# How to datamosh videos with data corruption

datamoshing.com/tag/mov

Watch Video At: https://youtu.be/Gtnke4Tjofg

Glitching videos with data corruption can be a tricky process. This is owing to the fact that video formats are substantially more complex than image formats. Since video formats contain audio and timing information in addition to visual information corrupting the wrong section of a video file can quickly render the file unplayable rather than delightfully distorted.

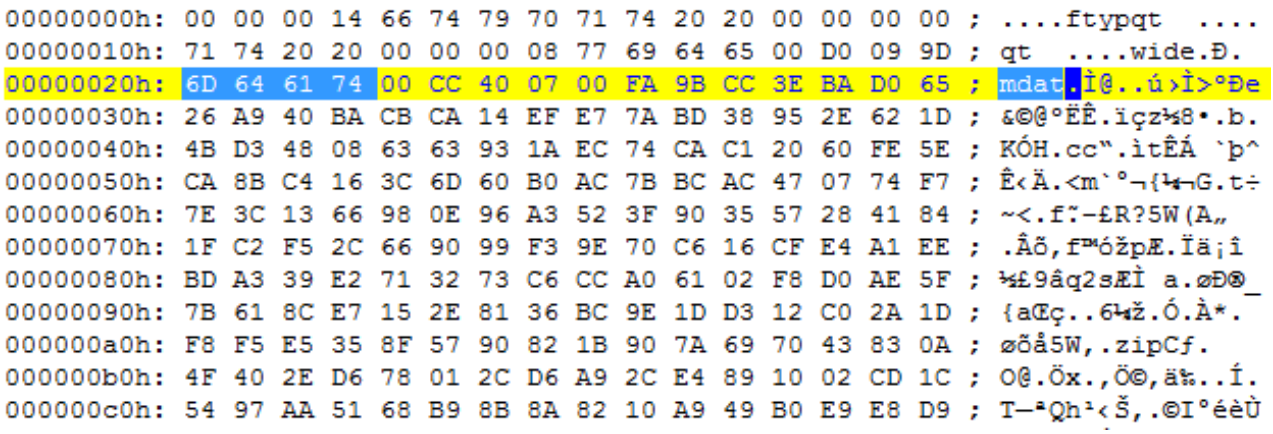
This tutorial will focus on glitching the popular MP4 and MOV formats containing video compressed with the H.264 standard. If the video you want to destroy is not in MP4 or MOV format already an easy way to convert it is to upload it to YouTube, let them convert it, and then download the result. If a video doesn't set off any copyright claims you can download it from the YouTube Video Manager in MP4 format.

Get started by making a copy of your MOV or MP4 and open the copy in a hex editor — never edit the original file. If you don't have a hex editor installed there are some freeware options listed at the bottom of this post. Hex editors allow us to view and edit the bytes of a file using hexadecimal. Editing the file using hex rather than text allows greater flexibility since we're no longer restricted to text characters (which are each represented by two hex digits).

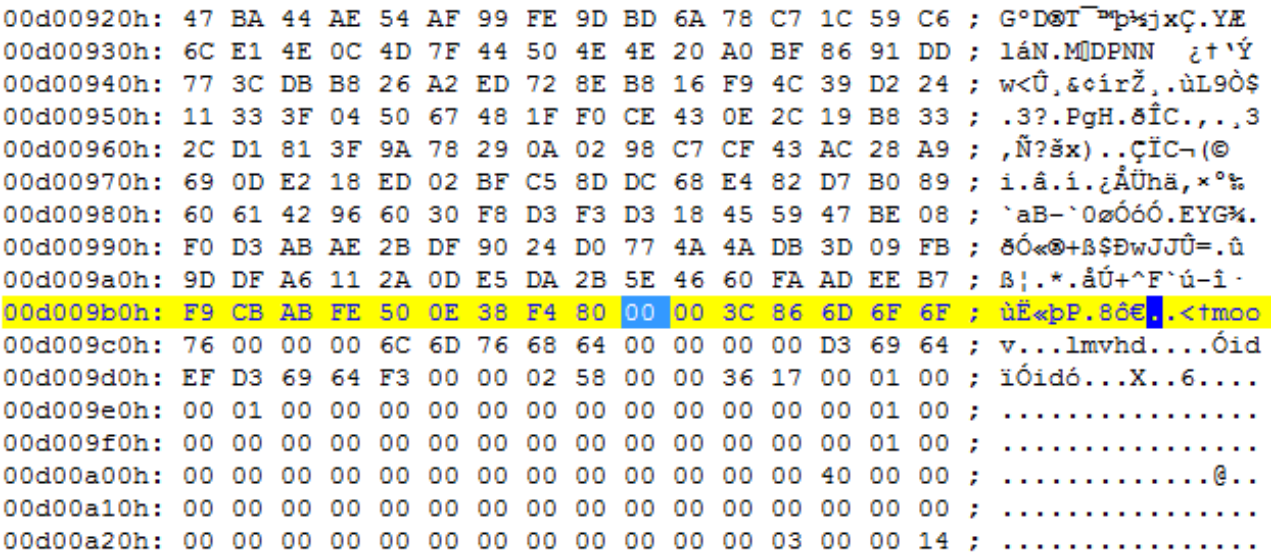
We know that we're looking at an MP4 or MOV file when we see the text **ftypqt** starting on the fifth byte of the file as illustrated in the example below.

```
00000000h: 00 00 00 14 66 74 79 70 71 74 20 20 00 00 00 00 ; ....ftypqt  ....
00000010h: 71 74 20 20 00 00 00 08 77 69 64 65 00 D0 09 9D ; qt  ....wide.Ð.
00000020h: 6D 64 61 74 00 CC 40 07 00 FA 9B CC 3E BA D0 65 ; mdat.Ì@..ú›Ì>°Ðe
00000030h: 26 A9 40 BA CB CA 14 EF E7 7A BD 38 95 2E 62 1D ; &©@°ËÊ.ïçz½8•.b.
00000040h: 4B D3 48 08 63 63 93 1A EC 74 CA C1 20 60 FE 5E ; KÓH.cc".ìtÊÁ `þ^
00000050h: CA 8B C4 16 3C 6D 60 B0 AC 7B BC AC 47 07 74 F7 ; Ê‹Ä.<m`°¬{¼¬G.t÷
00000060h: 7E 3C 13 66 98 0E 96 A3 52 3F 90 35 57 28 41 84 ; ~<.f˜.–£R?5W(A„
```

The MP4 and MOV (Quicktime) formats utilize a similar structure, the file is broken down into atoms or blocks of data. The atom which contains the raw frame and audio data can be identified by it's atom type string, which in this case is **mdat** (short for media data).




The data contained within the mdat atom is comprised of chunks, which are comprised of nal units, which are comprised of slices. For the purposes of this tutorial we won't delve that deep. Finding the mdat atom can be done by searching the file for the string "mdat" as seen in the above image. Notice that the contents of the mdat atom seem quite random when viewed in a hex editor, in contrast the contents of the other atoms in MP4 and MOV files are very structured. With this information we can easily find the end of the mdat atom by scrolling through it (or up from the bottom of the file) to see where the data starts to look random as illustrated in the image below.




Here we can see that the mdat atom is followed by the moov atom (and an mvhd atom after that), this is not always the case as the order of the atoms can be different. What's important to note though is that the file is visibly more structured after the mdat atom, this is how we can identify where the mdat atom ends. The mdat atom, in all cases, will either continue to the end of the file or it will be followed by another atom identified by a 4-character string such as the moov atom does in the above example. Using this method we can identify both the start and the end of the mdat atom, and in turn where we can corrupt only the mdat atom's contents and have a fair chance of the video still being playable.

Once we've identified the boundaries of mdat atom we can begin to copy and paste, replace or edit portions of the raw hex data (or the text ASCII data, either will work) of the video and check the result along the way by attempting to play the video. Making backup copies after every successful change will avoid heartaches when, not if, a misstep renders the video unplayable.

Some notes on successful corrupting:

- It doesn't take much corruption to add bizarre distortion to a video, even corrupting as little as 10% of a file, a couple of bytes here and there, has the potential to send playback into a wild frenzy.
- While not required, most data in the mdat will be in sequence so if we want to corrupt a specific portion of the video we can estimate the offset of the data for that portion is in the mdat based on its time.
- The mdat atom will also contain raw audio data so if the audio becomes distorted during playback we know we've gone too far, or started too early, in the mdat atom.
- Copying and pasting hex within nal units is probably the best way to corrupt H.264 video data as you'll be shuffling valid data around rather than adding outright gibberish.
- As mentioned previously the mdat is comprised of chunks, as chunks and the nal units contained within them have structure it's best to corrupt small portions of data in various spots rather than large swaths. Corrupting large regions of data will inevitably cross over structure boundaries and destroy important information regarding the type of nal unit or slice.
- To stay inside nal units look for hex **67**, **68** or **00 00 01** as these sequences can denote the beginning of a new nal unit.
- This definitely falls under the bull in a china shop category of datamoshing video so don't be discouraged if it takes a couple of restarts to get a playable result.

The video included in this post was glitched using this technique, however the audio was slowed down using traditional video editing.

As with any glitch-by-corruption technique, it requires a soft touch — too little has no effect, too much can destroy the file, but just enough results in glorious, glorious corruption. #corruptabsolutely