FIO42-C. Close files when they are no longer needed

A call to the `fopen()` or `freopen()` function must be matched with a call to `fclose()` before the lifetime of the last pointer that stores the return value of the call has ended or before normal program termination, whichever occurs first.

In general, this rule should also be applied to other functions with open and close resources, such as the POSIX `open()` and `close()` functions, or the Microsoft Windows `CreateFile()` and `CloseHandle()` functions.

Noncompliant Code Example

This code example is noncompliant because the file opened by the call to `fopen()` is not closed before function `func()` returns:

```
#include <stdio.h>

int func(const char *filename) {
  FILE *f = fopen(filename, "r");
  if (NULL == f) {
    return -1;
  }
  /* ... */
  return 0;
}
```

Compliant Solution

In this compliant solution, the file pointed to by `f` is closed before returning to the caller:

```
#include <stdio.h>

int func(const char *filename) {
  FILE *f = fopen(filename, "r");
  if (NULL == f) {
    return -1;
  }
  /* ... */
  if (fclose(f) == EOF) {
    return -1;
  }
  return 0;
}
```

Noncompliant Code Example (`exit()`)

This code example is noncompliant because the resource allocated by the call to `fopen()` is not closed before the program terminates. Although `exit()` closes the file, the program has no way of determining if an error occurs while flushing or closing the file.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE *f = fopen(filename, "w");
  if (NULL == f) {
    exit(EXIT_FAILURE);
  }
  /* ... */
  exit(EXIT_SUCCESS);
}
```

Compliant Solution (`exit()`)

In this compliant solution, the program closes `f` explicitly before calling `exit()`, allowing any error that occurs when flushing or closing the file to be handled appropriately:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  FILE *f = fopen(filename, "w");
  if (NULL == f) {
    /* Handle error */
  }
  /* ... */
  if (fclose(f) == EOF) {
    /* Handle error */
  }
  exit(EXIT_SUCCESS);
}
```

Noncompliant Code Example (POSIX)

This code example is noncompliant because the resource allocated by the call to `open()` is not closed before function `func()` returns:

```
#include <stdio.h>
#include <fcntl.h>

int func(const char *filename) {
  int fd = open(filename, O_RDONLY, S_IRUSR);
  if (-1 == fd) {
    return -1
  }
  /* ... */
  return 0;
}
```

Compliant Solution (POSIX)

In this compliant solution, `fd` is closed before returning to the caller:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int func(const char *filename) {
  int fd = open(filename, O_RDONLY, S_IRUSR);
  if (-1 == fd) {
    return -1
  }
  /* ... */
  if (-1 == close(fd)) {
    return -1;
  }
  return 0;
}
```

Noncompliant Code Example (Windows)

In this noncompliant code example, the file opened by the Microsoft Windows `CreateFile()` function is not closed before `func()` returns:

```
#include <Windows.h>

int func(LPCTSTR filename) {
  HANDLE hFile = CreateFile(filename, GENERIC_READ, 0, NULL,
                            OPEN_EXISTING,
                            FILE_ATTRIBUTE_NORMAL, NULL);
  if (INVALID_HANDLE_VALUE == hFile) {
    return -1;
  }
  /* ... */
  return 0;
}
```

Compliant Solution (Windows)

In this compliant solution, `hFile` is closed by invoking the `CloseHandle()` function before returning to the caller:

```
#include <Windows.h>

int func(LPCTSTR filename) {
  HANDLE hFile = CreateFile(filename, GENERIC_READ, 0, NULL,
                            OPEN_EXISTING,
                            FILE_ATTRIBUTE_NORMAL, NULL);
  if (INVALID_HANDLE_VALUE == hFile) {
    return -1;
  }
  /* ... */
  if (!CloseHandle(hFile)) {
    return -1;
  }

  return 0;
}
```

Risk Assessment

Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FIO42-C	Medium	Unlikely	Medium	**P4**	**L3**

Automated Detection

This rule is stricter than rule [fileclose] in ISO/IEC TS 17961:2013. Analyzers that conform to the technical standard may not detect all violations of this rule.

Tool	Version	Checker	Description
Astrée	19.04		Supported, but no explicit checker
CodeSonar	5.1p0	**ALLOC.LEAK**	Leak
Compass/ROSE			
Coverity	2017.07	**RESOURCE_LEAK (partial)**	Partially implemented
Klocwork	2018	**RH.LEAK**	
LDRA tool suite	9.7.1	**49 D**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_C-FIO42-a**	Ensure resources are freed
Polyspace Bug Finder	R2019b	CERT C: Rule FIO42-C	Checks for resource leak (rule partially covered)
PRQA QA-C	9.5	**2701, 2702, 2703**	
SonarQube C/C++ Plugin	3.11	**S2095**	

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	FIO51-CPP. Close files when they are no longer needed	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT Oracle Secure Coding Standard for Java	FIO04-J. Release resources when they are no longer needed	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961:2013	Failing to close files or free dynamic memory when they are no longer needed [fileclose]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-404, Improper Resource Shutdown or Release	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-459	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-772	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-773	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-775	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-403	2017-10-30:MITRE:Unspecified Relationship 2018-10-18:CERT:Partial overlap

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-773/CWE-775 and FIO42-C

CWE-773 = CWE-775

CWE-773 = Union(FIO42-C, list) where list =

- Failure to free resource handles besides files

CWE-404/CWE-459/CWE-771/CWE-772 and FIO42-C/MEM31-C

Intersection(FIO42-C, MEM31-C) = Ø

CWE-404 = CWE-459 = CWE-771 = CWE-772

CWE-404 = Union(FIO42-C, MEM31-C list) where list =

- Failure to free resources besides files or memory chunks, such as mutexes)

CWE-403 and FIO42-C

CWE-403 - FIO42-C = list, where list =

- A process opens and closes a sensitive file descriptor, but also executes a child process while the file descriptor is open.

FIO42-C - CWE-403 = SPECIAL_CASES, where SPECIAL_CASES =

- A program opens a file descriptor and fails to close it, but does not invoke any child processes while the file descriptor is open.

Bibliography

[IEEE Std 1003.1:2013]	XSH, System Interfaces, open