

Investigation: Coverity CID 1668646

Overview

CID: 1668646

Checker: OVERRUN

File: contrib/cpukit/fatfs/ff.c

Function: get_fileinfo()

Reported line: fno->altname[di++] = (TCHAR)wc;

Coverity reports a possible buffer overrun because fno->altname has 13 elements and claims that di can reach 13 before the write.

The relevant configuration and structure definitions are:

```
#define FF_SFN_BUF 12
```

and:

```
TCHAR altname[FF_SFN_BUF + 1]; /* Alternative object name */
```

Therefore, altname contains 13 elements with valid indices:

```
0 ... 12
```

The investigation below determines whether the reported write can actually occur with di == 13.

Coverity Finding

Coverity reports:

```
CID 1668646: Memory - corruptions (OVERRUN)
```

```
Overrunning array "fno->altname" of 13 bytes at byte offset 13  
using index "di++" (which evaluates to 13).
```

The reported code is:

```
while (si < 11) {  
    wc = dp->dir[si++];
```

```
...

if (si == 9 && di < FF_SFN_BUF)
    fno->altname[di++] = '.';

#if FF_LFN_UNICODE >= 1
...
#else
    fno->altname[di++] = (TCHAR)wc;
#endif
}

fno->altname[di] = 0;
```

Relevant Configuration

The current FatFs configuration contains:

```
#define FF_USE_LFN      0
#define FF_LFN_UNICODE 0
#define FF_SFN_BUF     12
```

Thus the ANSI/OEM output branch is active:

```
#else /* ANSI/OEM output */
fno->altname[di++] = (TCHAR)wc;
#endif
```

The **FILINFO** definition contains:

```
TCHAR altname[FF_SFN_BUF + 1]; /* Alternative object name */
```

With **FF_SFN_BUF = 12**, this gives:

```
altname size = 13 elements
valid indices = 0 through 12
```

Relevant Source Code

The complete relevant section of **get_fileinfo()** is:

```
si = di = 0;
while (si < 11) {           /* Get SFN from SFN entry */
    wc = dp->dir[si++];     /* Get a char */
    if (wc == ' ') continue; /* Skip padding spaces */
    if (wc == RDDEM) wc = DDEM; /* Restore replaced DDEM character */
    /*
        if (si == 9 && di < FF_SFN_BUF)
            fno->altname[di++] = '.'; /* Insert a . if extension is
exist */

#ifdef FF_LFN_UNICODE >= 1
    /* Unicode output */
    ...
#else
    /* ANSI/OEM output */
    fno->altname[di++] = (TCHAR)wc;
#endif
}

fno->altname[di] = 0; /* Terminate the SFN */
```

Bounds Analysis

1. Maximum number of SFN characters

The loop condition is:

```
while (si < 11)
```

and `si` is incremented immediately:

```
wc = dp->dir[si++];
```

Therefore, the loop processes at most 11 directory-entry character positions.

These correspond to the standard FAT short filename layout:

```
8 characters for the base name
3 characters for the extension
-----
11 directory-entry characters
```

Thus, at most **11 characters** can be copied from `dp->dir`.

2. Maximum number of inserted dots

The code can insert a `.` here:

```
if (si == 9 && di < FF_SFN_BUF)
    fno->altname[di++] = '.';
```

The condition `si == 9` can occur only once during the loop because `si` increases monotonically.

Therefore, at most **one additional character** can be inserted.

Hence:

```
Maximum copied SFN characters = 11
Maximum inserted dots         = 1
Maximum characters in output  = 12
```

Therefore:

```
Maximum final di = 12
```

Buffer Bounds

The buffer contains 13 elements:

```
TCHAR altname[13];
```

so the valid indices are:

```
0, 1, 2, ..., 11, 12
```

The maximum possible final value of `di` is 12.

Consequently, the final termination:

```
fno->altname[di] = 0;
```

can write at:

```
fno->altname[12] = 0;
```

which is valid.

The preceding character writes can use indices only up to 11.

Therefore, the maximum valid access pattern is:

```
Character writes: altname[0] ... altname[11]
NUL terminator:  altname[12]
```

No access to:

```
altname[13]
```

is possible from this loop.

Why Coverity Reports an Overrun

Coverity reports that the expression:

```
di++
```

may execute with:

```
di == 13
```

However, this requires the analysis to overlook the combined constraints imposed by:

```
while (si < 11)
```

and:

```
if (si == 9 && di < FF_SFN_BUF)
```

The loop permits at most 11 source-character writes and at most one additional dot insertion.

Thus, the total number of output characters cannot exceed 12.

The array has 13 elements specifically allowing those 12 characters plus the terminating NUL.

The reported `di == 13` state is therefore not reachable through the normal execution of this loop.

Configuration Verification

The relevant configuration was verified directly from:

```
contrib/cpukit/fatfs/ffconf.h
```

with:

```
#define FF_USE_LFN      0
#define FF_LFN_UNICODE 0
#define FF_SFN_BUF     12
```

The `FILINFO` definition was verified in:

```
contrib/cpukit/fatfs/ff.h
```

with:

```
TCHAR alname[FF_SFN_BUF + 1]; /* Alternative object name */
```

The reported ANSI/OEM assignment is therefore the active implementation path for this configuration.

Provenance

The source-history search:

```
git log -S"fno->alname[di++] = (TCHAR)wc" --oneline --all --
contrib/cpukit/fatfs/ff.c
```

returned:

```
99ea420b1b (HEAD) cpukit: Move FatFs core to contrib and update build
integration
```

This indicates that the code is part of the FatFs core moved into the `contrib` tree during the RTEMS FatFs refactor rather than being newly introduced as an RTEMS-specific implementation.

Conclusion

CID 1668646 is a Coverity false positive.

The `altname` buffer has 13 elements because:

```
#define FF_SFN_BUF 12
TCHAR altname[FF_SFN_BUF + 1];
```

The SFN loop can process at most 11 source characters and can insert at most one `.` character. Therefore, the output can contain at most 12 characters, leaving index 12 for the terminating NUL.

The maximum reachable state is:

```
di = 12
```

and not:

```
di = 13
```

Consequently, the reported access to `altname[13]` is unreachable under the loop's actual bounds.

Classification

Aspect	Result
Coverity finding	OVERRUN
Actual buffer size	13 elements
Maximum SFN characters	11
Maximum inserted dots	1
Maximum <code>di</code>	12
<code>altname[12]</code>	Valid
<code>altname[13]</code>	Unreachable
RTEMS-specific issue	No
Upstream FatFs issue	No
Final classification	Coverity false positive

Recommended Action

No source-code change is required for RTEMS or FatFs.

The finding can be documented as a false positive and closed with the bounds argument above. Adding an artificial bounds check solely to silence Coverity would unnecessarily modify the imported FatFs implementation.