MSC04-C. Use comments consistently and in a readable fashion

Noncompliant Code Example

Do not use the character sequence /* within a comment:

```c
/* Comment with end comment marker unintentionally omitted
security_critical_function(); /* Some other comment */
```

In this example, the call to the security-critical function is not executed. A reviewer examining this page could incorrectly assume that the code is executed.

If execution failure is the result of an accidental omission, it is useful to use an editor that provides syntax highlighting or formats the code to help identify issues like missing end-comment delimiters.

Because missing end delimiters are error prone and often viewed as a mistake, this approach is not recommended for commenting out code.

Compliant Solution (Preprocessor)

Instead of using /* and */ to comment out blocks of code, use conditional compilation (for example, #if, #ifdef, or #ifndef):

```c
#if 0 /*
 * Use of critical security function no
 * longer necessary.
 */
security_critical_function(); /* Some other comment */
#endif
```

The text inside a block of code commented out using #if, #ifdef, or #ifndef must still consist of valid preprocessing tokens. This means that the characters * and ' must each be paired just as in real C code, and the pairs must not cross line boundaries. In particular, an apostrophe within a contracted word looks like the beginning of a character constant. Consequently, natural-language comments and pseudocode should always be written between the comment delimiters /* and */ or following //.

Compliant Solution (Compiler)

This compliant solution takes advantage of the compiler's ability to remove unreachable (dead) code. The code inside the if block must remain acceptable to the compiler. If other parts of the program, such as macros, types, or function prototypes, later change in a way that would cause syntax errors, the unexecuted code must be brought up to date to correct the problem. Then, if it is needed again in the future, the programmer need only remove the surrounding if statement and the NOTREACHED comment.

The NOTREACHED comment tells some compilers and static analysis tools not to complain about this unreachable code. It also serves as documentation.

```c
if (0) {
    /*
     * Use of critical security function no
     * longer necessary, for now.
     */
    /*NOTREACHED*/
    security_critical_function();
    /* Some other comment */
}
```

This code is an instance of exception MSC07-C-EX2 to MSC07-C. Detect and remove dead code.

Noncompliant Code Example

Following are some additional examples of comment styles that are confusing and should be avoided:
Compliant Solution

Use a consistent style of commenting:

```c
/* Nice simple comment */

int i; /* Counter */
```

Risk Assessment

Confusion over which instructions are executed and which are not can lead to serious programming errors and vulnerabilities, including denial of service, abnormal program termination, and data integrity violation. This problem is mitigated by the use of interactive development environments (IDEs) and editors that use fonts, colors, or other mechanisms to differentiate between comments and code. However, the problem can still manifest, for example, when reviewing source code printed on a black-and-white printer.

<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>MSC04-C</td>
<td>Medium</td>
<td>Unlikely</td>
<td>Medium</td>
<td>P4</td>
<td>L3</td>
</tr>
</tbody>
</table>

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrée</td>
<td>19.04</td>
<td>mmline-comment</td>
<td>Partially checked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>smline-comment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>smline-comment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sline-splicing</td>
<td></td>
</tr>
<tr>
<td>GCC</td>
<td>4.3.5</td>
<td></td>
<td>Can detect violations of this rule when the –Wcomment flag is used</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.MSC04</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>119 S, 302 S, 611 S</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-MSC04-a, CERT_C-MSC04-b, CERT_C-MSC04-c, CERT_C-MSC04-d</td>
<td>The character sequence /* shall not be used within a C-style comment Line-splicing shall not be used in // comments</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT_C: Rec. MSC04-C</td>
<td>Checks for use of /* and // within a comment (rule partially covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>3108</td>
<td></td>
</tr>
</tbody>
</table>
RuleChecker 19.04 | mmline-comment smline-comment smline-comment sline-splicing | Partially checked

Related Vulnerabilities

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

Related Guidelines

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID MSC04-CPP. Use comments consistently and in a readable fashion</th>
</tr>
</thead>
</table>

Bibliography

[Summit 2005] Question 11.19