# CWE Detail – CWE-1429

## Description

The product has a hardware interface that silently discards operations
 in situations for which feedback would be security-relevant, such as
 the timely detection of failures or attacks.

## Extended Description

While some systems intentionally withhold feedback as a security
 measure, this approach must be strictly controlled to ensure it does
 not obscure operational failures that require prompt detection and
 remediation. Without these essential confirmations, failures go
 undetected, increasing the risk of data loss, security
 vulnerabilities, and overall system instability. Even when withholding
 feedback is an intentional part of a security policy designed, for
 example, to prevent attackers from gleaning sensitive internal
 details, the absence of expected feedback becomes a critical weakness
 when it masks operational failures that require prompt detection and
 remediation. For instance, certain encryption algorithms always return ciphertext
 regardless of errors to prevent attackers from gaining insight into
 internal state details. However, if such an algorithm fails to
 generate the expected ciphertext and provides no error feedback, the
 system cannot distinguish between a legitimate output and a
 malfunction. This can lead to undetected cryptographic failures,
 potentially compromising data security and system reliability. Without
 proper notification, a critical failure might remain hidden,
 undermining both the reliability and security of the process. Therefore, this weakness captures issues across various hardware
 interfaces where operations are discarded without any feedback, error
 handling, or logging. Such omissions can lead to data loss, security
 vulnerabilities, and system instability, with potential impacts
 ranging from minor to catastrophic. For some kinds of hardware products, some errors may be correctly
 identified and subsequently discarded, and the lack of feedback may
 have been an intentional design decision. However, this could result
 in a weakness if system operators or other authorized entities are not
 provided feedback about security-critical operations or failures that
 could prevent the operators from detecting and responding to an
 attack. For example: In a System-on-Chip (SoC) platform, write operations to reserved
 memory addresses might be correctly identified as invalid and
 subsequently discarded. However, if no feedback is provided to
 system operators, they may misinterpret the device's state, failing
 to recognize conditions that could lead to broader failures or
 security vulnerabilities. For example, if an attacker attempts
 unauthorized writes to protected regions, the system may silently
 discard these writes without alerting security mechanisms. This lack
 of feedback could obscure intrusion attempts or misconfigurations,
 increasing the risk of unnoticed system compromise Microcontroller Interrupt Systems: When interrupts are silently
 ignored due to priority conflicts or internal errors without
 notifying higher-level control, it becomes challenging to diagnose
 system failures or detect potential security breaches in a timely
 manner. Network Interface Controllers: Dropping packets - perhaps due to
 buffer overflows - without any error feedback can not only cause data
 loss but may also contribute to exploitable timing discrepancies
 that reveal sensitive internal processing details.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** [REF-1468]: Open source silicon root of trust (RoT) product does not immediately report when an integrity check fails for memory requests, causing the product to accept and continue processing data [REF-1468]

## Modes of Introduction

**•** Architecture and Design: This weakness can be introduced during the architecture and
design phase when the system does not incorporate proper mechanisms
for error reporting or feedback for discarded operations, such as when
handling reserved addresses or unexecuted instructions.

**•** Implementation: It can also arise during implementation if developers fail to
include appropriate feedback or logging for critical operations. This
leads to silent failures in certain scenarios like interrupt handling
or network buffer overflows.

**•** Requirements: A further layer of complexity emerges when considering
specifications. The weakness may stem either from ambiguous product
design specifications that fail to delineate when feedback should
occur or from implementations that do not adhere to existing
requirements. In either case, the result is the same: feedback that is
critical for detecting operational failures or security breaches is
missing.

## Common Consequences

**•** Impact: Read Memory, Read Files or Directories — Notes:

**•** Impact: Modify Memory, Modify Files or Directories — Notes:

**•** Impact: DoS: Resource Consumption (Memory), DoS: Crash, Exit, or Restart — Notes:

## Potential Mitigations

**•** Architecture and Design: Incorporate logging and feedback mechanisms during the
 design phase to ensure proper handling of discarded operations. (Effectiveness: High)

**•** Implementation: Developers should ensure that every critical operation
 includes proper logging or error feedback mechanisms. (Effectiveness: Moderate)

## Applicable Platforms

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

**•** C++ (Class: None, Prevalence: Undetermined)

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

**•** None (Class: Hardware Description Language, Prevalence: Undetermined)

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

## Demonstrative Examples

**•** The omission of feedback for the dropped lower-priority interrupt can
 cause developers to misinterpret the state of the system, leading to
 incorrect assumptions and potential system failures, such as missed
 sensor readings. Attackers might leverage this lack of visibility to induce conditions
 that lead to timing side-channels. For example, an attacker could
 intentionally flood the system with high-priority interrupts, forcing
 the system to discard lower-priority interrupts consistently. If these
 discarded interrupts correspond to processes executing critical
 security functions (e.g., cryptographic key handling), an attacker
 might measure system timing variations to infer when and how those
 functions are executing. This creates a timing side channel that could
 be used to extract sensitive information. Moreover, since these
 lower-priority interrupts are not reported, the system remains unaware
 that critical tasks such as sensor data collection or maintenance
 routines, are being starved of execution. Over time, this can lead to
 functional failures or watchdog time resets in real-time systems. One way to address this problem could be to use structured logging to
 provide visibility into discarded interrupts. This allows
 administrators, developers, or other authorized entities to track
 missed interrupts and optimize the system.

**•** For system security, if an uncorrectable error occurs but is not
 reported to the execution core and handled before the core attempts to
 consume the data that is read/written through the corrupted
 transactions, then this could enable silent data corruption (SDC)
 attacks. In the case of confidential compute technologies where system firmware
 is not a trusted component, error handling controls can be
 misconfigured to trigger this weakness and attack the assets protected
 by confidential compute.