Security Advisory
CFITSIO Library

Created by Adrian Denkiewicz

06/12/2025

Overview

This document summarizes the results of a vulnerability research activity aimed at discovering vulnerabilities affecting the CFITSIO Library. While security testing was not meant to be comprehensive in terms of attack and code coverage, we have identified six (6) findings that could lead to exploitable issues such as heap and stack overflows, including multiple out-of-bounds writes. Additionally, we identified a division by zero that can cause a DoS scenario.

The testing was performed on the latest release of the CFITSIO Library at that time. The v4.5 package was released in August 2024. We manually compiled the package to apply additional instrumentation. The code was slightly modified for fuzzing purposes.

The CFITSIO 4.6.2 release fixes 5 out of 6 reported issues. Details of the remaining issue (`CFITSIO-03`) have been redacted. The library maintainers confirmed that work on the final fix is ongoing.

About Us

Doyensec is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

CFITSIO-01. Invalid Memory is Dereferenced in ffr8fstr Due to Type Confusion	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	putcold.c
Status	Closed
Credits	Adrian Denkiewicz

Description

We discovered a type confusion issue leading to an information disclosure or invalid memory access issue.

The issue occurs within the `ffr8fstr` method called from the `fits_write_col_dbl` function. The following stack trace is present during the crash.

```
    #0 0x5555555f551c in printf_common(void*, char const*, __va_list_tag*)
asan_interceptors.cpp.o:?
    #1 0x5555555f5b83 in vsprintf ??:?
    #2 0x5555555f6c93 in __interceptor_sprintf ??:?
    #3 0x5555558e0a90 in ffr8fstr ??:?
    #4 0x5555558da935 in ffpcld ??:?
    #5 0x5555558c422e in ffpcl ??:?
    #6 0x555555695e33 in processTableHDU(fitsfile*, int, bool) ??:?
    #7 0x555555696f9d in processFITSFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, bool) ??:?
    #8 0x555555697cd2 in main ??:?
    #9 0x7ffff7829d8f in __libc_start_call_main csu/../sysdeps/nptl/
libc_start_call_main.h:58
    #10 0x7ffff7829e3f in __libc_start_main_impl csu/../csu/libc-start.c:392
    #11 0x5555555d38c4 in _start ??:?
```

When the code reaches the `ffcfmt` function, it attempts to *"convert the FITS format string for an ASCII Table extension column into the equivalent C format string that can be used in a printf statement, after the values have been read as a double."*

```
void ffcfmt(char *tform,    /* value of an ASCII table TFORMn keyword */
            char *cform)    /* equivalent format code in C language syntax */
{
        ...
        cform[0] = '%';

        strcpy(&cform[1], &tform[ii + 1]); /* append the width and decimal code */
```

```
    if (tform[ii] == 'A')
        strcat(cform, "s");
        ...
}
```

For our sample file, the `tform[ii+1]` is "A0001000", therefore the `cform` is set to the `%0001000s` string.

```
#0  ffcfmt (tform=0x7fffffff66d0 "A00010000", cform=cform@entry=0x7fffffff6710 "")
at fitscore.c:3237
#1  0x00005555558da07d in ffpcld (fptr=<optimized out>, colnum=<optimized out>,
firstrow=<optimized out>, firstelem=<optimized out>, nelem=<optimized out>,
array=<optimized out>, status=<optimized out>) at putcold.c:385
#2  0x00005555558c422f in ffpcl (fptr=0x7fffffff66d0, datatype=<optimized out>,
colnum=0, firstrow=140737488315649, firstelem=<optimized out>, nelem=<optimized
out>, array=0x611000000040, status=0x7fffffffdb80) at putcol.c:759
```

A bit further in the execution, inside the `ffpcld` function, a switch statement continues (`putcold.c:482`):

```
case (TSTRING):  /* numerical column in an ASCII table */

        if (cform[1] != 's')  /*  "%s" format is a string */
        {
                ffr8fstr(&array[next], ntodo, scale, zero, cform,
                ...
```

The code assumes that a special treatment is needed if `cform[1] != 's'`. This is precisely our case due to the additional digits present in the format identifier. The condition does not expect any additional flags. The `ffr8fstr` function executes the following code while assuming that the `input[ii]` variable is of the `char *` type:

```
sprintf(output, cform, input[ii]);
```

The passed `input[ii]` is a double number, not a memory pointer. When a number is cast to a pointer, an unexpected memory is dereferenced leading to either an information leak or a segmentation failure.

Reproduction Steps

To reproduce the issue, use the precompiled `FITSHarness` application (see Appendix A for additional details) along with the sample file stored in the `inputs/1/` directory.

```
FITSHarness inputs/1/afltriage_ASAN_stack-buffer-
overflow_READ_vsprintf_1681e25d1aa044ac9edf455c7a2c0a08.bin
```

The application will immediately crash. GDB can be used to debug the issue further.

Remediation

Properly parse the format string and ensure the right type is used.

CFITSIO-02. Division by Zero in FITS Image Compression Handling	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	imcompress.c getkey.c
Status	Closed
Credits	Adrian Denkiewicz

Description

A division by zero issue was identified in the `imcomp_calc_max_elem` function, leading to a floating point exception (SIGFPE). The issue is triggered when processing a malformed FITS file, causing the application to crash.

The crash occurs in `imcompress.c:1446`:

```
return (sizeof(short) * nx + nx / blocksize + 2 + 4);
```

If `blocksize` is 0, the statement results in an invalid division.

When the application reaches the `imcomp_get_compressed_image_par` function, it attempts to retrieve image compression parameters, ultimately calling `imcomp_calc_max_elem` without verifying the integrity of the `blocksize` parameter. The issue propagates through multiple function calls in the CFITSIO library, leading to an unhandled division by zero.

The crash is reproducible under AddressSanitizer as:

```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==232592==ERROR: AddressSanitizer: FPE on unknown address 0x56cf2fbffab5 (pc
0x56cf2fbffab5 bp 0x7ffeabdc58f0 sp 0x7ffeabdc56d8 T0)
    #0 0x56cf2fbffab5 in imcomp_calc_max_elem /home/adrian/cfitsio-4.5.0/
imcompress.c:1446:45
    #1 0x56cf2fc347d7 in imcomp_get_compressed_image_par /home/adrian/
cfitsio-4.5.0/imcompress.c:5457:12
    #2 0x56cf2f9c26f1 in ffbinit /home/adrian/cfitsio-4.5.0/fitscore.c:5155:9
    #3 0x56cf2f9bc170 in ffrhdu /home/adrian/cfitsio-4.5.0/fitscore.c:4415:13
    #4 0x56cf2f9a4159 in ffgext /home/adrian/cfitsio-4.5.0/fitscore.c:8057:13
    #5 0x56cf2f9a4159 in ffmahd /home/adrian/cfitsio-4.5.0/fitscore.c:7819:21
    #6 0x56cf2f9da8f8 in ffthdu /home/adrian/cfitsio-4.5.0/fitscore.c:8020:21
    #7 0x56cf2f903ab5 in processFITSFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, bool) /home/adrian/
FITSHarness/src/main.cpp:131:9
    #8 0x56cf2f904cd2 in main /home/adrian/FITSHarness/src/main.cpp:168:5
```

```
    #9 0x7079c5629d8f in __libc_start_call_main csu/../sysdeps/nptl/
libc_start_call_main.h:58:16
    #10 0x7079c5629e3f in __libc_start_main csu/../csu/libc-start.c:392:3
    #11 0x56cf2f8408c4 in _start (/home/adrian/FITSHarness/build-afl-asan/
FITSHarness+0x7f8c4) (BuildId: d550951926686c2b1ec33b6625f11090e95b7336)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE /home/adrian/cfitsio-4.5.0/imcompress.c:1446:45 in
imcomp_calc_max_elem
```

Further debugging reveals that the root cause might originate from the following code inside the `getkey.c:373`:

```
    else if (datatype == TLONG)
    {
        if (ffgkyjj(fptr, keyname, &longval, comm, status) <= 0)
        {
            if (longval > LONG_MAX || longval < LONG_MIN)
                *status = NUM_OVERFLOW;
            else
                *(int *) value = longval;
        }
        ffgkyj(fptr, keyname, (long *)value, comm, status);
    }
```

For the type `TLONG`, the `ffgkyjj` correctly reads the value and comment. However, the value is immediately ignored, and `ffgkyj` is called next. This time, the method does not return a correct value. If the interpreted `ZTILE2` value is 0, it propagates through multiple functions, ultimately being used as a divisor.

Reproduction Steps

To reproduce the issue, use the precompiled `FITSHarness` application (see Appendix A for additional details) along with the sample file stored in the `inputs/2/` directory.

```
FITSHarness inputs/2/
afltriage_SIGFPE_imcomp_get_compressed_image_par_e274303f5ed0a5775480324c82de7408.b
in
```

The application will immediately crash. GDB can be used to debug the issue further.

Alternatively, the issue can be reproduced using the official `fitsverify` binary.

Remediation

Ensure the values are properly parsed and validated, rejecting malformed or duplicate entries. Validate `blocksize` before performing division. Consider if the `ffgkyj` call is required.

CFITSIO-03. Heap Overflow in [redacted]	
Vendor	NASA's HEASARC
Severity	**Low**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	[redacted] [redacted]
Status	Open
Credits	Adrian Denkiewicz

Description

[redacted]

Reproduction Steps

[redacted]

Remediation

[redacted]

CFITSIO-04. Heap Overflow in fits_rdecomp	
Vendor	NASA's HEASARC
Severity	**Low**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	-
Status	Open
Credits	Adrian Denkiewicz

Description

A heap overflow issue was identified in the `fits_rdecomp` function. The issue is triggered when processing a malformed FITS file, causing the application to crash if Address Sanitizer (ASAN) is used.

The vulnerable code is found in `ricecomp.c:868`:

```
int fits_rdecomp (unsigned char *c,             /* input buffer                 */
              int clen,                  /* length of input               */
              unsigned int array[],     /* output array                  */
              int nx,                    /* number of output pixels       */
              int nblock)                /* coding block size             */
{
/* int bsize;  */
int i, k, imax;
int nbits, nzero, fs;
unsigned char *cend, bytevalue;
unsigned int b, diff, lastpix;
int fsmax, fsbits, bbits;
extern const int nonzero_count[];

    ...

    /* first 4 bytes of input buffer contain the value of the first */
    /* 4 byte integer value, without any encoding */

    lastpix = 0;
    bytevalue = c[0];
    lastpix = lastpix | (bytevalue<<24);
    bytevalue = c[1];
    lastpix = lastpix | (bytevalue<<16);

    bytevalue = c[2];
    lastpix = lastpix | (bytevalue<<8);
    bytevalue = c[3];
    lastpix = lastpix | bytevalue;
    ...
}
```

The `c` array is expected to contain at least 4 bytes. However, it was initially allocated as an 2-byte long buffer inside the `imcomp_decompress_tile` method.

The full ASAN message is presented below:

```
==1829377==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000001612
at pc 0x585bdcd9c7e7 bp 0x7ffe1a0adf00 sp 0x7ffe1a0adef8
READ of size 1 at 0x602000001612 thread T0
    #0 0x585bdcd9c7e6 in fits_rdecomp /home/adrian/cfitsio-4.5.0/ricecomp.c:936:17
    #1 0x585bdcd4cdbe in imcomp_decompress_tile /home/adrian/cfitsio-4.5.0/
imcompress.c:6311:23
    #2 0x585bdcd5e135 in fits_read_compressed_img /home/adrian/cfitsio-4.5.0/
imcompress.c:4654:19
    #3 0x585bdcb14e96 in ffgpxvll /home/adrian/cfitsio-4.5.0/getcol.c:114:17
    #4 0x585bdcb14598 in ffgpxv /home/adrian/cfitsio-4.5.0/getcol.c:43:5
    #5 0x585bdca32740 in processImageHDU(fitsfile*, int, bool) /home/adrian/
FITSHarness/src/main.cpp:44:9
    #6 0x585bdca34fe2 in processFITSFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, bool) /home/adrian/
FITSHarness/src/main.cpp:144:13
    #7 0x585bdca35cd2 in main /home/adrian/FITSHarness/src/main.cpp:168:5
    #8 0x71e041c29d8f in __libc_start_call_main csu/../sysdeps/nptl/
libc_start_call_main.h:58:16
    #9 0x71e041c29e3f in __libc_start_main csu/../csu/libc-start.c:392:3
    #10 0x585bdc9718c4 in _start (/home/adrian/test_fits/FITSHarness/bin/
FITSHarness-afl-asan+0x7f8c4) (BuildId: d550951926686c2b1ec33b6625f11090e95b7336)

0x602000001612 is located 0 bytes to the right of 2-byte region
[0x602000001610,0x602000001612)
allocated by thread T0 here:
    #0 0x585bdc9f470e in malloc (/home/adrian/test_fits/FITSHarness/bin/
FITSHarness-afl-asan+0x10270e) (BuildId: d550951926686c2b1ec33b6625f11090e95b7336)
    #1 0x585bdcd4c1f8 in imcomp_decompress_tile /home/adrian/cfitsio-4.5.0/
imcompress.c:6269:34
    #2 0x585bdcd5e135 in fits_read_compressed_img /home/adrian/cfitsio-4.5.0/
imcompress.c:4654:19
    #3 0x585bdcb14e96 in ffgpxvll /home/adrian/cfitsio-4.5.0/getcol.c:114:17
    #4 0x585bdcb14598 in ffgpxv /home/adrian/cfitsio-4.5.0/getcol.c:43:5
    #5 0x585bdca32740 in processImageHDU(fitsfile*, int, bool) /home/adrian/
FITSHarness/src/main.cpp:44:9
    #6 0x585bdca34fe2 in processFITSFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, bool) /home/adrian/
FITSHarness/src/main.cpp:144:13
    #7 0x585bdca35cd2 in main /home/adrian/FITSHarness/src/main.cpp:168:5
    #8 0x71e041c29d8f in __libc_start_call_main csu/../sysdeps/nptl/
libc_start_call_main.h:58:16

SUMMARY: AddressSanitizer: heap-buffer-overflow /home/adrian/cfitsio-4.5.0/
ricecomp.c:936:17 in fits_rdecomp
Shadow bytes around the buggy address:
  0x0c047fff8270: fa fa fd fa fa fa fd fa fa fa fd fa fa fa fd fa
  0x0c047fff8280: fa fa fd fa fa fa fd fa fa fa fd fa fa fa fd fa
  0x0c047fff8290: fa fa fd fa fa fa fd fa fa fa fd fa fa fa fd fa
  0x0c047fff82a0: fa fa fd fa fa fa fd fa fa fa fd fa fa fa fd fa
  0x0c047fff82b0: fa fa fd fa fa fa fd fa fa fa fd fa fa fa 04 fa
=>0x0c047fff82c0: fa fa[02]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff82d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff82e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff82f0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8300: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

```
  0x0c047fff8310: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==1829377==ABORTING
```

Without ASAN, no crash is observed despite the overflow. Without ASAN, no crash is observed despite the overflow. This is because a minimal allocated chunk size (0x20) can fit the data without overwriting other variables. However, it is internal `glibc` behavior and can change in the future.

Reproduction Steps

To reproduce the issue, use the precompiled `FITSHarness-afl-asan` application (see Appendix A for additional details) along with the sample file stored in the `inputs/4/` directory.

```
FITSHarness-afl-asan inputs/4/afltriage_ASAN_heap-buffer-
overflow_READ_fits_rdecomp_4b31de665c61a6fe9fe0408efc695051.bin
```

The application will immediately crash. GDB can be used to debug the issue further.

Remediation

Ensure that the allocated buffer has sufficient size to fit 4 bytes.

CFITSIO-05. Out-Of-Bounds Write in ffpdfl	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	-
Status	Closed
Credits	Adrian Denkiewicz

Description

Stack buffer-overflow (out-of-bounds write) was observed in the `ffpdfl` function. The issue is triggered when processing a malformed FITS file, causing the application to access memory outside the allocated buffer.

The vulnerable code is found in `fitscore.c:7071`:

```
int ffpdfl(fitsfile *fptr,      /* I - FITS file pointer */
           int *status)         /* IO - error status     */
{
    char chfill, fill[2880];
    LONGLONG fillstart;
    int nfill, tstatus, ii;

    ...

    fillstart = (fptr->Fptr)->datastart + (fptr->Fptr)->heapstart +
                (fptr->Fptr)->heapsize;

    nfill = (long) ((fillstart + 2879) / 2880 * 2880 - fillstart);

    if ((fptr->Fptr)->hdutype == ASCII_TBL)
        chfill = 32;          /* ASCII tables are filled with spaces */
    else
        chfill = 0;           /* all other extensions are filled with zeros */

    ...

    /* fill values are incorrect or have not been written, so write them */

    memset(fill, chfill, nfill);  /* fill the buffer with the fill value */

    ...
}
```

The `memset` function is called with `nfill=4416` which exceeds the `fill` buffer's size (hardcoded to 2880 bytes). The `nfill` is calculated using the following values: `(fptr->Fptr)->datastart=2880`, `(fptr->Fptr)->heapstart=10018838894356725760`, and `(fptr->Fptr)->heapsize=0`.

Note, that the modern `glibc` builds (when compiled with stack-protector flags or `_FORTIFY_SOURCE` enabled) include runtime checks. When functions such as `memset` write past the end of a buffer, the corrupted canary value is detected, and the runtime calls `__stack_chk_fail`, which aborts the program. Thus, the issue might not be exploitable on modern systems.

The ASAN output is presented below:

```
==1830103==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7ffd21c5fc20 at pc 0x5abc538a5d3c bp 0x7ffd21c5f0b0 sp 0x7ffd21c5e880
WRITE of size 4416 at 0x7ffd21c5fc20 thread T0
    #0 0x5abc538a5d3b in __asan_memset (/home/adrian/test_fits/FITSHarness/bin/
FITSHarness-afl-asan+0x101d3b) (BuildId: d550951926686c2b1ec33b6625f11090e95b7336)
    #1 0x5abc539b6560 in ffpdfl /home/adrian/cfitsio-4.5.0/fitscore.c:7071:5
    #2 0x5abc539b4df7 in ffchdu /home/adrian/cfitsio-4.5.0/fitscore.c:6668:9
    #3 0x5abc53986f45 in ffmahd /home/adrian/cfitsio-4.5.0/fitscore.c:7817:17
    #4 0x5abc539bd8f8 in ffthdu /home/adrian/cfitsio-4.5.0/fitscore.c:8020:21
    #5 0x5abc538e6ab5 in processFITSFile(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&, bool) /home/adrian/
FITSHarness/src/main.cpp:131:9
    #6 0x5abc538e7cd2 in main /home/adrian/FITSHarness/src/main.cpp:168:5
    #7 0x77c4d8c29d8f in __libc_start_call_main csu/../sysdeps/nptl/
libc_start_call_main.h:58:16
    #8 0x77c4d8c29e3f in __libc_start_main csu/../csu/libc-start.c:392:3
    #9 0x5abc538238c4 in _start (/home/adrian/test_fits/FITSHarness/bin/
FITSHarness-afl-asan+0x7f8c4) (BuildId: d550951926686c2b1ec33b6625f11090e95b7336)

Address 0x7ffd21c5fc20 is located in stack of thread T0 at offset 2912 in frame
    #0 0x5abc539b612f in ffpdfl /home/adrian/cfitsio-4.5.0/fitscore.c:7015

  This frame has 2 object(s):
    [32, 2912) 'fill' (line 7016)
    [3040, 3044) 'tstatus' (line 7018) <== Memory access at offset 2912 partially
underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind
mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/adrian/test_fits/
FITSHarness/bin/FITSHarness-afl-asan+0x101d3b) (BuildId:
d550951926686c2b1ec33b6625f11090e95b7336) in __asan_memset
Shadow bytes around the buggy address:
  0x100024383f30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100024383f40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100024383f50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100024383f60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100024383f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100024383f80: 00 00 00 00[f2]f2 f2 f2 f2 f2 f2 f2 f2 f2 f2 f2
  0x100024383f90: f2 f2 f2 f2 04 f3 f3 f3 00 00 00 00 00 00 00 00
  0x100024383fa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100024383fb0: f1 f1 f1 f1 00 00 00 00 00 00 00 00 00 00 01 f2
  0x100024383fc0: f2 f2 f2 f2 04 f3 f3 f3 00 00 00 00 00 00 00 00
  0x100024383fd0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
```

```
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==1830103==ABORTING
```

Reproduction Steps

To reproduce the issue, use the precompiled `FITSHarness` application (see Appendix A for additional details) along with the sample file stored in the `inputs/5/` directory.

```
FITSHarness inputs/5/afltriage_ASAN_stack-buffer-
overflow_WRITE_ffpdfl_08ae54fa58c1460752fdb4181f049273.bin
```

The application will immediately crash. GDB can be used to debug the issue further.

Remediation

Ensure that calculated `nfill` does not exceed `fill` buffer's size. Alternatively, consider dynamically allocating all necessary buffers.

CFITSIO-06. Out-Of-Bounds Write in ffpsvc	
Vendor	NASA's HEASARC
Severity	**Low**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	-
Status	Closed
Credits	Adrian Denkiewicz

Description

Stack buffer-overflow (out-of-bounds write) was observed in the `ffpsvc` function. The issue is triggered when processing a malformed FITS file, causing the application to access memory outside the allocated buffer.

The vulnerable code is found in `fitscore.c:1542`:

```
int ffpsvc(char *card,    /* I - FITS header card (nominally 80 bytes long) */
           char *value,   /* O - value string parsed from the card */
           char *comm,    /* O - comment string parsed from the card */
           int *status)   /* IO - error status   */
/*
  ParSe the Value and Comment strings from the input header card string.
  If the card contains a quoted string value, the returned value string
  includes the enclosing quote characters.  If comm = NULL, don't return
  the comment string.
*/
{
    ...
    else if (card[ii] == '\'' )  /* is this a quoted string value? */
    {
        value[0] = card[ii];
        for (jj=1, ii++; ii < cardlen && jj < FLEN_VALUE-1; ii++, jj++)
        {
            if (card[ii] == '\'')  /*  is this the closing quote?  */
            {
                if (card[ii+1] == '\'')  /* 2 successive quotes? */
                {
                   value[jj] = card[ii];
                   ii++;
                   jj++;
                }
                else
                {
                    value[jj] = card[ii];
                    break;   /* found the closing quote, so exit this loop  */
                }
            }
            value[jj] = card[ii];  /* copy the next character to the output */
```

```
            }

            if (ii == cardlen || jj == FLEN_VALUE-1)
            {
                jj = minvalue(jj, FLEN_VALUE-2);  /* don't exceed 70 char string
length */
                value[jj] = '\'';  /*  close the bad value string  */
                value[jj+1] = '\0';  /*  terminate the bad value string  */
                ffpmsg("This keyword string value has no closing quote:");
                ffpmsg(card);
                /*  May 2008 - modified to not fail on this minor error  */
/*                return(*status = NO_QUOTE);  */
            }
            else
            {
                value[jj+1] = '\0';  /*  terminate the good value string  */
                ii++;   /*  point to the character following the value  */
            }
        }
        ...
}
```

In our case, the loop terminates when the `jj < FLEN_VALUE-1` is no longer true. The `FLEN_VALUE` is defined as:

```
#define FLEN_VALUE     71  /* max length of a keyword value string */
```

After the loop is completed, the `jj` equals to 70. The `value` buffer is allocated in the `ffbinit` function (`fitscore.c:4962`) as:

```
  char value[FLEN_VALUE];
```

Therefore, the faulty assignment writes to out-of-bounds offset 70th. This is likely to result in zeroing the next frame variable, which is `comm`. Such an assignment will not immediately crash the application unless the Address Sanitizer is enabled.

Reproduction Steps

To reproduce the issue, use the precompiled `FITSHarness-afl-asan` application (see Appendix A for additional details) along with the sample file stored in the `inputs/6/` directory.

```
FITSHarness-afl-asan inputs/6/afltriage_ASAN_stack-buffer-
overflow_WRITE_ffpsvc_2759fe07350624fa20b791bbdfcf8915.bin
```

The application will immediately crash. GDB can be used to debug the issue further.

Remediation

Ensure the last array element offset is properly calculated and no out-of-bounds write occurs.

Appendix A - FITSHarness Application

The following application has been prepared for fuzzing purposes. It attempts to parse a FITS file provided as a first command line argument. Internally, it uses the `fits_open_file` method, therefore the CFITSIO's Extended Filename Syntax is supported.

The source code can be found within the `FITSHarness/src` directory. We're also sharing some helpful CMake files that can be used to build the solution:

```
# assuming /home/<user>/cfitsio-4.5.0 contains the precompiled cfitsio library
cmake -S . -B build-vanilla -DCFITSIO_DIR=/home/<user>/cfitsio-4.5.0 -
DBUILD_TYPE=vanilla
cmake --build build-vanilla --verbose
# run harness
./build-vanilla/FITSHarness <source-file>
```

Additionally, two precompiled binaries are shared in the `FITSHarness/bin` directory. The precompiled `FITSHarness-afl-asan` can be used to reproduce ASAN-specific findings. Building an ASAN version manually is also possible if `CFITSIO` and `FITSHarness` projects are compiled with the `ASAN` libraries linked.

Disclosure Timeline

Date	Event
02/12/2025	Issue reported to the maintainers
03/26/2025	5 out of 6 issues patched in the 4.6.2 release
05/28/2025	Maintainers informed us that the fix for CFITSIO-3 is delayed, as it proved more problematic than expected; there is currently no planned release for this fix
06/12/2025	Redacted report created