How this Coding Standard Is Organized

- Identifiers
- Noncompliant Code Examples and Compliant Solutions
- Exceptions
- Risk Assessment
- Automated Detection
- Related Vulnerabilities
- Related Guidelines
- Bibliography

This coding standard is organized into 11 chapters that contain rules in specific topic areas followed by four appendices. Appendix A contains the bibliography. Appendix B lists the definitions of terms used throughout the standard. Appendix C lists the relationships between the rules of this standard and two other related guidelines: MISRA C++ 2008: Guidelines for the Use of the C++ Language in Critical Systems [MISRA 2008] and MITRE's Common Weakness Enumeration (CWE) [MITRE 2010]. Appendix D lists the risk assessments associated with all of the rules in the coding standard.

Most rules have a consistent structure. Each rule in this standard has a unique identifier, which is included in the title. The title and the introductory paragraphs define the rule and are typically followed by one or more pairs of noncompliant code examples and compliant solutions. Each rule also includes a risk assessment, related guidelines, and a bibliography (where applicable). Rules may also include a table of related vulnerabilities.

Identifiers

Each rule and recommendation (see Rules Versus Recommendations) is given a unique identifier. These identifiers consist of three parts:

- A three-letter mnemonic representing the section of the standard
- A two-digit numeric value in the range of 00 to 99
- A suffix that represents the associated language or platform.
  - "-C" for the SEI CERT C Coding Standard
  - "-CPP" for the SEI CERT C++ Coding Standard
  - "-J" for the SEI CERT Oracle Coding Standard for Java
  - "-PL" for the SEI CERT Perl Coding Standard

The three-letter mnemonic is used to group similar coding practices and to indicate which category a coding practice belongs to.

The numeric value is used to give each coding practice a unique identifier. Numeric values in the range of 00 to 49 are reserved for recommendations, and values in the range of 50 to 99 are reserved for rules. (The values used for the SEI CERT C Coding Standard are different.) Rules and recommendations are frequently referenced from the guidelines in this standard by their identifier and title.

Here are some example identifiers with an explanation of each:

- INT50-CPP Do not cast to an out-of-range enumeration value
  - This identifier indicates a rule
  - "INT" stands for the Integer category
  - "50" is the unique identifier
  - "-CPP" stands for the C++ language
- EXP00-J Do not ignore values returned by methods
  - This identifier indicates a rule
  - "EXP" stands for the Expressions category
  - "00" is the unique identifier
  - "-J" stands for the Java language
- FLP00-C. Understand the limitations of floating-point numbers
  - This identifier indicates a recommendation
  - "FLP" stands for the Floating Point category
  - "00" is the unique identifier
  - "-C" stands for the C programming language

Noncompliant Code Examples and Compliant Solutions

Noncompliant code examples illustrate code that violates the guideline under discussion. It is important to note that these are only examples, and eliminating all occurrences of the example does not necessarily mean that the code being analyzed is now compliant with the guideline.

Noncompliant code examples are typically followed by compliant solutions, which show how the noncompliant code example can be recoded in a secure, compliant manner. Except where noted, noncompliant code examples should contain violations only of the guideline under discussion. Compliant solutions should comply with all of the secure coding rules but may on occasion fail to comply with a recommendation.

Exceptions

Any rule or recommendation may specify a small set of exceptions detailing the circumstances under which the guideline is not necessary to ensure the safety, reliability, or security of software. Exceptions are informative only and are not required to be followed.

Risk Assessment
Each guideline in the CERT C++ Coding Standard contains a risk assessment section that attempts to provide software developers with an indication of the potential consequences of not addressing a particular rule or recommendation in their code (along with some indication of expected remediation costs). This information may be used to prioritize the repair of rule violations by a development team. The metric is designed primarily for remediation projects. It is generally assumed that new code will be developed to be compliant with the entire coding standard and applicable recommendations.

Each rule and recommendation has an assigned priority. Priorities are assigned using a metric based on Failure Mode, Effects, and Criticality Analysis (FMECA) [IEC 60812]. Three values are assigned for each rule on a scale of 1 to 3 for severity, likelihood, and remediation cost.

**Severity**—How serious are the consequences of the rule being ignored?

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Examples of Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Denial-of-service attack, abnormal termination</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>Data integrity violation, unintentional information disclosure</td>
</tr>
<tr>
<td>3</td>
<td>High</td>
<td>Run arbitrary code</td>
</tr>
</tbody>
</table>

**Likelihood**—How likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability?

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unlikely</td>
</tr>
<tr>
<td>2</td>
<td>Probable</td>
</tr>
<tr>
<td>3</td>
<td>Likely</td>
</tr>
</tbody>
</table>

**Remediation Cost**—How expensive is it to comply with the rule?

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Detection</th>
<th>Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>High</td>
<td>Manual</td>
<td>Manual</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>Automatic</td>
<td>Manual</td>
</tr>
<tr>
<td>3</td>
<td>Low</td>
<td>Automatic</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

The three values are then multiplied together for each rule. This product provides a measure that can be used in prioritizing the application of the rules. The products range from 1 to 27, although only the following 10 distinct values are possible: 1, 2, 3, 4, 6, 8, 9, 12, 18, and 27. Rules and recommendations with a priority in the range of 1 to 4 are Level 3 rules, 6 to 9 are Level 2, and 12 to 27 are Level 1. The following are possible interpretations of the priorities and levels.

**Priorities and Levels**

<table>
<thead>
<tr>
<th>Level</th>
<th>Priorities</th>
<th>Possible Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1, 12, 18, 27</td>
<td>High severity, likely, inexpensive to repair</td>
</tr>
<tr>
<td>L2</td>
<td>6, 8, 9</td>
<td>Medium severity, probable, medium cost to repair</td>
</tr>
<tr>
<td>L3</td>
<td>1, 2, 3, 4</td>
<td>Low severity, unlikely, expensive to repair</td>
</tr>
</tbody>
</table>

Specific projects may begin remediation by implementing all rules at a particular level before proceeding to the lower priority rules, as shown in the following illustration:
Recommendations are not compulsory and are provided for information purposes only.

Automated Detection

On the wiki, both rules and recommendations frequently have sections that describe automated detection. These sections provide additional information on analyzers that can automatically diagnose violations of coding guidelines. Most automated analyses for the C++ programming language are neither sound nor complete, so the inclusion of a tool in this section typically means that the tool can diagnose some violations of this particular rule. The Secure Coding Validation Suite can be used to test the ability of analyzers to diagnose violations of rules from ISO/IEC TS 17961:2013, which is related to the rules in the SEI CERT C Coding Standard.

The information in the automated detection sections on this wiki may be

- provided by the vendors
- determined by CERT by informally evaluating the analyzer
- determined by CERT by reviewing the vendor documentation

Where possible, we try to reference the exact version of the tool for which the results were obtained. Because these tools evolve continuously, this information can rapidly become dated and obsolete.

Related Vulnerabilities

The risk assessment sections on the wiki also contain a link to search for related vulnerabilities on the CERT website. Whenever possible, CERT Vulnerability Notes are tagged with a keyword corresponding to the unique ID of the coding guideline. This search provides you with an up-to-date list of real-world vulnerabilities that have been determined to be at least partially caused by a violation of this specific guideline. These vulnerabilities are labeled as such only when the vulnerability analysis team at the CERT/CC is able to evaluate the source code and precisely determine the cause of the vulnerability. Because many vulnerability notes refer to vulnerabilities in closed-source software systems, it is not always possible to provide this additional analysis. Consequently, the related vulnerabilities field tends to be somewhat sparsely populated.

Related vulnerability sections are included only for specific rules in this standard, when the information is both relevant and interesting.

Related Guidelines


You can create a unique URL to get more information on CWEs by appending the relevant ID to the end of a fixed string. For example, to find more information about "CWE-192: Integer Coercion Error," you can append 192.html to http://cwe.mitre.org/data/definitions/ and enter the resulting URL in your browser: http://cwe.mitre.org/data/definitions/192.html.

The other referenced technical specifications, technical reports, and guidelines are commercially available.
Bibliography

Most guidelines have a small bibliography section that lists documents and sections in those documents that provide information relevant to the guideline.