The Intersection of Staying Secure with Full Speed Ahead

James Reinders, Intel, Engineer

intel®

Physical Separation

221B BAKER STREET
LONDON UK

PDP-8
Electromagnetic emissions –
AM radio plays!

User groups offered multiple “music” compilers

What is secure?

Two examples for you

Both very PERSONAL to me

OpenSSL

OpenSSL

- Time to decrypt
  - Revealed “how many zeros” were in the private key

OpenSSL

- Time to decrypt
  - Revealed “how many zeros” were in the private key

- One account – blamed it on multicore

Meltdown and Spectre

2017: Unprecedented research results find generic security holes that exist in virtually *all* processors.

server processors, desktop processors, tablet processors, cellphone processors, etc.
Intel, AMD, IBM, ARM (Qualcomm, Apple, Broadcom), etc.

Meltdown

Moritz Lipp[1], Michael Schwarz[1], Daniel Gruss[1], Thomas Prescher[2], Werner Haas[2], Stefan Mangard[1], Paul Kocher[3], Daniel Genkin[4], Yuval Yarom[5], Mike Hamburg[6]
[1] *Graz University of Technology*
[2] *Cyberus Technology GmbH*
[3] *Independent*
[4] *University of Pennsylvania and University of Maryland*
[5] *University of Adelaide and Data61*
[6] *Rambus, Cryptography Research Division*

Abstract

The security of computer systems fundamentally relies on memory isolation, e.g., kernel address ranges are marked as non-accessible and are protected from user access. In this paper, we present Meltdown. Meltdown exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations including personal data and passwords. Out-of-order execution is an indispensable performance feature and present in a wide range of modern processors. The attack is independent of the operating system, and it does not rely on any software vulnerabilities. Meltdown breaks all security assumptions given by address space isolation as well as paravirtualized environments and, thus, every security mechanism building upon this foundation. On affected systems, Meltdown enables an adversary to read memory of other processes or virtual machines in the cloud without any permissions or privileges, affecting millions of customers and virtually every user of a personal computer. We show that the KAISER defense mechanism for KASLR [8] has the important (but inadvertent) side effect of impeding Meltdown. We stress that KAISER must be deployed immediately to prevent large-scale exploitation of this severe information leakage.

1 Introduction

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.

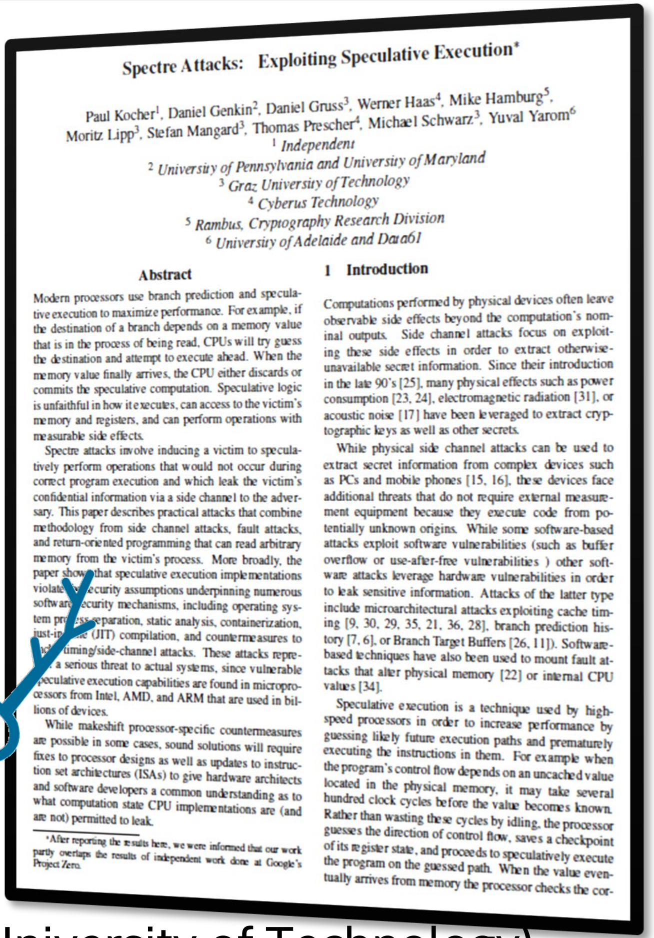
On modern processors, the isolation between the kernel and user processes is typically realized by a supervisor bit of the processor that defines whether a memory page of the kernel can be accessed or not. The basic idea is that this bit can only be set when entering kernel code and it is cleared when 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 Meltdown[1]. 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-

[1] This attack was independently found by the authors of this paper and Jann Horn from Google Project Zero.



Spectre Attacks: Exploiting Speculative Execution*

Paul Kocher[1], Daniel Genkin[2], Daniel Gruss[3], Werner Haas[4], Mike Hamburg[5], Moritz Lipp[3], Stefan Mangard[3], Thomas Prescher[4], Michael Schwarz[3], Yuval Yarom[6]
[1] *Independent*
[2] *University of Pennsylvania and University of Maryland*
[3] *Graz University of Technology*
[4] *Cyberus Technology*
[5] *Rambus, Cryptography Research Division*
[6] *University of Adelaide and Data61*

Abstract

Modern processors use branch prediction and speculative execution to maximize performance. For example, if the destination of a branch depends on a memory value that is in the process of being read, CPUs will try guess the destination and attempt to execute ahead. When the memory value finally arrives, the CPU either discards or commits the speculative computation. Speculative logic is unfaithful in how it executes, can access to the victim's memory and registers, and can perform operations with measurable side effects.

Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct program execution and which leak the victim's confidential information via a side channel to the adversary. This paper describes practical attacks that combine methodology from side channel attacks, fault attacks, and return-oriented programming that can read arbitrary memory from the victim's process. More broadly, the paper shows that speculative execution implementations violate the security assumptions underpinning numerous software security mechanisms, including operating system process separation, static analysis, containerization, just-in-time (JIT) compilation, and countermeasures to cache timing/side-channel attacks. These attacks represent a serious threat to actual systems, since vulnerable speculative execution capabilities are found in microprocessors from Intel, AMD, and ARM that are used in billions of devices.

While makeshift processor-specific countermeasures are possible in some cases, sound solutions will require fixes to processor designs as well as updates to instruction set architectures (ISAs) to give hardware architects and software developers a common understanding as to what computation state CPU implementations are (and are not) permitted to leak.

*After reporting the results here, we were informed that our work partly overlaps the results of independent work done at Google's Project Zero.

1 Introduction

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 in order to extract otherwise-unavailable secret information. Since their introduction in the late 90's [25], many physical effects such as power consumption [23, 24], electromagnetic radiation [31], or acoustic noise [17] have been leveraged to extract cryptographic keys as well as other secrets.

While physical side channel attacks can be used to extract secret information from complex devices such as PCs and mobile phones [15, 16], these devices face additional threats that do not require external measurement equipment because they execute code from potentially unknown origins. While some software-based attacks exploit software vulnerabilities (such as buffer overflow or use-after-free vulnerabilities) other software attacks leverage hardware vulnerabilities in order to leak sensitive information. Attacks of the latter type include microarchitectural attacks exploiting cache timing [9, 30, 29, 35, 21, 36, 28], branch prediction history [7, 6], or Branch Target Buffers [26, 11]). Software-based techniques have also been used to mount fault attacks that alter physical memory [22] or internal CPU values [34].

Speculative execution is a technique used by high-speed processors in order to increase performance by guessing likely future execution paths and prematurely executing the instructions in them. For example when the program's control flow depends on an uncached value located in the physical memory, it may take several hundred clock cycles before the value becomes known. Rather than wasting these cycles by idling, the processor guesses 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 processor checks the cor-

https://meltdownattack.com/ (system hosted at Graz University of Technology)

Meltdown and Spectre

"Architectural state"

a) Deduce secrets from out-of-order execution of instructions that are not committed. (Meltdown)

b) Deduce secrets from speculative execution of instructions on paths that are not executed (therefore instructions are not committed). (Spectre)

Assumptions – things change
Today's barriers can disappear

- Separations disappear - or are not real
- multiplexed communications can be hacked
- shared state: physical separations do not persist
- Cost to do an exploit
  - in resources, money, time
- Value of secrets
  - Value of available information becomes precious
- Assumptions are not always clear even when they are there

Side channel: Two steps

Ask to grab data that you are no allowed to see – in an instruction the processor will never commit or complete (because the processor knows that we should never see the data).

Quickly cause a side-effect based on the data, which will leave a detectable footprint behinds from which we can infer something about the protected data. *Warm cache line, warm TLB entry/page table, etc.*

The secret leaks though this side channel for later inspection, even if the processor and /or operating systems cancel the code that read data and caused the side effect.

Side channels lead to exploits

Meltdown and Spectre show the existence of “side channels.”

Side channels allow secrets to be inferred even though they are otherwise protected (not disclosed directly).

Power Consumption

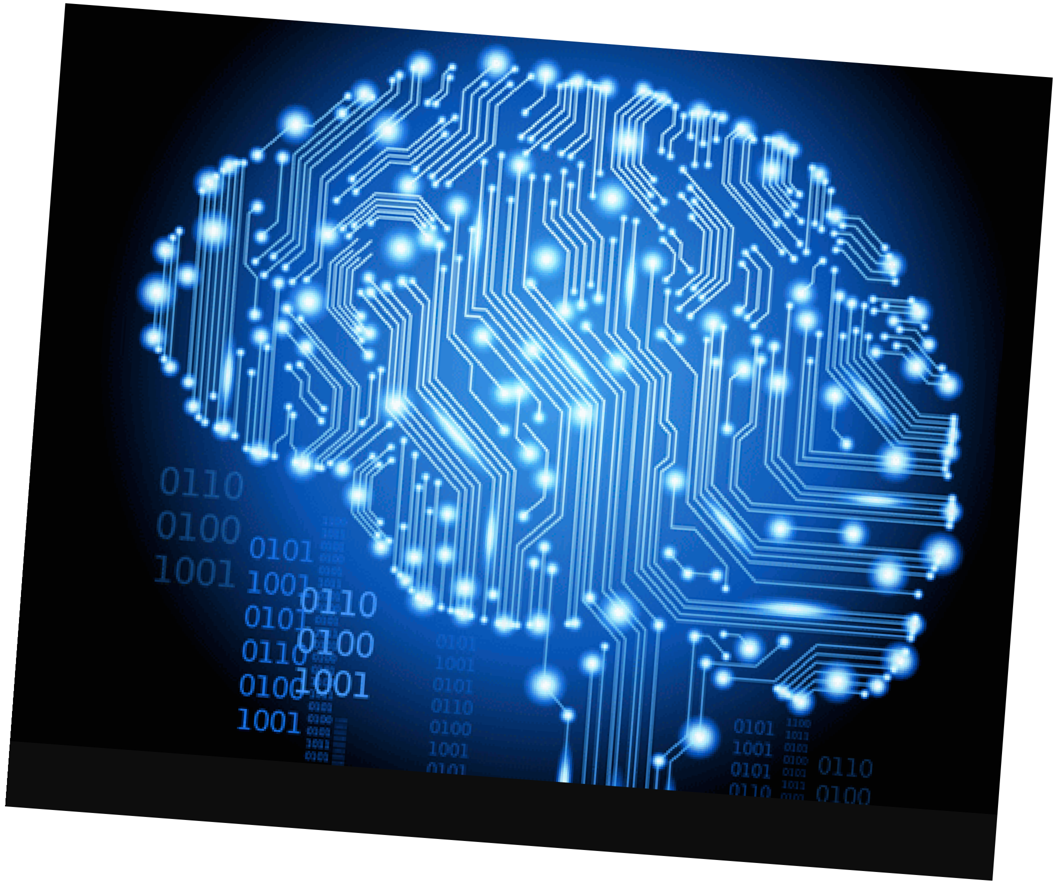
Electromagnetic Radiation

Injection of Faults

Acoustic Sounds

Just because we can see secrets, doesn’t mean we know how to exploit them (to use them for evil). Put another way: we know side channels exist, so we need to fear exploits – and we need to close side channels.

The obvious?
Quantum and AI



United – do we know?

Claims
Warrants
then "not possible"

Chris Roberts, a US security researcher, CLAIMS that he hacked the in-flight entertainment systems on several flights and on one flight gained access to the plane's thrust management computer and briefly changed its course.

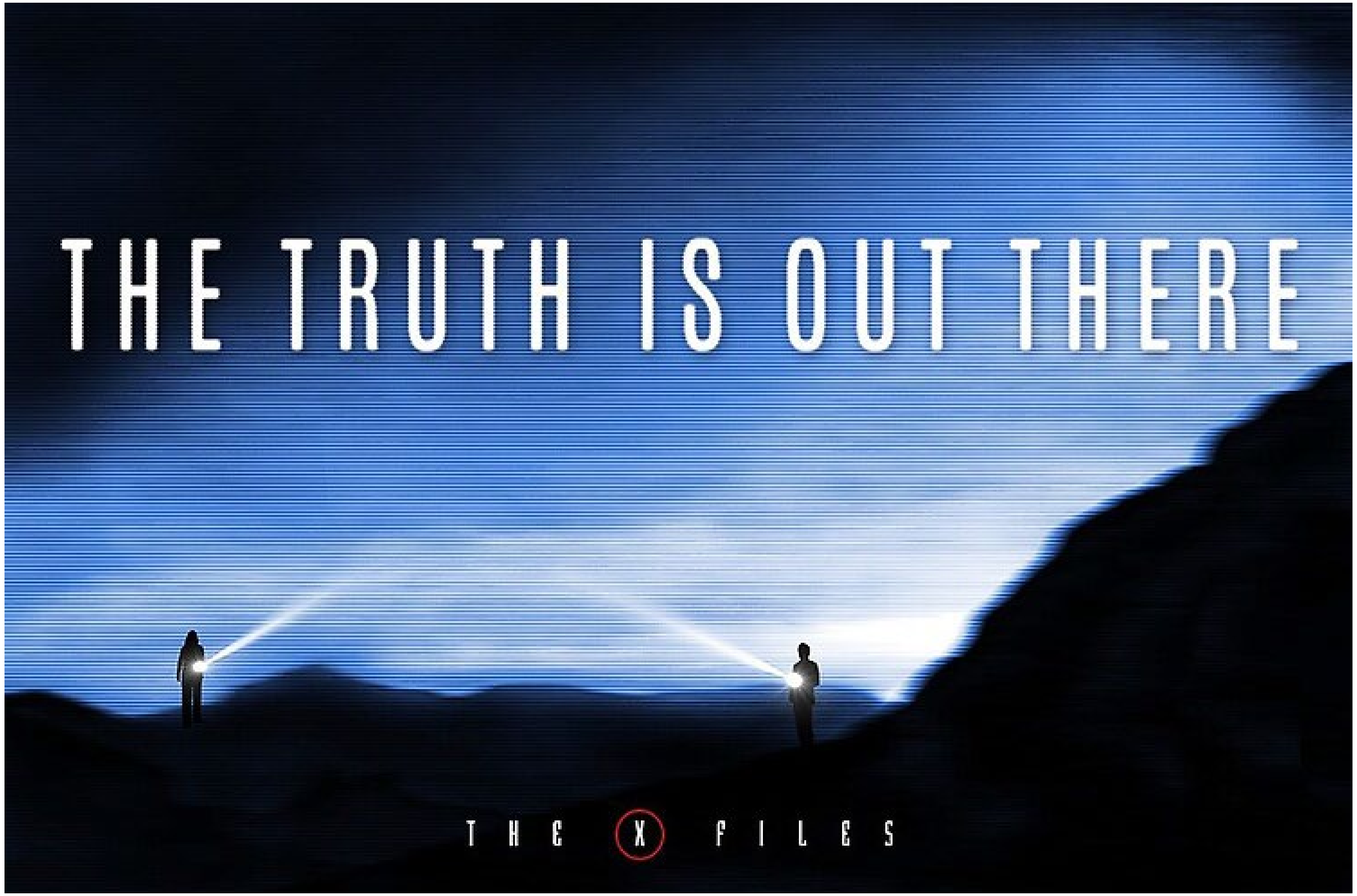
In a warrant application filed in April, FBI agent Mark Hurley said that Mr. Roberts made noticeable changes to the aircraft.
"(Roberts) stated that he thereby caused one of the airplane engines to climb resulting in a lateral or sideways movement of the plane during one of these flights," he wrote.

The document states that Roberts claimed to have compromised the in-flight entertainment systems of around 20 flights in the past four years. He achieved this by connecting his laptop to the electronics box under his seat after prying it open.

Operation Fortitude

Inflatable Tanks

What will you do to make things useful, safe, and make both qualities last?



More importantly: what are we NOT thinking about today, that we will be able to exploit in two decades?