# CWE Detail – CWE-1298

## Description

A race condition in the hardware logic results in undermining security guarantees of the system.

## Extended Description

A race condition in logic circuits typically occurs when a logic gate gets inputs from signals that have traversed different paths while originating from the same source. Such inputs to the gate can change at slightly different times in response to a change in the source signal. This results in a timing error or a glitch (temporary or permanent) that causes the output to change to an unwanted state before settling back to the desired state. If such timing errors occur in access control logic or finite state machines that are implemented in security sensitive flows, an attacker might exploit them to circumvent existing protections.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Related Attack Patterns (CAPEC)

* CAPEC-26

## Modes of Introduction

**•** Architecture and Design: N/A

**•** Implementation: N/A

## Common Consequences

**•** Impact: Bypass Protection Mechanism, Gain Privileges or Assume Identity, Alter Execution Logic — Notes:

## Potential Mitigations

**•** Architecture and Design: Adopting design practices that encourage designers to recognize and eliminate race conditions, such as Karnaugh maps, could result in the decrease in occurrences of race conditions. (Effectiveness: N/A)

**•** Implementation: Logic redundancy can be implemented along security critical paths to prevent race conditions. To avoid metastability, it is a good practice in general to default to a secure state in which access is not given to untrusted agents. (Effectiveness: N/A)

## Applicable Platforms

**•** Verilog (Class: None, Prevalence: Undetermined)

**•** VHDL (Class: None, Prevalence: Undetermined)

## Demonstrative Examples

**•** The buggy line of code, commented above, results in signal 'z' periodically changing to an unwanted state. Thus, any logic that references signal 'z' may access it at a time when it is in this unwanted state. This line should be replaced with the line shown below in the Good Code Snippet which results in signal 'z' remaining in a continuous, known, state. Reference for the above code, along with waveforms for simulation can be found in the references below.

**•** However, the above code [REF-1394] allows the values of pmpaddr\_i and pmpcfg\_i to be changed through DMA's input ports. This causes a race condition and will enable attackers to access sensitive addresses that the configuration is not associated with. Attackers can initialize the DMA access process (CTRL\_IDLE) using pmpcfg\_i for a non-privileged PMP address (pmpaddr\_i). Then during the loading state (CTRL\_LOAD), attackers can replace the non-privileged address in pmpaddr\_i with a privileged address without the requisite authorized access configuration. To fix this issue (see [REF-1395]), the value of the pmpaddr\_i and pmpcfg\_i signals should be stored in local registers (pmpaddr\_reg and pmpcfg\_reg at the start of the DMA access process and the pmp module should reference those registers instead of the signals directly. The values of the registers can only be updated at the start (CTRL\_IDLE) or the end (CTRL\_DONE) of the DMA access process, which prevents attackers from changing the PMP address in the middle of the DMA access process.