MSC01-C. Strive for logical completeness

Software vulnerabilities can result when a programmer fails to consider all possible data states.

Noncompliant Code Example (`if` Chain)

This noncompliant code example fails to test for conditions where a is neither b nor c. This behavior may be correct in this case, but failure to account for all the values of a can result in logic errors if a unexpectedly assumes a different value.

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

Compliant Solution (`if` Chain)

This compliant solution explicitly checks for the unexpected condition and handles it appropriately:

```
if (a == b) {
  /* ... */
}
else if (a == c) {
  /* ... */
}
else {
  /* Handle error condition */
}
```

Noncompliant Code Example (`switch`)

This noncompliant code example fails to consider all possible cases. Failure to account for all valid values of type `Color` will result in a logic error. Because valid values of an enumerated type include all those of its underlying integer type, unless enumeration constants have been provided for all those values, the `default` label is appropriate and necessary.

```
typedef enum { Red, Green, Blue } Color;
const char* f(Color c) {
  switch (c) {
    case Red: return "Red";
    case Green: return "Green";
    case Blue: return "Blue";
  }
}

void g() {
  Color unknown = (Color)123;
  puts(f(unknown));
}
```

Implementation Details

Microsoft Visual C++ .NET with `/W4` does not warn when assigning an integer value to an `enum` type or when the `switch` statement does not contain all possible values of the enumeration.

Compliant Solution (`switch`)

This compliant solution takes care to provide the `default` label to handle all valid values of type `Color`:

```
typedef enum { Red, Green, Blue } Color;
const char* f(Color c) {
  switch (c) {
    case Red: return "Red";
    case Green: return "Green";
    case Blue: return "Blue";
    default: return "Unknown color";   /* Necessary */
  }
}
```

Note that adding a default case to a `switch` statement, even when all possible `switch` labels are specified, is an exception (MSC07-C-EX1) to MSC07-C. Detect and remove dead code.

An alternative compliant solution to the noncompliant code example is to provide a `return` statement after the `switch` statement. Note, however, that this solution may not be appropriate in all situations.

```
typedef enum { Red, Green, Blue } Color;
const char* f(Color c) {
  switch (c) {
    case Red: return "Red";
    case Green: return "Green";
    case Blue: return "Blue";
  }
  return "Unknown color";   /* Necessary */
}
```

Historical Discussion

This practice has been a subject of debate for some time, but a clear direction has emerged.

Originally, the consensus among those writing best practices was simply that each `switch` statement should have a `default` label. Eventually, emerging compilers and static analysis tools could verify that a `switch` on an `enum` type contained a `case` label for each enumeration value, but only if no `default` label existed. This led to a shift toward purposely leaving out the `default` label to allow static analysis. However, the resulting code was then vulnerable to `enum` variables being assigned `int` values outside the set of `enum` values.

These two practices have now been merged. A `switch` on an `enum` type should now contain a `case` label for each `enum` value but should also contain a d`efault` label for safety. This practice does not add difficulty to static analysis.

Existing implementations are in transition, with some not yet analyzing `switch` statements with `default` labels. Developers must take extra care to check their own `switch` statements until the new practice becomes universal.

Noncompliant Code Example (Zune 30)

This noncompliant code example shows incomplete logic when converting dates. The code appeared in the Zune 30 media player, causing many players to lock up on December 30, 2008, at midnight PST. This noncompliant code example comes from the `ConvertDays` function in the real-time clock (RTC) routines for the MC13783 PMIC RTC. It takes the number of days since January 1, 1980, and computes the correct year and number of days since January 1 of the correct year.

The flaw in the code occurs when `days` has the value 366 because the loop never terminates. This bug manifested itself on the 366th day of 2008, which was the first leap year in which this code was active.

```
#define ORIGINYEAR 1980
UINT32 days = /* Number of days since January 1, 1980 */
int year = ORIGINYEAR;
/* ... */

while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  }
  else {
    days -= 365;
    year += 1;
  }
}
```

Compliant Solution (Zune 30)

The following proposed rewrite is provided at http://winjade.net/2009/01/lesson-on-infinite-loops. The loop is guaranteed to exit, because `days` decreases for each iteration of the loop, unless the `while` condition fails and the loop terminates.

```
#define ORIGINYEAR 1980
UINT32 days = /* Input parameter */
int year = ORIGINYEAR;
/* ... */

int daysThisYear = (IsLeapYear(year) ? 366 : 365);
while (days > daysThisYear) {
  days -= daysThisYear;
  year += 1;
  daysThisYear = (IsLeapYear(year) ? 366 : 365);
}
```

This compliant solution is for illustrative purposes and is not necessarily the solution implemented by Microsoft.

Risk Assessment

Failing to account for all possibilities within a logic statement can lead to a corrupted running state, potentially resulting in unintentional information disclosure or abnormal termination.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
MSC01-C	Medium	Probable	Medium	P8	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**missing-else** **switch-default**	Partially checked

Compass /ROSE			Can detect some violations of this recommendation. In particular, it flags switch statements that do not have a default clause. ROSE should detect "fake switches" as well (that is, a chain of `if` statements each checking the value of the same variable). These `if` statements should always end in an `else` clause, or they should mathematically cover every possibility. For instance, consider the following:
			`if (x > 0) {`
			`  /* ... */`
			`} else if (x < 0) {`
			`  /* ... */`
			`} else if (x == 0) {`
			`  /* ... */`
			`}`

GCC	4.3.5		Can detect some violations of this recommendation when the `-Wswitch` and `-Wswitch-default` flags are used
Klocwork	2018	**CWARN.EMPTY.LABEL** **LA_UNUSED** **MISRA.IF.NO_ELSE** **MISRA.SWITCH.** **WELL_FORMED.DEFAULT.** **2012** **INFINITE_LOOP.GLOBAL** **INFINITE_LOOP.LOCAL** **INFINITE_LOOP.MACRO**	
LDRA tool suite	9.7.1	**48 S, 59 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_C-MSC01-a** **CERT_C-MSC01-b**	All 'if...else-if' constructs shall be terminated with an 'else' clause The final clause of a switch statement shall be the default clause
Polyspace Bug Finder	R2019a	CERT C: Rec. MSC01-C	Checks for missing case for switch condition (rule partially covered)
PRQA QA-C	9.5	**2000, 2002, 2004**	Fully implemented
PVS-Studio	6.23	**V517**, **V533**, **V534**, **V535**, **V547**, **V556**, **V560**, **V577**, **V590**, **V600**, **V612**, **V695**, **V696**, **V719**, **V722**, **V747**, **V785**, **V786**	
RuleChecker	19.04	**missing-else** **switch-default**	Partially checked
SonarQube C/C++ Plugin	3.11	**ElseIfWithoutElse**, **SwitchWithoutDefault**	

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	VOID MSC01-CPP. Strive for logical completeness
CERT Oracle Secure Coding Standard for Java	MSC57-J. Strive for logical completeness
ISO/IEC TS 17961	Use of an implied default in a switch statement [swtchdflt]
ISO/IEC TR 24772	Switch Statements and Static Analysis [CLL]

Bibliography

[Hatton 1995]	Section 2.7.2, "Errors of Omission and Addition"
[Viega 2005]	Section 5.2.17, "Failure to Account for Default Case in Switch"
[Zadegan 2009]	"A Lesson on Infinite Loops"