# CWE Detail – CWE-1421

## Description

A processor event may allow transient operations to access
 architecturally restricted data (for example, in another address
 space) in a shared microarchitectural structure (for example, a CPU
 cache), potentially exposing the data over a covert channel.

## Extended Description

Many commodity processors have Instruction Set Architecture (ISA)
 features that protect software components from one another. These
 features can include memory segmentation, virtual memory, privilege
 rings, trusted execution environments, and virtual machines, among
 others. For example, virtual memory provides each process with its own
 address space, which prevents processes from accessing each other's
 private data. Many of these features can be used to form
 hardware-enforced security boundaries between software components. Many commodity processors also share microarchitectural resources that
 cache (temporarily store) data, which may be confidential. These
 resources may be shared across processor contexts, including across
 SMT threads, privilege rings, or others. When transient operations allow access to ISA-protected data in a
 shared microarchitectural resource, this might violate users'
 expectations of the ISA feature that is bypassed. For example, if
 transient operations can access a victim's private data in a shared
 microarchitectural resource, then the operations' microarchitectural
 side effects may correspond to the accessed data. If an attacker can
 trigger these transient operations and observe their side effects
 through a covert channel [REF-1400], then the attacker may be able to infer the
 victim's private data. Private data could include sensitive program
 data, OS/VMM data, page table data (such as memory addresses), system
 configuration data (see Demonstrative Example 3), or any other data
 that the attacker does not have the required privileges to access.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2017-5715: A fault may allow transient user-mode operations to
 access kernel data cached in the L1D, potentially exposing the data
 over a covert channel.

**•** CVE-2018-3615: A fault may allow transient non-enclave operations to
 access SGX enclave data cached in the L1D, potentially exposing the
 data over a covert channel.

**•** CVE-2019-1135: A TSX Asynchronous Abort may allow transient operations
 to access architecturally restricted data, potentially exposing the
 data over a covert channel.

## Modes of Introduction

**•** Architecture and Design: This weakness can be introduced during hardware architecture and
 design if a data path allows architecturally restricted data to
 propagate to operations that execute before an older mis-prediction or
 processor event (such as an exception) is caught.

**•** Implementation: This weakness can be introduced during system software
 implementation if state-sanitizing operations are not invoked when
 switching from one context to another, according to the hardware
 vendor's recommendations for mitigating the weakness.

**•** System Configuration: This weakness can be introduced if the system has not been
 configured according to the hardware vendor's recommendations for
 mitigating the weakness.

**•** Architecture and Design: This weakness can be introduced when an access control check
 (for example, checking page permissions) can proceed in parallel with
 the access operation (for example, a load) that is being checked. If
 the processor can allow the access operation to execute before the
 check completes, this race condition may allow subsequent transient
 operations to expose sensitive information.

## Common Consequences

**•** Impact: Read Memory — Notes:

## Potential Mitigations

**•** Architecture and Design: Hardware designers may choose to engineer the processor's
 pipeline to prevent architecturally restricted data from being used by
 operations that can execute transiently. (Effectiveness: High)

**•** Architecture and Design: Hardware designers may choose not to share
 microarchitectural resources that can contain sensitive data, such as
 fill buffers and store buffers. (Effectiveness: Moderate)

**•** Architecture and Design: Hardware designers may choose to sanitize specific
 microarchitectural state (for example, store buffers) when the
 processor transitions to a different context, such as whenever a
 system call is invoked. Alternatively, the hardware may expose
 instruction(s) that allow software to sanitize microarchitectural
 state according to the user or system administrator's threat
 model. These mitigation approaches are similar to those that address
 CWE-226; however, sanitizing microarchitectural state may not be the
 optimal or best way to mitigate this weakness on every processor
 design. (Effectiveness: Moderate)

**•** Architecture and Design: The hardware designer can attempt to prevent transient
 execution from causing observable discrepancies in specific covert
 channels. (Effectiveness: Limited)

**•** Architecture and Design: Software architects may design software to enforce strong
 isolation between different contexts. For example, kernel page table
 isolation (KPTI) mitigates the Meltdown vulnerability [REF-1401] by
 separating user-mode page tables from kernel-mode page tables, which
 prevents user-mode processes from using Meltdown to transiently access
 kernel memory [REF-1404]. (Effectiveness: Limited)

**•** Build and Compilation: If the weakness is exposed by a single instruction (or a
 small set of instructions), then the compiler (or JIT, etc.) can be
 configured to prevent the affected instruction(s) from being
 generated, and instead generate an alternate sequence of instructions
 that is not affected by the weakness. (Effectiveness: Limited)

**•** Build and Compilation: Use software techniques (including the use of
 serialization instructions) that are intended to reduce the number of
 instructions that can be executed transiently after a processor event
 or misprediction. (Effectiveness: Incidental)

**•** Implementation: System software can mitigate this weakness by invoking
 state-sanitizing operations when switching from one context to
 another, according to the hardware vendor's recommendations. (Effectiveness: Limited)

**•** System Configuration: Some systems may allow the user to disable (for example,
 in the BIOS) sharing of the affected resource. (Effectiveness: Limited)

**•** System Configuration: Some systems may allow the user to disable (for example,
 in the BIOS) microarchitectural features that allow transient access
 to architecturally restricted data. (Effectiveness: Limited)

**•** Patching and Maintenance: The hardware vendor may provide a patch to sanitize the
 affected shared microarchitectural state when the processor
 transitions to a different context. (Effectiveness: Moderate)

**•** Patching and Maintenance: This kind of patch may not be feasible or
 implementable for all processors or all weaknesses. (Effectiveness: Limited)

**•** Requirements: Processor designers, system software vendors, or other
 agents may choose to restrict the ability of unprivileged software to
 access to high-resolution timers that are commonly used to monitor
 covert channels. (Effectiveness: Defense in Depth)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** Vulnerable processors may return kernel data from a shared
 microarchitectural resource in line 4, for example, from the
 processor's L1 data cache. Since this vulnerability involves a race
 condition, the mov in line 4 may not always return kernel data (that
 is, whenever the check "wins" the race), in which case this
 demonstration code re-attempts the access in line 6. The accessed data
 is multiplied by 4KB, a common page size, to make it easier to observe
 via a cache covert channel after the transmission in line 7. The use
 of cache covert channels to observe the side effects of transient
 execution has been described in [REF-1408].

**•** N/A

**•** N/A