

# FIO10-C. Take care when using the rename() function

The `rename()` function has the following prototype:

```
int rename(const char *src_file, const char *dest_file);
```

If the file referenced by `dest_file` exists prior to calling `rename()`, the behavior is [implementation-defined](#). On POSIX systems, the destination file is removed. On Windows systems, the `rename()` fails. Consequently, issues arise when trying to write portable code or when trying to implement alternative behavior.

## Preserve Existing Destination File

If the desired behavior is to ensure that the destination file is not erased or overwritten, POSIX programmers must implement additional safeguards.

## Noncompliant Code Example (POSIX)

This code example is noncompliant because any existing destination file is removed by `rename()`:

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;
if (rename(src_file, dest_file) != 0) {
    /* Handle error */
}
```

## Compliant Solution (POSIX)

If the programmer's intent is not to remove an existing destination file, the POSIX `access()` function can be used to check for the existence of a file [[IEEE Std 1003.1:2013](#)]. This compliant solution renames the source file only if the destination file does not exist:

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;

if (access(dest_file, F_OK) != 0) {
    if (rename(src_file, dest_file) != 0) {
        /* Handle error condition */
    }
}
else {
    /* Handle file-exists condition */
}
```

This code contains an unavoidable race condition between the call to `access()` and the call to `rename()` and can consequently be safely executed only when the destination file is located within a secure directory. (See [FIO15-C. Ensure that file operations are performed in a secure directory.](#))

On file systems where the program does not have sufficient permissions in the directory to view the file, `access()` may return `-1` even when the file exists. In such cases, `rename()` will also fail because the program lacks adequate permissions to perform the operation.

In situations where the source file is supposed not to be a directory or symbolic link, an alternative solution is to use `link()` to link the source file to the destination file and then use `unlink()` (or `remove()`) to delete the source file. Because `link()` fails if the destination file exists, the need for calling `access()` is avoided. However, this solution has two race conditions related to the source file. First, before calling `link()`, the program must use `lstat()` to check that the source file is not a directory or symbolic link. Second, the source file could change during the time window between the `link()` and the `unlink()`. Consequently, this alternative solution can be safely executed only when the source file is located within a secure directory.

## Compliant Solution (Windows)

On Windows, the `rename()` function fails if a

*file or directory specified by `newname` already exists or could not be created (invalid path).* [[MSDN](#)]

Consequently, it is unnecessary to explicitly check for the existence of the destination file before calling `rename()`.

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;
if (rename(src_file, dest_file) != 0) {
    /* Handle error */
}
```

## Remove Existing Destination File

If the desired behavior is to ensure that the destination file is erased by the `rename()` operation, Windows programmers must write additional code.

### Noncompliant Code Example (Windows)

If the intent of the programmer is to remove the file referenced by `dest_file` if it exists prior to calling `rename()`, this code example is noncompliant on Windows platforms because `rename()` will fail:

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;
if (rename(src_file, dest_file) != 0) {
    /* Handle error */
}
```

### Compliant Solution (Windows)

On Windows systems, it is necessary to explicitly remove the destination file before calling `rename()` if the programmer wants the file to be overwritten and the `rename()` operation to succeed:

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;

if (_access_s(dest_file, 0) == 0) {
    if (remove(dest_file) != 0) {
        /* Handle error condition */
    }
}

if (rename(src_file, dest_file) != 0) {
    /* Handle error condition */
}
```

This code contains unavoidable race conditions between the calls to `_access_s()`, `remove()`, and `rename()` and can consequently be safely executed only within a secure directory. (See [FIO15-C. Ensure that file operations are performed in a secure directory.](#)) Another option would be to use the [MoveFileEx](#) API and pass in the `MOVEFILE_REPLACE_EXISTING` flag:

```
const char *src_file = /* ... */;
const char *dest_file = /* ... */;

if (!MoveFileEx(src_file, dest_file, MOVEFILE_REPLACE_EXISTING)) {
    /* Handle error condition */
}
```

Although this code is not portable, it does avoid the race condition when using `_access_s()`, `remove()`, and `rename()`.

### Compliant Solution (POSIX)

On POSIX systems, if the destination file exists prior to calling `rename()`, the file is automatically removed:

```

const char *src_file = /* ... */;
const char *dest_file = /* ... */;
if (rename(src_file, dest_file) != 0) {
    /* Handle error condition */
}

```

## Portable Behavior

A programmer who wants an application to behave the same on any C [implementation](#) must first determine what behavior to implement.

### Compliant Solution (Remove Existing Destination File)

This compliant solution ensures that any destination file is portably removed:

```

const char *src_file = /* ... */;
const char *dest_file = /* ... */;

(void)remove(dest_file);

if (rename(src_file, dest_file) != 0) {
    /* Handle error condition */
}

```

This code contains an unavoidable race condition between the call to `remove()` and the call to `rename()` and consequently can be safely executed only within a secure directory. (See [FIO15-C. Ensure that file operations are performed in a secure directory.](#))

The return value of `remove()` is deliberately not checked because it is expected to fail if the file does not exist. If the file exists but cannot be removed, the `rename()` call will also fail, and the error will be detected at that point. This is a valid exception (EXP12-C-EX1) to [EXP12-C. Do not ignore values returned by functions.](#)

### Compliant Solution (Preserve Existing Destination File)

This compliant solution renames the source file only if the destination file does not exist:

```

const char *src_file = /* ... */;
const char *dest_file = /* ... */;

if (!file_exists(dest_file)) {
    if (rename(src_file, dest_file) != 0) {
        /* Handle error condition */
    }
}
else {
    /* Handle file-exists condition */
}

```

This code contains an unavoidable race condition between the call to `file_exists()` and the call to `rename()` and can consequently be safely executed only within a secure directory. (See [FIO15-C. Ensure that file operations are performed in a secure directory.](#))

The `file_exists()` function is provided by the application and is not shown here because it must be implemented differently on different platforms. (On POSIX systems, it would use `access()`; on Windows, `_access_s()`; and on other platforms, whatever function is available to test file existence.)

## Risk Assessment

Calling `rename()` has [implementation-defined](#) behavior when the new file name refers to an existing file. Incorrect use of `rename()` can result in a file being unexpectedly overwritten or other [unexpected behavior](#).

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
FIO10-C	Medium	Probable	Medium	P8	L2

## Automated Detection

Tool	Version	Checker	Description
<a href="#">CodeSonar</a>	5.2p0	<b>(customization)</b>	Users can add a custom check for all uses of <code>rename ( )</code> .
<a href="#">LDRA tool suite</a>	9.7.1	<b>592 S</b>	Fully Implemented
<a href="#">PRQA QA-C</a>	9.7	<b>5015</b>	Partially implemented

## Related Vulnerabilities

Search for [vulnerabilities](#) resulting from the violation of this rule on the [CERT website](#).

## Related Guidelines

<a href="#">SEI CERT C++ Coding Standard</a>	<a href="#">VOID FIO10-CPP. Take care when using the rename() function</a>
--	--

## Bibliography

<a href="#">[IEEE Std 1003.1:2013]</a>	XSH, System Interfaces, <a href="#">access</a>
<a href="#">[MSDN]</a>	<a href="#">rename ( )</a>

