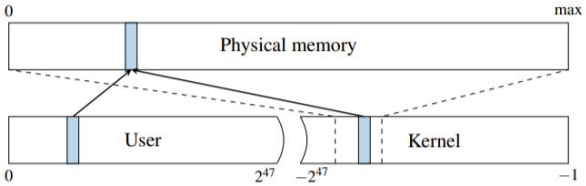
Meltdown Security Exploit

CS 305/341 Course Project

Bhaskar Gupta - 180050022
Niraj Mahajan - 180050069
Shivam Goel - 180050098
Tathagat Verma - 180050111

What is Meltdown?

- An attack that allows overcoming memory isolation completely by providing a way for any user process to read the entire kernel memory
- Programs can get hold of secrets stored in memory of other running programs by reading kernel memory which has mappings of the entire physical memory



- Meltdown does not exploit any software vulnerability and hence works on all major operating systems
- Exploits out-of-order execution of hardware to read inaccessible data

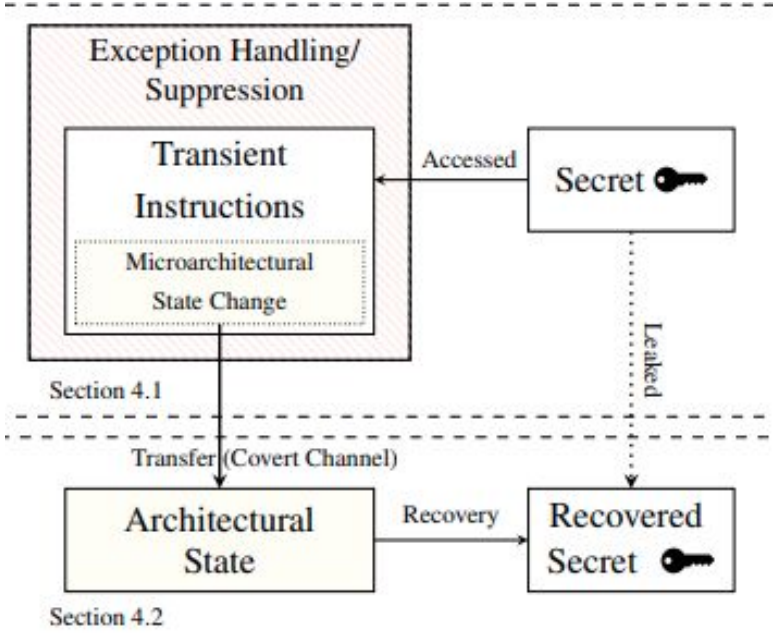
What is Out of Order Execution?

- Out-of-order execution is used to maximize the usage of the CPU
- The CPU executes instructions whenever their required resources(variable values) are available, rather than sequentially processing them
- The CPU does operations on the subsequent instructions but commits them to memory only when it is certain that operation has to be run
- For example branching conditions can lead to out-of-order executed instructions being wasted

Building Blocks of Meltdown

- Transient instructions introduce an exploitable side channel if their operation depends on a secret value
- The parent process forks a child which accesses a protected memory location containing the “secret”, and subsequently is terminated
- This secret can be recovered by the parent process by observing the microarchitectural state

Building Blocks of Meltdown (ctd)



The Meltdown Attack

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

- Let rcx represent the kernel address, and rbx the array using which we affect the cache.
- On line 4 we access the first byte of rcx and store it in the first byte of rax register which raises the exception

The Meltdown Attack

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

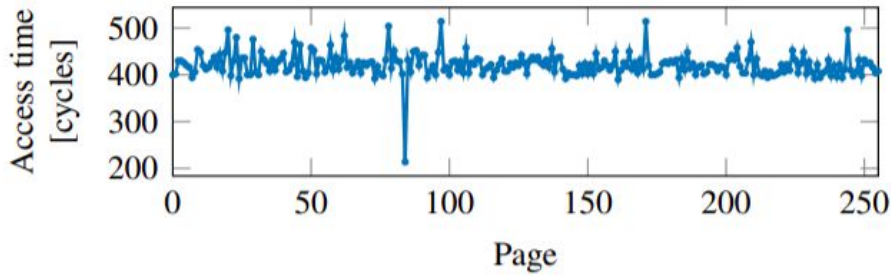
- On line 5 we do a left shift of rax register by 12 bits(so that each data element of the array will be stored on a different page)
- The final step is accessing the address rbx+rax, ie, the rax’th index of the probe array pointed by rbx

The Meltdown Attack

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

- This leads to a race condition between the exception being handled and the array being accessed in the final step
- If the last step of the algorithm wins the race, the cache shall be changed.
- The secret is recovered by observing the cache using Flush+Reload attack

The Meltdown Attack - Flush+Reload



- The probe array (of size 256*4096) maps onto 256 pages and there are 256 possible values for a byte
- We now access the pages of the probe array sequentially and note access time in every case
- We will have only one hit in the cache, which corresponds to the page accessed during the out-of-order execution

The Meltdown Attack - Optimisations

- On reading the data if we get the value 0, we reiterate until fault is raised to check if it is actually zero(instead of 0 due to the error)
- The probability of error will reduce and we can dump the physical memory with more accuracy
- Exception Handling(122 KB/s) vs Exception Suppression with Intel TSX or Branch Predictors(502 KB/s)
- Single Bit Transmission - 256 data accesses in Flush+Reload is the bottleneck

Protection from Meltdown

- **Disabling out-of-order execution:** Though this is the most obvious change this shall not let us reap the benefits of parallelism.
- **Permission check before register fetch:** This shall completely prevent meltdown as the memory access in the last step of meltdown shall never occur but this is inefficient as until the permission is checked we will need to stall.
- **Hard split between user and kernel space:** Let's say the kernel space shall lie in the top half while user space on the bottom half. This hard split shall save time as we can understand if the address is restricted just on the basis of virtual memory.

Protection from Meltdown - KASLR

- **KASLR** (Kernel Address Space Layout Randomization) randomizes the offsets where drivers are located on every boot, making attacks harder as they now require to guess the location of kernel data structures
- This randomized offset can be found out in a few seconds as randomisation is limited to 40bits, hence KASLR is not a suitable protection against Meltdown

Protection from Meltdown - KAISER Page Table Isolation (PTI)

- **KAISER** is a software change which already exists and prevents meltdown attacks to a certain extent
- It is a kernel modification to not have the kernel mapped in user space
- It prevents meltdown attacks as there are no valid mappings to kernel space or physical memory available in user-space
- A few privileged memory locations are required to be mapped in user space which still leaves room for a meltdown attack
- Although KAISER is not full-proof, it is the best short-time solution currently available

Appendix

- **Transient Instruction:** An instruction which is executed out of order, leaving measurable side effects, is called a transient instruction
- **Architectural State:** Architectural state includes register files and memory
- **Microarchitectural State:** Microarchitectural state represents internal state that has not yet been exposed outside the processor (e.g., instruction queue state).
- **Side-channel attack:** A side-channel attack is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself

References

- Meltdown: Reading Kernel Memory from User Space, Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg Appears in Proceedings of the 27th USENIX Security Symposium, 2018
- Institute of Applied Information Processing and Communications (IAIK). Meltdown (security vulnerability). url: https://github.com/IAIK/meltdown.
- Wikipedia. Meltdown (security vulnerability). url: https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability).