Flush+Flush: A Fast and Stealthy Cache Attack

Daniel Gruss, Clémentine Maurice, Klaus Wagner and Stefan Mangard
Graz University of Technology

July 8, 2016 — DIMVA 2016

Motivations

- cache side channels are stealthy attacks $\rightarrow$ not detected by IDS

Motivations

- cache side channels are stealthy attacks → not detected by IDS
- but cache attacks have a large impact on the cache

→ use performance counters for detection [HF15; Pay16]

→ detect abnormal behavior

Contributions

- novel attack Flush+Flush
  - stealthier than previous cache attacks
  - faster than previous cache attacks
- framework to evaluate cache attacks
  - in terms of performance and detectability

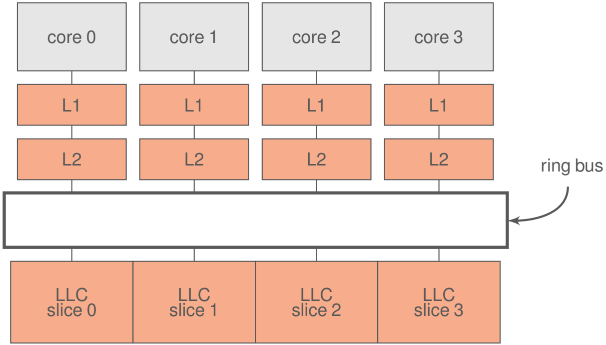
→ `https://github.com/IAIK/flush_flush`

1. Introduction

2. Flush+Flush

3. Conclusion and future work

Caches on Intel CPUs



- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive
  - hash function for addressing [MLSNHF15]

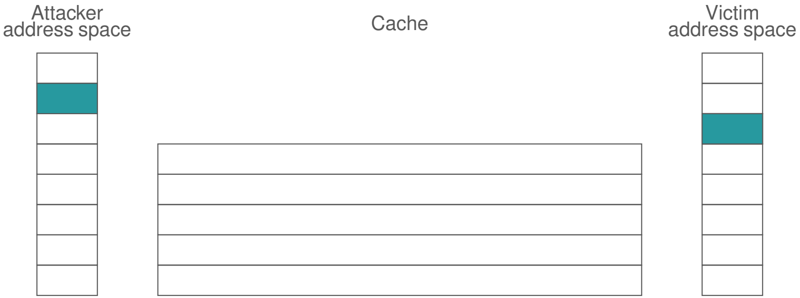
Access-driven cache attacks

Attacker monitors its own activity to find sets accessed by victim.

Flush+Reload	Prime+Probe
[GBK11]	[Per05]
[YF14]	[LYGHL15]
[GSM15]	[MNHF15]

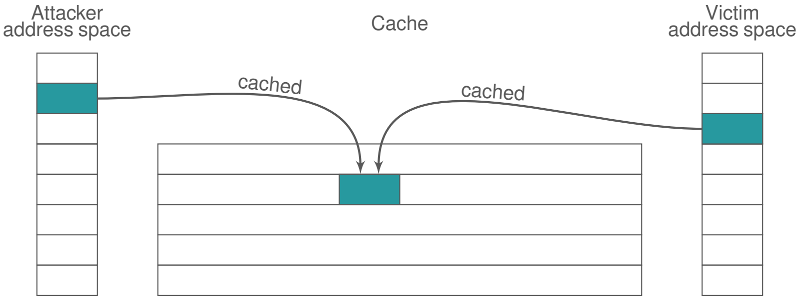
Exploit timing difference between cache hits and cache misses

Flush+Reload



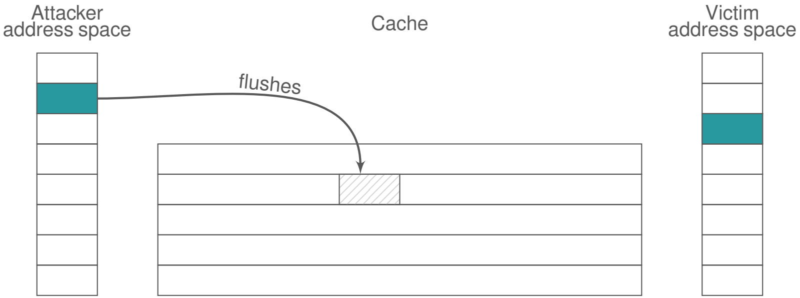
step 0: attacker maps shared library → shared memory, shared in cache

Flush+Reload



step 0: attacker maps shared library → shared memory, shared in cache

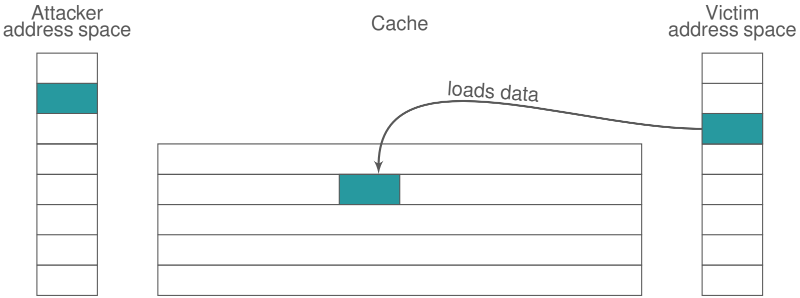
Flush+Reload



step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line with `clflush`

Flush+Reload

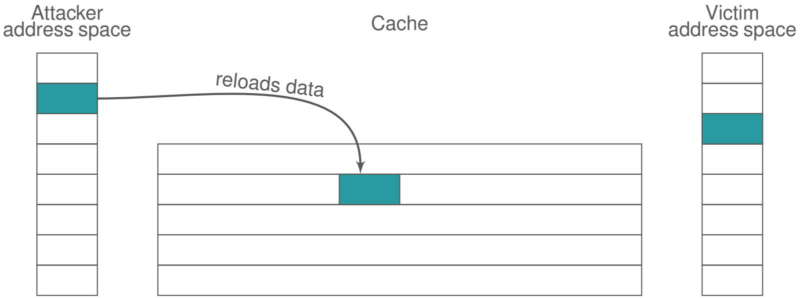


step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line with `clflush`

step 2: victim loads data while performing encryption

Flush+Reload



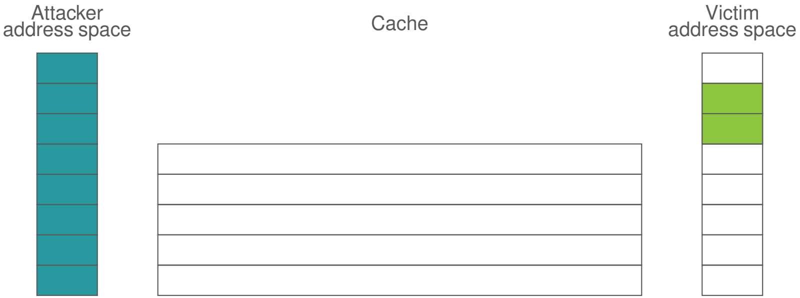
step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line with `clflush`

step 2: victim loads data while performing encryption

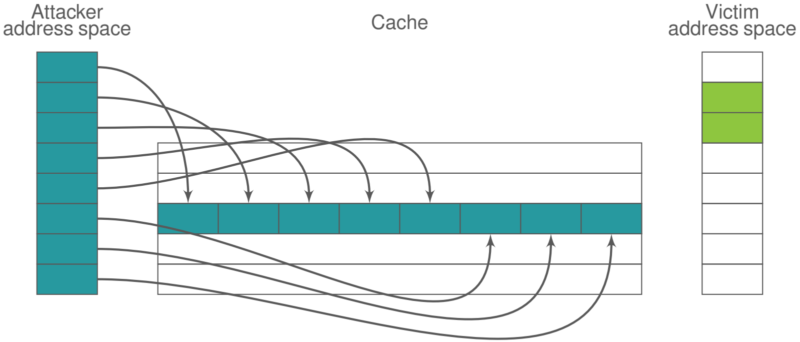
step 3: attacker reloads data → fast access if the victim loaded the line

Prime+Probe



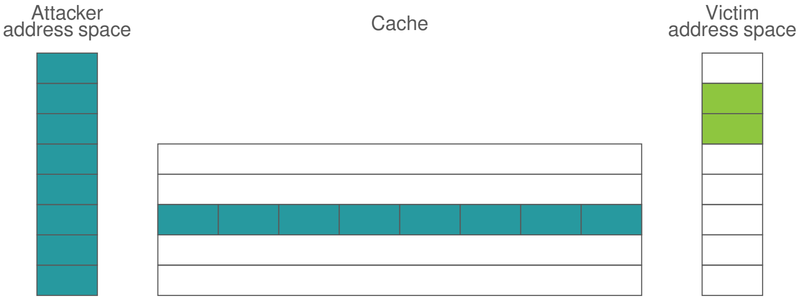
step 0: attacker fills the cache (prime)

Prime+Probe



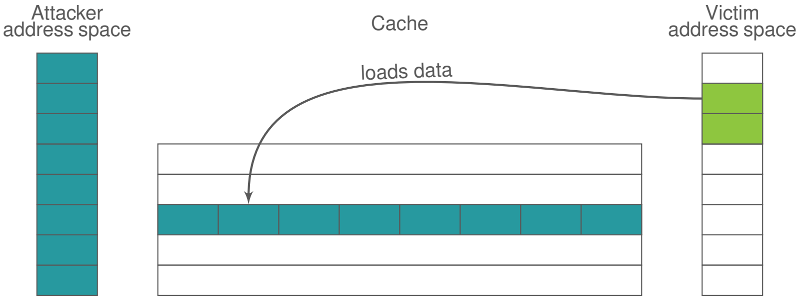
step 0: attacker fills the cache (prime)

Prime+Probe



step 0: attacker fills the cache (prime)

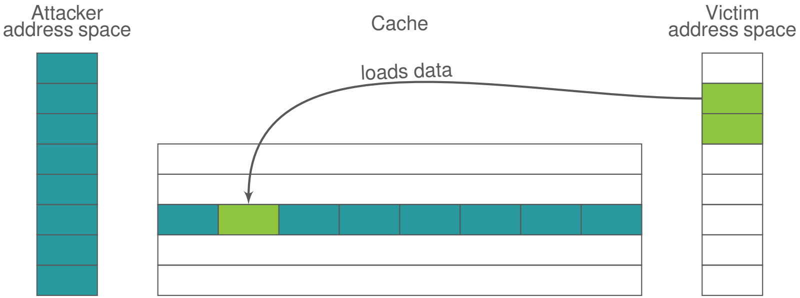
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

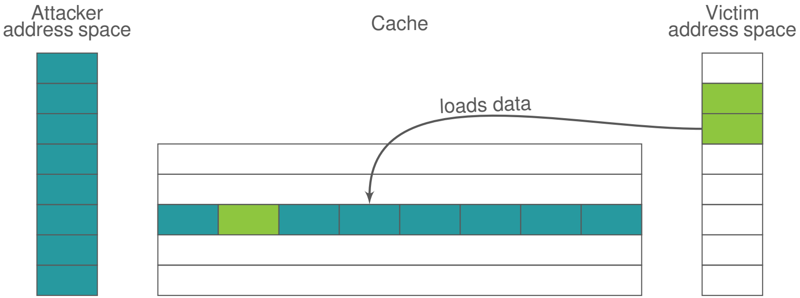
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

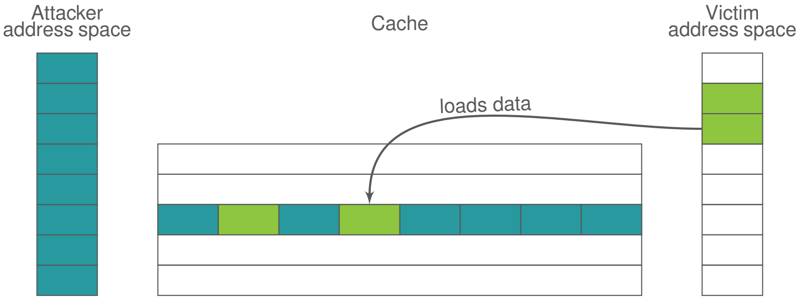
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

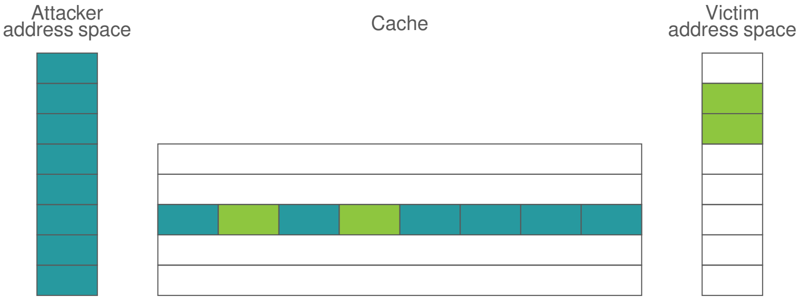
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

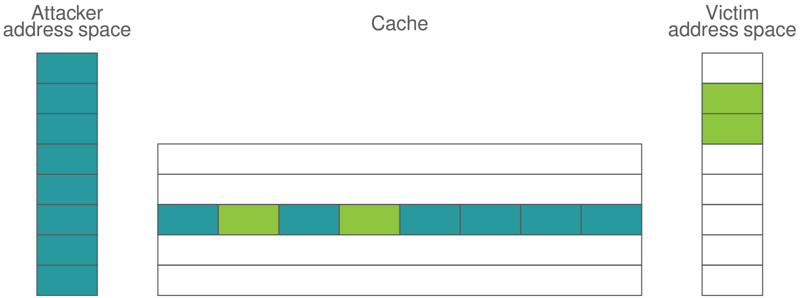
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

Prime+Probe

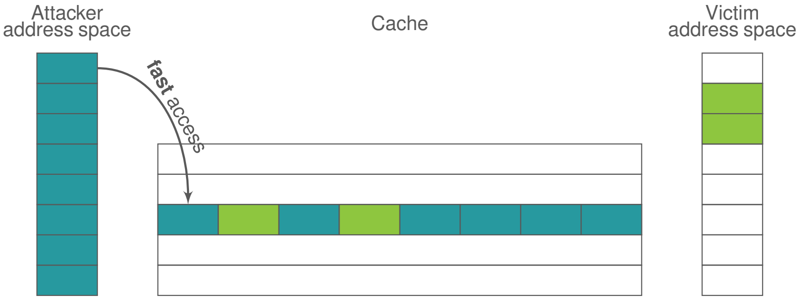


step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Prime+Probe

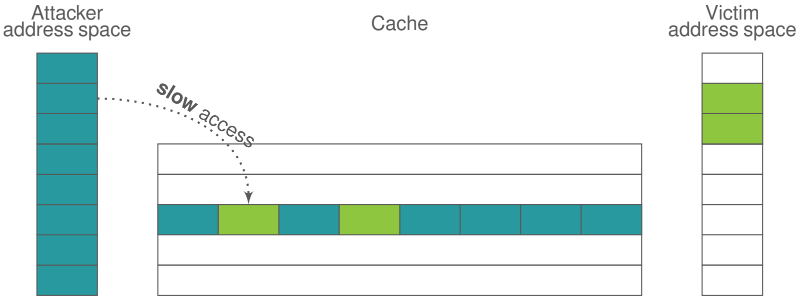


step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Examples of cache attacks

Covert channel:

- cross-VM, cross-core: 600kbps [LYGHL15]

Side channel:

- cross-VM, cross-core side-channel attacks on crypto algorithms
  - RSA: 96.7% of key bits in 1 signature [YF14]
- tracking user behavior in the browser, in JavaScript [OKSK15]

1. Introduction

2. Flush+Flush

3. Conclusion and future work

Which performance counters (1)

We analyzed events w.r.t cache attacks and benign programs:

- 23 hardware and cache performance events with `perf_event_open`
- `UNC_CBO_CACHE_LOOKUP` of CBo with MSRs

→ rmq: only possible to monitor 4 events simultaneously

Which performance counters (2)

Detecting cache attacks and Rowhammer

1. `CACHE_MISSES` → occur after data is flushed
2. `CACHE_REFERENCES` → occur when reaccessing memory
3. `L1D_RM` → occur after data is flushed
4. `LL_RA` → occur when reaccessing memory

Which performance counters (3)

Detecting cache attacks and Rowhammer without false positives

Which performance counters (3)

Detecting cache attacks and Rowhammer without false positives

- heavy activity on the cache

Which performance counters (3)

Detecting cache attacks and Rowhammer without false positives

- heavy activity on the cache
- very short loops of code $\rightarrow$ low pressure on the iTLB

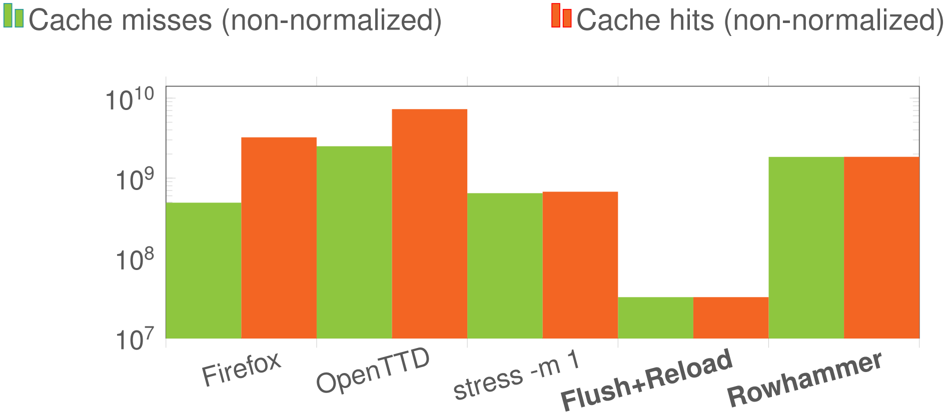
Which performance counters (3)

Detecting cache attacks and Rowhammer without false positives

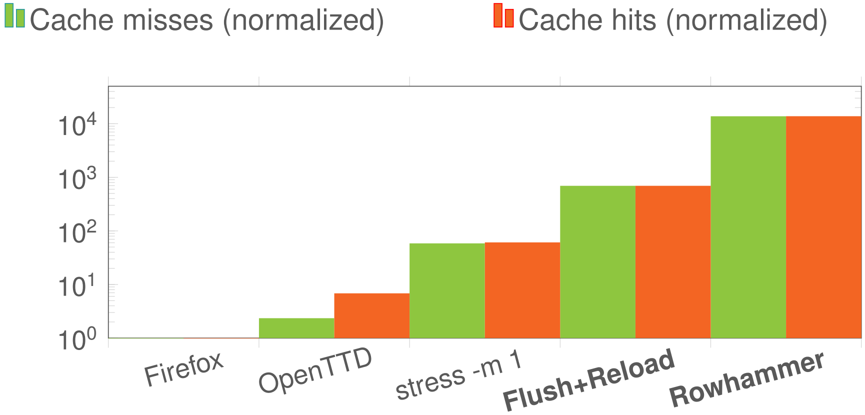
- heavy activity on the cache
- very short loops of code → low pressure on the iTLB

→ normalize the events by `ITLB_RA+ITLB_RM`

Detection, non-normalized



Detection, normalized



Stealthier cache attack: Flush+Flush

- motivation: detecting cache attacks with perf counters is not enough

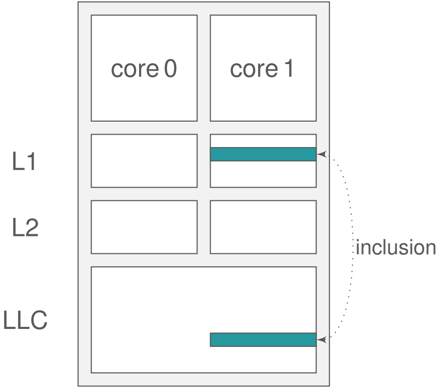
Stealthier cache attack: Flush+Flush

- motivation: detecting cache attacks with perf counters is not enough

→ Flush+Flush: new cache attack, based on `clflush` timing leakage

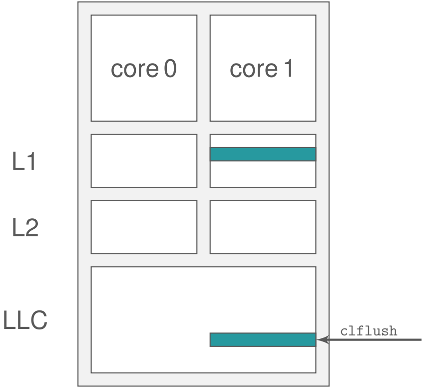
  → stealthier than Prime+Probe and Flush+Reload
  → faster than Prime+Probe and Flush+Reload

`clflush` timing leakage (1)



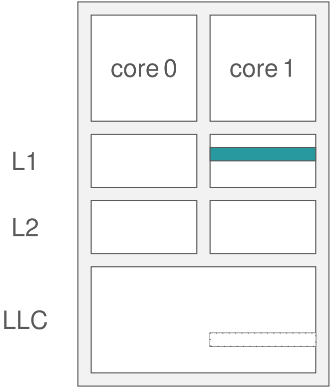
- `clflush` on cached data

`clflush` timing leakage (1)



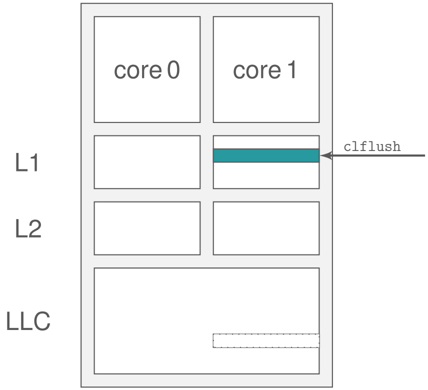
- `clflush` on cached data
  - goes to LLC, flushes line

`clflush` timing leakage (1)



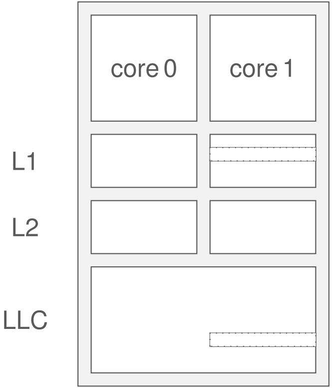
- `clflush` on cached data
  - goes to LLC, flushes line

`clflush` timing leakage (1)



- `clflush` on cached data
  - goes to LLC, flushes line
  - flushes line in L1-L2

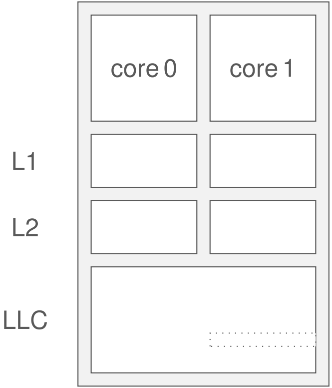
`clflush` timing leakage (1)



- `clflush` on cached data
  - goes to LLC, flushes line
  - flushes line in L1-L2

→ slow

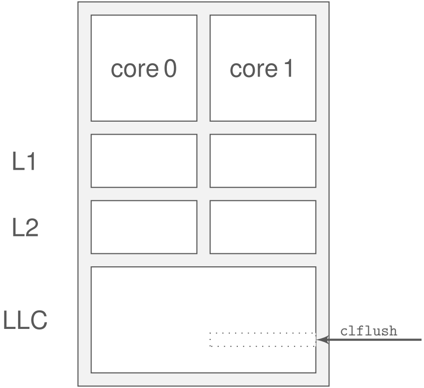
`clflush` timing leakage (1)



- `clflush` on cached data
  - goes to LLC, flushes line
  - flushes line in L1-L2

  → slow
- `clflush` on non-cached data

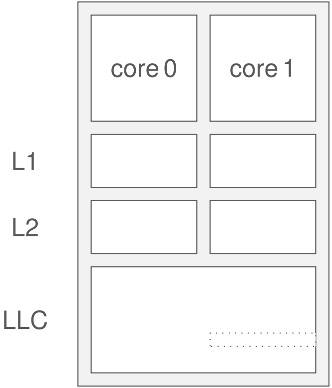
`clflush` timing leakage (1)



- `clflush` on cached data
  - goes to LLC, flushes line
  - flushes line in L1-L2

  → slow
- `clflush` on non-cached data
  - goes to LLC, does nothing

`clflush` timing leakage (1)



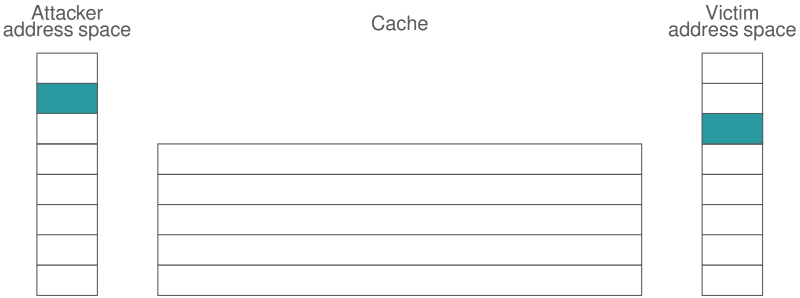
- `clflush` on cached data
  - goes to LLC, flushes line
  - flushes line in L1-L2

  → slow
- `clflush` on non-cached data
  - goes to LLC, does nothing

  → fast

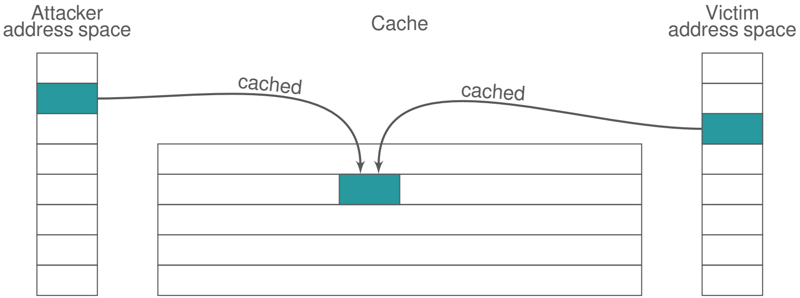
`clflush` timing leakage (2)

Flush+Flush



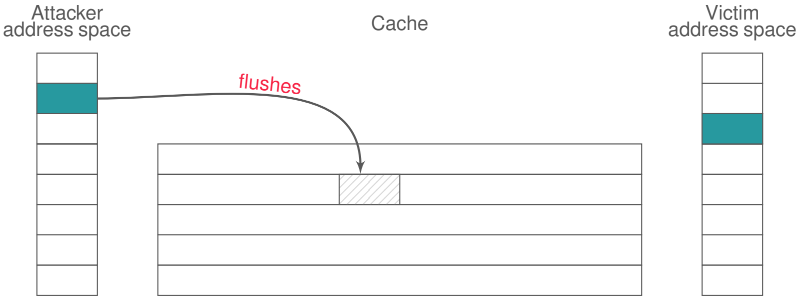
step 0: attacker maps shared library → shared memory, shared in cache

Flush+Flush



step 0: attacker maps shared library → shared memory, shared in cache

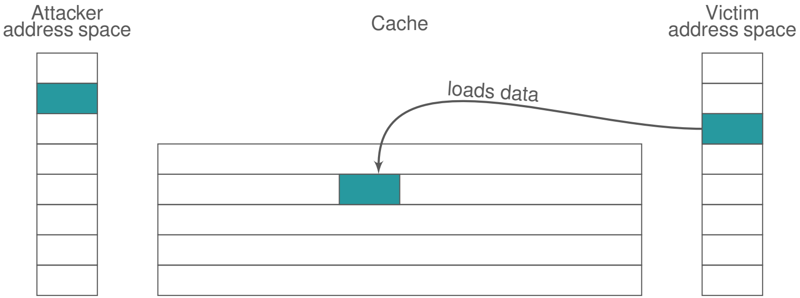
Flush+Flush



step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

Flush+Flush

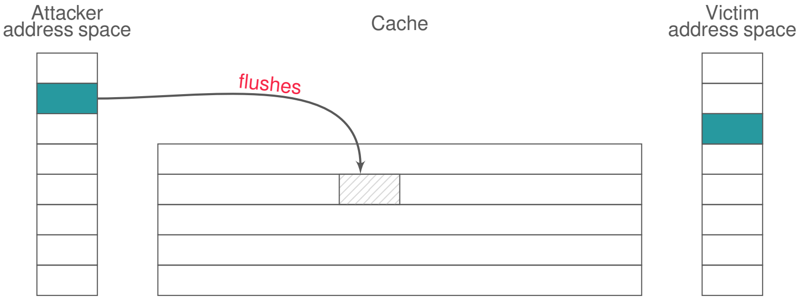


step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

Flush+Flush



step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

step 3: attacker flushes data → high execution time if the victim loaded the line

Flush+Reload vs. Flush+Flush

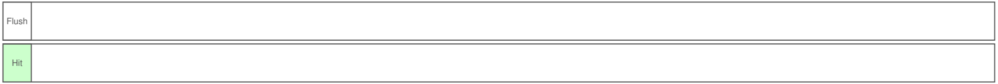
Flush+Reload spy

Flush+Flush spy

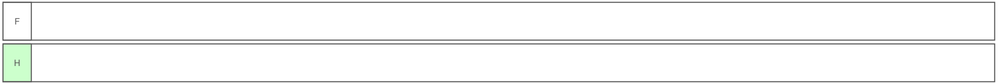
Victim

Flush+Reload vs. Flush+Flush

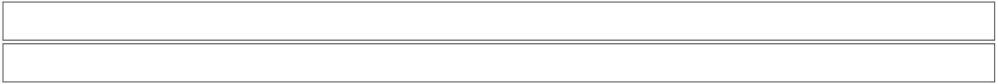
Flush+Reload spy



Flush+Flush spy

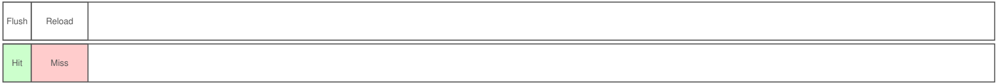


Victim

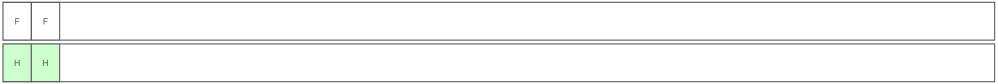


Flush+Reload vs. Flush+Flush

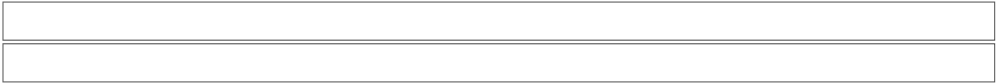
Flush+Reload spy



Flush+Flush spy

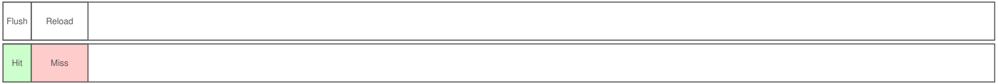


Victim

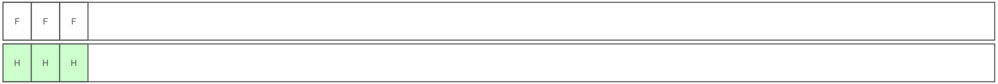


Flush+Reload vs. Flush+Flush

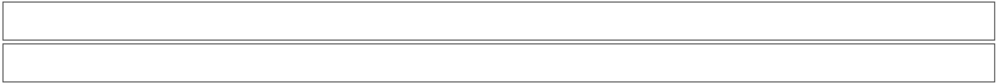
Flush+Reload spy



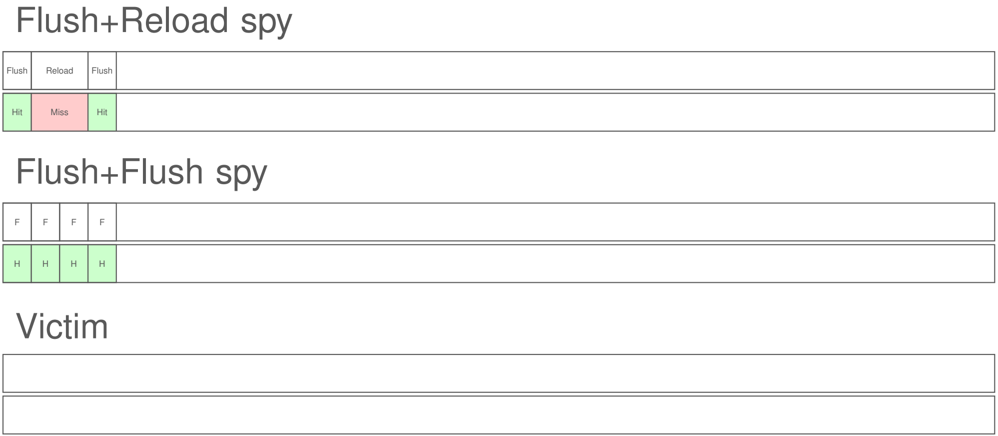
Flush+Flush spy



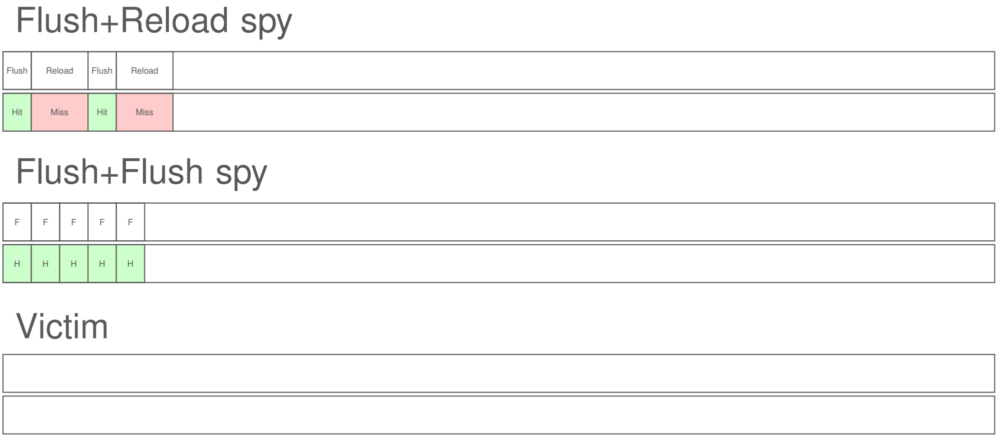
Victim



Flush+Reload vs. Flush+Flush



Flush+Reload vs. Flush+Flush



Flush+Reload vs. Flush+Flush

Flush+Reload spy

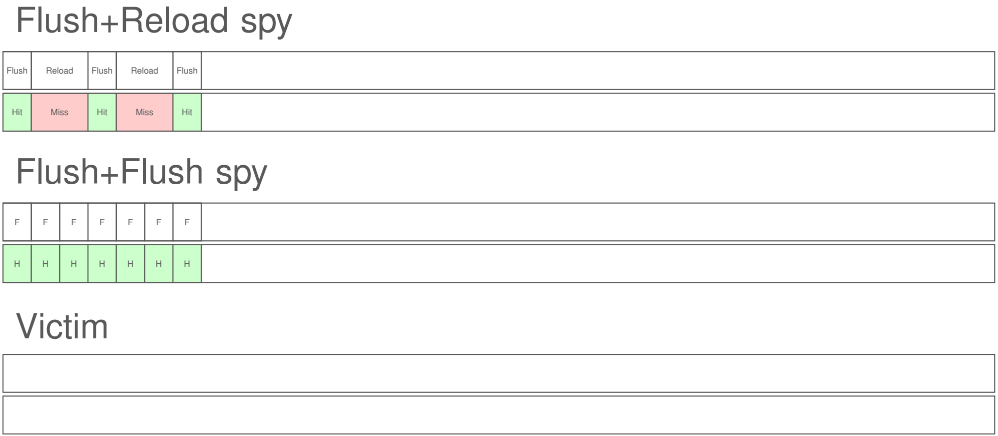
Flush	Reload	Flush	Reload	
Hit	Miss	Hit	Miss	

Flush+Flush spy

F	F	F	F	F	F	
H	H	H	H	H	H	

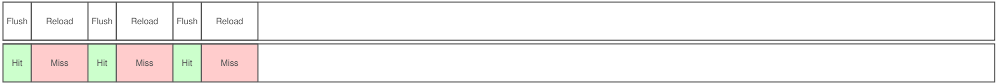
Victim

Flush+Reload vs. Flush+Flush

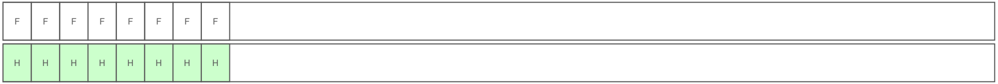


Flush+Reload vs. Flush+Flush

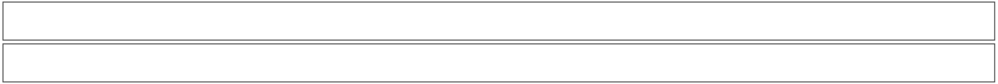
Flush+Reload spy



Flush+Flush spy

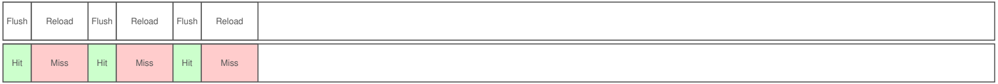


Victim

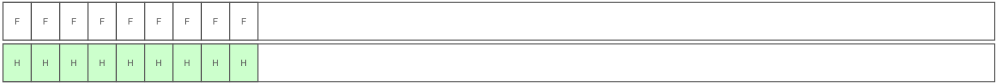


Flush+Reload vs. Flush+Flush

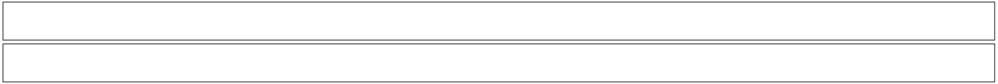
Flush+Reload spy



Flush+Flush spy



Victim



Flush+Reload vs. Flush+Flush

Flush+Reload spy

Flush	Reload	Flush	Reload	Flush	Reload	Flush	
Hit	Miss	Hit	Miss	Hit	Miss	Hit	

Flush+Flush spy

F	F	F	F	F	F	F	F	F	F	
H	H	H	H	H	H	H	H	H	H	

Victim

Flush+Reload vs. Flush+Flush

Flush+Reload spy

Flush	Reload	Flush	Reload	Flush	Reload	Flush	Reload	
Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	

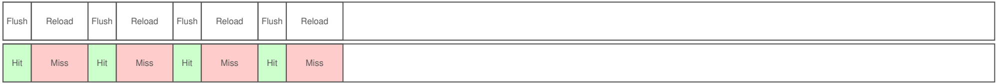
Flush+Flush spy

F	F	F	F	F	F	F	F	F	F	F	
H	H	H	H	H	H	H	H	H	H	H	

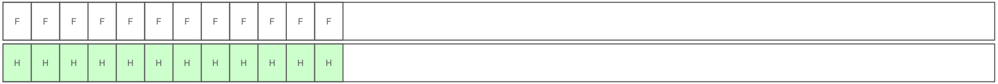
Victim

Flush+Reload vs. Flush+Flush

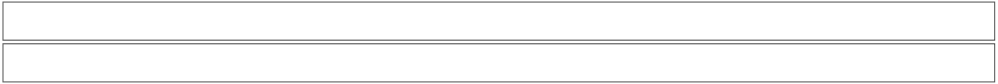
Flush+Reload spy



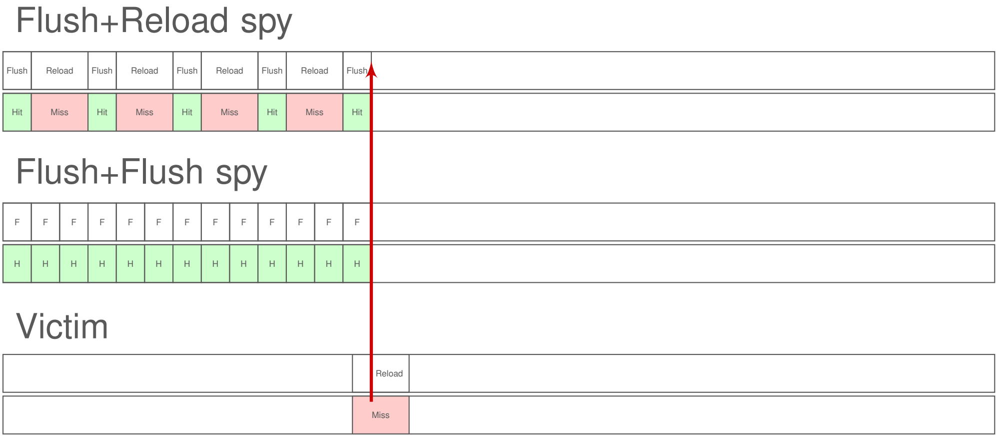
Flush+Flush spy



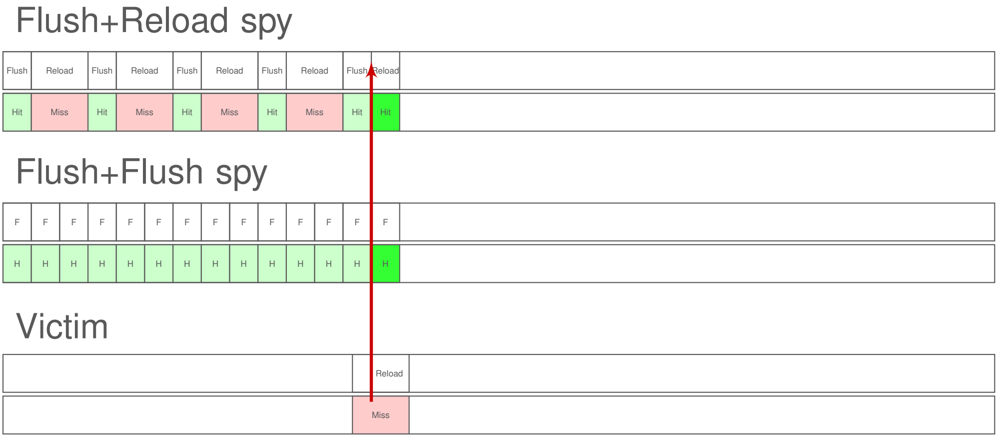
Victim



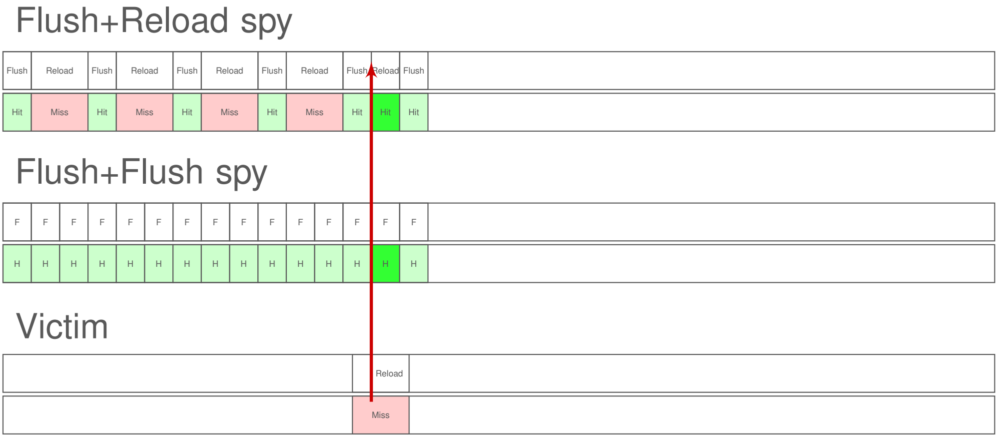
Flush+Reload vs. Flush+Flush



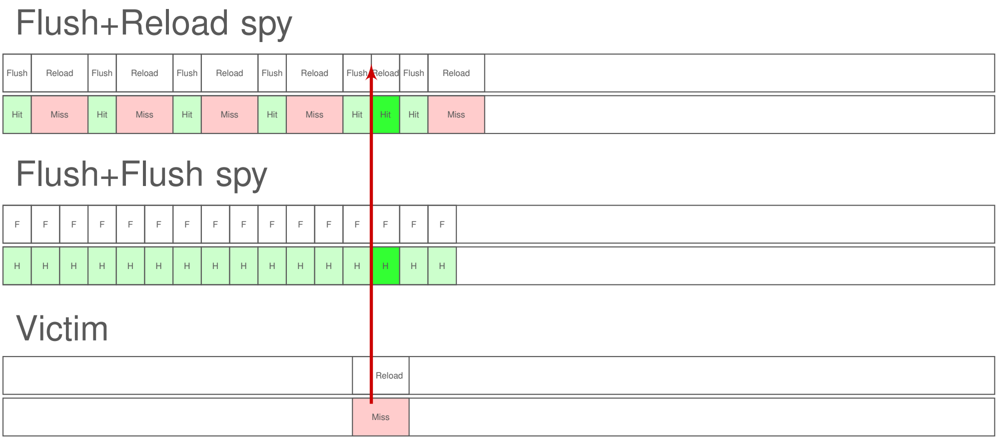
Flush+Reload vs. Flush+Flush



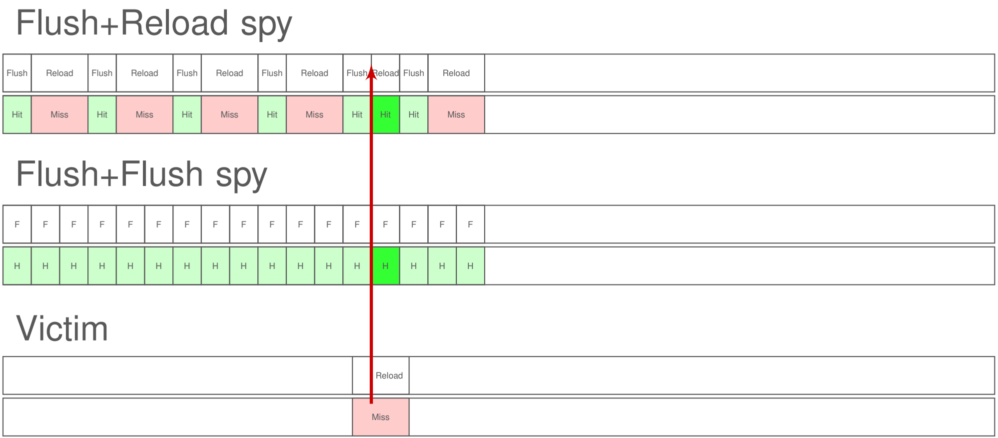
Flush+Reload vs. Flush+Flush



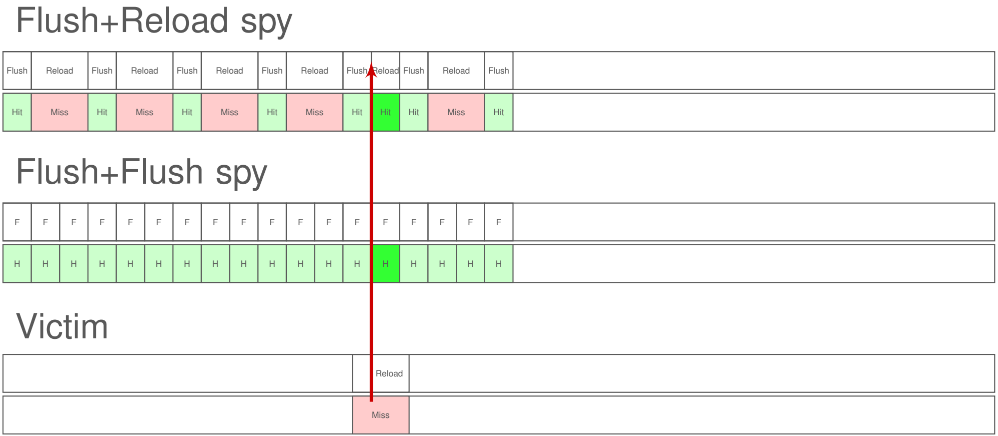
Flush+Reload vs. Flush+Flush



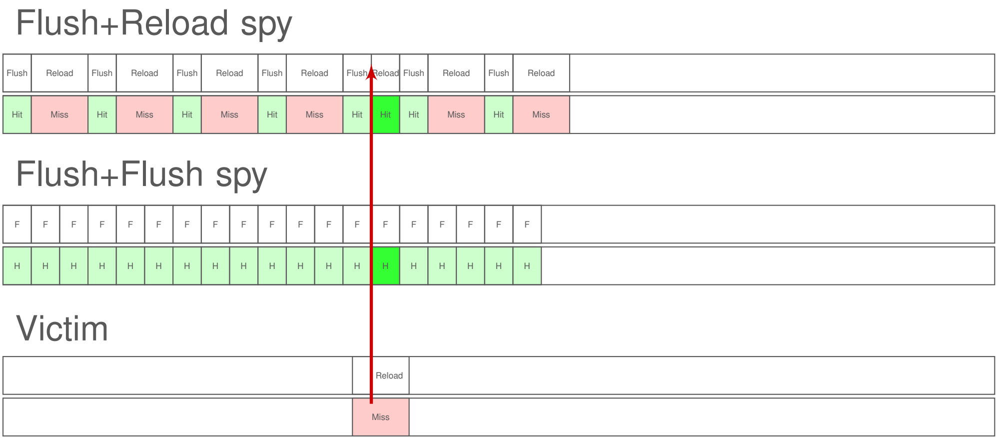
Flush+Reload vs. Flush+Flush



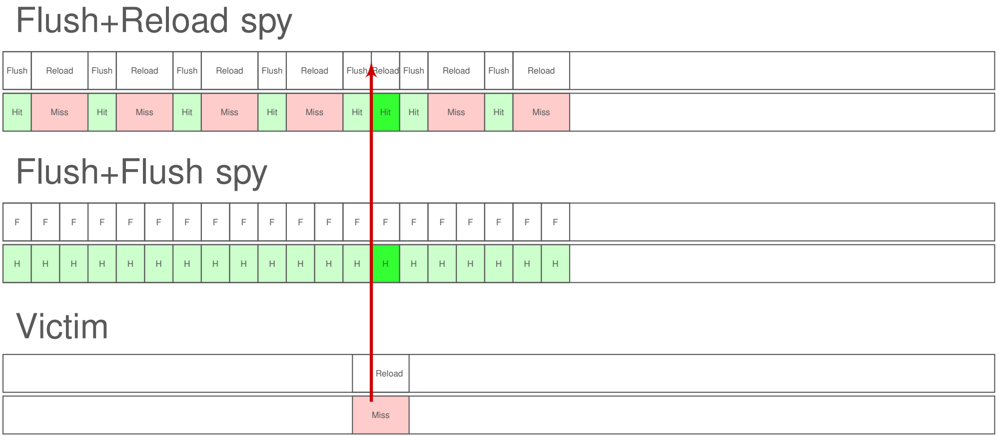
Flush+Reload vs. Flush+Flush



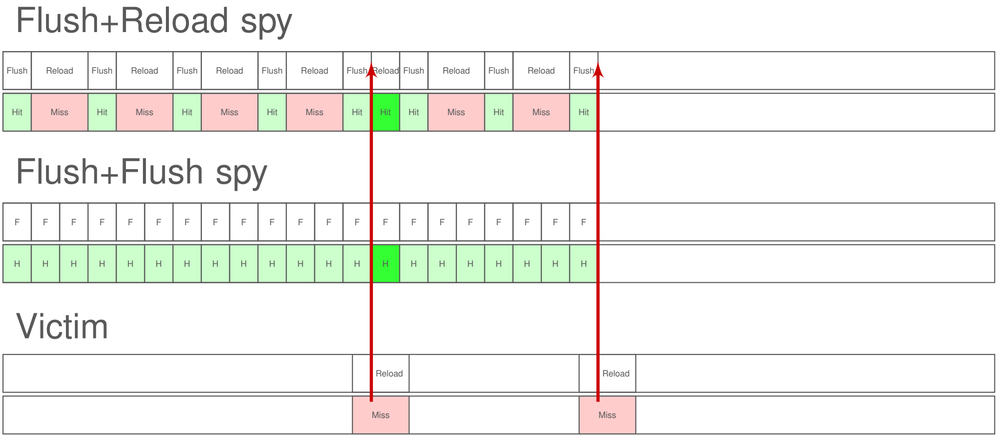
Flush+Reload vs. Flush+Flush



Flush+Reload vs. Flush+Flush



Flush+Reload vs. Flush+Flush



Clémentine Maurice, Graz University of Technology

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Clémentine Maurice, Graz University of Technology

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Reload vs. Flush+Flush

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28			
Flush+Reload	28			

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496		
Flush+Reload	28	298		

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	
Flush+Reload	28	298	✗	

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗
Flush+Reload	4			
Flush+Flush	4			
Prime+Probe	4			

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗
Flush+Reload	4	54		
Flush+Flush	4	52		
Prime+Probe	4	34		

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗
Flush+Reload	4	54	✗	
Flush+Flush	4	52	✓	
Prime+Probe	4	34	✗	

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗
Flush+Reload	4	54	✗	✓
Flush+Flush	4	52	✓	✓
Prime+Probe	4	34	✗	✗

Flush+Flush: Covert channel

technique	packet size	capacity (KBps)	receiver stealth	sender stealth
Flush+Flush	28	496	✓	✗
Flush+Reload	28	298	✗	✗
Flush+Reload	4	54	✗	✓
Flush+Flush	4	52	✓	✓
Prime+Probe	4	34	✗	✗

Flush+Flush: Side channel on AES T-tables (1)

Number of encryptions to determine the upper 4 bits of a key byte

technique	number of encryptions
Flush+Reload	250
Flush+Flush	350
Prime+Probe	4 800

→ same performance for Flush+Flush and Flush+Reload

Flush+Flush: Side channel on AES T-tables (2)

Stealth comparison on 256 million encryptions

technique	time (s)	stealth
Flush+Reload	215	✗
Prime+Probe	234	✗
Flush+Flush	163	✓

→ Flush+Flush is the only stealthy spy process
→ others need to be slowed down too much to be practical

Countermeasures

- `clflush`
  - unprivileged line eviction

Countermeasures

- `clflush`
  - unprivileged line eviction → make it privileged

Countermeasures

- `clflush`
    - unprivileged line eviction $\rightarrow$ make it privileged
    - leaks timing information

Countermeasures

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time

Countermeasures

- `clflush`
  - unprivileged line eviction $\rightarrow$ make it privileged
  - leaks timing information $\rightarrow$ make it constant-time
- `rdtsc`
  - unprivileged timing

Countermeasures

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged timing → make it privileged

Countermeasures

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged timing → make it privileged
  - fine-grained timing

Countermeasures

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged timing → make it privileged
  - fine-grained timing → make it coarse-grained

Countermeasures

- `clflush`
  - unprivileged line eviction $\rightarrow$ make it privileged
  - leaks timing information $\rightarrow$ make it constant-time
- `rdtsc`
  - unprivileged timing $\rightarrow$ make it privileged
  - fine-grained timing $\rightarrow$ make it coarse-grained
- disable shared memory

1. Introduction

2. Flush+Flush

3. Conclusion and future work

Conclusion and future work

- `clflush` leaks enough information to enable a side channel
- Flush+Flush is a novel cache attack that makes no memory access

  → faster and stealthier than previous cache attacks

Conclusion and future work

- `clflush` leaks enough information to enable a side channel
- Flush+Flush is a novel cache attack that makes no memory access

  $\rightarrow$ faster and stealthier than previous cache attacks

Future work

- new detection mechanism
- exploit timing difference of slices

Flush+Flush: A Fast and Stealthy Cache Attack

Daniel Gruss, Clémentine Maurice, Klaus Wagner and Stefan Mangard
Graz University of Technology

July 8, 2016 — DIMVA 2016

References I

[BM15] S. Bhattacharya and D. Mukhopadhyay. “Who watches the watchmen?: Utilizing Performance Monitors for Compromising keys of RSA on Intel Platforms”. In: CHES’15 (2015).

[DMSTWSS13] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. “On the feasibility of online malware detection with performance counters”. In: ACM SIGARCH Computer Architecture News 41.3 (2013), pp. 559–570.

[GBK11] D. Gullasch, E. Bangerter, and S. Krenn. “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”. In: S&P’11. 2011.

[GSM15] D. Gruss, R. Spreitzer, and S. Mangard. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: USENIX Security Symposium. 2015.

References II

[HF15] N. Herath and A. Fogh. “These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security”. In: Black Hat 2015 Briefings. 2015. URL: https://www.blackhat.com/docs/us-15/materials/us-15-Herath-These-Are-Not-Your-Grand-Daddys-CPU-Performance-Counters-CPU-Hardware-Performance-Counters-For-Security.pdf.

[LYGHL15] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. “Last-Level Cache Side-Channel Attacks are Practical”. In: S&P’15. 2015.

[MLSNHF15] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID. 2015.

[MNHF15] C. Maurice, C. Neumann, O. Heen, and A. Francillon. “C5: Cross-Cores Cache Covert Channel”. In: DIMVA’15. 2015.

[OKSK15] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: CCS’15. 2015.

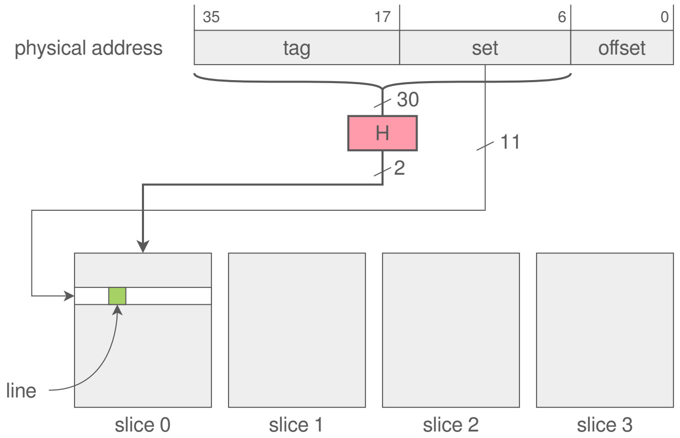
References III

[Pay16] M. Payer. "HexPADS: a platform to detect "stealth" attacks". In: ESSoS'16. 2016.

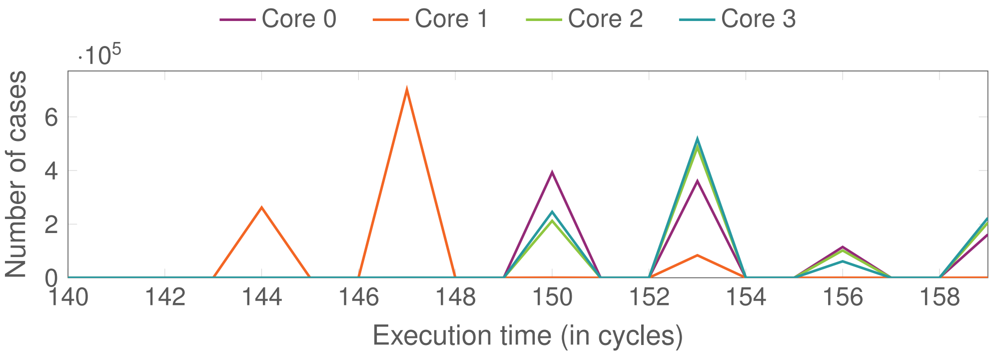
[Per05] C. Percival. "Cache missing for fun and profit". In: Proceedings of BSDCan. 2005.

[YF14] Y. Yarom and K. Falkner. "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: USENIX Security Symposium. 2014.

Last-level cache addressing



Bonus: Even more timing leakage



Performance counters: Usages

- answers “what’s happening in the system?”
- performance tuning $\rightarrow$ find bottlenecks
- side-channels [BM15]
- reverse-engineering microarchitecture [MLSNHF15]
- attack detection [DMSTWSS13; HF15; Pay16]