# CWE Detail – CWE-1281

## Description

Specific combinations of processor instructions lead to undesirable behavior such as locking the processor until a hard reset performed.

## Extended Description

If the instruction set architecture (ISA) and processor logic are not designed carefully and tested thoroughly, certain combinations of instructions may lead to locking the processor or other unexpected and undesirable behavior. Upon encountering unimplemented instruction opcodes or illegal instruction operands, the processor should throw an exception and carry on without negatively impacting security. However, specific combinations of legal and illegal instructions may cause unexpected behavior with security implications such as allowing unprivileged programs to completely lock the CPU.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2021-26339: A bug in AMD CPU's core logic allows a potential DoS by using a specific x86 instruction sequence to hang the processor

**•** CVE-1999-1476: A bug in some Intel Pentium processors allow DoS (hang) via an invalid "CMPXCHG8B" instruction, causing a deadlock

## Related Attack Patterns (CAPEC)

* CAPEC-212

## Modes of Introduction

**•** Architecture and Design: Unexpected behavior from certain instruction combinations can arise from bugs in the ISA

**•** Implementation: Unexpected behavior from certain instruction combinations can arise because of implementation details such as speculative execution, caching etc.

## Common Consequences

**•** Impact: Varies by Context — Notes:

## Potential Mitigations

**•** Testing: Implement a rigorous testing strategy that incorporates randomization to explore instruction sequences that are unlikely to appear in normal workloads in order to identify halt and catch fire instruction sequences. (Effectiveness: N/A)

**•** Patching and Maintenance: Patch operating system to avoid running Halt and Catch Fire type sequences or to mitigate the damage caused by unexpected behavior. See [REF-1108]. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** N/A

**•** N/A

**•** N/A

**•** Atomicity is especially critical when the variable to be modified is a mutex, counting semaphore, or similar piece of data that controls access to shared resources. Failure to ensure atomicity may result in two processors accessing a shared resource simultaneously, permanent lock-up, or similar disastrous behavior.