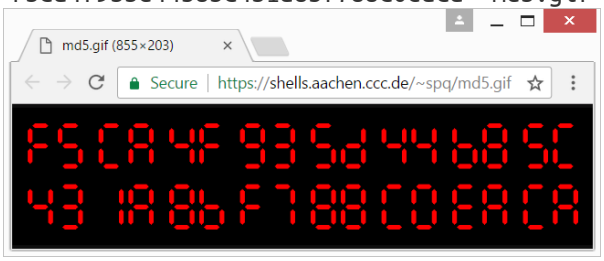
14:11 This GIF shows its own MD5!

by Kristoffer "spq" Janke

The recent successful attack on the SHA-1 hash algorithm[38] has led to a resurgence of interest in hash collisions and their consequences.

A particularly well-broken hash algorithm is MD5, which allows for a myriad of ways to play with it. Here, we demonstrate how to assemble an animated GIF image that displays its own MD5 hash.[39]

```
$ md5sum md5.gif
f5ca4f935d44b85c431a8bf788c0eaca  md5.gif
```



The GIF89a file format

A GIF89a file consists of concatenated blocks. A parser can read these blocks from the file in a serial fashion without needing to keep state.

A GIF file is made up of three parts.

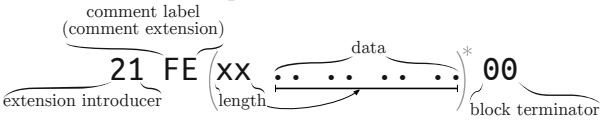
Header Signature, Version and basic info like the Canvas Size and (optional) Color Map.

Body Image, Comment, Text and Extension blocks, in any order.

Trailer The byte `0x3b`.

Of particular interest to us is the format of comment blocks. They begin with the two bytes `0x21 0xfe`, followed by any number of comment chunks. Every chunk consists of one length byte and <length> bytes of arbitrary data. The end of the comment block is marked with a chunk having zero length.

This means that, by controlling the length bytes, we can make the parser skip any number of non-displayable bytes in comment chunks. These skipped bytes, of course, still affect the file's MD5 hash. So two GIF files can show different content, while their skipped bytes are manipulated to make them have the same MD5 hash values. With some careful stitching, here we'll build just such files—MD5 GIF collision pairs.



MD5 collisions

For MD5, appending the same data to both colliding files will still produce the same hash value. The same is true for appending another collision pair. So we can have four different files all having the same MD5 hash with this method.

Or, instead of producing multiple files, we can produce just one file but later change one of the collisions in the produced file. This is the technique we'll use here.

Fastcoll is a MD5 collision generator, created by Marc Stevens.[40] From any input file, it generates two different output files, both having the same MD5 hash.

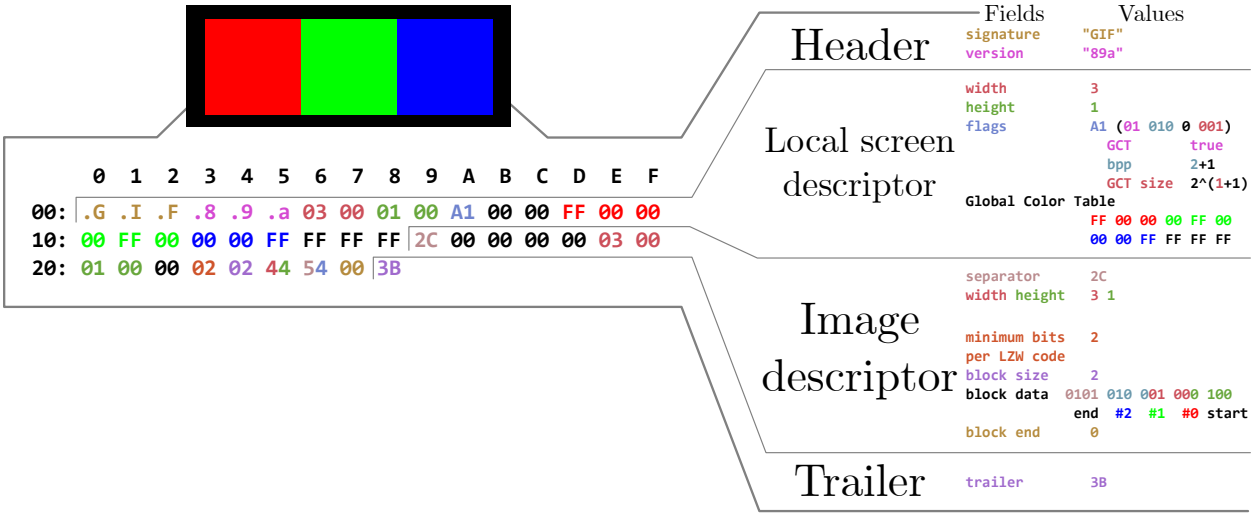
These output files consist of the 64-byte aligned, zero-padded input file, followed by 128 bytes of collision data generated by Fastcoll. Every byte from the generated collision data of both files appears to be random. Comparing these last 128 bytes in both output files, we can see that only nine bytes differ. These bytes can be found at indices 19, 45, 46, 59, 83, 109, 110 and 123. While the bytes at 46 and 110 do not show any pattern, the other bytes differ only and exactly in their most significant bit. This can be used to construct GIF comment chunks of different sizes.

Showing two different images

The GIF comment block format and the collisions generated by Fastcoll allow for the creation of two GIF files that have the same MD5 hash, but are interpreted differently.

By constructing the GIF such that one of the differing bytes in the collision data is interpreted as the length of a comment chunk, the interpretation

[38] `unzip pocorgtfo14.pdf shattered.pdf`
[39] `unzip pocorgtfo14.pdf md5.gif`
[40] `unzip pocorgtfo14.pdf fastcoll-v1.0.0.5-1.zip`



of the remaining file will be different across the two colliding files.

Here, we chose the last differing byte at position 123. Due to the most significant bit having been flipped between the two collisions, the byte's value differs by 128. In order to align this byte to the Length byte of comment chunk #2, the previous comment chunk #1 needs to contain the first 123 bytes of the collision data. As the collision is 64-byte aligned, the comment chunk #1 should contain some padding bytes. We'll refer to these two colliding blocks as (X) and (Y).

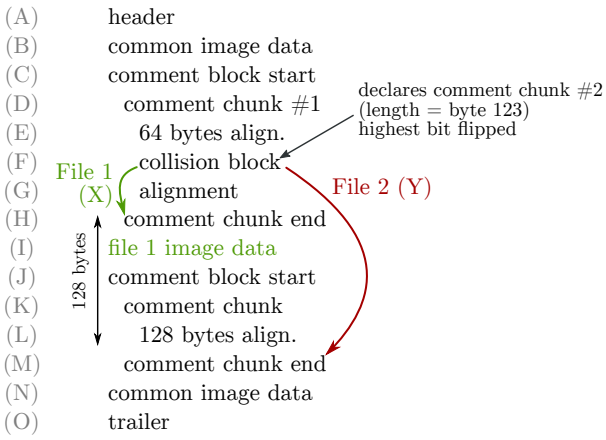
One limitation arises when the value of the byte controlling the length of #2 is smaller than 4. The reason for this limitation is that the comment chunk #2 needs to contain at least the remaining collision data (four bytes) in both files. When this requirement is not met, a new collision needs to be generated.

We now have two files with different-sized comment chunks, but the same MD5 hash. We can use this in one of the collisions by ending the comment block and starting an image block. The image block is followed by another comment block, which is sized such that it skips the remaining bytes of the difference to 128 and both collisions are aligned from there.

The diagram to the right shows the contents of the GIF file, which is interpreted differently depending upon which of the colliding blocks is found at Point **F**.

The file with the collision block **X** will have the body blocks **B**, **I** and **N** interpreted, while the file with **Y** will only have **B** and **N** interpreted, with **I** skipped over as part of a comment. In order to yield two GIFs with completely different images, one could use the blocks **B** and **N** for the two images and one or more dummy image with very high animation delay in block **I**. The result is a pair of animated GIF files, both having the desired images as first and last frames, but only the variant with **X** would have a delay of multiple minutes between the two frames.


```
$ md5sum md5_avp_loop.gif
8895af74c2b5478c547cfb85f7475f0b  md5_avp_loop.gif
```



Showing the MD5 hash

For my PoC, I decided to use 7-segment optics. For displaying the MD5 hash, I need 32 digits, each having seven segments. The background image with all 224 (32×7) segments visible is put into block (B), block (N) can be left empty. We repeat the blocks (D)…(L) for every single segment and put an image masking that segment into block (I). Generating all 224 collisions required thirty minutes on my PC. When the file is completely generated, we calculate its MD5 hash. This will be the final hash, which the GIF file itself should show.

Every masking image will only be shown when the corresponding collision block is (X), otherwise a parser will only see comment chunks. We can switch between collision blocks (X) and (Y) for every image masking one of the segments. This switch will not change the MD5 hash value of the file but it allows us to control what is displayed. Once we have the final hash value, we choose the right collision for each segment and replace it in the file.[41]

That's it![42] :)

[41] `unzip pocorgtfo14.pdf md5_avp_loop.gif`

[42] Between this article's writing and publication, a friendly neighbor Rogdham created his own PoC with detailed write-up and script, which are available at `http://www.rogdham.net/2017/03/12/gif-md5-hashquine.en` and in this issue's ZIP contents.