Introduction

This coding standard consists of *rules* and *recommendations*, collectively referred to as *guidelines*. Rules are meant to provide normative requirements for code, whereas recommendations are meant to provide guidance that, when followed, should improve the safety, reliability, and security of software systems. However, a violation of a recommendation does not necessarily indicate the presence of a defect in the code.

Rules

Rules must meet the following criteria:

1. Violation of the guideline is likely to result in a defect that may adversely affect the safety, reliability, or security of a system, for example, by introducing a **security flaw** that may result in an exploitable **vulnerability**.
2. The guideline does not rely on source code annotations or assumptions.
3. Conformance to the guideline can be determined through automated analysis (either static or dynamic), formal methods, or manual inspection techniques.

Rules are identified by the label *rule*.

Recommendations

Recommendations are suggestions for improving code quality. Guidelines are defined to be recommendations when all of the following conditions are met:

1. Application of a guideline is likely to improve the safety, reliability, or security of software systems.
2. One or more of the requirements necessary for a guideline to be considered a rule cannot be met.

The set of recommendations that a particular development effort adopts depends on the requirements of the final software product. Projects with stricter requirements may decide to dedicate more resources to ensuring the safety, reliability, and security of a system and consequently are likely to adopt a broader set of recommendations.

Recommendations are identified by the label *recommendation*.

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.

Identifiers

Each rule and recommendation is given a unique identifier. These identifiers consist of three parts:
Supported Languages and Platforms

See the table below for a summary of supported languages and platforms:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Language/Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>-C</td>
<td>C</td>
</tr>
<tr>
<td>-CPP</td>
<td>C++</td>
</tr>
<tr>
<td>-J</td>
<td>Java</td>
</tr>
<tr>
<td>-PL</td>
<td>Perl</td>
</tr>
</tbody>
</table>

Examples

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 recommendation
  
  “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