Security Advisory
CFITSIO Library

Created by Adrian Denkiewicz

04/16/2026

Overview

Doyensec researchers discovered and reported sixteen (16) vulnerabilities in the CFITSIO Library. The work was conducted on the **cfitsio-4.6.3** (*386e719*) release.

It is important to reiterate that this report represents a snapshot of the environment's security posture at a point in time.

The findings describe numerous memory corruption issues. The root causes are primarily classic buffer overflows on both the stack and the heap, together with a smaller number of related memory-corruption bugs. Some of the issues reliably cause application crashes and denial of service. Others are more likely to be weaponizable into remote code execution. The ultimate exploitability depends strongly on the build and runtime environment. The fewer platform and compiler mitigations that are present, the higher the chance an attacker can turn one of these defects into a fully weaponized exploit.

A notable factor increasing exposure is the CFITSIO Extended Filename Syntax feature. Depending on the program architecture, this feature might directly accept filenames supplied by an external caller. In such a case, because the input is attacker-controlled at the entry point, all the reported bugs would be reachable directly from an attacker-supplied filename. This makes remote compromise realistic in configurations where the library processes filenames coming from untrusted sources. Although we did not rate any single finding as Critical at this stage, it remains plausible that an adversary could chain or adapt these bugs to achieve full code execution or system compromise, depending on environment and mitigations.

This report is a direct follow-up to Doyensec's CFITSIO research and advisory (published June 12, 2025). This time our assessment concentrated specifically on the CIFTSIO Extended Filename Syntax and its attacker-exposed parsing surface, which increases the reachability of several memory-corruption conditions described above.

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-EFS-01. Stack Buffer Overflow in ffifile2 During Row Filter Parsing	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	cfileio.c
Status	Closed
Credits	Adrian Denkiewicz

Description

A stack buffer overflow was identified in the `ffifile2` function while parsing row filter expressions in the Extended Filename Syntax. When copying a substring between square brackets, the function performs:

```
strncat(rowfilterx, ptr1 + 1, (ptr2 - ptr1 - 1));
```

The code tries to protect the `strncat` call with:

```
if (strlen(rowfilterx) + (ptr2-ptr1 + (*rowfilterx)?4:0) > FLEN_FILENAME - 1)
```

However, the `?:` operator has lower precedence than `+`, so this evaluates as `((strlen(rowfilterx) + (ptr2 - ptr1) + (*rowfilterx)) ? 4 : 0 > FLEN_FILENAME - 1)` and never accounts for the pending copy length `(ptr2 - ptr1 - 1)`. Crafted inputs such as `file.fits[2:f[R:f...]` could therefore bypass the guard and `strncat` would write bytes past `rowfilterx`, producing the ASAN crash.

Reproduction Steps

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

```
./fits-opener inputs/1/*
```

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

Remediation

Validate the computed substring length before concatenation to ensure it does not exceed `FLEN_FILENAME`:

We propose the following fix:

```
if (strlen(rowfilterx) + (ptr2 - ptr1 - 1) + ((*rowfilterx) ? 4 : 0) >
FLEN_FILENAME - 1) {
    free(infile);
    return (*status = URL_PARSE_ERROR);
}
```

This ensures the row filter never writes beyond the destination buffer.

CFITSIO-EFS-02. Stack Buffer Overflow in ffexts During Image Column Parsing	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	cfileio.c
Status	Closed
Credits	Adrian Denkiewicz

Description

A stack buffer overflow was identified in the `ffexts` function while parsing the "image-in-table" syntax (`;COLNAME(expr)`) of the Extended Filename Syntax. When copying the column name before the opening parenthesis, the code performs:

```
strncat(imagecolname, ptr1, ptr2 - ptr1);
```

The destination `imagecolname` is a 71-byte (`FLEN_VALUE`) stack buffer in `ffopen`, but the guard compared against `FLEN_FILENAME` (1025 bytes):

```
if (ptr2 - ptr1 > FLEN_FILENAME - 1)
```

As a result, column names longer than 70 characters bypass the check and `strncat` writes beyond `imagecolname`, clobbering the adjacent `rowexpress` buffer. In our case, ASAN reports a stack-buffer-overflow in `strncat` from `ffexts` with a WRITE of size 877.

Reproduction Steps

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

```
./fits-opener inputs/2/*
```

The application will immediately crash with an ASAN stack overflow report in `ffexts`.

Remediation

Ensure the substring length is bounded by the true capacity of `imagecolname`. Doyensec proposes the following fix:

```
if (ptr2 - ptr1 > FLEN_VALUE - 1) {
    return (*status = URL_PARSE_ERROR);
}
strncat(imagecolname, ptr1, ptr2 - ptr1);
```

CFITSIO-EFS-03. Null Pointer Dereference in file_openfile Tilde Expansion	
Vendor	NASA's HEASARC
Severity	**Low**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	drvrfile.c
Status	Closed
Credits	Adrian Denkiewicz

Description

When opening paths of the form `~user/file.fits`, the `file_openfile` function copies the username into a fixed 80-byte buffer and invokes `getpwnam` to resolve the home directory:

```
cptr = filename + 1;
while (*cptr && (*cptr != '/')) {
    user[ii] = *cptr;
    cptr++;
    ii++;
}
user[ii] = '\0';
pwd = getpwnam(user);
strcpy(tempname, pwd->pw_dir);
```

The copy loop doesn't enforce the buffer size, and the result of `getpwnam` isn't checked. A crafted filename with an overly long username (or a nonexistent account) leaves `pwd` as `NULL`, yet the code still dereferences `pwd->pw_dir`, crashing with a SIGSEGV in `file_openfile` (drvrfile.c:263).

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/3/` directory.

```
./fits-opener inputs/3/*
```

The application immediately terminates with a SIGSEGV inside `file_openfile` while expanding the `~user` prefix.

Remediation

To fix the issue, guard both the username copy and the `getpwnam` invocation:

```
while (*cptr && (*cptr != '/')) {
    if (ii >= (int)sizeof(user) - 1)
        return FILE_NOT_OPENED;
    user[ii++] = *cptr++;
}
user[ii] = '\0';

pwd = getpwnam(user);
if (!pwd || !pwd->pw_dir)
    return FILE_NOT_OPENED;
```

By rejecting usernames that don't fit in the buffer or that do not resolve to existing users, the tilde expansion no longer results in NULL pointer dereference.

CFITSIO-EFS-04. Double Free in Do_Deref During Pixel Filter Evaluation	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.y eval_y.c
Status	Closed
Credits	Adrian Denkiewicz

Description

The `Do_Deref` helper, which dereferences array expressions inside pixel filters (`PIX[...]`), allocates an output buffer (`this->value.data.ptr`) and then frees it whenever an error occurs (e.g., null index, index out of range, etc.):

```
if (theDims[i]->value.undef[row]) {
    yyerror(..., "Null encountered as vector index");
    free(this->value.data.ptr);
    break;
}
```

Rows are processed sequentially, so multiple rows can hit the same error and each call `free(this->value.data.ptr)` even though the buffer was already released. Subsequent cleanup paths also free the pointer unconditionally, leading to an ASAN-reported double free inside `Do_Deref` (eval.y:5464) when a crafted pixel filter expression causes repeated null-index errors.

Reproduction Steps

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

```
./fits-opener inputs/4/*
```

When the malformed pixel filter is parsed, AddressSanitizer reports a double free in `Do_Deref`.

Remediation

The issue could be addressed by introducing an internal "freed" flag within `Do_Deref` so error paths only release `this->value.data.ptr` once, and null the pointer afterward. For instance:

```
static void deref_free_value(Node *node, int *freed) {
    if (!*freed && node->value.data.ptr) {
        free(node->value.data.ptr);
```

```
            node->value.data.ptr = NULL;
            *freed = 1;
        }
    }

    /* ... inside Do_Deref ... */
    int valueFreed = 0;
    /* ... */
    yyerror(...);
    deref_free_value(this, &valueFreed);
```

This is a preliminary safeguard to stop the immediate double free. The HEASARC team may want to refactor the error-handling logic or centralize the cleanup to ensure no other paths leave partially freed parser nodes.

CFITSIO-EFS-05. Heap Buffer Overflow in ffcalchist During Histogram Generation	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	histo.c
Status	Closed
Credits	Adrian Denkiewicz

Description

When computing multi-dimensional histograms, `ffcalchist` walks through each row/element pair using a single running index `ii`:

```
for (elem = 1; elem <= histData->repeat; elem++, ii++) {
    if (colptr[0][ii] == DOUBLENULLVALUE) continue;
    /* ... */
    pix = (colptr[0][ii] - histData->amin1) / histData->binsize1;
```

The histogram setup assumes every iterator column delivers exactly `repeat * nrows` doubles, but this isn't validated. If the iterator's repeat (vector length) is smaller than the expected repeat, something an attacker can trigger via crafted `BIN(...)` or weight expressions, `ii` walks past the end of `colptr[k]`, and the subsequent reads (`colptr[0][ii]`, `colptr[1][ii]`, etc.) overrun the heap buffer.

In our case, ASAN reports a heap-buffer-overflow in `ffcalchist` (histo.c:3145) when fuzzed filters generate inconsistent repeats (~224 KB allocation overrun).

Reproduction Steps

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

```
./fits-opener inputs/5/*
```

The run immediately aborts under ASAN, showing a READ overflow at `ffcalchist`.

Remediation

Track each iterator column's actual capacity (`repeat * nrows`) via `fits_iter_get_repeat` when gathering `colptr`, and check `ii` against that maximum before every access. If any column provides fewer elements than expected, abort operation with the `BAD_DIMEN` error instead of reading out of bounds.

CFITSIO-EFS-06. Heap Buffer Overflow in fits_copy_image_section	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	cfileio.c
Status	Closed
Credits	Adrian Denkiewicz

Description

The image-section parser accepts expressions such as `[fpix:lpix:inc]` and computes the output size via `(smax - smin + sinc) / sinc`. These arithmetic expressions use signed long values and overflow when `sinc` (the stride) approaches `LONG_MAX` or when `fpixels`/`lpixels` are inverted with giant magnitudes. The wrap-around collapses the computed `outnaxes[0]` to 0 or 1 even though the requested slice is huge, so the subsequent row buffer allocation uses only a single byte:

```
outsize = outnaxes[0];
buffer = malloc((abs(bitpix) / 8) * outsize);  // often 1 byte
```

Later, `ffgsve` and `ffgcle` stream the entire requested slice (thousands of floats) into that tiny buffer, and `ffgbyt` performs an unchecked `memcpy` past the allocation.

In our case, ASAN reports WRITE of size 4 immediately after the single byte returned by `malloc` during `fits_select_image_section`.

Reproduction Steps

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

```
./fits-opener inputs/6/*
```

The crafted section string requests an enormous stride in the first axis, triggering the overflow and subsequent heap-buffer-overflow inside `ffgbyt`.

Remediation

Clamp all section-length computations (both when populating `outnaxes[]` and when computing the row/slice/cube iteration counts) to safe ranges by:

1. Recording differences using 64-bit absolute values.
2. Validating the stride is non-zero before dividing.
3. Falling back to `LONG_MAX` if the slice length exceeds what fits in a `long` type.

Reject zero increments up front and reuse the same guarded helper for every axis. This prevents wrap-around and keeps the row buffer sized for the true number of pixels, eliminating the overflow.

CFITSIO-EFS-07. Use-After-Free in Pixel Filter Vector Normalization	
Vendor	NASA's HEASARC
Severity	**High**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.y eval_f.c eval_y.c
Status	Closed
Credits	Adrian Denkiewicz

Description

The parser builds vector nodes by appending child nodes and later "closing" the vector via `Close_Vec`. When finalizing, `Close_Vec` coerces each child to the vector's target type by calling `New_Unary`.

`New_Unary` may allocate additional parser nodes, and a successful `Alloc_Node` can `realloc` the entire `ParseData::Nodes` array, invalidating any pointers into it. `Close_Vec` cached `Node *this = lParse->Nodes + vecNode` before the loop and continued to dereference it after `New_Unary` potentially reallocated the array, leaving `this` dangling and writing to freed memory.

Malformed pixel-filter expressions that force type promotions trigger deterministic heap-use-after-free crashes.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/7/` directory.

```
./fits-opener inputs/7/*
```

ASAN reports a heap-use-after-free in `Close_Vec` as soon as the expression coerces elements of a vector literal.

Remediation

Refresh the cached node pointer after each coercion: store the result of `New_Unary`, re-index `this = lParse->Nodes + vecNode`, and update the `SubNodes[]` entry before continuing. Do not use cached values.

This ensures `Close_Vec` always refers to the current `Nodes` array even if it was reallocated mid-loop.

CFITSIO-EFS-08. Stack Buffer Overflow in ffbinse Column Name Parsing	
Vendor	NASA's HEASARC
Severity	**High**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	histo.c
Status	Closed
Credits	Adrian Denkiewicz

Description

When parsing the `BIN(...)` portion of an Extended Filename Syntax URL, `ffbinse` copies each comma-separated column name into fixed-size `char colname[4][FLEN_VALUE]` buffers using `strncat`. The copy length is derived solely from `strcspn(ptr, " ,)")`, so a malicious binning string can provide a token hundreds of bytes long and overflow the stack-allocated column name.

This results in a deterministic stack-buffer-overflow in `strncat` when fuzzed URLs include deliberately long column names.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/8/` directory.

```
./fits-opener inputs/8/*
```

The process aborts inside `ffbinse` as soon as it copies the first oversized column token.

Remediation

Clamp token lengths before copying: compare the `strcspn` result against `FLEN_VALUE`, and reject the binning specification with `URL_PARSE_ERROR` if the column name exceeds the fixed buffer. This prevents `strncat` from ever writing past `colname[axis]`.

We propose the following fix:

```
slen = strcspn(ptr, " ,)");
if (slen >= FLEN_VALUE) {
    ffpmsg("column name too long in binning specification");
    ffpmsg(binspec);
    return (*status = URL_PARSE_ERROR);
}
strncat(colname[ii], ptr, slen);
```

With this bound check in place, untrusted binning expressions no longer corrupt the stack, and the parser safely rejects overly long column tokens.

CFITSIO-EFS-09. Stack Buffer Overflows in ffourl Output URL Parsing	
Vendor	NASA's HEASARC
Severity	**High**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	cfileio.c
Status	Closed
Credits	Adrian Denkiewicz

Description

The Extended Filename Syntax lets callers specify an output file, optional template `()` block, and compression spec `[]`. The `ffourl` function copies those segments into fixed-size stack buffers (`tmplfile`, `compspec`) using `strncat`, but it never ensures that the cumulative copy stays within the buffer's capacity.

A crafted filename with hundreds of characters between `( )` or `[ ]` causes `strncat` to write past the end of the local arrays, clobbering adjacent stack data in `ffinit` before control returns to the caller. Because `ffopen` accepts untrusted URLs and this overflow happens before any privilege boundary, an attacker can reliably smash the stack and, on systems without modern mitigations, seize control of execution.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/9/` directory.

```
./fits-opener inputs/9/*
```

The process aborts with stack-buffer-overflow in `ffourl` as soon as the overlong template/compression substring is appended.

Remediation

Add explicit length checks before each `strncat` in the template/compression parsing pathway of `ffourl`, rejecting any segment whose length would exceed `FLEN_FILENAME` when added to the current buffer contents.

Additionally, ensure the stack buffers provided by callers (e.g., `compspec` in `ffinit`) are actually sized to `FLEN_FILENAME`, so the parser's bounds checks match the destination's real capacity.

CFITSIO-EFS-10. Stack Buffer Overflow in Lexical Parser (fits_parser_yylex)	
Vendor	NASA's HEASARC
Severity	**High**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.l eval_l.c
Status	Closed
Credits	Adrian Denkiewicz

Description

CFITSIO's expression engine accepts literals of the form `0b(...)`, `0o(...)`, `0x(...)` when evaluating arithmetic expressions. Those expressions can be invoked directly through APIs such as `fits_calc_binningde` or indirectly via Extended Filename Syntax (EFS) elements like `BIN(...)`, `PIXT`, or row filters.

For octal/hex literals, the lexer expands each digit to a 3- or 4-character binary string and concatenates it into `char bitstring[256]` on the stack. The code uses repeated `strcat` calls without enforcing the cumulative length, so an attacker-controlled literal with a few hundred digits overflows `bitstring`, overwriting adjacent stack data before the parser returns. Since EFS funnels user-supplied histogram/pixel-filter expressions into this parser, a malicious FITS URL can trigger the overflow simply by embedding a long `0x…` literal inside `BIN(...)`.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/10/` directory.

```
./fits-opener inputs/10/*
```

Observe stack-buffer-overflow at `fits_parser_yylex`, originating from the repeated `strcat(bitstring, HEX_X)` calls.

Remediation

The issue can be mitigated by implementing the following checks:

1. In the lexer rules (inside the `eval.l` file), track the total number of output bits while expanding literals. Before concatenating each chunk, verify that `bitlen + chunk_len <= sizeof(bitstring) - 1`. If not, raise `PARSE_SYNTAX_ERR` (e.g., via `yylParse->status` and `ffpmsg`) instead of writing past the buffer.
2. Replace the `strcat` loops with guarded `memcpy`/`strncpy` to avoid repeated scanning and ensure exact bounds control.

3. Regenerate `eval_l.c` (or manually mirror the safeguard) so the runtime lexer matches the fixed logic.

CFITSIO-EFS-11. Integer Division Overflow in Do_BinOp_lng	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.y
Status	Closed
Credits	Adrian Denkiewicz

Description

CFITSIO's expression engine (invoked via Extended Filename Syntax features such as `BIN(...)` and `PIXT ...`) implements arithmetic on 64-bit integers in `Do_BinOp_lng`. The division (`/`) and modulus (`%`) cases only guard against divide-by-zero. They do not handle the well-known overflow when dividing the minimum signed long (`LONG_MIN`) by `-1`. On two's-complement hardware this raises a `SIGFPE` (integer overflow trap) before the parser can mark the value undefined. A crafted pixel-filter or binning expression that evaluates `LONG_MIN / -1` therefore crashes the host process, making denial of service trivial.

Any FITS URL that embeds such an expression (e.g., `filename.fits[PIXT ... LONG_MIN/-1 ...]`) exercises this parser path and can be used to remotely crash the process instance.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/11/` directory.

```
./fits-opener inputs/11/*
```

Observe that the process aborts with SIGFPE in `Do_BinOp_lng`.

Remediation

Add support for the special-cases: `LONG_MIN / -1` and `LONG_MIN % -1` inside `Do_BinOp_lng`.

CFITSIO-EFS-12. Divide-by-Zero in fits_calc_binningde Integer Bin Calculation	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	histo.c
Status	Closed
Credits	Adrian Denkiewicz

Description

When parsing Extended Filename Syntax `BIN(...)` clauses, `fits_calc_binningde` derives a histogram bin width from user-controlled bounds. If the expression forces `amax == amin` (or otherwise collapses the range), the fallback `(amax - amin) / 10` logic produces a floating-point `binsize[ii]` of exactly `0.0`. Because integer histograms require integral widths, the code casts this to `ibin` and immediately evaluates `(imax - imin) / ibin`, yielding a deterministic `SIGFPE`.

Crafted URLs such as `file.fits[BIN ...]` therefore terminate any CFITSIO process that opens attacker-controlled strings.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/12/` directory.

```
./fits-opener inputs/12/*
```

The process crashes in `fits_calc_binningde` when it attempts to divide by the zero-width `ibin`.

Remediation

Reject zero-width bins after all range adjustments but before integer math is performed.

We propose the following fix:

```
if (binsize[ii] == 0.) {
    ffpmsg("error: computed histogram binsize = 0");
    return (*status = ZERO_SCALE);
}
```

This turns malicious BIN specifications into parse errors rather than runtime faults.

CFITSIO-EFS-13. Null Pointer Dereference in Do_BinOp_dbl Vector Operands	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.y eval_y.c
Status	Closed
Credits	Adrian Denkiewicz

Description

Extended Filename Syntax row filters are parsed before opening a FITS HDU. Malicious expressions can steer the parser into building binary-operation nodes whose vector operands never receive valid column buffers. When `Do_BinOp_dbl` evaluates such a node, it unconditionally dereferences `thatX->value.data.dblptr` and `thatX->value.undef`. This might result in read from the zero page, producing a deterministic SIGSEGV during `fits_open_file`.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/13/` directory.

```
./fits-opener inputs/13/*
```

The NULL page access causes segmentation fault in `Do_BinOp_dbl` while evaluating the supplied row expression.

Remediation

Before performing vector math, verify that each non-constant operand has valid backing storage. Reject evaluation if either `dblptr` or `undef` is null or still points into the guard page, returning `PARSE_SYNTAX_ERR`.

CFITSIO-EFS-14. Out-of-Bounds Read in Do_Offset Column Shifts	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval.y eval_y.c
Status	Closed
Credits	Adrian Denkiewicz

Description

Extended Filename Syntax allows expressions such as `COLUMN{constant}` to reference rows with a fixed offset. The parser enforces only that the offset is a constant integer. In `Do_Offset` the constant is multiplied by the column repeat count and used to shift row pointers without checking for overflow or bounds. A malicious row-expression can supply a very large offset, causing the computed index to wrap around or exceed the buffered table window, so `col->value.data.*` accesses memory past the valid column buffer.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/14/` directory.

```
./fits-opener inputs/14/*
```

The process segfaults inside `Do_Offset` while copying the overlapped rows.

Remediation

Reject offsets that would index outside the buffered data.

CFITSIO-EFS-15. Null Pointer Dereference in Histogram Iterator Initialization	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	putcol.c
Status	Closed
Credits	Adrian Denkiewicz

Description

Extended Filename Syntax histogram expressions can steer CFITSIO into calling the iterator framework with zero or malformed iterator columns.

`ffiter` assumes `cols[0].fptr` always references a valid FITS handle and immediately calls `ffghdt(cols[0].fptr, …)` to query the HDU type. When the column array is empty or its first element lacks an attached fitsfile, `cols[0].fptr` is null and `ffghdt` dereferences `NULL` page.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/15/` directory.

```
./fits-opener inputs/15/*
```

The process terminates due to a null pointer dereference.

Remediation

Add defensive checks to `ffiter`:

1. Reject `n_cols == 0`
2. Ensure every non-temporary iterator column supplies a non-null `fptr`
3. Determine the "base" FITS pointer by scanning for the first valid column instead of blindly using `cols[0]`

CFITSIO-EFS-16. Null Pointer Crash in Pixel-Filter Column Lookup	
Vendor	NASA's HEASARC
Severity	**Medium**
Vulnerability Class	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
Component	eval_f.c histo.c
Status	Closed
Credits	Adrian Denkiewicz

Description

Extended Filename Syntax expressions that reference image pseudo-columns force the parser down the pixel-filter path inside `find_column`. That routine assumes `lParse->pixFilter` is a valid structure because `ffiprs` preserves it across a `memset`. When callers instantiate `ParseData` on the stack without clearing it - as happens along the default EFS path - `pixFilter` can hold stray stack bytes (such as `0x1`). `find_column` then dereferences `pixFilter->count` and `pixFilter->tag[]`, crashing in `cmp DWORD PTR [rdi],0x0` assembler instruction, before any sanity checks fire.

Reproduction Steps

To reproduce the issue, use the precompiled `fits-opener` application (see Appendix A for additional details) along with the sample file stored in the `inputs/16/` directory.

```
./fits-opener inputs/16/*
```

The process segfaults in `find_column`, showing `lParse->pixFilter == 0x1` (or other stack value).

Remediation

Ensure every parser entry point zero-initializes its `ParseData` before calling `ffiprs`.

Declaring locals as `ParseData lParse = {0};` keeps `pixFilter` NULL unless a valid filter is explicitly installed, allowing `find_column` to reject malformed EFS input via its existing guard.

Appendix A - FITS-Opener Test Application

The following application has been prepared for fuzzing purposes. It attempts to open a sample FITS file using the CFITSIO's Extended Filename Syntax provided as a first command line argument. The syntax is fuzzed using AFL++ to stress the library and find various unexpected edge cases.

```
#include <cstdio>
#include <cstdlib>
#include <cstdint>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>

extern "C" {
#include "fitsio.h"
}

int main(int argc, char** argv) {
    if (argc < 2) {
        // no input file specified -> silent non-zero exit
        return 1;
    }

    const char* input_path = argv[1];
    const bool verbose = (argc >= 3 && std::string(argv[2]) == "-v");

    // Read the entire file (binary) into a buffer
    std::ifstream ifs(input_path, std::ios::in | std::ios::binary);
    if (!ifs) {
        if (verbose) std::fprintf(stderr, "fuzz_open: failed to open '%s'\n",
input_path);
        return 1;
    }

    ifs.seekg(0, std::ios::end);
    std::streamsize size = ifs.tellg();
    ifs.seekg(0, std::ios::beg);

    const std::streamsize MAX_SZ = 1 << 20; // 1 MiB cap
    if (size < 0) size = 0;
    if (size > MAX_SZ) size = MAX_SZ;

    std::vector<char> buf;
    buf.resize(static_cast<size_t>(size) + 1);
    std::streamsize actually_read = 0;
    if (size > 0) {
        ifs.read(buf.data(), size);
        actually_read = ifs.gcount();
    }
    buf[actually_read] = '\0'; // null-terminate

    // Create a std::string from the buffer (this will stop at the first NUL)
    std::string fname(buf.data(), static_cast<size_t>(actually_read));

    // Strip all newline characters (both LF and CR) from the filename string.
    // Filenames should not contain newlines; removing them helps harness inputs.
```

```
    fname.erase(std::remove_if(fname.begin(), fname.end(),
                [](char c){ return c == '\n' || c == '\r'; }),
                fname.end());

    if (fname.empty()) {
        if (verbose) std::fprintf(stderr, "fuzz_open: filename empty after
stripping newlines\n");
        return 1;
    }

    fitsfile* fptr = nullptr;
    int status = 0;

    // Call only fits_open_file (READONLY)
    fits_open_file(&fptr, fname.c_str(), READONLY, &status);

    // If opened, close it (fits_close_file will update status on error)
    if (fptr != nullptr) {
        fits_close_file(fptr, &status);
    }

    if (verbose && status != 0) {
        char errtxt[FLEN_ERRMSG];
        fits_get_errstatus(status, errtxt);
        std::fprintf(stderr, "cfitsio status=%d: %s\n", status, errtxt);
    }

    // Return status as exit code. Clamp to 0..255 to be a valid POSIX exit status.
    int exit_code = status & 0xFF;
    return exit_code;
}
```

The application along with the precompiled binaries are provided in a separate archive.

Disclosure Timeline

Date	Event
11/17/2025	Issues reported to the CFITSIO maintainers
11/17/2025	Maintainers acknowledged the report
03/31/2026	All findings closed
04/16/2026	Public disclosure