New issue

Uncontrolled allocation (OOM) in drflac__read_and_decode_metadata (dr_flac.h:6750) #298

Closed

kapnull opened on Mar 5

In line with the project's security policy favoring public and transparent handling, I am opening this issue to coordinate a fix.

Summary

`drflac__read_and_decode_metadata` reads a 4-byte `mimeLength` field from a PICTURE metadata block and immediately calls `malloc(mimeLength + 1)` before checking that `mimeLength` fits within the block's declared byte range. An attacker supplying a crafted FLAC stream can force a single allocation of up to ~4 GiB from a 54-byte input, and from a 78-byte hand-crafted standalone input, causing denial of service. The same ordering mistake is repeated for `descriptionLength` at line 6772.

Affected entry points: any `drflac_open_*_with_metadata()` variant that accepts a non-NULL metadata callback. `drflac_open_memory()` and `drflac_open_file()` (no callback) are not affected - the PICTURE body is only parsed when `onMeta != NULL` (line 6728).

- **Version:** v0.13.3 (latest)
- **Platform:** Ubuntu 24.04, clang 18

Root Cause

Inside the `DRFLAC_METADATA_BLOCK_TYPE_PICTURE` case of `drflac__read_and_decode_metadata`, after reading 4 bytes for `pictureType` and 4
bytes for `mimeLength`, the allocation happens at line 6750 before the bounds check at line 6756.

`blockSizeRemaining` tracks how many bytes remain in the PICTURE block (maximum ~16 MB, derived from a 3-byte blockSize field). It is never compared against `mimeLength` before the allocation. A PICTURE block with a small declared `blockSize` and a large `mimeLength` field produces a multi-GiB `malloc` from a tiny input.

The same pattern repeats at line 6772 for `descriptionLength`.

Note: `mimeLength = 0xFFFFFFFF` causes `mimeLength + 1` to wrap to 0 as `drflac_uint32` before reaching `malloc`, resulting in `malloc(0)`. The fix below also prevents this case since `blockSizeRemaining` is always less than `0xFFFFFFFF`.

Fuzzer Discovery

The vulnerability was found by fuzzing `drflac_open_memory_with_metadata` with libFuzzer. A 54-byte minimized crash input (`minimized_oom.flac`) was produced.

fuzzer.zip

Build:

```
clang -fsanitize=address,undefined,fuzzer -O1 -g -o repro repro.c
./repro minimized_oom.flac
```

▶ repro.c

Fuzzer output:

```
$ clang -fsanitize=address,undefined,fuzzer -O1 -g -o repro repro.c && ./repro
minimized_oom.flac
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 352751702
INFO: Loaded 1 modules   (15743 inline 8-bit counters): 15743 [0x5ca91a851af8,
0x5ca91a855877),
INFO: Loaded 1 PC tables (15743 PCs): 15743 [0x5ca91a855878,0x5ca91a893068),
./repro: Running 1 inputs 1 time(s) each.
Running: minimized_oom.flac
==4929== ERROR: libFuzzer: out-of-memory (malloc(4282580993))
   To change the out-of-memory limit use -rss_limit_mb=<N>

    #0 0x5ca91a703b65 in __sanitizer_print_stack_trace
(/home/ana/dr_libs/fuzz/FINAL/repro+0x191b65) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #1 0x5ca91a65d67c in fuzzer::PrintStackTrace()
(/home/ana/dr_libs/fuzz/FINAL/repro+0xeb67c) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #2 0x5ca91a642515 in fuzzer::Fuzzer::HandleMalloc(unsigned long)
(/home/ana/dr_libs/fuzz/FINAL/repro+0xd0515) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
```

```
    #3 0x5ca91a64241f in fuzzer::MallocHook(void const volatile*, unsigned long)
(/home/ana/dr_libs/fuzz/FINAL/repro+0xd041f) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #4 0x5ca91a70b426 in __sanitizer::RunMallocHooks(void*, unsigned long)
(/home/ana/dr_libs/fuzz/FINAL/repro+0x199426) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #5 0x5ca91a660c62 in __asan::Allocator::Allocate(unsigned long, unsigned long,
__sanitizer::BufferedStackTrace*, __asan::AllocType, bool)
(/home/ana/dr_libs/fuzz/FINAL/repro+0xeec62) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #6 0x5ca91a660597 in __asan::asan_malloc(unsigned long,
__sanitizer::BufferedStackTrace*) (/home/ana/dr_libs/fuzz/FINAL/repro+0xee597) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #7 0x5ca91a6f8e63 in malloc (/home/ana/dr_libs/fuzz/FINAL/repro+0x186e63) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #8 0x5ca91a73a9ff in drflac__read_and_decode_metadata
/home/ana/dr_libs/fuzz/FINAL/./dr_flac.h:6750:36
    #9 0x5ca91a73a9ff in drflac_open_with_metadata_private
/home/ana/dr_libs/fuzz/FINAL/./dr_flac.h:8090:14
    #10 0x5ca91a7406a7 in drflac_open_memory_with_metadata
/home/ana/dr_libs/fuzz/FINAL/./dr_flac.h:9008:13
    #11 0x5ca91a77ccb3 in LLVMFuzzerTestOneInput
/home/ana/dr_libs/fuzz/FINAL/repro.c:14:21
    #12 0x5ca91a644cd4 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
long) (/home/ana/dr_libs/fuzz/FINAL/repro+0xd2cd4) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #13 0x5ca91a62de06 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long)
(/home/ana/dr_libs/fuzz/FINAL/repro+0xbbe06) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #14 0x5ca91a6338ba in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long)) (/home/ana/dr_libs/fuzz/FINAL/repro+0xc18ba) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #15 0x5ca91a65e076 in main (/home/ana/dr_libs/fuzz/FINAL/repro+0xec076) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)
    #16 0x7f8745e2a1c9 in __libc_start_call_main
csu/../sysdeps/nptl/libc_start_call_main.h:58:16
    #17 0x7f8745e2a28a in __libc_start_main csu/../csu/libc-start.c:360:3
    #18 0x5ca91a6289d4 in _start (/home/ana/dr_libs/fuzz/FINAL/repro+0xb69d4) (BuildId:
406f5e653ce2e0cfb772db107fe84e2de3d50b2f)

SUMMARY: libFuzzer: out-of-memory
```

libFuzzer's `HandleMalloc` hook fires when a single allocation request exceeds the RSS limit. The `malloc(4282580993)` request (≈4 GiB) originates at `dr_flac.h:6750` from `mimeLength = 0xFF430000` in `minimized_oom.flac`.

Standalone Reproducer

To confirm the finding is reachable without any fuzzer runtime, a standalone reproducer constructs the 78-byte attack payload entirely in code and calls the vulnerable API directly from `main()`.

Note: the fuzzer-minimized input uses `mimeLength = 0xFF430000` (4,282,580,992); the standalone uses `0xFFFFFFFE` (4,294,967,294), the largest value that does not wrap to zero on `mimeLength + 1`. Both are attacker-controlled values that trigger the same unchecked allocation at line 6750.

Build and run:

```
clang -fsanitize=address -O1 -g -o repro_main repro_main.c
./repro_main
```

repro_main.c

Output:

```
dr_flac PICTURE mimeLength OOM reproducer
Crafted mimeLength: 0xFFFFFFFE = 4294967294 bytes (4.00 GiB)

Calling drflac_open_memory_with_metadata()...
  [on_meta] type=0
    #0 0x62c5abb1cf25 in __sanitizer_print_stack_trace
    #1 0x62c5abb50770 in __sanitizer_malloc_hook        repro_main.c:43
    #2 0x62c5abb247d2 in __sanitizer::RunMallocHooks(void*, unsigned long)
    #3 0x62c5aba7a022 in __asan::Allocator::Allocate(...)
    #4 0x62c5aba79957 in __asan::asan_malloc(...)
    #5 0x62c5abb12223 in malloc
    #6 0x62c5abb519e4 in drflac__malloc_from_callbacks  dr_flac.h:6374
    #7 0x62c5abb519e4 in drflac__read_and_decode_metadata  dr_flac.h:6750
    #8 0x62c5abb519e4 in drflac_open_with_metadata_private  dr_flac.h:8090
    #9 0x62c5abb67e49 in drflac_open_memory_with_metadata  dr_flac.h:9008
   #10 0x62c5abb67e49 in main                           repro_main.c:94
   #11 0x79b2f7a2a1c9 in __libc_start_call_main
   #12 0x79b2f7a2a28a in __libc_start_main
   #13 0x62c5aba77374 in _start
Aborted (core dumped)
```

`__sanitizer_malloc_hook` is an ASan weak symbol called by the allocator on every allocation. The hook aborts when the requested size exceeds 512 MiB, printing the symbolised call stack. Frame `#7` confirms the allocation originates at `dr_flac.h:6750` inside `drflac__read_and_decode_metadata`, reached directly from `main()` with no fuzzer runtime present.

▶ repro_main.c

GDB Verification

Both sessions were run against the standalone binary (`repro_main`, built with `-fsanitize=address -O0 -g`). No fuzzer runtime is present anywhere in the call stack.

Session 1 - allocation size at the call site

A breakpoint was placed at `dr_flac.h:6750`. Execution was stepped into `drflac__malloc_from_callbacks` to read the argument directly:

```
Temporary breakpoint 2, drflac__read_and_decode_metadata (onRead=0x55555565e720
<drflac__on_read_memory>, onSeek=0x55555565e950 <drflac__on_seek_memory>,
    onTell=0x55555565eb30 <drflac__on_tell_memory>, onMeta=0x55555567ea50 <on_meta>,
pUserData=0x7ffff5e00020, pUserDataMD=0x0, pFirstFramePos=0x7ffff6401320,
    pSeektablePos=0x7ffff6401340, pSeekpointCount=0x7ffff6401360,
pAllocationCallbacks=0x7ffff6401370) at ./dr_flac.h:6750
6750                        pMime =
(char*)drflac__malloc_from_callbacks(metadata.data.picture.mimeLength + 1,
pAllocationCallbacks); /* +1 for null terminator. */
(gdb) list 6746,6762
6746                        }
6747                        blockSizeRemaining -= 4;
6748                        metadata.data.picture.mimeLength =
drflac__be2host_32(metadata.data.picture.mimeLength);
6749
6750                        pMime =
(char*)drflac__malloc_from_callbacks(metadata.data.picture.mimeLength + 1,
pAllocationCallbacks); /* +1 for null terminator. */
6751                        if (pMime == NULL) {
6752                            result = DRFLAC_FALSE;
6753                            goto done_flac;
6754                        }
6755
6756                        if (blockSizeRemaining < metadata.data.picture.mimeLength ||
onRead(pUserData, pMime, metadata.data.picture.mimeLength) !=
metadata.data.picture.mimeLength) {
6757                            result = DRFLAC_FALSE;
6758                            goto done_flac;
6759                        }
6760                        blockSizeRemaining -= metadata.data.picture.mimeLength;
6761                        pMime[metadata.data.picture.mimeLength] = '\0';  /* Null
terminate for safety. */
6762                        metadata.data.picture.mime = (const char*)pMime;
(gdb) step
drflac__malloc_from_callbacks (sz=4294967295, pAllocationCallbacks=0x7ffff6401370) at
./dr_flac.h:6369
6369        if (pAllocationCallbacks == NULL) {
(gdb)
```

`sz=4294967295` confirms `malloc(mimeLength + 1) = malloc(0xFFFFFFFE + 1)` is called before any bounds check.

Session 2 - causation confirmed by patching mimeLength at runtime

Breakpoints were placed at both `dr_flac.h:6750` (the malloc call) and `dr_flac.h:6756` (the bounds check). `mimeLength` was overwritten with 100 at line 6750, then execution was resumed. The breakpoint at line 6756 fired next, confirming that (a) malloc succeeded for the small allocation, (b) `blockSizeRemaining` was directly read as 24, and (c) the condition `24 < 100` is true, so short-circuit evaluation prevents `onRead` from being called and the function takes `goto done_flac`.

```
dr_flac PICTURE mimeLength OOM reproducer
Crafted mimeLength: 0xFFFFFFFE = 4294967294 bytes (4.00 GiB)

Calling drflac_open_memory_with_metadata()...
  [on_meta] type=0

Breakpoint 1, drflac__read_and_decode_metadata (onRead=0x555555556490
<drflac__on_read_memory>, onSeek=0x555555556580 <drflac__on_seek_memory>,
    onTell=0x5555555566a0 <drflac__on_tell_memory>, onMeta=0x5555555623d0 <on_meta>,
pUserData=0x7fffffffdc88, pUserDataMD=0x0, pFirstFramePos=0x7fffffffc9b0,
    pSeektablePos=0x7fffffffc9a8, pSeekpointCount=0x7fffffffc9a4,
pAllocationCallbacks=0x7fffffffc980) at ./dr_flac.h:6750
6750                        pMime =
(char*)drflac__malloc_from_callbacks(metadata.data.picture.mimeLength + 1,
pAllocationCallbacks); /* +1 for null terminator. */
(gdb) p metadata.data.picture.mimeLength
$1 = 4294967294
(gdb) set metadata.data.picture.mimeLength = 100
(gdb) p metadata.data.picture.mimeLength
$2 = 100
(gdb) c
Continuing.

Breakpoint 2, drflac__read_and_decode_metadata (onRead=0x555555556490
<drflac__on_read_memory>, onSeek=0x555555556580 <drflac__on_seek_memory>,
    onTell=0x5555555566a0 <drflac__on_tell_memory>, onMeta=0x5555555623d0 <on_meta>,
pUserData=0x7fffffffdc88, pUserDataMD=0x0, pFirstFramePos=0x7fffffffc9b0,
    pSeektablePos=0x7fffffffc9a8, pSeekpointCount=0x7fffffffc9a4,
pAllocationCallbacks=0x7fffffffc980) at ./dr_flac.h:6756
6756                        if (blockSizeRemaining < metadata.data.picture.mimeLength ||
onRead(pUserData, pMime, metadata.data.picture.mimeLength) !=
metadata.data.picture.mimeLength) {
(gdb) p blockSizeRemaining
$3 = 24
(gdb) p metadata.data.picture.mimeLength
$4 = 100
(gdb) c
Continuing.

Result: NULL returned
[Inferior 1 (process 5644) exited normally]
```

With `mimeLength` set to 100, execution reaches the breakpoint at line 6756 directly. `blockSizeRemaining = 24` and `mimeLength = 100` are read from the live process; since `24 < 100`, the bounds check condition is true and the function takes `goto done_flac`, returning NULL. The process exits normally.

Suggested Fix

Move the bounds check for `mimeLength` to before the allocation at line 6750, matching the pattern already used correctly for `pictureDataSize` at line 6824:

```
/* After line 6748: */
metadata.data.picture.mimeLength = drflac__be2host_32(metadata.data.picture.mimeLength);

/* ADD before line 6750: */
if (blockSizeRemaining < metadata.data.picture.mimeLength) {
    result = DRFLAC_FALSE;
    goto done_flac;
}

pMime = (char*)drflac__malloc_from_callbacks(metadata.data.picture.mimeLength + 1,
                                             pAllocationCallbacks);
```

Apply the same fix at line 6772 for `descriptionLength`:

```
/* After line 6770: */
metadata.data.picture.descriptionLength = drflac__be2host_32(...);

/* ADD before line 6772: */
if (blockSizeRemaining < metadata.data.picture.descriptionLength) {
    result = DRFLAC_FALSE;
    goto done_flac;
}

pDescription = (char*)drflac__malloc_from_callbacks(
    metadata.data.picture.descriptionLength + 1, pAllocationCallbacks);
```

kapnull on Mar 23 — Author

CVE-2026-32836 has been assigned to this issue.

mackron added 3 commits that reference this issue 4 days ago

dr_flac: Fix a possible overflow error when parsing picture metadata. ···	663239a
dr_flac: Fix a possible overflow error when parsing picture metadata. ···	fefced4
dr_flac: Add some bounds checking when parsing metadata. ···	4f5a4cd

mackron 4 days ago — Owner

Thanks. This should be addressed in the master branch. Could you verify that?

I've added an additional more general check to compare the block size against the size of the file to use that as an upper bound limit. This should also address unbounded mallocs for other block types.

It looks like this was generated by AI. In future please cut down on the verbiage and keep the report simple and concise. I'll ask follow up questions if I have any.

kapnull 4 days ago · edited by kapnull Edits Author •••

Yeah I just verified, the bounds check at line 6768 fires first and I get a NULL return with no allocation attempt. Fix works.

Thanks for the feedback. The fuzzing and GDB sessions were done manually but I used an LLM to draft the report from my notes and output rather than writing it up by hand. I'll keep it concise in the future.

mackron 4 days ago Owner •••

Thanks for checking that. Will go ahead and close this one now.

mackron closed this as completed 4 days ago

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants