Spectre Attacks: Exploiting Speculative Execution

IEEE Symposium on Security and Privacy 2019

Paul Kocher¹, Jann Horn², Anders Fogh³, Daniel Genkin⁴, Daniel Gruss⁵, Werner Haas⁶, Mike Hamburg⁷, Moritz Lipp⁵, Stefan Mangard⁵, Thomas Prescher⁶, Michael Schwarz⁵, Yuval Yarom⁸

¹ Independent (www.paulkocher.com), ² Google Project Zero, ³ G DATA Advanced Analytics, ⁴ University of Pennsylvania and University of Maryland, ⁵ Graz University of Technology, ⁶ Cyberus Technology, ⁷ Rambus, Cryptography Research Division, ⁸ University of Adelaide and Data61
What Is Spectre?

Spectre exploits traces left in covert channels (e.g., cache) by speculative execution to leak sensitive data. Vulnerability to such attacks is widespread and hard to fix.
Background
Out-of-Order Execution

`mov (%ebx), %edx`  
`add %edx, %eax`  
`imul %esi, %esi`  
`imul %edi, %edi`  
`add %esi, %edi`

Note: assume that all registers have a known initial value
Out-of-Order Execution

\[
\begin{align*}
\text{mov}\ (\%ebx), \ %edx \\
\text{imul}\ %esi, \ %esi \\
\text{imul}\ %edi, \ %edi \\
\text{add}\ %esi, \ %edi \\
\text{add}\ %edx, \ %eax
\end{align*}
\]

Note: assume that all registers have a known initial value
Speculative Execution

cmp (%eax), %esi
jeq target
imul %edi, %ebx
...
:target
add %(%edx), %ecx
mov %(%ecx), %eax
add %eax, %ecx
...
Speculative Execution

cmp (%eax), %esi
jeq target
imul %edi, %ebx
...
:target
add %(%edx), %ecx
mov %(%ecx), %eax
add %eax, %ecx
...

Instruction pointer

Loading (%eax)).= %esi → take branch

Correct branch → time saved
Speculative Execution

cmp (%eax), %esi
jeq target
imul %edi, %ebx
...
:target
add %(%edx), %ecx
mov %(%ecx), %eax
add %eax, %ecx
...

Instruction pointer
Loading (%eax). ≠ %esi → don’t take branch

Transient instructions
Incorrect branch → rollback
Branch Prediction

cmp (%eax), %esi
jeq target
imul %edi, %ebx
...
:target
add %(%edx), %ecx
...
Cache (Flush+Reload / Evict+Reload)

That was quick. The victim has probably used X.
Cache (Flush+Reload / Evict+Reload)

Memory
- A
- B
- C
- D
- W
- X
- Y
- Z

Cache
- Y
- X

Victim process
- use Y

Attacker process
- That took a while.
  The victim probably has not used X.
Performing Spectre Attacks
Exploiting Conditional Branch Misprediction

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

• The attacker wants to determine the secret byte array1[x] for some out of bounds x
• The attacker is permitted to access array1 and array2
• array1[x] should be cached, but not array1_size and array2
• The attacker should have trained the processor to take the next branch
• Using Flush+Reload, the attacker can determine the loaded entry from array2 and deduce array1[x]
Example Implementation in C

• A huge variety of processors are vulnerable to the exploitation of conditional branch misprediction

• This method was implemented in C

• Data leakage rate is in the order of 10kB/s

• Very low error rate (<0.01%)
Poisoning Indirect Branches

• Find a snippet of code in the victim that can be maliciously used to leak data (called a Spectre gadget)

• Requirements to set up the attack:
  • The attacker controls registers R1 and R2 and knows their initial values
  • R1 contains the address of the desired secret byte
  • The Spectre gadget adds (or other arithmetic) the memory at address R1 to R2 and then loads from memory at address R2
  • The attacker has trained the branch predictor to the gadget’s address

• Using Evict+Reload, the attacker can deduce the secret data
Other Varieties of Spectre Attacks
Instruction Timing

if (false but mispredicts as true)
    multiply R1, R2  // R1 and R2 are secret
multiply R3, R4   // R3 and R4 are public

• Advantages: Independent from the cache

• Disadvantages: May not provide concrete value of secret data
Contention on the Register File

\[
\begin{align*}
\text{if (false but mispredicts as true)} \\
\quad \text{if (condition on R1)} & \quad // \text{R1 is secret} \\
\quad \text{if(condition)} \\
\end{align*}
\]

• If the secret data meets the condition in the second if statement, three checkpoints of registers will be kept for speculative execution rather than only two; the attacker may then notice slowdown

• Advantage: Independent from the cache

• Disadvantage: Some processors may not allow multiple layers of speculative execution
Exploiting Interrupt Returns

• Poison the return from an interrupt instead of an indirect branch in the victim’s code

• Initiate attack by causing an interrupt

• Advantage: Less dependent on victim’s code

• Disadvantage: Not applicable to Intel CPUs
Arbitrary Observable Effects

if (x < array1_size) // Mispredict as true
    y = array1[x];
    // Do something using y that leaves observable side effects

• Advantages: Less picky with required code snippets, not dependent on any single microarchitectural element

• Disadvantage: May require more specialized code on the attacker’s side
Preventing Spectre Attacks
Disabling Speculative Execution

• Disable speculative execution entirely

• Protect specific branches with lfence instructions

• Advantages: Easy to implement, very effective

• Disadvantage: Reduces performance
Separate Processes (Web Browsers)

• Run websites in separate processes

• Used by Chrome

• Advantages: Can be implemented on current hardware, reasonably effective

• Disadvantage: Requires a lot of resources
Index Masking

if (x < array1_size)
    y = array2[array1[x & 0xff] * 4096];

• AND array indexes with a mask
• With a proper mask, resulting indexes may be no larger than 2 times
  the array length
• Advantage: Very cheap
• Disadvantages: Doesn’t protect from all types of Spectre attacks, data
  right behind the array can still be loaded
Preventing Sensitive Data From Entering Covert Channels

• While speculatively executing, do not allow operations on sensitive data that leave detectable traces, e.g., usage as a memory address.

• Advantages: Effective, good trade-off between performance and security.

• Disadvantage: Not supported by current processors.
Degrading the Clock

• Add jitter or reduce resolution

• Sometimes done in JavaScript

• Advantages: Very cheap, slows potential attacks

• Disadvantages: Can be detrimental to benign processes, can be weakened by repeated use of Evict+Reload
Preventing Branch Poisoning

• Prevent effects of branches from less privileged processes

• Added to ISA by Intel and AMD

• Advantage: Implemented on current processors

• Disadvantage: Ineffective against other variants of Spectre attacks
Conclusion
Conclusion

Spectre attacks can exploit traces of incorrectly speculatively executed instruction to deduce secret data on various CPUs. Due to the variety of performance improving components of modern processors, Spectre vulnerabilities are nearly impossible to fix.
Strengths
Strengths

• Widespread vulnerability

• Variety of possible covert channels

• Hard to fully prevent on modern hardware, especially without greatly reducing performance

• Simplicity
Weaknesses
Weaknesses

• Needs to know the location of secret data and exploitable pieces of code

• Can struggle with complex networks of processes

• Limited by its permissions
Alternatives and Possible Improvements
Alternatives and Improvements

• Give processes with sensitive data an entire CPU core

• Do not modify the cache during speculative execution

• Flag branches where branch mispredictions occurred at least n times
Discussion
Possible Discussion Starters

• How to detect Spectre attacks

• What data would you want to extract with / protect from Spectre

• Any other suggestions for improvements in attack / defense
Discussion: How to Detect Spectre Attacks

That process is acting sus
Discussion: What Sensitive Data to Extract / Protect
Discussion: Ideas for Attack or Defense