Welcome to our presentation on Spectre & Meltdown.

In this brief presentation we will try to explain what these vulnerabilities are, how they work, what they affect and what can be done about them.

Background
The Spectre vulnerability was discovered in early 2018.
Initially three variants were discovered, of which Meltdown is one.
Since then, an avalanche of subvariants (vulnerabilities that exploit the same attack vector) have been discovered and written about extensively in academic articles.

Simply, the vulnerability exploits speculative and OoO mechanisms ubiquitous in modern processors.
This has the implication that billions of devices are affected globally.
The hardware-nature of the exploit means that systems are affected regardless of operating system or other software.
Software mitigations do exist however, which will be covered later on.

Technical Background
A **Microarchitecture** is simply the way a particular ISA is implemented on actual hardware.
An important assumption previously held is that changes to the microarchitectural state were inconsequential, since only functionally correct results are passed to higher levels.

Speculative Execution is the dispatch of instructions prior to resolving control-flow dependencies.
Misspeculated results are reverted to maintain the functional correctness mentioned previously.
Instructions can speculatively execute when control-flow condition invalid, and even when an instruction causes an exception (relevant to Meltdown).

Transient Instructions are instructions that are executed as a result of misspeculation, and are of course always aborted.
Though changes are reverted, the microarchitectural state (ie. cache contents) may be changed.

Side-Channel Attacks are a general term for secret information derived from the specific implementation of a computer system.
In this case, the microarchitectural state canbe leaked by an attacker program sharing resources with the victim.
The most common shared resource is the cache, which is vulnerable to so-called ”timing-attacks”. These measure the access time to a given line, determine hit/miss and using this determine if a given piece of information is within the cache.

Memory Isolation

Spectre Overview
Spectre attacks basically trick the processor into (speculatively) executing instruction sequences not permitted under correct program execution, in order to leak information from within the victim memory address space.

At a high level this in the following way:
*An attacker trains a prediction mechanism to predict a given outcome, by providing the branch condition with valid inputs.

*The processor is then intentionally made to misspeculate by giving it an invalid input, while it predicts that it is valid.

*Instructions normally guarded by the branch condition are then speculatively executed, saving potential secrets in the microarchitectural state.

The secret is then derived (or reconstructed) via side-channel attacks, making the secret architecturally visible.

Spectre In-Depth
To explain in terms of a more concrete example, the two first Specter variants are demonstrated.

Spectre V1
The first variant is used to speculatively exceed the bounds of a given array, accessing potentially private memory.

An attacker first trains the branch in the first line using valid values for ”untrusted_offset”. An specially selected, invalid value is then provided, making the processor misspeculate.

Potentially secret information is then speculatively accessed (not saved! as loads will be reverted)

A secondary access then touches a predictable memory location, dependent on the value of the secret from the previous line.

Side-channel attacks are then used to determine which cache-line has been affected.
The affected cache-line will be dependent on the value of the secret, which then allows it to be reconstructed!
The misspeculation is eventually reverted, and ”val” and ”x” are discarded.

Spectre V2

The second Spectre variant exploits the Branch Target Buffer as the predictive mechanism, as opposed to the branch predictor in V1.

The attacker trains a specific jump instruction to speculatively branch to a particular address, chosen by the attacker.

The destination usually contains specific instruction sequences, or ’gadgets’, that execute desired operations, followed by a return. These speculative jumps can then be chained together for almost arbitrary execution.

After seizing control-flow, information is leaked just as in V1.

Spectre: Scope

The attack, as shown, is simple but affects almost all modern processors.

Intel, AMD, ARM as well as IBM are affected.

The assumption that runtime and language constructs can protect data in shared address space is comprehensively disproven.

Hardware mitigation is challenging without careful hardware-software cooperation.

For example, many ”managed” languages that have software protection mechanisms do not reveal this to the hardware context.

Forthcoming ISA extensions to address this are likely slow to implement and may result in decreased performance.

Spectre V1 mitigation

Spectre V1 can be mitigated in both hardware and software.

An obvious solution may be to prevent speculative execution beyond bounds checks.

This results in a severe performance penalty, however.

The preferred approach is to speculatively clamp the array index (”untrusted_offset” in the example) to within the array bounds using a software macro.

Meltdown Overview

.

.

.

. Anna takes over..

.

.

Summary: Spectre vs. Meltdown

Both exploit speculative execution.

Spectre exploits transient execution following control- or data-flow-misprediction.

Whereas Meltdown exploits transient execution following exception-causing instructions.

Spectre bypasses software-defined security policies and can only compute on data which the victim program is premitted to access.

Meltdown on the other hand, bypasses hardware enforced security and can speculatively execute on the results of security-bounds-violating instructions, which normally raise an exception.

Conclusion

Spectre attacks affect almost all modern processors, from PC's to mobile phones and beyond. Furthermore, they fundamentally challenge long-held assumptions about computer correctness

Transient instructions DO leave measurable side-effects in the microarchitecture, a fact that has been shown to be not-inconsequential.
That means that "timing-independent functional correctness"is no longer a sufficient metric when it comes to security.

Mitigation is possible in many cases, but likely come with a performance hit.

The main question is: can Spectre be eliminated altogether?

Academic research indicates that this may be impossible in software, leaving many existing systems vulnerable indefinately.

Even in hardware, accounting for all variants is a daunting proposition, especially since they are still being discovered!

Many Thanks!!!