ETH zürich

Spectre Attacks: Exploiting Speculative Execution

Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin,Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp,Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom

Presented at: 2019 IEEE Symposium on Security and Privacy (SP)

Presentation By Pascal Brunner

ETH zürich

Executive Summary

- **Problem:**
  - Speculative execution can be exploited to leak secret information of other processes
  - Performance was primary focus of processor development over the last couple of decades, while neglecting security implications
    - e.g., Branch prediction & speculative execution
- **Goal:**
  - Abuse branch prediction and speculative execution and use side channel attacks to collect confidential information.
- **Novelty & Key Approach:**
  - First use of speculative execution and branch prediction to leak secret information on modern high performance processors
- **Results:**
  - Numerous real proof of concept implementations (C code and JavaScript)
  - Few possible countermeasures:
    - Some fixable by micro-architecture updates
    - Others need hardware changes or even ISA updates

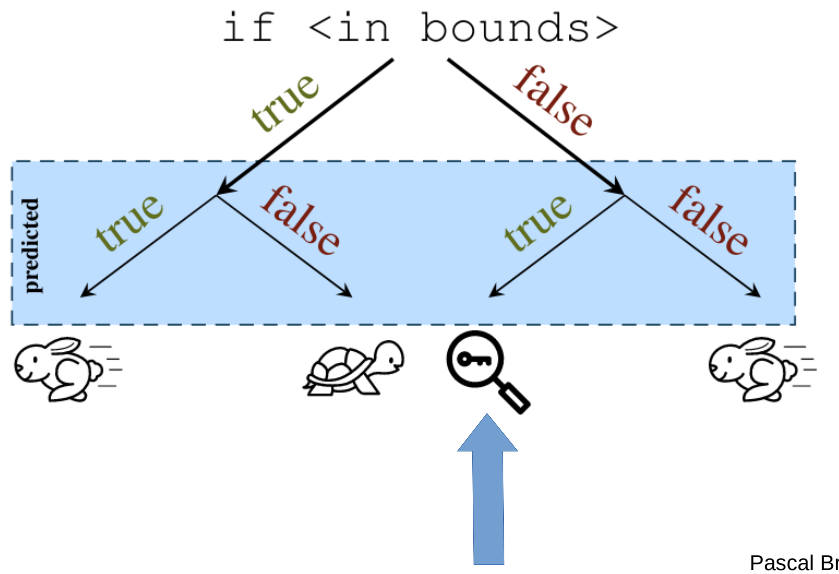
Outline

Background – Out of Order Execution

- Prevent waste of CPU cycles and increase processor utilization
- Order of execution is different from the instruction order in the code
- Micro-ops used to implement ISA
  - Commit changes in program order using reorder buffer
  - Micro-ops are retired when all micro-ops of an instruction and all previous instructions are completed

Background – Speculative Execution

- On branch prediction result
- Run code in predicted branch:
  - **Correct prediction**: leads to significant speedup
  - **Wrong prediction**: throw away changes and execute correct branch, same performance as stalling
- Miss-predictions are not side-effect free



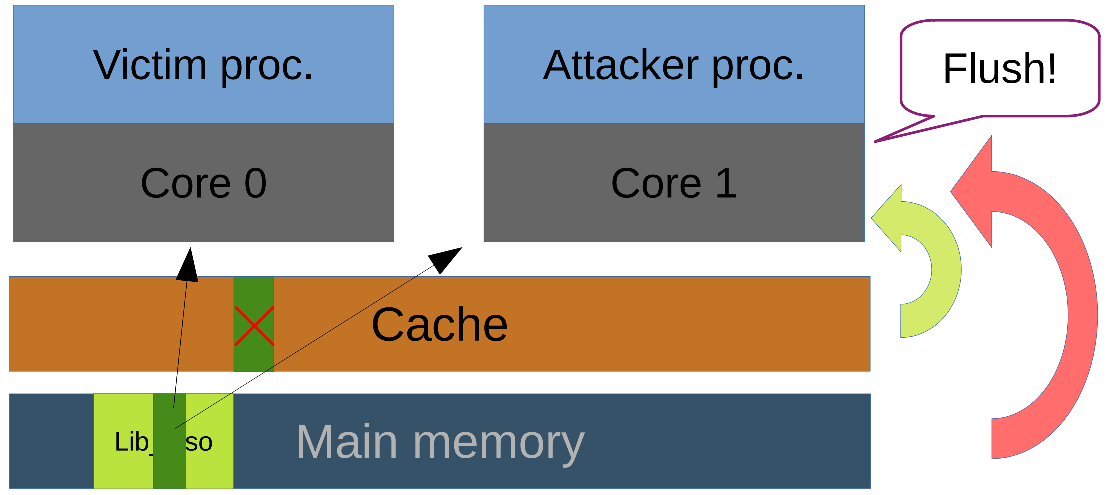
Background – Branch Prediction

- Predictions are made about branching instructions
- More correct outcome predictions lead to improved performance
- Multiple prediction mechanisms used (for different branch types)
  - Direct branches
  - Indirect branches
    - Branch Target Buffer (BTB)
    - Return Stack Buffer (RSB)

ETH zürich

Background – Micro-architectural Side-Channel Attacks

- Use side effects of using the same hardware
- Many types and variants
  - Timing based
  - Micro-architectural state changes based
    - Instruction cache
    - L1 and lower level caches
    - Branch history
- Flush + Reload
  - Evict + Reload



Outline

Novelty & Key Idea

- First time to show the use of speculative execution for ex-filtration of sensitive/secret data of another process
- “Violate memory isolation boundaries by combining speculative execution with data exfiltration via micro-architectural covert channels.”

Implementation Details

- **Variant** 1 – Exploitation of Conditional Branches
- **Variant** 2 – Exploitation of Indirect Branches
- Indicating further possible variants by variations in the method used for speculative execution and the covert channel method

Attack

1) Mistrain branch prediction & setup side channel
2) Enforce speculative execution, transferring secret data to the side-channel
3) Use side-channel to recover secret data

Implementation Details – Variant I (Conditional Branches)

- Code similar to Listing 1 found in victim (e.g., system call or library)
  - *x* comes from untrusted source (e.g., Input)

```
if (x < array1_size)
   y = array2[array1[x] * 4096];
```

Listing 1: Conditional Branch Example

- If *array1_size* is not cached, the processor will speculatively execute the code inside the if-branch
- The access to *array1[x]* can now be out of bounds, to a secret value
  - To do this set: x = (address of secret byte to load) – (base address of array1)
- The value of the secret byte can now be determined, by detecting which element of *array2* was accessed

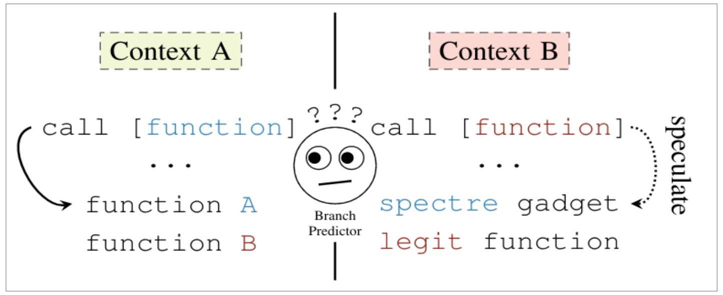
Implementation Details – Variant I (Conditional Branches)

1) Train the branch predictor by running the above code on many valid inputs for *x*
2) Choose *x* maliciously s.t. *array[x]* = *k* is secret information k
   → x = (address of secret byte to load) – (base address of array1)
3) Make sure *array1_size* and *array2* are not cached
   → Flush the elements from cache
4) Run code with malicious x to cause speculative execution of *if*-branch
   → array2[k * 4096] will be cached
5) Measure which location was brought into cache
   → Flush + Reload

Implementation Details – Variant II (Indirect Branches)

- **Gadget:** small snippet of code that can be called
- Simple example with 2 registers *(R1,R2)* input and 2 instructions
  1) **ALU operation** between two registers (e.g., XOR R1 R2)
  2) **Access memory** at register location **R2**
  - R1 provides control over the address to leak
  - R2 control over how mapped memory maps to address

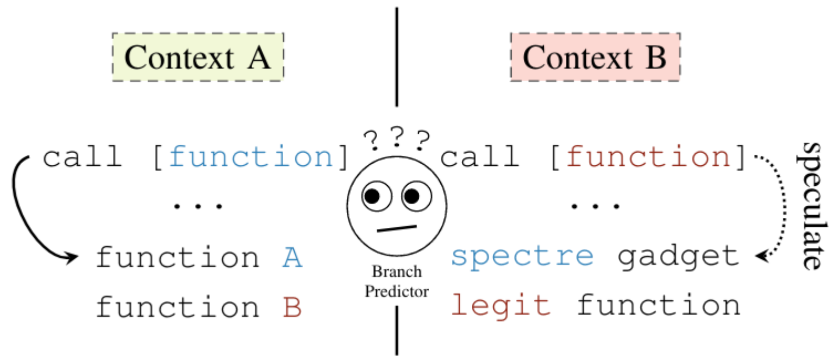
- How to mistrain the branch predictor
  - Learn how branch predictor gets updated
    - Authors reverse engineered branch history buffer update
  - Call function at the same location (in another context) to function at the same location as the gadget continuously



Implementation Details – Variant II (Indirect Branches)

<u>Attack:</u>

1) Find gadget and calculate arguments for desired secret
2) Mistrain the branch predictor on a gadget location
3) Prepare side-channel
4) Invoke function with predetermined arguments
5) Ex-filtrate data over side channel



- Similarity to return oriented programming (ROP) but without need for correct termination
- Easy to find gadgets, especially with mapped shared libraries

ETH zürich

Outline

Results & Evaluation

- Multiple micro-architectures
  - x86
  - ARM
- Multiple environments
  - Google Chrome
  - User-space
  - Virtual machine
- Multiple Platforms
  - Intel Ivy Bridge – Kaby Lake
  - AMD Ryzen
- Multiple OSes
  - Linux
  - Windows

Version & Lang.	Bandwidth	Error Rate	unreadible/ wrong Data
V1: C impl. [1]	~ 10 kB/s	< 0.01%	-
V1: JavaScript [2]	-	-	-
V1: eBPF (Linux Kernel) [1][3]	2-5 kB/s	-	-
V2: C impl. [1]	41 B/s	~ 2%	-
V2: KVM [4]	1809 B/s	-	1.7%

[1] Intel i7-1650U (Haswell)
[2] Google Chrome v62.0.3202
[3] AMD Pro A8-9600 R7
[4] Intel Xeon Haswell E5-1650 v3

Mitigation

- Prevention of speculative execution
  - Modes to deactivate speculative execution (only on future processors)
  - Serialization and blocking instructions in software
  - Unlikely to provide immediate fix
  - New hardware might need to be designed
  - Would alleviate the problem
- Prevention of access to secret data
  - ex. Google Chrome uses one process per website
  - Not very useful, when program runtime environment can't restrict program access
  - Most useful for JIT compiler, interpreters and other language based protections

Mitigation

- Prevent data from entering covert channels
  - Track data and prevent use in subsequent operations (future processors only)
  - Hard to know all covert channels
  - Needs new hardware (current hardware doesn’t have the capability)
  - Allows for speculative execution and security
- Limit data extraction from covert channels
  - ex. Degraded timer resolution in JavaScript
  - Does not guarantee that attacks aren’t possible
  - Current system lack features/capabilities
  - Would alleviate some problems/concerns

Mitigation

- Prevent Branch Poisoning
  - IBRS (Indirect Branch Restricted Speculation) mode
  - STIBP (Single Thread Indirect Branch Prediction)
  - Require OS or BIOS support
  - Performance impact still there
    - Between a few percent (~2-3 %) to factors of 4x
  - Can be done by microcode path

Conclusion

- Transient instructions executed because by speculative execution leave behind information traces
- Side channel attacks allow the extraction of secret information from other processes.
- Multiple variants of the attack exist and even more will be exploitable in the future.
- Caused by the continued focus on performance and the neglect of security details in processor design.
- Mitigation is difficult and certain measures can only be applied to future processors or instruction set architectures
- Proof of concepts show the real world applicability of this paper.

Outline

- Background
- Novelty & Key Idea
- Implementation Details
- Results & Evaluation
- Conclusion
- Takeaways
- Strengths & Weaknesses
- Further & Related Works
- Discussion

Takeaways

- We need to find a trade-off between performance and security desires, especially for certain applications
- Most systems are vulnerable to at least some Spectre attacks and mitigations aren’t possible at the end-user
- One of the biggest system vulnerabilities in the last decade with huge media coverage

Meltdown and Spectre: ‘worst ever’ CPU bugs affect virtually all computers

Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors – and fix could slow devices

- Spectre and Meltdown processor security flaws – explained

The Guardian, Jan. 2018

Strengths

- High applicability and impact to real world situations
- Code examples and proof of concepts across ISAs and hardware manufacturers
- Generality of attack (vectors)
- Well comprehensible higher level explanations
- Good summary of used concepts
- Many papers and works followed on this foundation
- Proof of concept openly available:
  - Try it yourself: https://gist.github.com/anonymous/99a72c9c1003f8ae0707b4927ec1bd8a

Weaknesses

- Local execution of code required
- Certain details not as well explained
- Sudden jumps between abstraction levels
  - Can be confusing & inhibits reading flow

Further & Related Works

- Lipp, Moritz, et al., "**Meltdown**: Reading kernel memory from user space," in USENIX Security, 2018.
- Van Bulck, Jo, et al. "**Foreshadow**: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution." 27th {USENIX} Security Symposium ({USENIX} Security 18). 2018.
- Hunt, Tyler, et al. "**Ryoan**: A distributed sandbox for untrusted computation on secret data." ACM Transactions on Computer Systems (TOCS) 35.4 (2018): 1-32.
- Chen, Guoxing, et al., "**Sgxpectre attacks**: Leaking enclave secrets via speculative execution," arXiv:1802.09085, 2018.

Q&A

Discussion

- How do you think these vulnerabilities will influence future processor design? More focus on security? Keep focus on performance?

- Do you think there will continue to be future observable side effects discovered? What about after some CPUs have been patched?

- Will we find further flaws, similar to Spectre, caused by constant performance optimizations on our devices in the future? Where?