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 une xpected behavior.

Operator	Context
`if`	Controlling expression
`while`	Controlling expression
`do ... while`	Controlling expression
`for`	Second operand
`?:`	First operand
`?:`	Second or third operands, where the ternary expression is used in any of these contexts
`&&`	Either operand
`||`	either operand
`,`	Second operand, when the comma expression is used in any of these contexts

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:

```
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:

```
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:

```
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:

```
 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:

```
 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:

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

Compliant Solution (Intentional Assignment)

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

```
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:

```
 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:

```
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.

```
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 `'\t'`, which is invalid and produces a diagnostic message.

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

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

```
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:

```
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:

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

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

```
if ((v = w) && 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`:

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

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

```
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:

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

Risk Assessment

Errors of omission can result in unintended program flow.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
EXP45-C	Low	Likely	Medium	**P6**	**L2**

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**assignment-conditional**	Fully checked
Axivion Bauhaus Suite	6.9.0	**CertC-EXP45**	
Clang	3.9	`-Wparentheses`	Can detect some instances of this rule, but does not detect all
CodeSonar	5.0p0	**LANG.STRUCT.CONDASSIG** **LANG.STRUCT.SE.COND** **LANG.STRUCT.USEASSIGN**	Assignment in conditional Condition contains side effects Assignment result in expression
Compass/ROSE			Could detect violations of this recommendation by identifying any assignment expression as the top-level expression in an `if` or `while` statement
ECLAIR	1.2	**CC2.EXP18** **CC2.EXP21**	Fully implemented
GCC	4.3.5		Can detect violations of this recommendation when the `-Wall` flag is used
Klocwork	2018	**ASSIGCOND.CALL** **ASSIGCOND.GEN** **MISRA.ASSIGN.COND**	
LDRA tool suite	9.7.1	**114 S, 132 S**	Enhanced Enforcement
Parasoft C/C++test	10.4.2	**CERT_C-EXP45-a** **CERT_C-EXP45-b** **CERT_C-EXP45-c** **CERT_C-EXP45-d**	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 '&', or with a parenthesised parameter list, which may be empty Assignment operators shall not be used in expressions that yield a Boolean value
Polyspace Bug Finder	R2018a	Invalid use of = operator	Assignment in conditional statement
PRQA QA-C	9.5	**3314, 3326, 3344, 3416**	Partially implemented
PRQA QA-C++	4.3	**4071, 4074**	
PVS-Studio	6.23	**V559**, **V633**, **V699**	
RuleChecker	19.04	**assignment-conditional**	Fully checked
SonarQube C/C++ Plugin	3.11	**AssignmentInSubExpression**	

Related Vulnerabilities

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

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
CERT C	EXP19-CPP. Do not perform assignments in conditional expressions	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT Oracle Secure Coding Standard for Java	EXP51-J. Do not perform assignments in conditional expressions	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TR 24772:2013	Likely Incorrect Expression [KOA]	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961	No assignment in conditional expressions [boolasgn]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-480, Use of Incorrect Operator	2017-07-05: CERT: Rule subset of CWE
CWE 2.11	CWE-481	2017-07-05: CERT: Rule subset of CWE

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"