FIO19-C. Do not use fseek() and ftell() to compute the size of a regular file

Understanding the difference between text mode and binary mode is important when using functions that operate on file streams. (See FIO14-C. Understand the difference between text mode and binary mode with file streams for more information.)

Subclause 7.21.9.2 of the C Standard [ISO/IEC 9899:2011] specifies the following behavior for `fseek()` when opening a binary file in binary mode:

> *A binary stream need not meaningfully support fseek calls with a whence value of SEEK_END.*

In addition, footnote 268 of subclause 7.21.3 says:

> *Setting the file position indicator to end-of-file, as with `fseek(file, 0, SEEK_END)`, has undefined behavior for a binary stream (because of possible trailing null characters) or for any stream with state-dependent encoding that does not assuredly end in the initial shift state.*

Seeking to the end of a binary stream in binary mode with `fseek()` is not meaningfully supported and is not a recommended method for computing the size of a file.

Subclause 7.21.9.4 of the C Standard [ISO/IEC 9899:2011] specifies the following behavior for `ftell()` when opening a text file in text mode:

> *For a text stream, its file position indicator contains unspecified information, usable by the `fseek` function for returning the file position indicator for the stream to its position at the time of the `ftell` call.*

Consequently, the return value of `ftell()` for streams opened in text mode should never be used for offset calculations other than in calls to `fseek()`.

POSIX [IEEE Std 1003.1:2013] provides several guarantees that the problems described in the C Standard do not occur on POSIX systems.

First, the `fopen` page says:

> *The character 'b' shall have no effect, but is allowed for ISO C standard conformance.*

This guarantees that binary files are treated the same as text files in POSIX.

Second, the `fwrite` page says:

> *For each object,* size *calls shall be made to the* fputc() *function, taking the values (in order) from an array of* ***unsigned char*** *exactly overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written.*

This means that the file position indicator, and consequently the file size, is directly based on the number of bytes actually written to a file.

Noncompliant Code Example (Binary File)

This code example attempts to open a binary file in binary mode and use `fseek()` and `ftell()` to obtain the file size. This code is noncompliant on systems that do not provide the same guarantees as POSIX. On these systems, setting the file position indicator to the end of the file using `fseek()` is not guaranteed to work for a binary stream, and consequently, the amount of memory allocated may be incorrect, leading to a potential vulnerability.

```
FILE *fp;
long file_size;
char *buffer;

fp = fopen("foo.bin", "rb");
if (fp == NULL) {
  /* Handle error */
}

if (fseek(fp, 0 , SEEK_END) != 0) {
  /* Handle error */
}

file_size = ftell(fp);
if (file_size == -1) {
  /* Handle error */
}

buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

Compliant Solution (POSIX `ftello()`)

If the code needs to handle large files, it is preferable to use `fseeko()` and `ftello()` because, for some implementations, they can handle larger file offsets than `fseek()` and `ftell()` can handle. If they are used, the `file_size` variable should have type `off_t` to avoid the possibility of overflow when assigning the return value of `ftello()` to it. This solution works only with regular files.

```
FILE* fp;
int fd;
off_t file_size;
char *buffer;
struct stat st;

fd = open("foo.bin", O_RDONLY);
if (fd == -1) {
  /* Handle error */
}

fp = fdopen(fd, "r");
if (fp == NULL) {
  /* Handle error */
}

/* Ensure that the file is a regular file */
if ((fstat(fd, &st) != 0) || (!S_ISREG(st.st_mode))) {
  /* Handle error */
}

if (fseeko(fp, 0 , SEEK_END) != 0) {
  /* Handle error */
}

file_size = ftello(fp);
if (file_size == -1) {
  /* Handle error */
}

buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

Compliant Solution (POSIX `fstat()`)

This compliant solution uses the size provided by the POSIX `fstat()` function, rather than by `fseek()` and `ftell()`, to obtain the size of the binary file. This solution works only with regular files.

```
off_t file_size;
char *buffer;
struct stat stbuf;
int fd;

fd = open("foo.bin", O_RDONLY);
if (fd == -1) {
  /* Handle error */
}

if ((fstat(fd, &stbuf) != 0) || (!S_ISREG(stbuf.st_mode))) {
  /* Handle error */
}

file_size = stbuf.st_size;

buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

Compliant Solution (Windows)

This compliant solution uses the Windows `_filelength()` function to determine the size of the file on a 32-bit operating system. For a 64-bit operating system, consider using `_filelengthi64` instead.

```
int fd;
long file_size;
char *buffer;

_sopen_s(&fd, "foo.bin", _O_RDONLY, _SH_DENYRW, _S_IREAD);
if (fd == -1) {
  /* Handle error */
}

file_size = _filelength(fd);
if (file_size == -1) {
  /* Handle error */
}

buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

Compliant Solution (Windows)

This compliant solution uses the Windows `GetFileSizeEx()` function to determine the size of the file on a 32- or 64-bit operating system:

```
HANDLE file;
LARGE_INTEGER file_size;
char *buffer;


file = CreateFile(TEXT("foo.bin"), GENERIC_READ, 0, NULL,
                  OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (INVALID_FILE_HANDLE == file) {
  /* Handle error */
}

if (!GetFileSizeEx(file, &file_size)) {
  /* Handle error */
}

/*
 * Note: 32-bit portability issue with LARGE_INTEGER
 * truncating to a size_t.
 */
buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

Noncompliant Code Example (Text File)

This noncompliant code example attempts to open a text file in text mode and use `fseek()` and `ftell()` to obtain the file size:

```
FILE *fp;
long file_size;
char *buffer;

fp = fopen("foo.txt", "r");
if (fp == NULL) {
  /* Handle error */
}

if (fseek(fp, 0 , SEEK_END) != 0) {
  /* Handle error */
}

file_size = ftell(fp);
if (file_size == -1) {
  /* Handle error */
}

buffer = (char*)malloc(file_size);
if (buffer == NULL) {
  /* Handle error */
}

/* ... */
```

However, the file position indicator returned by `ftell()` with a file opened in text mode is useful only in calls to `fseek()`. As such, the value of `file_siz` may not necessarily be a meaningful measure of the number of characters in the file, and consequently, the amount of memory allocated may be incorrect, leading to a potential vulnerability.

The Visual Studio documentation for `ftell()` [MSDN] states:

> *The value returned by `ftell` may not reflect the physical byte offset for streams opened in text mode, because text mode causes carriage return-linefeed translation. Use `ftell` with `fseek` to return to file locations correctly.*

Again, this indicates that the return value of `ftell()` for streams opened in text mode is useful only in calls to `fseek()` and should not be used for any other purpose.

Compliant Solution (Windows)

The compliant solution used for binary files on Windows can also be used for text files.

Compliant Solution (POSIX)

Because binary files are treated the same as text files in POSIX, either compliant solution can be used for determining the size of a binary file under POSIX to determine the size of a text file as well.

Risk Assessment

Understanding the difference between text mode and binary mode with file streams is critical when working with functions that operate on them. Setting the file position indicator to end-of-file with `fseek()` has undefined behavior for a binary stream. In addition, the return value of `ftell()` for streams opened in text mode is useful only in calls to `fseek()`, not for determining file sizes or for any other use. As such, `fstat()` or other platform-equivalent functions should be used to determine the size of a file.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
FIO19-C	Low	Unlikely	Medium	P2	L3

Automated Detection

Tool	Version	Checker	Description
LDRA tool suite	9.7.1	44 S	Enhanced Enforcement

Bibliography

[IEEE Std 1003.1:2013]	XSH, System Interfaces, `fopen` XSH, System Interfaces, `fwrite`
[ISO/IEC 9899:2011]	Section 7.21.3, "Files" Section 7.21.9.2, "The `fseek` Function" Section 7.21.9.4, "The `ftell` Function"
[MSDN]	"ftell"