MSC00-C. Compile cleanly at high warning levels

Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.

According to the C Standard, subclause 5.1.1.3 [ISO/IEC 9899:2011].

A conforming implementation shall produce at least one diagnostic message (identified in an implementation-defined manner) if a preprocessing translation unit or translation unit contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as undefined or implementation-defined. Diagnostic messages need not be produced in other circumstances.

Assuming a conforming implementation, eliminating diagnostic messages will eliminate any syntactic or constraint violations.

If suitable source-code-checking tools are available, use them regularly.

Compilers can produce diagnostic messages for correct code, as is permitted by C. It is usually preferable to rewrite code to eliminate compiler warnings, but if the code is correct, it is sufficient to provide a comment explaining why the warning message does not apply. Some compilers provide ways to suppress warnings, such as suitably formatted comments or pragmas, which can be used sparingly when the programmer understands the implications of the warning and the code is correct.

Do not simply quiet warnings by adding type casts or other means. Instead, understand the reason for the warning and consider a better approach, such as using matching types and avoiding type casts whenever possible.

Noncompliant Code Example (Windows)

Using the default warning specifier with #pragma warning resets the behavior of a warning to its default value, which may not be the same as its previous behavior. Programmers commonly, but incorrectly, use the default warning specifier to restore previous warning messages after a message is temporarily disabled.

```plaintext
#pragma warning(disable:4705)
#pragma warning(disable:4706)
#pragma warning(disable:4707)
/* Unnecessarily flagged code */
#pragma warning(default:4705)
#pragma warning(default:4706)
#pragma warning(default:4707)
```

Compliant Solution (Windows)

Instead of using the default warning specifier, the current state of the warnings should be saved and then restored after the unnecessarily flagged code.

```plaintext
#pragma warning(push)
#pragma warning(disable:4705)
#pragma warning(disable:4706)
#pragma warning(disable:4707)
/* Unnecessarily flagged code */
#pragma warning(pop)
```

The `#pragma warning(push)` stores the current warning state for every warning. The `#pragma warning(pop)` pops the last warning state pushed onto the stack. Any changes made to the warning state between the push and pop are undone.

Risk Assessment

Eliminating violations of syntax rules and other constraints can eliminate serious software vulnerabilities that can lead to the execution of arbitrary code with the permissions of the vulnerable process.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC00-C</td>
<td>Medium</td>
<td>Probable</td>
<td>Medium</td>
<td>P8</td>
<td>L2</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Automated Detection
<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSonar</td>
<td>6.0p0</td>
<td>BUILD.WALL, BUILD.WERROR</td>
<td>Not All Warnings Are Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Warnings Not Treated As Errors</td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.13</td>
<td>V665</td>
<td></td>
</tr>
<tr>
<td>SonarQube C/C++ Plugin</td>
<td>3.11</td>
<td>S1762, S973</td>
<td>Warns when the default warning specifier is used with #pragma warning.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Requires documentation of #pragma uses</td>
</tr>
</tbody>
</table>

**Related Guidelines**

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID MSC00-CPP. Compile cleanly at high warning levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITRE CWE</td>
<td>CWE-563, Unused variable, CWE-570, Expression is always false, CWE-571, Expression is always true</td>
</tr>
</tbody>
</table>

**Bibliography**

- ISO/IEC 9899:2011: Section 5.1.1.3, "Diagnostics"
- Seacord 2013: Chapter 9, "Recommended Practices"
- Sutter 2005: Item 1