EXP19-C. Use braces for the body of an if, for, or while statement

Opening and closing braces for if, for, and while statements should always be used even if the statement's body contains only a single statement.

If an if, while, or for statement is used in a macro, the macro definition should not conclude with a semicolon. (See PRE11-C. Do not conclude macro definitions with a semicolon.)

Braces improve the uniformity and readability of code. More important, when inserting an additional statement into a body containing only a single statement, it is easy to forget to add braces because the indentation gives strong (but misleading) guidance to the structure.

Braces also help ensure that macros with multiple statements are properly expanded. Such a macro should be wrapped in a do-while loop. (See PRE10-C. Wrap multistatement macros in a do-while loop.) However, when the do-while loop is not present, braces can still ensure that the macro expands as intended.

Noncompliant Code Example

This noncompliant code example uses an if statement without braces to authenticate a user:

```c
int login;
if (invalid_login())
    login = 0;
else
    login = 1;
```

A developer might add a debugging statement to determine when the login is valid but forget to add opening and closing braces:

```c
int login;
if (invalid_login())
    login = 0;
else
    printf("Login is valid\n");  /* Debugging line added here */
    login = 1;                   /* This line always gets executed
    /* regardless of a valid login! */
```

Because of the indentation of the code, it is difficult to tell that the code will not function as intended by the programmer, potentially leading to a security breach.

Compliant Solution

In the compliant solution, opening and closing braces are used even when the body is a single statement:

```c
int login;
if (invalid_login()) {
    login = 0;
} else {
    login = 1;
}
```

Noncompliant Code Example

This noncompliant code example has an if statement nested in another if statement without braces around the if and else bodies:
int privileges;
if (invalid_login())
  if (allow_guests())
    privileges = GUEST;
else
  privileges = ADMINISTRATOR;

The indentation could lead the programmer to believe that a user is given administrator privileges only when the user's login is valid. However, the else statement actually attaches to the inner if statement:

int privileges;
if (invalid_login())
  if (allow_guests())
    privileges = GUEST;
else
  privileges = ADMINISTRATOR;

This is a security loophole: users with invalid logins can still obtain administrator privileges.

Compliant Solution
In the compliant solution, adding braces removes the ambiguity and ensures that privileges are correctly assigned:

int privileges;
if (invalid_login()) {
  if (allow_guests()) {
    privileges = GUEST;
  } else {
    privileges = ADMINISTRATOR;
  }
}

Noncompliant Code Example (empty block)
This noncompliant code example has a while statement with no block:

while (invalid_login());

Note that if invalid_login() has no side effects (such as warning the user if their login failed), this code also violates MSC12-C. Detect and remove code that has no effect or is never executed.

Compliant Solution (empty block)
This compliant solution features an explicit empty block, which clarifies the developer's intent:

while (invalid_login()) {}
### Table

<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>compound-ifelse</td>
<td>Fully checked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compound-loop</td>
<td></td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-EXP19</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>MISRA_IF_NO_COMPOUND MISRA_STMT. NO_COMPOUND</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>11 S, 12 S, 428 S</td>
<td>Fully Implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-EXP19-a</td>
<td>The statement forming the body of a 'switch', 'while', 'do...while' or 'for' statement shall be a compound statement</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rec. EXP19-C</td>
<td>Checks for iteration or selection statement body not enclosed in braces (rec. fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>2212</td>
<td></td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V563, V628, V640, V705</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>compound-ifelse</td>
<td>Fully checked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compound-loop</td>
<td></td>
</tr>
<tr>
<td>SonarQube C/C++ Plugin</td>
<td>3.11</td>
<td>S121</td>
<td></td>
</tr>
</tbody>
</table>

### Related Vulnerabilities

CVE-2014-1266 was due, in large part, to failing to follow this recommendation. There is a spurious "goto fail" statement on line 631 of `sslKeyExchange.c`. This "goto" gets executed unconditionally, even though it is indented as if it were part of the preceding "if" statement. As a result, the call to `sslRawVerify` (which performs the actual signature verification) is rendered dead code. [ImperialViolet 2014]. If the body of the "if" statement had been enclosed in braces, then this defect likely would not have happened.

### Related Guidelines

| MISRA C:2012 | Rule 15.6 (required) |

### Bibliography

- [GNU 2010] Coding Standards, Section 5.3, "Clean Use of C Constructs"