ISO/IEC JTC 1/SC 22/WG 23 N 0313

Proposed vulnerability descriptions YUK and SUK

Date	March 21, 2011
Contributed by	Erhard Ploedereder
Original file name	AI-16-06-YUK-and-SUC.doc
Notes	Responds to AI 16-06

I wrote up two vulnerabilites instead of one.

The first one deals with the suppression of runtime checks (as I was tasked to do).

The second one deals with the de-facto suppression of compile-time checks and with inherently unsafe operations that the language might provide.

I simply could not find a good way of combining all three in a single vulnerability, although they are of the same general ilk. All attempts ended in complexity of description.

Suppression of Language-Defined Run-Time Checking (YUK)

Description of application vulnerability

Some languages include the provision for runtime checking to prevent vulnerabilities to arise. Canonical examples are bounds or length checks on array operations or null-value checks upon dereferencing pointers or references. In most cases, the reaction to a failed check is the raising of a language-defined exception.

As run-time checking requires execution time and as some project guidelines exclude the use of exceptions, languages may define a way to optionally suppress such checking for regions of the code or the entire program. Analogously, compiler options may be used to achieve this effect.

Cross reference

Mechanism of Failure

The vulnerabilities that should have been prevented by the checks re-emerge whenever the suppressed checks would have failed. For their description, see the respective subsections.

Applicable language characteristics

This vulnerability description is intended to be applicable to languages with the following characteristics:

- Languages that define runtime checks to prevent certain vulnerabilities and

- Languages that allow the above checks to be suppressed, or

- Languages, whose compilers or interpreters provide options to omit the above checks

Avoiding the vulnerability

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

- Do not suppress checks or restrict such suppression to the most performance-critical sections of the code.

- Where checks are suppressed, verify that the suppressed checks could not have failed.

- Clearly identify code sections where checks are suppressed.

- Do not assume that checks in code verified to satisfy all checks could not fail nevertheless due to hardware faults.

Provision of Inherently Unsafe Operations (SUK)

Description of application vulnerability

Languages define semantic rules to be obeyed by legal programs. Compilers enforce these rules and reject violating programs.

A canonical example are the rules of type checking, intended among other reasons to prevent semantically incorrect assignments, such as characters to pointers, meter to feet, euro to dollar, real numbers to booleans, or complex numbers to two-dimensional coordinates.

Yet, occasionally there arises a need to step outside the rules of the type model to achieve needed functionally. A typical such situation is the casting of memory as part of the implementation of a heap allocator to the type of object for which the memory is allocated. A type-safe assignment is impossible for this functionality. Thus, a capability for unchecked "type casting" between arbitrary types to interpret the bits in a different fashion is a necessary but inherently unsafe operation, without which the type-safe allocator cannot be programmed.

Another example is the provision of operations known to be inherently unsafe, such as the deallocation of heap memory without prevention of dangling references.

A third example is any interfacing with another language, since the checks ensuring type-safeness rarely extend across language boundaries.

These inherently unsafe operations constitute a vulnerability, since they can (and will) be used by programmers in situations where their use is neither necessary nor appropriate. As the knowledge of the programmer about implementation details may be incomplete or incorrect, unintended execution semantics may result.

The vulnerability is eminently exploitable to violate program security.

Cross reference

Mechanism of Failure

Suppression of checks of the use of inherently unsafe operations circumvents the checks that are normally applied to ensure safe execution. Control flow, data values, and memory accesses can be corrupted as a consequence. See the respective vulnerabilities resulting from such corruption.

Applicable language characteristics

This vulnerability description is intended to be applicable to languages with the following characteristics:

- Languages that allow compile-time checks for the prevention of vulnerabilities to be suppressed by compiler or interpreter options or by language constructs, or

- Languages that provide inherently unsafe operations

Avoiding the vulnerability

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

- Restrict the suppression of compile-time checks to where the suppression is functionally essential.

- Use inherently unsafe operations only when they are functionally essential.

- Clearly identify program code that suppresses checks or uses unsafe operations.