Zia us samad

15-28591-1

Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure.

How it works :

One of the central security features of today’s operating

systems is memory isolation. Operating systems ensure

that user applications cannot access each other’s memories

and prevent user applications from reading or writing

kernel memory. This isolation is a cornerstone of our

computing environments and allows running multiple applications

on personal devices or executing processes of

multiple users on a single machine in the cloud.

On modern processors, the isolation between the kernel

and user processes is typically realized by a supervisor

supervisor

bit of the processor that defines whether a memory

page of the kernel can be accessed or not. The basic

idea is that this bit can only be set when entering kernel

code and it is cleared when switching to user processes.

This hardware feature allows operating systems to map

the kernel into the address space of every process and

to have very efficient transitions from the user process

to the kernel, e.g., for interrupt handling. Consequently,

in practice, there is no change of the memory mapping

when switching from a user process to the kernel.

In this work, we present Meltdown1. Meltdown is a

novel attack that allows overcoming memory isolation

completely by providing a simple way for any user process

to read the entire kernel memory of the machine it

executes on, including all physical memory mapped in

the kernel region. Meltdown does not exploit any software

vulnerability, i.e., it works on all major operating

systems. Instead, Meltdown exploits side-channel information

available on most modern processors, e.g., modern

Intel microarchitectures since 2010 and potentially

on other CPUs of other vendors.

While side-channel attacks typically require very specific

knowledge about the target application and are tailored

to only leak information about its secrets, Meltdown

allows an adversary who can run code on the vulnerable

processor to obtain a dump of the entire kernel

address space, including any mapped physical memory.

The root cause of the simplicity and strength of Meltdown

are side effects caused by out-of-order execution.

Out-of-order execution is an important performance

feature of today’s processors in order to overcome latencies

of busy execution units, e.g., a memory fetch unit

needs to wait for data arrival from memory. Instead of

stalling the execution, modern processors run operations

out-of-order i.e., they look ahead and schedule subsequent

operations to idle execution units of the processor.

However, such operations often have unwanted side.

#### Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre.Spectre is harder to exploit than Meltdown, but it is also harder to mitigate.

How it works:

Computations performed by physical devices often leave

observable side effects beyond the computation’s nominal

outputs. Side-channel attacks focus on exploiting these side

effects to extract otherwise-unavailable secret information.

Since their introduction in the late 90’s [43], many physical

effects such as power consumption [41, 42], electromagnetic

radiation [58], or acoustic noise [20] have been leveraged to

extract cryptographic keys as well as other secrets.

Physical side-channel attacks can also be used to extract

secret information from complex devices such as PCs and

mobile phones [21, 22]. However, because these devices

often execute code from a potentially unknown origin, they

face additional threats in the form of software-based attacks,

which do not require external measurement equipment. While

some attacks exploit software vulnerabilities (such as buffer

overflows [5] or double-free errors [12]), other software attacks.

leverage hardware vulnerabilities to leak sensitive information.

Attacks of the latter type include microarchitectural attacks

exploiting cache timing [8, 30, 48, 52, 55, 69, 74], branch

prediction history [1, 2], branch target buffers [14, 44] or open

DRAM rows [56]. Software-based techniques have also been

used to mount fault attacks that alter physical memory [39] or

internal CPU values [65].

Several microarchitectural design techniques have facilitated

the increase in processor speed over the past decades. One such

advancement is speculative execution, which is widely used

to increase performance and involves having the CPU guess

likely future execution directions and prematurely execute

instructions on these paths. More specifically, consider an

example where the program’s control flow depends on an

uncached value located in external physical memory. As this

memory is much slower than the CPU, it often takes several

hundred clock cycles before the value becomes known. Rather

than wasting these cycles by idling, the CPU attempts to guess

the direction of control flow, saves a checkpoint of its register

state, and proceeds to speculatively execute the program on the

guessed path. When the value eventually arrives from memory,

the CPU checks the correctness of its initial guess. If the

guess was wrong, the CPU discards the incorrect speculative

execution by reverting the register state back to the stored

checkpoint, resulting in performance comparable to idling.

However, if the guess was correct, the speculative execution

results are committed, yielding a significant performance gain

as useful work was accomplished during the delay.

From a security perspective, speculative execution involves

executing a program in possibly incorrect ways. However,

because CPUs are designed to maintain functional correctness

by reverting the results of incorrect speculative executions to

their prior states, these errors were previously assumed to be

safe.