FIO41-C. Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects

Do not invoke `getc()` or `putc()` or their wide-character analogues `getwc()` and `putwc()` with a stream argument that has side effects. The stream argument passed to these macros may be evaluated more than once if these functions are implemented as unsafe macros. (See PRE31-C. Avoid side effects in arguments to unsafe macros for more information.)

This rule does not apply to the character argument in `putc()` or the wide-character argument in `putwc()`, which is guaranteed to be evaluated exactly once.

Noncompliant Code Example (`getc()`)

This noncompliant code example calls the `getc()` function with an expression as the stream argument. If `getc()` is implemented as a macro, the file may be opened multiple times. (See FIO24-C. Do not open a file that is already open.)

```
#include <stdio.h>

void func(const char *file_name) {
  FILE *fptr;

  int c = getc(fptr = fopen(file_name, "r"));
  if (feof(stdin) || ferror(stdin)) {
    /* Handle error */
  }

  if (fclose(fptr) == EOF) {
    /* Handle error */
  }
}
```

This noncompliant code example also violates ERR33-C. Detect and handle standard library errors because the value returned by `fopen()` is not checked for errors.

Compliant Solution (`getc()`)

In this compliant solution, `fopen()` is called before `getc()` and its return value is checked for errors:

```
#include <stdio.h>

void func(const char *file_name) {
  int c;
  FILE *fptr;

  fptr = fopen(file_name, "r");
  if (fptr == NULL) {
    /* Handle error */
  }

  c = getc(fptr);
  if (c == EOF) {
    /* Handle error */
  }

  if (fclose(fptr) == EOF) {
    /* Handle error */
  }
}
```

Noncompliant Code Example (`putc()`)

In this noncompliant example, `putc()` is called with an expression as the stream argument. If `putc()` is implemented as a macro, this expression might be evaluated multiple times.

```
#include <stdio.h>

void func(const char *file_name) {
  FILE *fptr = NULL;
  int c = 'a';

  while (c <= 'z') {
    if (putc(c++, fptr ? fptr :
         (fptr = fopen(file_name, "w"))) == EOF) {
      /* Handle error */
    }
  }

  if (fclose(fptr) == EOF) {
    /* Handle error */
  }
}
```

This noncompliant code example might appear safe even if the `putc()` macro evaluates its stream argument multiple times, as the ternary conditional expression ostensibly prevents multiple calls to `fopen()`. However, the assignment to `fptr` and the evaluation of `fptr` as the controlling expression of the ternary conditional expression can take place between the same sequence points, resulting in undefined behavior (a violation of EXP30-C. Do not depend on the order of evaluation for side effects). This code also violates ERR33-C. Detect and handle standard library errors because it fails to check the return value from `fopen()`.

Compliant Solution (`putc()`)

In this compliant solution, the stream argument to `putc()` no longer has side effects:

```
#include <stdio.h>

void func(const char *file_name) {
  int c = 'a';
  FILE *fptr = fopen(file_name, "w");

  if (fptr == NULL) {
    /* Handle error */
  }

  while (c <= 'z') {
    if (putc(c++, fptr) == EOF) {
      /* Handle error */
    }
  }

  if (fclose(fptr) == EOF) {
    /* Handle error */
  }
}
```

The expression `c++` is perfectly safe because `putc()` guarantees to evaluate its character argument exactly once.

NOTE: The output of this compliant solution differs depending on the character set. For example, when run on a machine using an ASCII-derived code set such as ISO-8859 or Unicode, this solution will print out the 26 lowercase letters of the English alphabet. However, if run with an EBCDIC-based code set, such as Codepage 037 or Codepage 285, punctuation marks or symbols may be output between the letters.

Risk Assessment

Using an expression that has side effects as the stream argument to `getc()`, `putc()`, or `getwc()` can result in unexpected behavior and abnormal program termination.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FIO41-C	Low	Unlikely	Medium	**P2**	**L3**

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**stream-argument-with-side-effects**	Fully checked
LDRA tool suite	9.7.1	**35 D, 1 Q, 9 S, 30 S, 134 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_C-FIO41-a** **CERT_C-FIO41-b** **CERT_C-FIO41-c** **CERT_C-FIO41-d** **CERT_C-FIO41-e**	The value of an expression shall be the same under any order of evaluation that the standard permits Don't write code that depends on the order of evaluation of function arguments Don't write code that depends on the order of evaluation of function designator and function arguments Don't write code that depends on the order of evaluation of expression that involves a function call A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects
Polyspace Bug Finder	R2019b	CERT C: Rule FIO41-C	Checks for stream arguments with possibly unintended side effects (rule fully covered)
PRQA QA-C	9.5	**5036**	
PRQA QA-C++	4.3	**3225, 3229**	
RuleChecker	19.04	**stream-argument-with-side-effects**	Fully checked

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 Secure Coding Standard	FIO24-C. Do not open a file that is already open	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT C Secure Coding Standard	EXP30-C. Do not depend on the order of evaluation for side effects	Prior to 2018-01-12: CERT: Unspecified Relationship