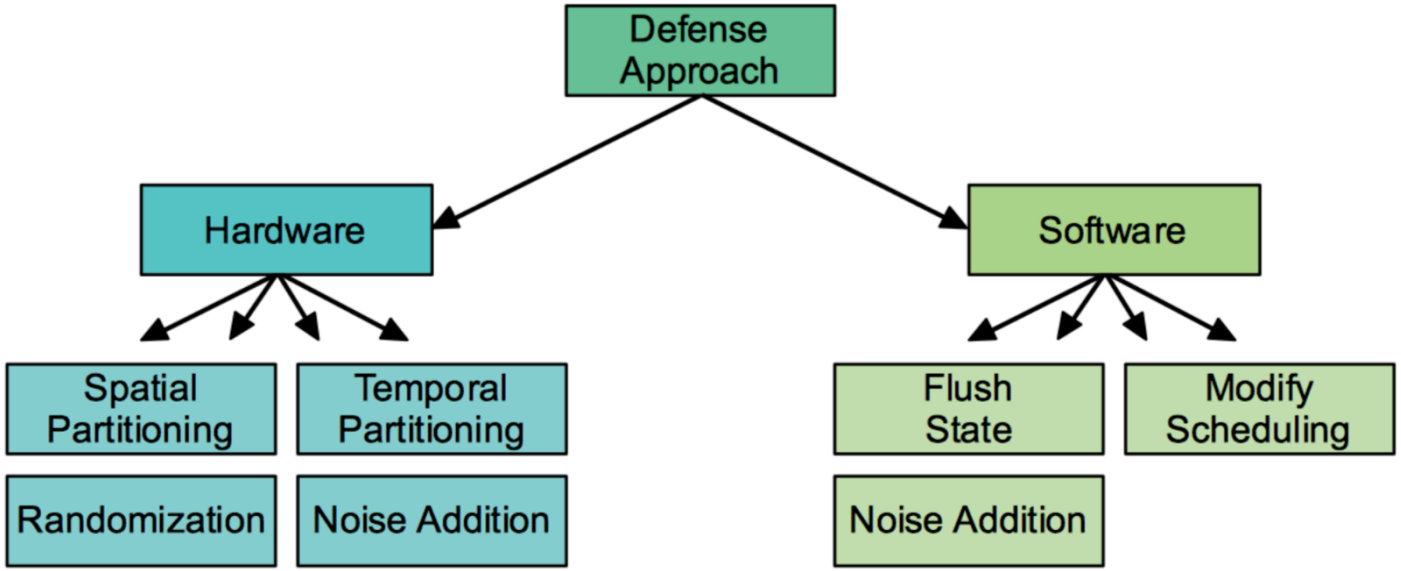
The Orange Book, also called the Trusted Computer System Evaluation Criteria (TCSEC), specifies that a channel bandwidth exceeding a rate of **100 bps is a high bandwidth channel**.



© Jakub Szefer 2019

Timing Channel Defense Strategies

Hardware and software based defenses are possible. Most will result in performance degradation.



© Jakub Szefer 2019

Side Channels as Attack Detectors

Side channels can be used to detect or observe system's operation

- Measure timing, power, EM, etc. to detect unusual behavior
- Similar to using performance counters
- Attacker doesn't have a way to prevent the side channels, otherwise the problem of side channels would have been solved

Tension between **side channels as attack vectors vs. detection tools**.

- Side channels are mostly used for attack today
- Desire to eliminate side (and covert) channels – but it precludes use of the channels for defense

Observer can use side-channels to deduce the presence and behavior of the attacker

Attacker's behavior modifies processor state or created emanations

Emoji Image:
https://www.emojione.com/emoji/1f468-1f4bc
https://www.emojione.com/emoji/1f469-1f4bc

© Jakub Szefer 2019

Industry Standards for Evaluating System's Security

Orange Book or the Trusted Computer System Evaluation Criteria (TCSEC)

- Replaced by Common Criteria
- Standard for assessing the effectiveness of a computer system's security controls

Common Criteria

- Standard for computer security certification

FIPS 140-2

- Standard defining security levels for cryptogrphic modules

© Jakub Szefer 2019

Timing Side Channels which Use Speculation

© Jakub Szefer 2019

Timing Side Channels which Use Speculation

- Modern computer architectures gain performance by using prediction mechanisms:
  - Successful prediction = fast execution and performance gain
  - Mis-prediction = slow execution and performance loss

- The prediction units (e.g., branch predictor, prefetcher, memory disambiguation prediction, etc.) make prediction based on prior history of executed instructions and data
- The prediction units are often shared between threads in SMT cores
- Victim's execution history can affect the prediction observed by the attacker thread, and the attacker can observe the timing difference

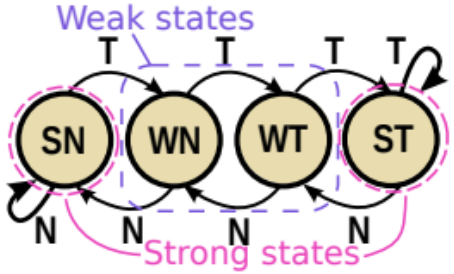
- **This type of side channels are different from the transient executions attacks**
  - In transient execution attacks, secrets are accessed during mis-prediction
  - In timing side channels using speculation victim's behavior is leaked to the attacker through the mis-prediction (or lack there of) by the attacker

© Jakub Szefer 2019

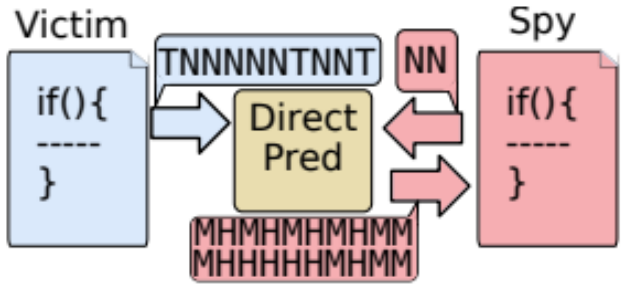
Pattern History Table (PHT) : BranchScope

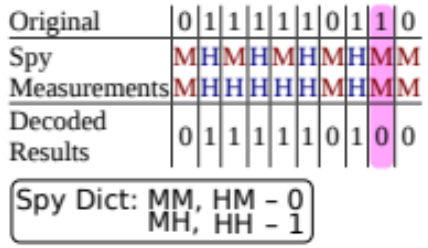
D. Evtyushkin, et al., "BranchScope: A New Side-Channel Attack on Directional Branch Predictor", 2018
D. Evtyushkin, et al., "Covert Channels Through Branch Predictors: A Feasibility Study", 2015

- PHT is shared among all processes on core, and is not flushed on context switches
- The branch predictor stores its history in the form of a 2-bit saturating counter in a pattern history table (PHT)



- The PHT entry used is a simple function of the branch address
- Prime+Probe Strategy
- Attacks:
  - Covert channels
  - Attack SGX enclave code






```
int sec_data[]
= {1,0,1,1,..};
i = 0;
void victim_f(){
 //Victim Branch
 if(sec_data[i])
  asm("nop;nop");
 i++;
}
```

Code 1. Pseudo-code of the victim.

```
int probe_array [2] = {1, 1};//Not-taken
int main(){
  for(int i = 0; i < N_BITS; i++){
    randomize_pht();//(1)
    usleep(SLEEP_TIME);//Wait for victim
    spy_function(probe_arr); } }
void spy_function(int array [2]){
  for(int i = 0; i < 2; i++){
    a = read_branch_mispred_counter();
    if(array[i])// <- Spy branch
      asm("nop; nop; nop;");
    b = read_branch_mispred_counter();
    store_branch_mispred_data(b - a); } }
```

Code 2. Pseudo-code of the attacker.

© Jakub Szefer 2019

Branch Target Buffer (BTB): Jump Over ASLR

D. Evtyushkin, et al., "Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR", 2016

- The BTB stores target addresses of recently executed branch instructions, so that those addresses can be obtained directly from a BTB lookup

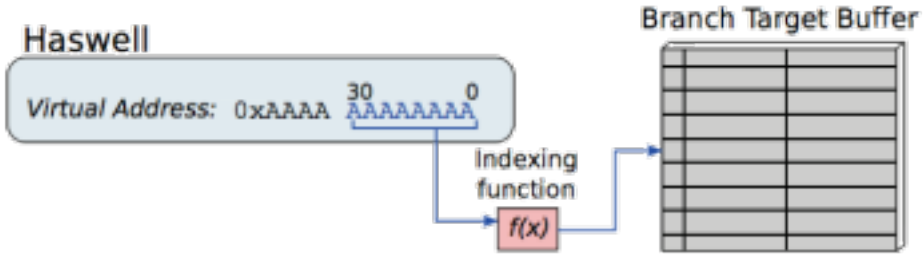


Fig. 4: BTB addressing scheme in Haswell processor

- Same-Domain Collisions (SDC)
- BTB collisions between two processes executing in the same protection domain

- Attacks:
  - Attack KASLR (Kernel address space layout randomization)

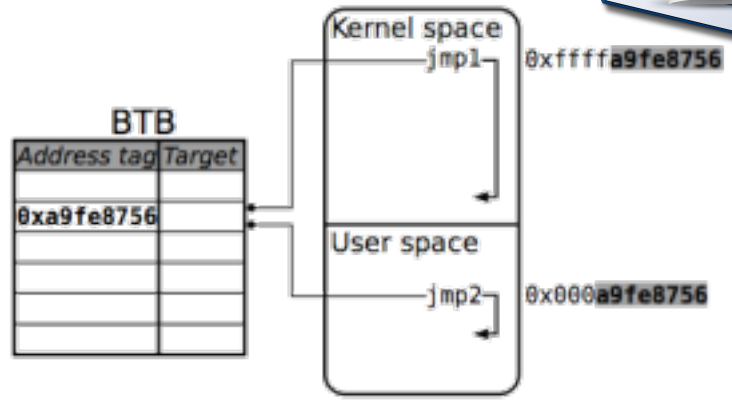


Fig. 5: CDC Example

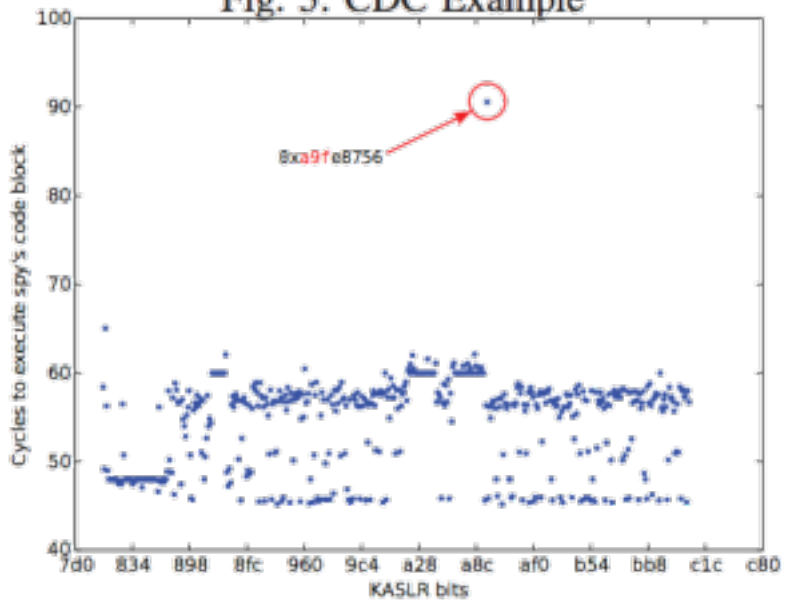


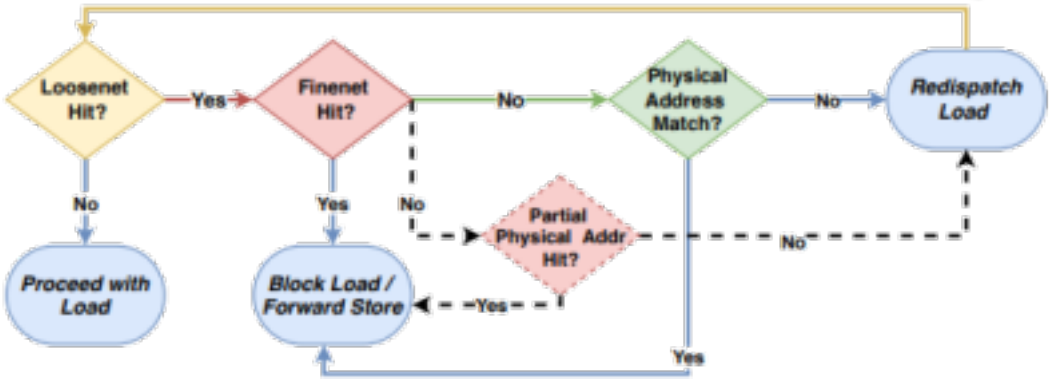
Fig. 7: Results of the BTB-based Attack on KASLR

© Jakub Szefer 2019

Memory Disambiguation: Spoiler Attack

S. Islam, et al., "SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks", 2019

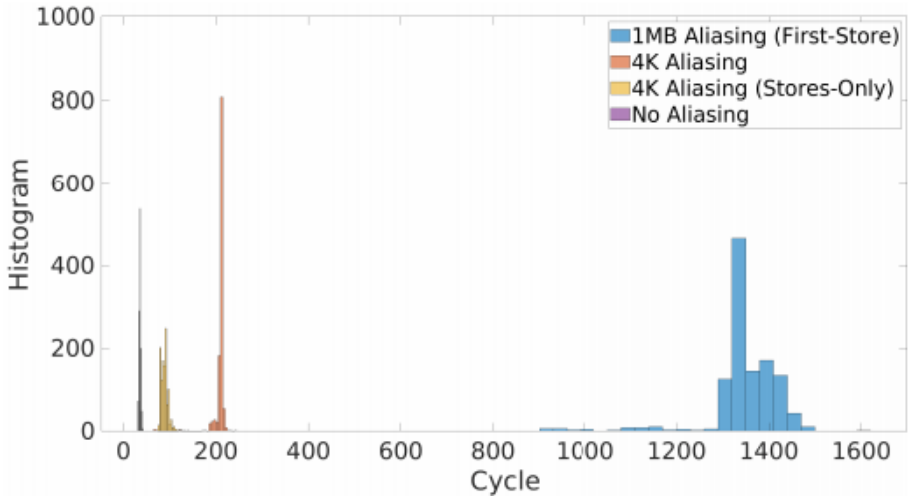
- The processor executes the load speculatively before the stores, and forwards the data of a preceding store to the load if there is a potential dependency



- The finenet check may be implemented based on checking the partial physical address bits

- 1MB aliasing in Intel processors

- Attacks: Leakage of the Physical Address Mapping
  - Efficient eviction set finding for Prime+Probe attacks in LLC
  - Helps to conduct DRAM row conflicts



© Jakub Szefer 2019

Related Attacks

"Unveiling Hardware-based Data Prefetcher, a Hidden Source of Information Leakage", Y. Shin, et al., CCS 2018

Prefetchers have been abused for timing attacks

- E.g. IP-based stride prefetcher, has been used to break cryptographic algorithm implementations
- Any cryptographic algorithm implementation that utilizes a lookup table is subject to the attack
  - Pattern of accesses in the table will be revealed by the data that is prefetched
- Prefetching is a type of prediction or speculation

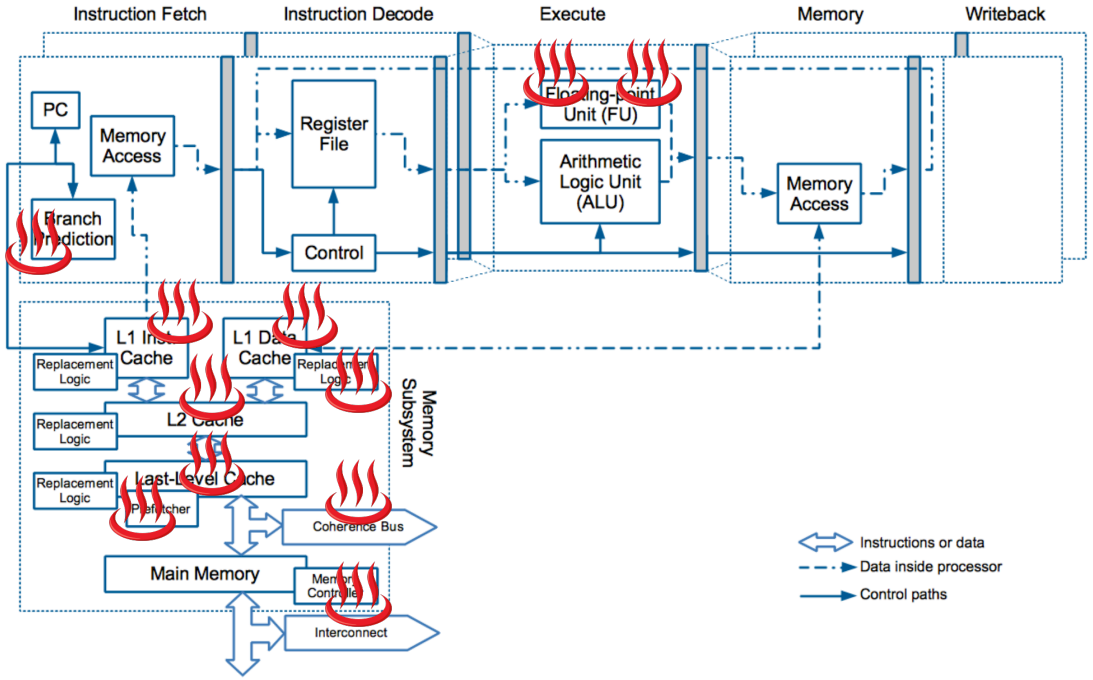
Prefetchers in Intel processors

No.	Hardware prefetcher	Detection technique	Cache Level	Bit # in MSR 0x1a4
1	Streamer	Stream	L2	0
2	Spatial prefetcher	Adjacent-line	L2	1
3	DCU prefetcher	Next-line	L1	2
4	IP-based stride prefetcher	Stride	L1	3

© Jakub Szefer 2019

Summary

- Side and covert channels continue to pose danger to processors
- Timing channels don't require physical access to the machine
- Among others, shared components or ones with behavior based on prior execution history contribute a lot to timing channels and are not easy to eliminate (without performance penalty)
- Most units in a processor somehow contribute to timing channels
- Channels are both classical and ones used as part of transient execution attacks



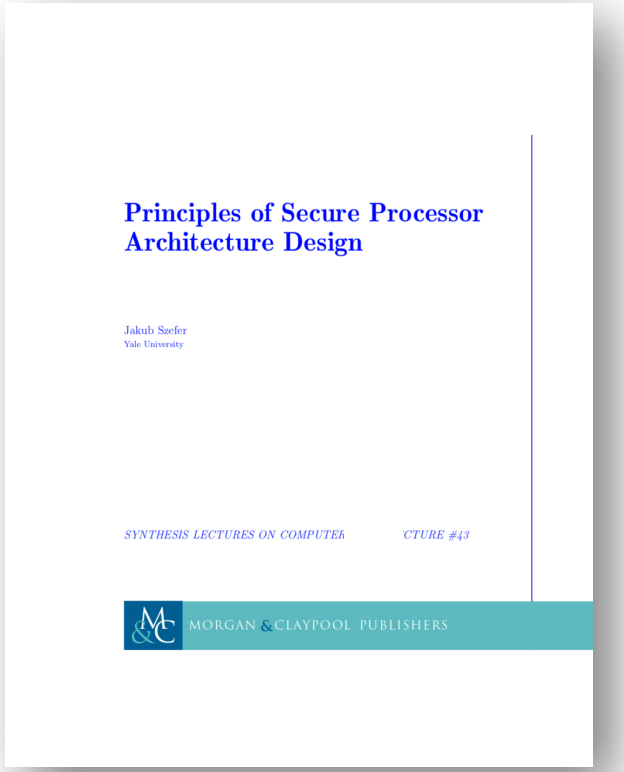
© Jakub Szefer 2019

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/



© Jakub Szefer 2019