SPECTRE ATTACKS: EXPLOITING SPECULATIVE EXECUTION

PAPER BY KOCHER ET AL.

PRESENTATION BY MATTHEW COPELAND

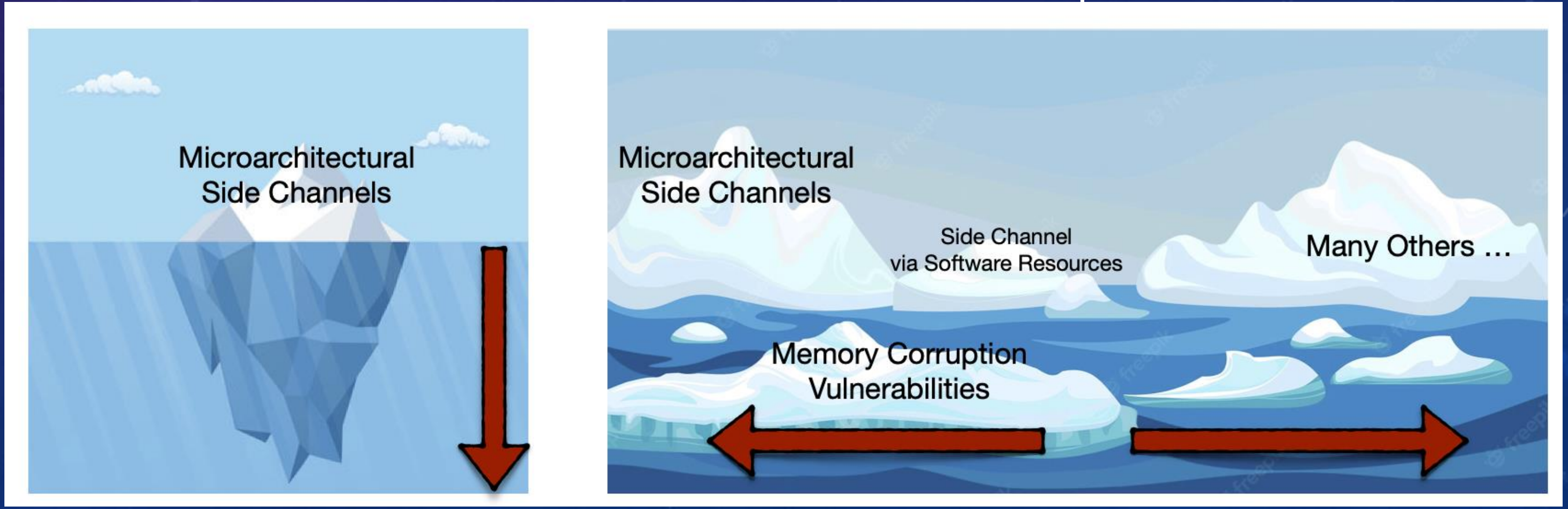
SPECTRE BACKGROUND

- Optimizing modern processors proves increasingly difficult as developers require complex mechanisms to achieve speed-ups
- Microarchitectural advancements are required to achieve these speed-ups but may enable microarchitectural side-channel attacks
- Spectre induces a victim to speculatively perform operations that would not occur during slower serialized instruction processing
    - Setup: Spectre attacks mistrain the processor to make an erroneous speculative prediction and prepares the covert channel for data extraction
    - Execution: The processor runs the instructions speculatively which transfers information into a microarchitectural channel
    - Recovery: The sensitive data is recovered by timing cache access

MICROARCHITECTURAL SIDE-CHANNEL ATTACKS

- Microarchitectural components improve processor performance by predicting future program behavior
  - Assumes future behavior is related to past behavior
- Programs executing on the same hardware changes microarchitectural states which may affect the other programs
  - May result in unintended information leaks between programs
  - Can be seen if attacker can monitor cache usage
- Flush+Reload or Evict+Reload used in this paper to leak information
  - Flush uses machine instructions to clear the cache
  - Evict uses forced contention by accessing other memory that are then loaded into the cache

Microarchitectural attack landscape


Microarchitectural Side Channels
Microarchitectural Side Channels
Side Channel via Software Resources
Many Others …
Memory Corruption Vulnerabilities

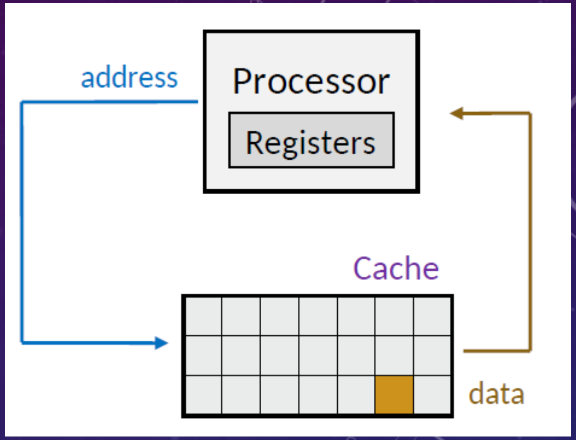
https://www.sigarch.org/looking-beyond-microarchitectural-side-channels/

MICROARCHITECTURAL COMPONENTS

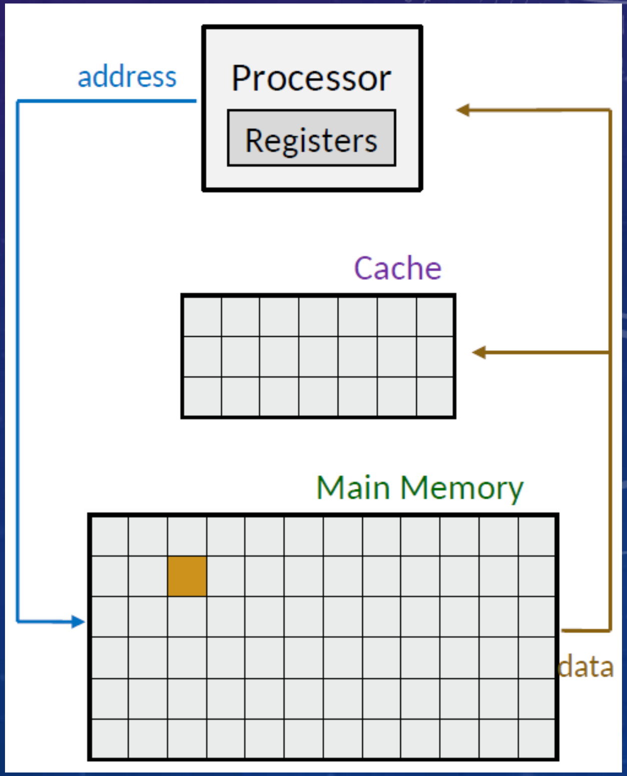
MEMORY HIERARCHY

- Memory access latency limits the processor's throughput
- Memory Hierarchy – Keeping relevant data in small fast-access caches reduces latency when data is in cache
    - Cache Hit – Data was previously loaded into low-latency cache and can be fetched with in minimal clock cycles
    - Cache Miss - Data needs to be fetched from high-latency memory, but takes order of magnitude more clock cycles
- Attackers can determine if a memory access is a cache hit or miss by timing the memory read operation since the hit and miss times differ so much

Cache Hit



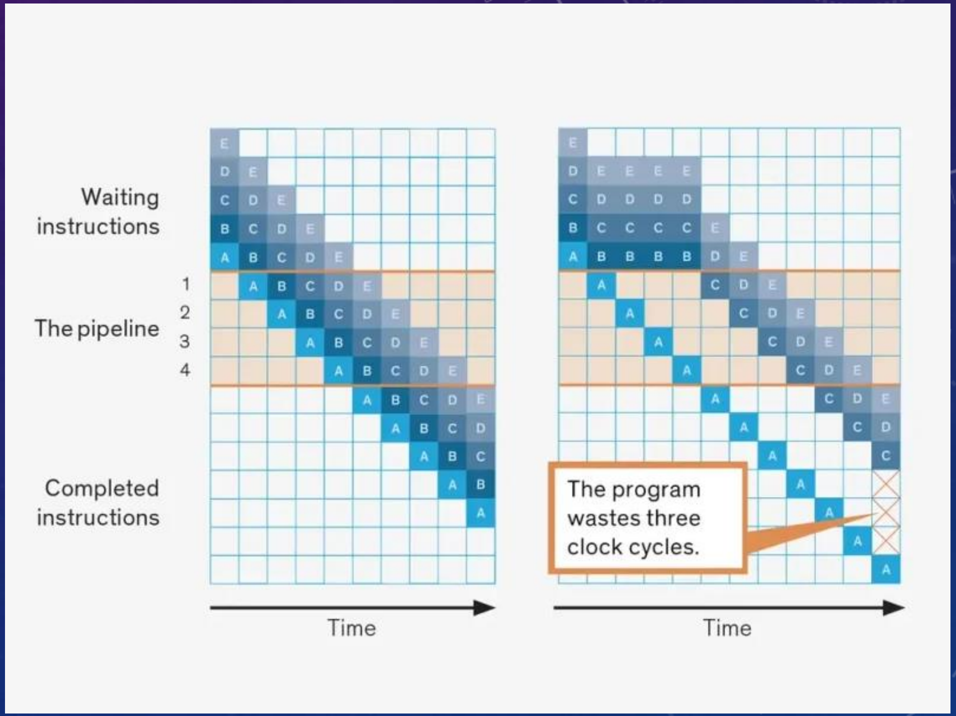
Cache Miss



Cache attacks: Jens-Peter Kaps ECE 699

OUT-OF-ORDER EXECUTION

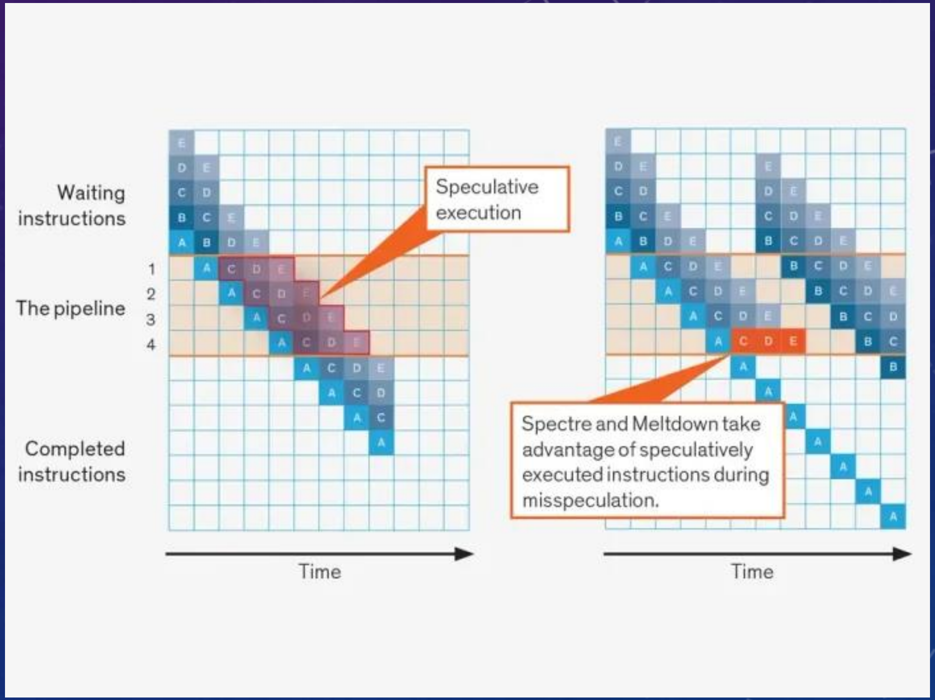
- How can CPU designers ensure the cache contains memory that is needed for the upcoming instructions?
  - Optimized cache replacement policy – Ensure the data we need remains in cache
  - Out-of-Order Execution – Continue to execute instructions that occur after the memory access which would normally halt the processor
- Instructions for modern processors are broken into micro-operations
- Many micro-ops can be completed in parallel or before preceding ops but an instruction is only retired (committed to registers and freeing buffer space) after all the previous and current instruction micro-ops have completed



https://spectrum.ieee.org/how-the-spectre-and-meltdown-hacks-really-worked

SPECULATIVE EXECUTION

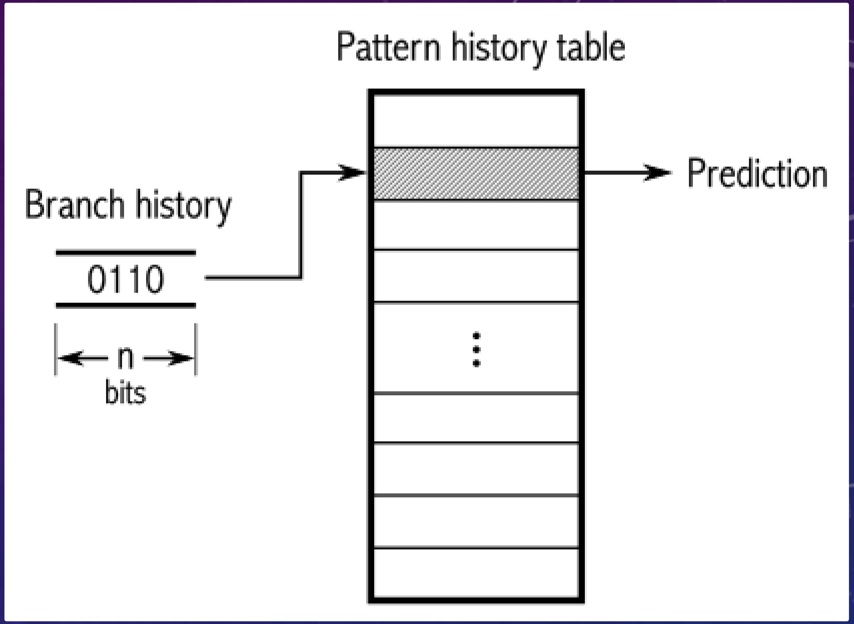
- Out-of-Order execution may lead to states where future instructions are unknown
  - statements based on previous instructions such as conditionals or arbitrary jumps
- Speculative execution allows the processor to make a predication about the outcome of the conditional statement and speculatively execute instructions along that path
  - If the prediction is correct, the new states are committed to memory
  - If the prediction is erroneous, the transient instructions are discarded
- Transient instructions do not have their results saved, but can leave microarchitectural traces which leak information which can be exploited with spectre


Waiting instructions
The pipeline
1
2
3
4
Completed instructions
Speculative execution
Time
Spectre and Meltdown take advantage of speculatively executed instructions during misspeculation.
Time

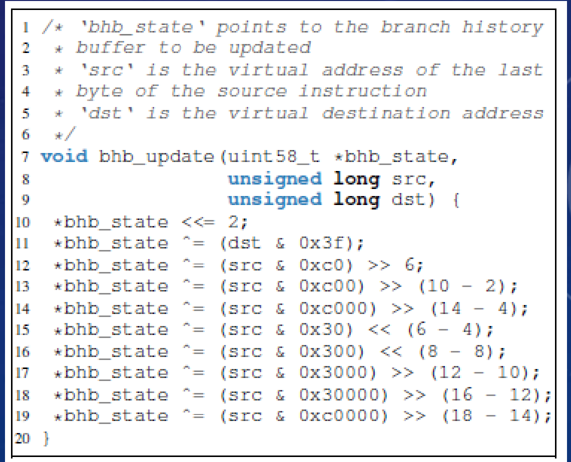
https://spectrum.ieee.org/how-the-spectre-and-meltdown-hacks-really-worked

BRANCH PREDICTION

- Two types of predictions in modern processors
- Indirect Call / Jump Prediction – Predicts based on recent program behavior
    - Indirect calls and jumps – jump to arbitrary addresses computed at runtime (e.g. register, memory, or stack)
- Conditional Branch Prediction - Predicts based on the branch target and whether the branch will be taken
- Branch Target Buffer (BTB) keeps a mapping from recent branch instructions to their destination addresses which can be used to predict future code addresses before decoding instructions
- Conditional branches don't need to map addresses since the condition is calculated at runtime instead of the address. instead records of both outcomes need to be maintained



https://www.cs.cmu.edu/afs/cs/academic/class/15213-f00/docs/mpr-branchpredict.pdf



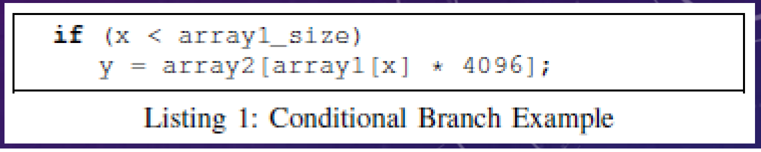
```
/* 'bhb_state' points to the branch history
 * buffer to be updated
 * 'src' is the virtual address of the last
 * byte of the source instruction
 * 'dst' is the virtual destination address
 */
void bhb_update(uint58_t *bhb_state,
                unsigned long src,
                unsigned long dst) {
 *bhb_state <<= 2;
 *bhb_state ^= (dst & 0x3f);
 *bhb_state ^= (src & 0xc0) >> 6;
 *bhb_state ^= (src & 0xc00) >> (10 - 2);
 *bhb_state ^= (src & 0xc000) >> (14 - 4);
 *bhb_state ^= (src & 0x30) << (6 - 4);
 *bhb_state ^= (src & 0x300) << (8 - 8);
 *bhb_state ^= (src & 0x3000) >> (12 - 10);
 *bhb_state ^= (src & 0x30000) >> (16 - 12);
 *bhb_state ^= (src & 0xc0000) >> (18 - 14);
}
```

Listing 4: Pseudocode for updating the branch history buffer state when a branch is encountered.

SPECTRE ATTACK

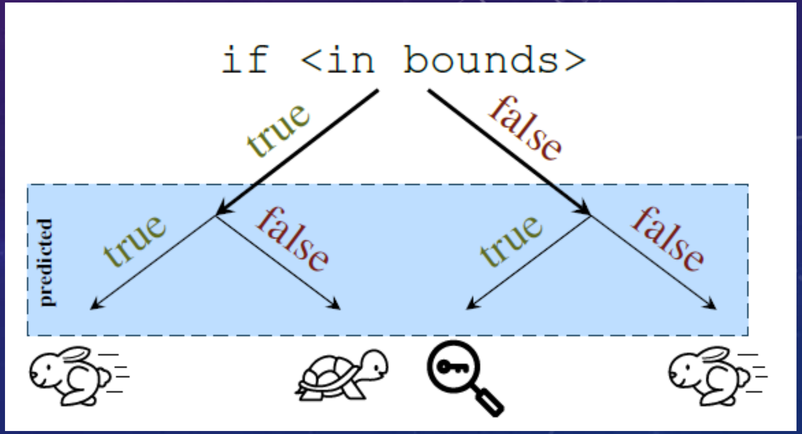
SPECTRE VARIANT 1: CONDITIONAL BRANCHES

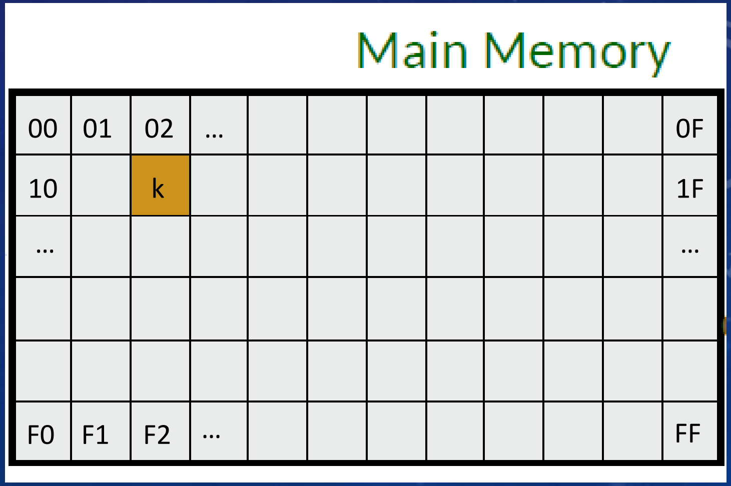
- Bounds check needed to ensure array1 does not access sensitive memory
- Processor may speculate if cache miss, congested processor, complex arithmetic dependencies, or nested speculations
- Implementation – Spectre V1
    - the value of x is maliciously chosen (out-of-bounds), such that array1[x] resolves to a secret byte k somewhere in the victim's memory;
    - array1_size and array2 are uncached, but k is cached
    - previous operations received values of x that were valid, leading the branch predictor to assume the if will likely be true.
    - The secret memory at cached address k is read erroneously by the speculative execution logic
    - Array 2 is read using the secret memory byte to compute the address, but will result in a cache miss
    - The processor catches up and realizes the branch was incorrect then rewinds the register state, but since the array 2 lookup was already in flight it will affect the cache in an address-specific manner where the address depends on k
    - Attacker uses Flush+Reload / Evict+Reload to recover k by finding the cached value of array2



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

Listing 1: Conditional Branch Example



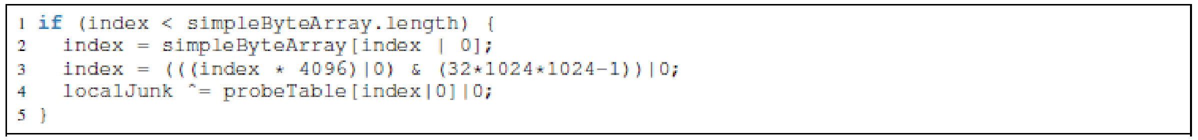


Main Memory

00	01	02	…								0F
10		k									1F
…											…
F0	F1	F2	…								FF

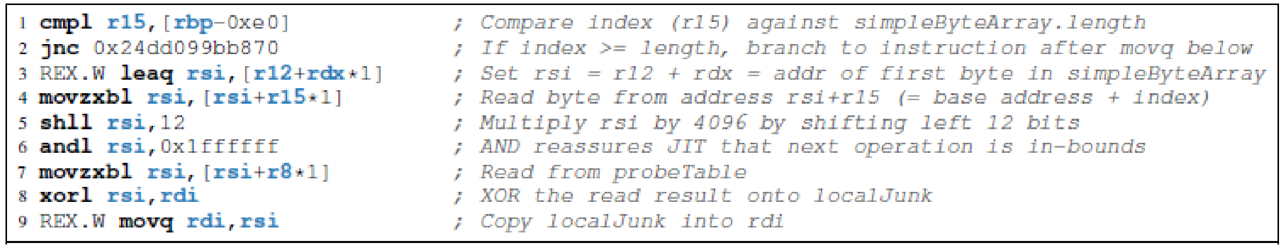
SPECTRE V1: CONT

- Can read up to 10 KB/s with < 0.01% error rate in C program on intel I7-4650U
- JavaScript proof-of-concept implementation tested in chrome
  - Large pool of sensitive data in browser and could be implemented by attackers remotely
  - CLFLUSH not accessible so cache eviction must be used
  - RDTSCP not accessible and chrome degrades accuracy of high-resolution timer intentionally to dissuade timing attacks when using performance.now()
  - Web Workers feature of HTML5 enables creating a thread for timing in shared memory that yields a high-resolution timer
- High accuracy data recovery, but this can be reduced by low-accuracy timing measurements, or unexpected caching of array2 data
  - Multiple iterations may be required to improve confidence at the cost of reduced data rate



```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

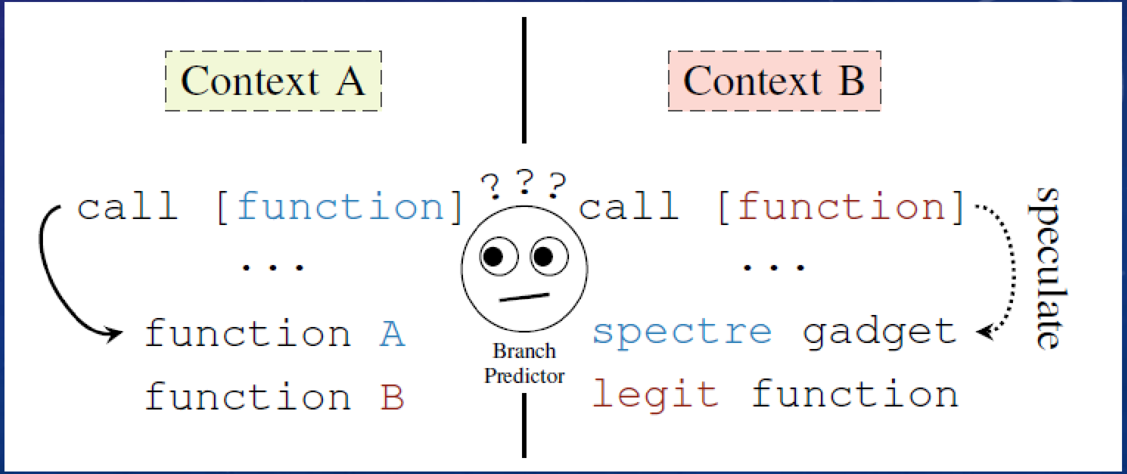


```
cmpl r15,[rbp-0xe0]            ; Compare index (r15) against simpleByteArray.length
jnc 0x24dd099bb870             ; If index >= length, branch to instruction after movq below
REX.W leaq rsi,[r12+rdx*1]     ; Set rsi = r12 + rdx = addr of first byte in simpleByteArray
movzxbl rsi,[rsi+r15*1]        ; Read byte from address rsi+r15 (= base address + index)
shll rsi,12                    ; Multiply rsi by 4096 by shifting left 12 bits
andl rsi,0x1ffffff             ; AND reassures JIT that next operation is in-bounds
movzxbl rsi,[rsi+r8*1]         ; Read from probeTable
xorl rsi,rdi                   ; XOR the read result onto localJunk
REX.W movq rdi,rsi             ; Copy localJunk into rdi
```

Listing 3: Disassembly of JavaScript Example from Listing 2.

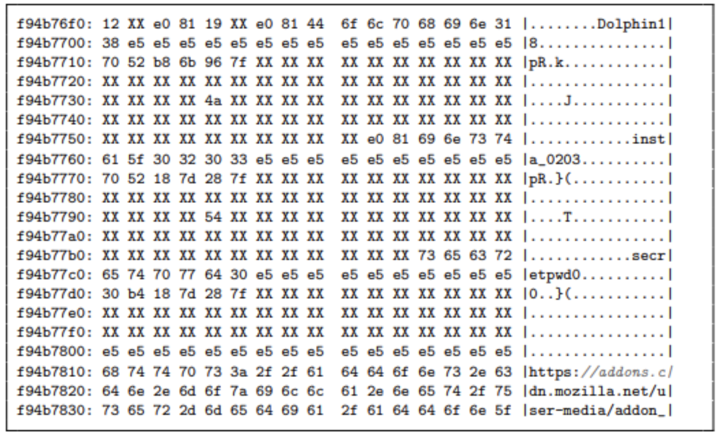
SPECTRE VARIANT 2: INDIRECT BRANCHES

- Mistrains the branch predictor with malicious destinations such that the speculative execution continues in a location chosen by the attacker
- Allows for exposing victim memory even without exploitable conditional branch misprediction
- Attacker that seeks to read victim memory requires control over two registers (R1 & R2) when the indirect branch occurs and a “spectre gadget”
    - Often these registers are ignored by the called function and simply pushed onto the stack
    - The spectre gadget is a code fragment whose speculative execution will transfer info to a covert channel
- Spectre gadget adds / XORs / etc. the R1 onto R2 followed by accessing the memory at R2
- R1 controls the address that will be leaked
- R2 controls how the content will be mapped to an address
- The gadget must reside in memory executable by the victim



MELTDOWN + OTHER VARIANTS

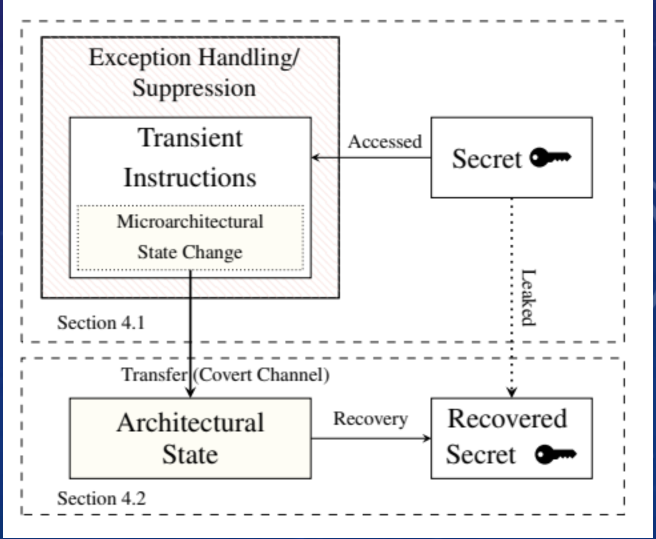
- Spectre assumes no page fault or exception, so data must be in the programs memory space
- Meltdown functions orthogonally to spectre and can be paired to read from kernel memory instead of program memory
    - Exploits out-of-order execution, but does not use branch prediction
    - When an instruction causes a trap, following instructions are executed out-of-order before being terminated
    - Also exploits a vulnerability on intel and ARM processors which allows certain speculative instructions to bypass memory protection
    - Combining these allows meltdown to access kernel memory from user space which is leaked through the covert cache channel
- Other Variations: Mistraining return instructions, leaking info via timing variations, contention on arithmetic units, speculation in store-to-load forwarding logic, Evict+Time, instruction timing, register file contention, or arbitrary secondary observable effects



```
f94b76f0: 12 XX e0 81 19 XX e0 81 44  6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX XX  XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX XX  XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |............inst|
f94b7760: 61 5f 30 32 30 33 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7790: XX XX XX XX 54 XX XX XX XX  XX XX XX XX XX XX XX |....T...........|
f94b77a0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77b0: XX XX XX XX XX XX XX XX XX  XX XX XX 73 65 63 72 |............secr|
f94b77c0: 65 74 70 77 64 30 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |etpwd0..........|
f94b77d0: 30 b4 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |0..}(...........|
f94b77e0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77f0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7800: e5 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |................|
f94b7810: 68 74 74 70 73 3a 2f 2f 61  64 64 6f 6e 73 2e 63 |https://addons.c|
f94b7820: 64 6e 2e 6d 6f 7a 69 6c 6c  61 2e 6e 65 74 2f 75 |dn.mozilla.net/u|
f94b7830: 73 65 72 2d 6d 65 64 69 61  2f 61 64 64 6f 6e 5f |ser-media/addon_|
```

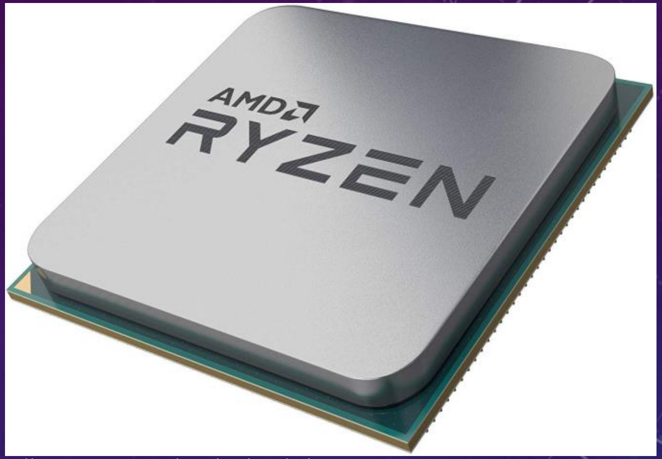
Listing (4) Memory dump of Firefox 56 on Ubuntu 16.10 on a Intel Core i7-6700K disclosing saved passwords.

https://meltdownattack.com/meltdown.pdf

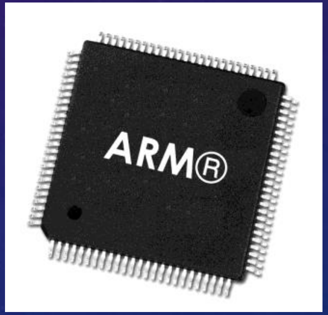


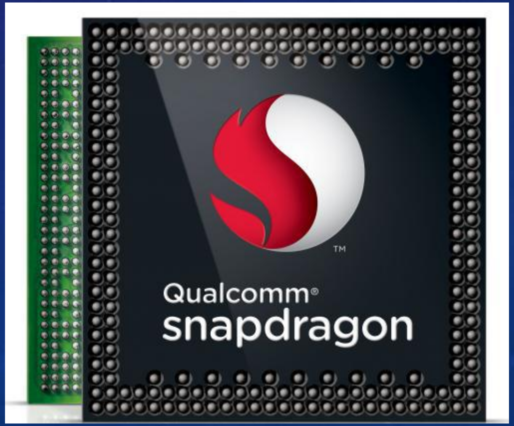
VULNERABLE HARDWARE

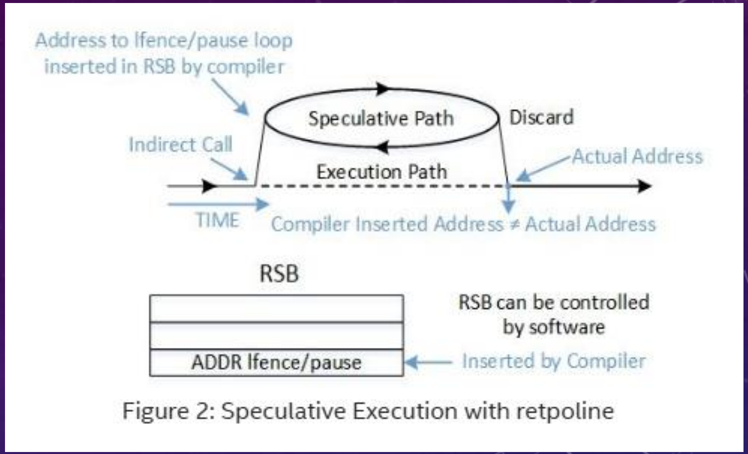
- Empirically verified on several Intel processors
  - Ivy Bridge, Haswell, Broadwell, Skylake, Kaby Lake
- Verified on AMD Ryzen CPUs
- Several ARM-based Samsung and Qualcomm processors in mobile phones
- Responsible disclosure - The Spectre family of attacks is documented under CVE-2017-5753 and CVE-2017-5715.



https://www.qualcomm.com/news/onq/2013/01/qualcomm-snapdragon-chip-best-mobile-processor-2012
https://www.pcmag.com/news/which-cpu-should-you-buy-intel-core-i5-vs-i7
https://copperhilltech.com/blog/a-brief-introduction-to-the-arm-cortex-m3-processor/
https://www.amd.com/en/products/processors/desktops/ryzen.html






Address to lfence/pause loop inserted in RSB by compiler
Speculative Path
Discard
Indirect Call
Actual Address
Execution Path
TIME
Compiler Inserted Address ≠ Actual Address
RSB
RSB can be controlled by software
ADDR lfence/pause
Inserted by Compiler
Figure 2: Speculative Execution with retpoline

https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/retpoline-branch-target-injection-mitigation.html

MITIGATION OPTIONS

- Preventing speculative execution
  - Would cause degradation in performance and unsupported on modern processors (2018), so you would need to change processors to a lower end version that does not have speculative execution
- Preventing access to secret data
  - Chrome executes each site in a different process, webkit replaces array bound checks with index masking
- Preventing data from entering covert channels
  - Future processors could trach the data from speculative operations and prevent its use in subsequent operations
- Limiting data extraction from covert channels
  - Degraded timer accuracy such as in the JavaScript example, but this just lowers data exfiltration throughput
  - Future processors may be able to eliminate covert channels completely
- Preventing branch poisoning
  - Extended ISA for controlling indirect branches (Intel / AMD) prevents branches in privileged code from being affected by branches in less privileged code
  - Restricts sharing of branch prediction between hyperthreads on a single core
  - Flushing the BTB state with a branch prediction barrier
  - Retpolines – code sequences that replace indirect branches with return instructions proposed by Google

PAPERS REFERENCES

- **P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom,** “Spectre Attacks: Exploiting Speculative Execution,” in *Proc. 40th IEEE Symp. Security and Privacy (S&P)*, 2019.
  - **M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg,** “Meltdown: Reading Kernel Memory from User Space,” in *Proc. 27th USENIX Security Symp. (USENIX Security 18)*, 2018.
  - Image sources seen throughout