Scope

The CERT C++ Coding Standard was developed specifically for versions of the C++ programming language defined by the following:


Although the guidelines for this standard were developed for C++14, they can also be applied to earlier versions of the C++ programming language, including C++11. Variations between versions of the C++ Standard that would affect the proper application of these guidelines are noted where applicable.

Most guidelines have a noncompliant code example that is a C++14-conforming program to ensure that the problem identified by the guideline is within the scope of the standard. Most guidelines also include a compliant solution that applies to multiple platforms. Language and library extensions that have been published as ISO/IEC technical reports or technical specifications are frequently given precedence. Occasionally, interesting or illustrative implementation-specific behaviors are described.

Rationale

A coding standard for the C++ programming language can create the highest value for the longest period of time by focusing on the C++ Standard (C++14) and any relevant post-C++14 technical reports.

The C++ Standard documents existing practice where possible. That is, most features must be tested in an implementation before being included in the standard. The CERT C++ Coding Standard has a secondary purpose: to establish a set of best practices, which sometimes requires introducing new practices that may not be widely known or used when existing practices are inadequate. To put it a different way, the CERT C++ Coding Standard attempts to drive change rather than just document it.

Some vendors have extensions to C++, and some have implemented only part of the C++ Standard before stopping development. Consequently, it is not possible to discuss only C++98, C++03, or C++11. The vendor support equation is too complicated to draw a line and say that a certain compiler supports exactly a certain standard. Whatever demarcation point is selected, different vendors are on opposite sides of it for different parts of the language. Supporting all possibilities would require testing the cross-product of each compiler with each language feature. Consequently, we have selected a demarcation point that is the most recent in time so that the rules and recommendations defined by the standard will be applicable for as long as possible.

As a result of the variations in support, source-code portability is enhanced when the programmer uses only the features specified by C++14. This is one of many trade-offs between security and portability inherent to C++ language programming.

The value of forward-looking information increases with time before it starts to decrease. The value of backward-looking information starts to decrease immediately.

For all of these reasons, the priority of this standard is to support new code development using C++14. A close-second priority is supporting remediation of old code using C++11.

This coding standard does make contributions to support older compilers when these contributions can be significant and doing so does not compromise other priorities. The intent is not to capture all deviations from the standard but to capture only a few important ones.

Issues Not Addressed

A number of issues are not addressed by this secure coding standard.

Coding Style

Coding style issues are subjective, and it has proven impossible to develop a consensus on appropriate style guidelines. Consequently, the CERT C++ Secure Coding Standard does not require the enforcement of any particular coding style but only suggests that development organizations define or adopt style guidelines and apply these guidelines consistently. The easiest way to apply a coding style consistently is to use a code-formating tool. Many interactive development environments (IDEs) provide such capabilities.

Controversial Rules

In general, the CERT coding standards try to avoid the inclusion of controversial rules that lack a broad consensus.