**EXP45-C. Do not perform assignments in selection statements**

Do not use the assignment operator in the contexts listed in the following table because doing so typically indicates programmer error and can result in unexpected behavior.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>Controlling expression</td>
</tr>
<tr>
<td>while</td>
<td>Controlling expression</td>
</tr>
<tr>
<td>do ... while</td>
<td>Controlling expression</td>
</tr>
<tr>
<td>for</td>
<td>Second operand</td>
</tr>
<tr>
<td>?:</td>
<td>First operand</td>
</tr>
<tr>
<td>??</td>
<td>Second or third operands, where the ternary expression is used in any of these contexts</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Either operand</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>Second operand, when the comma expression is used in any of these contexts</td>
</tr>
</tbody>
</table>

Performing assignment statements in other contexts do not violate this rule. However, they may violate other rules, such as **EXP30-C. Do not depend on the order of evaluation for side effects**.

**Noncompliant Code Example**

In this noncompliant code example, an assignment expression is the outermost expression in an `if` statement:

```c
if (a = b) {
    /* ... */
}
```

Although the intent of the code may be to assign `b` to `a` and test the value of the result for equality to 0, it is frequently a case of the programmer mistakenly using the assignment operator `=` instead of the equals operator `==`. Consequently, many compilers will warn about this condition, making this coding error detectable by adhering to **MSC00-C. Compile cleanly at high warning levels**.

**Compliant Solution (Unintentional Assignment)**

When the assignment of `b` to `a` is not intended, the conditional block is now executed when `a` is equal to `b`:

```c
if (a == b) {
    /* ... */
}
```

**Compliant Solution (Intentional Assignment)**

When the assignment is intended, this compliant solution explicitly uses inequality as the outermost expression while performing the assignment in the inner expression:

```c
if ((a = b) != 0) {
    /* ... */
}
```
It is less desirable in general, depending on what was intended, because it mixes the assignment in the condition, but it is clear that the programmer intended the assignment to occur.

**Noncompliant Code Example**

In this noncompliant code example, the expression $x = y$ is used as the controlling expression of the `while` statement:

```c
    do { /* ... */ } while (foo(), x = y);
```

The same result can be obtained using the `for` statement, which is specifically designed to evaluate an expression on each iteration of the loop, just before performing the test in its controlling expression:

```c
    for (; x; foo(), x = y) { /* ... */ }
```

**Compliant Solution (Unintentional Assignment)**

When the assignment of $y$ to $x$ is not intended, the conditional block should be executed only when $x$ is equal to $y$, as in this compliant solution:

```c
    do { /* ... */ } while (foo(), x == y);
```

**Compliant Solution (Intentional Assignment)**

When the assignment is intended, this compliant solution can be used:

```c
    do { /* ... */ } while (foo(), (x = y) != 0);
```

**Noncompliant Code Example**

In this noncompliant example, the expression $p = q$ is used as the controlling expression of the `while` statement:

```c
    do { /* ... */ } while (x = y, p = q);
```

**Compliant Solution**

In this compliant solution, the expression $x = y$ is not used as the controlling expression of the `while` statement:

```c
    do { /* ... */ } while (x = y, p == q);
```

**Noncompliant Code Example**

This noncompliant code example has a typo that results in an assignment rather than a comparison.

```c
    while (ch = '\t' || ch == ' ' || ch == '\n') {
        /* ... */
    }
```

Many compilers will warn about this condition. This coding error would typically be eliminated by adherence to MSC00-C. Compile cleanly at high warning levels. Although this code compiles, it will cause unexpected behavior to an unsuspecting programmer. If the intent was to verify a string such as a password, user name, or group user ID, the code may produce significant vulnerabilities and require significant debugging.

**Compliant Solution (RHS Variable)**

When comparisons are made between a variable and a literal or const-qualified variable, placing the variable on the right of the comparison operation can prevent a spurious assignment.
In this code example, the literals are placed on the left-hand side of each comparison. If the programmer were to inadvertently use an assignment operator, the statement would assign `ch` to `'	'`, which is invalid and produces a diagnostic message.

```c
while ('\t' == ch || ' ' == ch || '\n' == ch) {
    /* ... */
}
```

Due to the diagnostic, the typo will be easily spotted and fixed.

```c
while ('\t' == ch || ' ' == ch || '\n' == ch) {
    /* ... */
}
```

As a result, any mistaken use of the assignment operator that could otherwise create a vulnerability for operations such as string verification will result in a compiler diagnostic regardless of compiler, warning level, or implementation.

Exceptions

**EXP45-C-EX1**: Assignment can be used where the result of the assignment is itself an operand to a comparison expression or relational expression. In this compliant example, the expression `x = y` is itself an operand to a comparison operation:

```c
if ((x = y) != 0) { /* ... */ }
```

**EXP45-C-EX2**: Assignment can be used where the expression consists of a single primary expression. The following code is compliant because the expression `x = y` is a single primary expression:

```c
if ((x = y)) { /* ... */ }
```

The following controlling expression is noncompliant because `&&` is not a comparison or relational operator and the entire expression is not primary:

```c
if ((w = v) && flag) { /* ... */ }
```

When the assignment of `v` to `w` is not intended, the following controlling expression can be used to execute the conditional block when `v` is equal to `w`:

```c
if ((v == w) && flag) { /* ... */ });
```

When the assignment is intended, the following controlling expression can be used:

```c
if (((v = w) != 0) && flag) { /* ... */ });
```

**EXP45-C-EX3**: Assignment can be used in a function argument or array index. In this compliant solution, the expression `x = y` is used in a function argument:

```c
if (foo(x = y)) { /* ... */ }
```

Risk Assessment

Errors of omission can result in unintended program flow.

<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>EXP45-C</td>
<td>Low</td>
<td>Likely</td>
<td>Medium</td>
<td>P6</td>
<td>L2</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>assignment-conditional</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-EXP45</td>
<td></td>
</tr>
<tr>
<td>Clang</td>
<td>3.9</td>
<td>--parentheses</td>
<td>Can detect some instances of this rule, but does not detect all</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>LANG_STRUCTCONDASSIGN LANG_STRUCTSE_COND LANG_STRUCTUSEASSIGN</td>
<td>Assignment in conditional Condition contains side effects Assignment result in expression</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td>Could detect violations of this recommendation by identifying any assignment expression as the top-level expression in an <code>if</code> or <code>while</code> statement</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.EXP18 CC2.EXP21</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>GCC</td>
<td>4.3.5</td>
<td></td>
<td>Can detect violations of this recommendation when the <code>--Wall</code> flag is used</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>ASSIGCOND.CALL ASSIGCOND.GEN MISRA.ASSIGN.COND</td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>114 S, 132 S</td>
<td>Enhanced Enforcement</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT.C-EXP45-a CERT.C-EXP45-b CERT.C-EXP45-c CERT.C-EXP45-d</td>
<td>Avoid conditions that always evaluate to the same value Assignment operators shall not be used in conditions without brackets A function identifier shall only be used with either a preceding <code>&amp;</code>, or with a parenthesised parameter list, which may be empty Assignment operators shall not be used in expressions that yield a Boolean value</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule EXP45-C</td>
<td>Checks for invalid use of <code>=</code> (assignment) operator (rule partially covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>3314, 3326, 3344, 3416</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>PRQA QA-C++</td>
<td>4.4</td>
<td>4071, 4074</td>
<td></td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V559, V633, V699</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>assignment-conditional</td>
<td>Fully checked</td>
</tr>
<tr>
<td>SonarQube C/C++ Plugin</td>
<td>3.11</td>
<td>AssignmentInSubExpression</td>
<td></td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://www.cert.org).

**Related Guidelines**

*Key here* (explains table format and definitions)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Taxonomy Item</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C</td>
<td>EXP19-CPP. Do not perform assignments in conditional expressions</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>CERT Oracle Secure Coding Standard for Java</td>
<td>EXP51-J. Do not perform assignments in conditional expressions</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>ISO/IEC TS 17961</td>
<td>No assignment in conditional expressions [boolasgn]</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>CWE 2.11</td>
<td>CWE-480, Use of Incorrect Operator</td>
<td>2017-07-05: CERT: Rule subset of CWE</td>
</tr>
<tr>
<td>CWE 2.11</td>
<td>CWE-481</td>
<td>2017-07-05: CERT: Rule subset of CWE</td>
</tr>
</tbody>
</table>
CERT-CWE Mapping Notes

Key here for mapping notes

**CWE-480 and EXP45-C**

Intersection( EXP45-C, EXP46-C) = Ø

CWE-480 = Union( EXP45-C, list) where list =
  * Usage of incorrect operator besides s/===/

**CWE-569 and EXP45-C**

CWE-480 = Subset( CWE-569)

Bibliography

| [Dutta 03] | *Best Practices for Programming in C* |
| [Hatton 1995] | Section 2.7.2, *Errors of Omission and Addition* |