Talos Vulnerability Report

TALOS-2022-1486

HDF5 Group libhdf5 gif2h5 out-of-bounds read vulnerability

AUGUST 16, 2022

CVE NUMBER

CVE-2022-25942

SUMMARY

An out-of-bounds read vulnerability exists in the gif2h5 functionality of HDF5 Group libhdf5 1.10.4. A specially-crafted GIF file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

HDF5 Group libhdf5 1.10.4

PRODUCT URLS

libhdf5 - https://www.hdfgroup.org

CVSSV3 SCORE

7.8 - CVSS:3.0/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CWE

CWE-125 - Out-of-bounds Read

DETAILS

HDF5 is a file format that is maintained by a non-profit organization, the HDF Group. HDF5 is designed to store and organize large amounts of scientific data. It is used to exchange data structures between applications in industries (such as the GIS industry) via libraries such as GDAL, OGR or as part of software like ArcGIS.

The vulnerability exists in the HDF5 gif2h5 tool/library, specifically in ReadGifHeader() function while attempting to convert GIF files to their HDF5 format. The vulnerability exists due to their failure to check the file size compared to the Global Palette Size specified in the GIF file. This leads to repeated data filling the gifHead->HDFPalette and returning what should be a pointer to the GIF data from the original file. Instead, it points just past the palette data in the destination structure.

In gif2mem.c:

```
96     /* Read the GIF image file header information */
97     ReadGifHeader(gifHead, &MemGif);
```

We can see this occur in the ReadGifHeader() function within gifread.c. The size of the file isn't checked against the table size before attempting to read palette data. In this example, we specified a pallete size of 0x100 (max) bytes with a file that is only 0x5e bytes long

```
 79     /* Check if a Global Color Table is present */
 80     if (GifHead->PackedField & 0x80) {
 81         /* Read number of color table entries */
 82         tableSize          = (GIFWORD)(1L << ((GifHead->PackedField & 0x07) +
1));
 83         GifHead->TableSize = tableSize;
 84
 85         /* Read the Global Color Table */
 86
 87         /*
 88          * There are some changes made here apart from just reading in the
 89          * global color table as would seem intuitively obvious.  The colors
 90          * are stored in the bottom part of the palette as opposed to the top
 91          */
 92
 93         for (i = 0; i < tableSize; i++) {
 94             GifHead->HDFPalette[i][0] = *(*MemGif2)++;
 95             GifHead->HDFPalette[i][1] = *(*MemGif2)++;
 96             GifHead->HDFPalette[i][2] = *(*MemGif2)++;
 97         }
 98     }
```

The MemGif2 structure exists at 0x0000555555a838dd, which points to the palette data in the original GIF.

When i == 0x0 (Before consuming Memory)

```
gef➤  hexdump *MemGif2
0x0000555555a838dd     10 00 00 0a 03 00 f4 e4 2c 46 38 39 61 0a 00 03
........,F89a...
0x0000555555a838ed     00 80 76 00 ff fa 2c 00 00 00 00 00 00 03 00 ff
..v...,.........
0x0000555555a838fd     fe ff ef 00 01 e4 2c 46 38 39 61 0a 00 03 00 80
......,F89a.....
0x0000555555a8390d     76 f4 fe fa 2c 00 00 00 00 00 00 03 00 ff ff 00
v...,...........
```

The GifHead structure exists at 0x0000555555a83940, which is only 0x63 bytes away

```
gef➤  hexdump GifHead
0x0000555555a83940     ff 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00
................
0x0000555555a83950     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0x0000555555a83960     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0x0000555555a83970     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
```

The loop will read for a total of 0x300 bytes, incrementing the MemGif2 pointer until it eventually points into the GifHead structure itself and begins copying bytes from the GifHead structure into the HDFPalette data.

We can see this occur after the loop by dumping the beginning of the HDFPalette data. The same data from GifHead structure has been copied into the HDFPalette member starting 0x63 (99) bytes into the HDFPalette data at address 0x0000555555a839b7.

```
gef➤  hexdump GifHead->HDFPalette 128
0x0000555555a83954     10 00 00 0a 03 00 f4 e4 2c 46 38 39 61 0a 00 03
........,F89a...
0x0000555555a83964     00 80 76 00 ff fa 2c 00 00 00 00 00 00 03 00 ff
..v...,.........
0x0000555555a83974     fe ff ef 00 01 e4 2c 46 38 39 61 0a 00 03 00 80
......,F89a.....
0x0000555555a83984     76 f4 fe fa 2c 00 00 00 00 00 00 03 00 ff ff 00
v...,...........
0x0000555555a83994     00 2c 00 00 00 9c 01 00 00 01 00 00 02 ff 0e 64
.,.............d
0x0000555555a839a4     3b 00 00 00 00 00 00 00 00 00 00 31 03 00 00 00
;..........1....
0x0000555555a839b4     00 00 00 ff 00 00 00 00 00 00 00 00 01 00 00 00
................  // <<-- Here
0x0000555555a839c4     00 00 00 00 00 00 00 10 00 00 0a 03 00 f4 e4 2c
...............,

// The GifHead structure itself
gef➤  p/x *GifHead
$37 = {
  PackedField = 0xff,
  TableSize = 0x100,
  ImageCount = 0x0,
  CommentCount = 0x0,
  ApplicationCount = 0x0,
  PlainTextCount = 0x0,
  HDFPalette = {{0x10, 0x0, 0x0}, {0xa, 0x3, 0x0}, {0xf4, 0xe4, 0x2c}, {0x46, 0x38,
0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0x0, 0xff}, {0xfa, 0x2c, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x3}, {0x0, 0xff, 0xfe}, {0xff, 0xef, 0x0}, {0x1, 0xe4,
0x2c}, {0x46, 0x38, 0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0xf4, 0xfe},
{0xfa, 0x2c, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x3}, {0x0, 0xff, 0xff}, {0x0, 0x0,
0x2c}, {0x0, 0x0, 0x0}, {0x9c, 0x1, 0x0}, {0x0, 0x1, 0x0}, {0x0, 0x2, 0xff}, {0xe,
0x64, 0x3b}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x31, 0x3},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0xff, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0,
0x0}, {0x1, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x10}, {0x0,
0x0, 0xa}, {0x3, 0x0, 0xf4}, {0xe4, 0x2c, 0x46}, {0x38, 0x39, 0x61}, {0xa, 0x0,
0x3}, {0x0, 0x80, 0x76}, {0x0, 0xff, 0xfa}, {0x2c, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0,
0x3, 0x0}, {0xff, 0xfe, 0xff}, {0xef, 0x0, 0x1}, {0xe4, 0x2c, 0x46}, {0x38, 0x39,
0x61}, {0xa, 0x0, 0x3}, {0x0, 0x80, 0x76}, {0xf4, 0xfe, 0xfa}, {0x2c, 0x0, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x3, 0x0}, {0xff, 0xff, 0x0}, {0x0, 0x2c, 0x0}, {0x0, 0x0,
0x9c}, {0x1, 0x0, 0x0}, {0x1, 0x0, 0x0}, {0x2, 0xff, 0xe}, {0x64, 0x3b, 0x0}, {0x0,
0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x31, 0x3, 0x0}, {0x0, 0x0, 0x0},
{0x0, 0x0, 0xff}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x1}, {0x0, 0x0,
0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x10, 0x0}, {0x0, 0xa, 0x3}, {0x0,
0xf4, 0xe4}, {0x2c, 0x46, 0x38}, {0x39, 0x61, 0xa}, {0x0, 0x3, 0x0}, {0x80, 0x76,
0x0}, {0xff, 0xfa, 0x2c}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x3, 0x0, 0xff}, {0xfe,
0xff, 0xef}, {0x0, 0x1, 0xe4}, {0x2c, 0x46, 0x38}, {0x39, 0x61, 0xa}, {0x0, 0x3,
0x0}, {0x80, 0x76, 0xf4}, {0xfe, 0xfa, 0x2c}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0},
{0x3, 0x0, 0xff}, {0xff, 0x0, 0x0}, {0x2c, 0x0, 0x0}, {0x0, 0x9c, 0x1}, {0x0, 0x0,
0x1}, {0x0, 0x0, 0x2}, {0xff, 0xe, 0x64}, {0x3b, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0,
0x0, 0x0}, {0x0, 0x0, 0x31}, {0x3, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0xff, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x1, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0},
{0x0, 0x0, 0x0}, {0x10, 0x0, 0x0}, {0xa, 0x3, 0x0}, {0xf4, 0xe4, 0x2c}, {0x46, 0x38,
0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0x0, 0xff}, {0xfa, 0x2c, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x3}, {0x0, 0xff, 0xfe}, {0xff, 0xef, 0x0}, {0x1, 0xe4,
0x2c}, {0x46, 0x38, 0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0xf4, 0xfe},
{0xfa, 0x2c, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x3}, {0x0, 0xff, 0xff}, {0x0, 0x0,
```

```
0x2c}, {0x0, 0x0, 0x0}, {0x9c, 0x1, 0x0}, {0x0, 0x1, 0x0}, {0x0, 0x2, 0xff}, {0xe,
0x64, 0x3b}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x31, 0x3},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0xff, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0,
0x0}, {0x1, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x10}, {0x0,
0x0, 0xa}, {0x3, 0x0, 0xf4}, {0xe4, 0x2c, 0x46}, {0x38, 0x39, 0x61}, {0xa, 0x0,
0x3}, {0x0, 0x80, 0x76}, {0x0, 0xff, 0xfa}, {0x2c, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0,
0x3, 0x0}, {0xff, 0xfe, 0xff}, {0xef, 0x0, 0x1}, {0xe4, 0x2c, 0x46}, {0x38, 0x39,
0x61}, {0xa, 0x0, 0x3}, {0x0, 0x80, 0x76}, {0xf4, 0xfe, 0xfa}, {0x2c, 0x0, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x3, 0x0}, {0xff, 0xff, 0x0}, {0x0, 0x2c, 0x0}, {0x0, 0x0,
0x9c}, {0x1, 0x0, 0x0}, {0x1, 0x0, 0x0}, {0x2, 0xff, 0xe}, {0x64, 0x3b, 0x0}, {0x0,
0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x31, 0x3, 0x0}, {0x0, 0x0, 0x0},
{0x0, 0x0, 0xff}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x1}, {0x0, 0x0,
0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x10, 0x0}, {0x0, 0xa, 0x3}, {0x0,
0xf4, 0xe4}, {0x2c, 0x46, 0x38}, {0x39, 0x61, 0xa}, {0x0, 0x3, 0x0}, {0x80, 0x76,
0x0}, {0xff, 0xfa, 0x2c}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x3, 0x0, 0xff}, {0xfe,
0xff, 0xef}, {0x0, 0x1, 0xe4}, {0x2c, 0x46, 0x38}, {0x39, 0x61, 0xa}, {0x0, 0x3,
0x0}, {0x80, 0x76, 0xf4}, {0xfe, 0xfa, 0x2c}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0},
{0x3, 0x0, 0xff}, {0xff, 0x0, 0x0}, {0x2c, 0x0, 0x0}, {0x0, 0x9c, 0x1}, {0x0, 0x0,
0x1}, {0x0, 0x0, 0x2}, {0xff, 0xe, 0x64}, {0x3b, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0,
0x0, 0x0}, {0x0, 0x0, 0x31}, {0x3, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0xff, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x1, 0x0}, {0x0, 0x0, 0x0}, {0x0, 0x0, 0x0},
{0x0, 0x0, 0x0}, {0x10, 0x0, 0x0}, {0xa, 0x3, 0x0}, {0xf4, 0xe4, 0x2c}, {0x46, 0x38,
0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0x0, 0xff}, {0xfa, 0x2c, 0x0},
{0x0, 0x0, 0x0}, {0x0, 0x0, 0x3}, {0x0, 0xff, 0xfe}, {0xff, 0xef, 0x0}, {0x1, 0xe4,
0x2c}, {0x46, 0x38, 0x39}, {0x61, 0xa, 0x0}, {0x3, 0x0, 0x80}, {0x76, 0xf4, 0xfe},
{0xfa, 0x2c, 0x0}},
  HeaderDump = {0x47, 0x49, 0x46, 0x38, 0x39, 0xb4},
  LSDDump = {0xa, 0x80, 0x0, 0x2, 0xff, 0x76, 0x0}
}
```

When we return back to the Gif2Mem() function within gif2mem.c, parsing of the image data begins using the MemGif pointer used previously:

```
 97     ReadGifHeader(gifHead, &MemGif);
           /* <snip> */
117     for (;;) {
118         Identifier = *MemGif++;
119
120         switch (Identifier) {
121             case 0x3B: /* Trailer */
122                 /*
123                  * The counts are stored to make it easier while putting stuff
124                  * into the HDF file and then deallocating space.
```

However, MemGif instead points into the GifHead struct itself. We can see this at address 0x0000555555a83c54, which is the gifHead->HeaderDump member and so on, meaning that the gifHead data will be interpreted and parsed as image data.

```
gef➤  hexdump gifHead->HeaderDump
0x0000555555a83c54     47 49 46 38 39 b4 0a 80 00 02 ff 76 00 00 00 00
GIF89......v....
0x0000555555a83c64     00 00 00 00 a1 03 02 00 00 00 00 00 00 00 00 00
................
0x0000555555a83c74     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0x0000555555a83c84     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................


gef➤  hexdump MemGif 128
0x0000555555a83bdd     00 00 00 00 00 03 00 ff ff 00 00 2c 00 00 00 9c
...........,....
0x0000555555a83bed     01 00 00 01 00 00 02 ff 0e 64 3b 00 00 00 00 00
.........d;.....
0x0000555555a83bfd     00 00 00 00 00 31 03 00 00 00 00 00 00 ff 00 00
.....1..........
0x0000555555a83c0d     00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00
................
0x0000555555a83c1d     00 10 00 00 0a 03 00 f4 e4 2c 46 38 39 61 0a 00
.........,F89a..
0x0000555555a83c2d     03 00 80 76 00 ff fa 2c 00 00 00 00 00 00 03 00
...v...,........
0x0000555555a83c3d     ff fe ff ef 00 01 e4 2c 46 38 39 61 0a 00 03 00
.......,F89a....
0x0000555555a83c4d     80 76 f4 fe fa 2c 00 47 49 46 38 39 b4 0a 80 00
.v...,.GIF89.... // <<-- Here
```

This can even corrupt the heap, when parsing the remaining data as image data (for example, when attempting to reallocate image memory in gif2mem.c because “multiple” images were found, when in fact they are merely copies of the same image data):

```
152                 if (ImageCount > ImageArray) {
153                     aTemp      = ImageArray;
154                     ImageArray = (GIFBYTE)((ImageArray << 1) + 1);
155                     if (!(gifImageDesc =
156                               (GIFIMAGEDESC **)realloc(gifImageDesc,
sizeof(GIFIMAGEDESC *) * ImageArray))) {
157                         printf("Out of memory!");
158                         exit(EXIT_FAILURE);
159                     }
```

The reallocation can shift the next image pointer into the middle of this palette data. From here, arbitrary memory can be freed when returning to the main function:

```
113     /* Free all buffers */
114     /* replacing int32 with long */
115     ImageCount = (long)((GifMemoryStruct.GifHeader)->ImageCount);
116
117     for (i = 0; i < ImageCount; i++) {
118         gifImageDesc = *(GifMemoryStruct.GifImageDesc[i]);
119
120         if (gifImageDesc.Image != NULL)
121             free(gifImageDesc.Image);

gef➤  p gifImageDesc.Image
$3 = (GIFBYTE *) 0x4141414141414141 <error: Cannot access memory at address
0x4141414141414141>

Program received signal SIGSEGV, Segmentation fault.
__GI___libc_free (mem=0x4141414141414141) at malloc.c:3102
3102      p = mem2chunk (mem);
```

TIMELINE

2022-03-23 - Vendor Disclosure

2022-08-16 - Public Release

CREDIT

Discovered by Dave McDaniel of Cisco Talos.
