ISO/IEC JTC 1/SC 22/WG 23 N 0328

Revised proposed rewrite of NZN

Date	2011-03-25
Contributed by	Bob Karlin
Original file name	
Notes	Revision of N0319. Also see N0329

6.36 Returning Status [NZN]

6.36.1 Description of application vulnerability

Programming languages provide multiple mechanisms for the reporting the success or failure of a process or operation. When the returned information is insufficient or misleading it can mask other vulnerabilities, and lead to corrupt data and incorrect process flow.

6.36.2 Cross reference

6.36.3 Mechanism of failure

Some languages provide mechanisms to return the failure of a process or operation by raising exceptions, creating and raising error objects, or setting of environmental variables or semaphores. If none of these mechanisms exist, processes and operations can return error status, or set global variables to indicate that an error occurred. The mechanisms of failure include:

- not properly transmitting error conditions to the calling routines
- not propagating errors that occur within the process or operation
- improperly documenting error return codes, variables, or exception conditions
- not providing enough status information to determine the nature of an error

6.36.4 Applicable language characteristics

This vulnerability is endemic to all languages, to one extent or another. Those languages that do not provide a language defined method of returning errors are more vulnerable to improper documentation and insufficient status information.

6.36.5 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Always return status, in some form or another
- Always check status from any subroutine or subprogram, and propagate the error to the calling process

  NOTE In many cases it is appropriate to propagate a status that informs the calling process of an error that occurred even if that error has been corrected within the called process.

- When creating error statuses, be consistent throughout the project, and provide enough detail to determine what caused the error
- When a language provided error mechanism does not allow the granularity or environmental detail to properly debug the error, create global or status arrays to return that data

6.36.6 Implications for standardization

In future standardization activities, the following items should be considered:

- Providing language features to return error status
- Providing language features that allow extended environmental data that is relevant to the error, such as data in error, data base record number, data items that may be affected, etc.

6.nn Handling Exception Conditions

6.nn.1 Description of application vulnerability [???]

Most software environments provide a method for detecting conditions that do not correspond to those expected. These may be environment related, such as hardware failures, or unexpected communication results, data related, such as a non-numeric item input to a numeric calculation, or software related, such as unexpected numeric overflows, or improper index calculations. Many other vulnerabilities are detectable within the software environment. Ignoring this detection, or providing a generic error handling routine may lead to corrupt code, and may provide a vehicle for malicious attack.

6.nn.2 Cross reference

6.nn.3 Mechanism of failure

Improperly handling exception conditions can provoke failure in a number of ways:

- Unhandled exception conditions may end up being ignored, thereby corrupting data or process flow.
- Unhandled exception conditions may be considered fatal by the operating environment, thereby terminating execution without finalizing the process, leaving open or corrupt

databases, other processes blocked while waiting for results, or resources left in a locked state.

- Generically handled exceptions may not capture appropriate environmental and program state to allow the error to be properly investigated
- Generically handled exceptions may not free local resources properly
- Generically handled exceptions may not be able to correct a specific correctable error and return to processing
- Some error handling mechanisms may allow exiting an error-handling subroutine without correcting an error, or by directly exiting to an inappropriate target address.

6.nn.4 Applicable language characteristics

Different programming languages handle error conditions in a number of manners. The main error handling methodologies are:

- Try-Catch uses a block structure to "catch" errors that occur with the block.
- Error-Condition sets a predefined error condition code that can be checked with a conditional expression
- Error-Status sets a user defined variable to a predefined code that specifies the error
- Error-Object creates an object that defines the error, along with environmental conditions
- Declaratives are routines that will be executed when particular error conditions occur.

6.nn.5 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

• Always check error status and error conditions.

• Check errors with the smallest possible granularity. Keep the block that is tested to the smallest number of operations.

• Always close and release any open resources during error processing when the error is uncorrectable

• When an error is uncorrectable, propagate the error upwards when possible, after cleaning up open resources

• When an error is corrected, propagate a successful status that indicates that an error was corrected, whenever possible.

• Never disable error handling without extreme provocation.

• When multiple error handling methodologies are provided, try to standardize on a single methodology for the project.

6.nn.6 Implications for standardization

In future standardization activities, the following items should be considered:

• A standardized set of mechanisms for detecting and treating error conditions should be developed so that all languages to the extent possible could use them. This does not mean that all languages should use the same mechanisms as there should be a variety, but each of the mechanisms should be standardized.