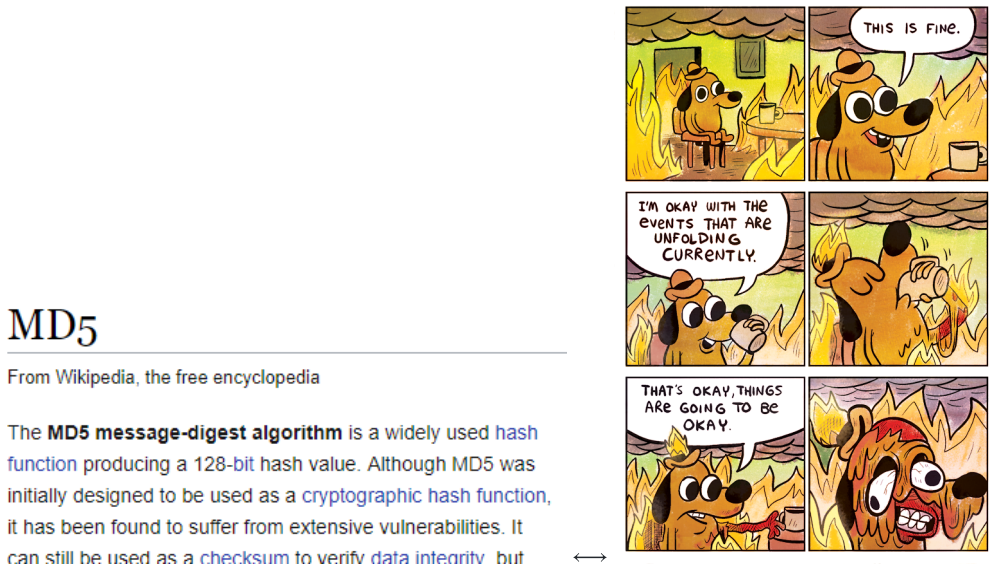
TL;DR getting an MD5 collision of these 2 images is now(*) trivial and instant.



Don't play with fire, don't rely on MD5.

(*) Colliding any pair of files has been possible for many years, but it takes several hours each time, with no shortcut. This page provide tricks specific to file formats and pre-computed collision prefixes to make collision **instant**. `git clone`. Run Script. Done.

Introduction

This part of the repository is focused on hash collisions exploitation for MD5 and SHA1.

This is a collaboration with Marc Stevens.

The goal of this page is to explore extensively existing attacks - and show on the way how weak MD5 is (instant collisions of any JPG, PNG, PDF, MP4, PE...) - and also explore in detail common file formats to determine how they can be exploited with present or with future attacks.

Indeed, the same file format trick can be used on several hashes (the same JPG tricks were used for MD5, malicious SHA-1 and SHA1), as long as the collisions follow the same byte patterns.

This document is **not** about new attacks (the most recent one was documented in 2012), but about new forms of exploitations of existing attacks.

Status

Current status - as of December 2018 - of known attacks:

- get a file to get another file's hash or a given hash: **impossible**
  - it's still even not practical with MD2.
  - works for simpler hashes(*)
- get 2 different files with the same MD5: **instant**
  - examples: 1 ⟷ 2
- make 2 arbitrary files get the same MD5: **a few hours** (72 hours.core)

    - examples: 1 ⟷ 2
- make 2 arbitrary files of specific file formats (PNG, JPG, PE...) get the same MD5: **instant**
    - read below
- get two different files with the same SHA1: 6500 years.core
    - get two different PDFs with the same SHA-1 to show a different picture: instant (the prefixes are already computed)

(*) example with crypt - thanks Sven!

```
>>> import crypt
>>> crypt.crypt("5dUD&66", "br")
'brokenOz4KxMc'
>>> crypt.crypt("O!>',%$", "br")
'brokenOz4KxMc'
```

Attacks

MD5 and SHA1 work with blocks of 64 bytes.

If 2 contents A & B have the same hash, then appending the same contents C to both will keep the same hash.

```
hash(A) = hash(B) -> hash(A + C) = hash(B + C)
```

Collisions work by inserting at a block boundary a number of computed collision blocks that depends on what came before in the file. These collision blocks are very random-looking with some minor differences (that follow a specific pattern for each attack) and they will introduce tiny differences while eventually getting hashes the same value after these blocks.

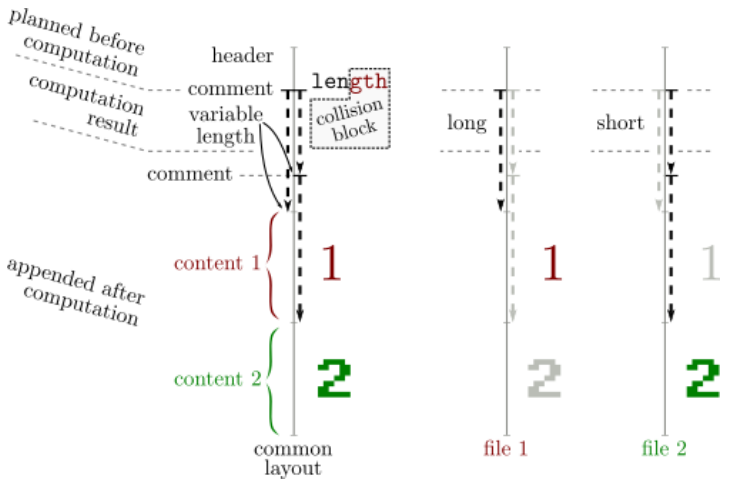
These differences are abused to craft valid files with specific properties.

File formats also work top-down, and most of them work by byte-level chunks.

Some 'comment' chunks can be inserted to align file chunks to block boundaries, to align specific structures to collision blocks differences, to hide the rest of the collision blocks randomness from the file parsers, and to hide otherwise valid content from the parser (so that it will see another content).

These 'comment' chunks are often not officially real comments: they are just used as data containers that are ignored by the parser (for example, PNG chunks with a lowercase-starting ID are ancillary, not critical).

Most of the time, a difference in the collision blocks is used to modify the length of a comment chunk, which is typically declared just before the data of this chunk: in the gap between the smaller and the longer version of this chunk, another comment chunk is declared to jump over one file's content `A`. After this file content `A`, just append another file content `B`.



Since file formats usually define a terminator that will make parsers stop after it, `A` will terminate parsing, which will make the appended content `B` ignored.

So typically at least 2 comments are needed:

1. alignment
2. hide collision blocks
3. hide one file content (for re-usable collisions)

These common properties of file formats make it possible - they are not typically seen as weaknesses, but they can be detected or normalized out:

- dummy chunks - used as comments
- more than 1 comment
- huge comments (lengths: 64b for MP4, 32b for PNG -> trivial collisions. 16b for JPG, 8b for GIF -> no generic collision for GIF, limited for JPG)
- store any data in a comment (UTF8 could be enforced)
- store anything after the terminator (usually used only for malicious purposes)
- no integrity check. CRC32 in PNG are usually ignored, which would prevent PNG re-usable collisions otherwise.
- flat structure: ASN.1 defines parent structure with the length of all the enclosed substructures, which prevents these constructs: you'd need to abuse a length, but also the length of the parent.
- put a comment before the header - this makes generic re-usable collisions possible.

Identical prefix

1. Define an arbitrary prefix - its content and length don't matter.
2. The prefix is padded to the next 64-byte block.
3. Collision block(s) are computed depending on the prefix and appended. Both sides are very random. The differences are predetermined by the attack.
4. After this[these] block[s], the hash value is the same despite the file differences.
5. Any arbitrary identical suffix can be added.

Prefix	**=**	**Prefix**
Collision *A*	≠	Collision *B*
Suffix	=	Suffix

Both files are almost identical (their content have only a few bits of differences)

Exploitation:

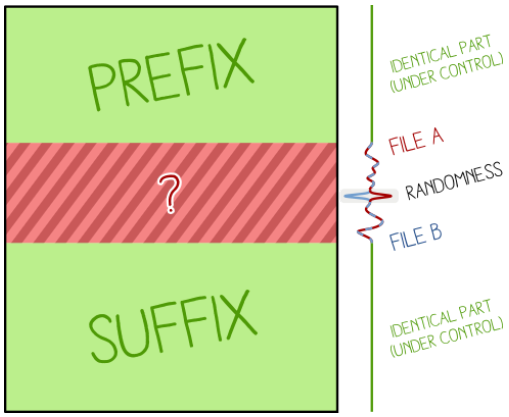
Bundle 2 contents, then either:

- Data exploit: run code that checks for differences and displays one or the other (typically trivial since differences are known in advance).
- Structure exploit: exploit file structure (typically, the length of a comment) to hide one content or show the other (depends on the file format and its parsers).

Two files with this structure:

Prefix	**=**	**Prefix**
Collision *A*	≠	Collision *B*
A	=	~~**A**~~
~~**B**~~	=	**B**

will show either A or B.



FastColl (MD5)

Final version in 2009.

- time: a few seconds of computation
- space: 2 blocks
- differences: no control before, no control after. FastColl difference mask:

```
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. X. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. X. .X .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

- exploitation: hard

The differences aren't near the start/end of the blocks, so it's very hard to exploit since you don't control any nearby byte. A potential solution is to brute-force the surrounding bytes - cf PoCGTFO 14:10.

Examples:

With an empty prefix:

```
00:  37 75 C1 F1-C4 A7 5A E7-9C E0 DE 7A-5B 10 80 26  7u┴±─ºZτ£α▌z[►Ç&
10:  02 AB D9 39-C9 6C 5F 02-12 C2 7F DA-CD 0D A3 B0  ☻½┘9╔l_☻↕┬⌂┌═♪ú░
20:  8C ED FA F3-E1 A3 FD B4-EF 09 E7 FB-B1 C3 99 1D  îφ·≤ßú²┤∩○τ√▒├Ö↔
30:  CD 91 C8 45-E6 6E FD 3D-C7 BB 61 52-3E F4 E0 38  ═æ╚Eμn²=╠╗aR>⌠α8

40:  49 11 85 69-EB CC 17 9C-93 4F 40 EB-33 02 AD 20  I◄à↕δ╠↨£ôÒ@δ3☻¡
50:  A4 09 2D FB-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E  ñ○─√§· ↔╤▀↨═▌)Y▲
60:  39 89 9E F6-79 46 9F E6-8B 85 C5 EF-DE 42 4F 46  9ë₧÷yFƒμïà┼∩▐BOF
70:  C2 78 75 9D-8B 65 F4 50-EA 21 C5 59-18 62 FF 7B  ┬xu¥ïe⌠PΩ!┼Y↑b {
```

- MD5: `fe6c446ee3a831ee010f33ac9c1b602c`
- SHA256: `c5dd2ef7c74cd2e80a0fd16f1dd6955c626b59def888be734219d48da6b9dbdd`

```
00:  37 75 C1 F1-C4 A7 5A E7-9C E0 DE 7A-5B 10 80 26  7u┴±─ºZτ£α▌z[►Ç&
10:  02 AB D9 B9-C9 6C 5F 02-12 C2 7F DA-CD 0D A3 B0  ☻½┘╣╔l_☻↕┬⌂┌═♪ú░
20:  8C ED FA F3-E1 A3 FD B4-EF 09 E7 FB-B1 43 9A 1D  îφ·≤ßú²┤∩○τ√▒CÜ↔
30:  CD 91 C8 45-E6 6E FD 3D-C7 BB 61 D2-3E F4 E0 38  ═æ╚Eμn²=╠╗a╥>⌠α8

40:  49 11 85 69-EB CC 17 9C-93 4F 40 EB-33 02 AD 20  I◄à↕δ╠↨£ôÒ@δ3☻¡
50:  A4 09 2D 7B-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E  ñ○─{§· ↔╤▀↨═▌)Y▲
60:  39 89 9E F6-79 46 9F E6-8B 85 C5 EF-DE C2 4E 46  9ë₧÷yFƒμïà┼∩▐╥NF
70:  C2 78 75 9D-8B 65 F4 50-EA 21 C5 D9-18 62 FF 7B  ┬xu¥ïe⌠PΩ!┼┘↑b {
```

- MD5: `fe6c446ee3a831ee010f33ac9c1b602c`
- SHA256: `e27cf3073c704d0665da42d597d4d20131013204eecb6372a5bd60aeddd5d670`

Other examples, with an identical prefix: 1 ⟷ 2

Variant: there is a single-block MD5 collision but it takes five weeks of computation.

UniColl (MD5)

Documented in 2012, implemented in 2017

UniColl lets you control a few bytes in the collision blocks, before and after the first difference, which makes it an identical-prefix collision with some controllable differences, almost like a chosen prefix collision. This is very handy, and even better the difference can be very predictable: in the case of `m2+= 2^8` (a.k.a. `N=1` / `m2 9` in HashClash poc_no.sh script), the difference is +1 on the 9th byte, which makes it very exploitable, as you can even think about the collision in your head: the 9th character of that sentence will be replaced with the next one: `0` replaced by `1`, `a` replaced by `b`..

- time: a few minutes (depends on the amount of byte you want to control)
- space: 2 blocks
- differences:

```
.. .. .. .. DD .. .. .. ..
.. .. .. .. +1 .. .. .. ..
```

- exploitation: very easy - controlled bytes before and after the difference, and the difference is predictable. The only restrictions are alignment and that you 'only' control 10 bytes after the difference.

Examples with `N=1` and 20 bytes of set text in the collision blocks:

```
00:  55 6E 69 43-6F 6C 6C 20-31 20 70 72-65 66 69 78  UniColl 1 prefix
10:  20 32 30 62-F5 48 34 B9-3B 1C 01 9F-C8 6B E6 44   20bJH4╣;└☺ƒ╚kµD
20:  FE F6 31 3A-63 DB 99 3E-77 4D C7 5A-6E B0 A6 88  ■÷1:c█Ö>wM╟Zn░ªê
30:  04 05 FB 39-33 21 64 BF-0D A4 FE E2-A6 9D 83 36  ♦♣√93!d┐♪ñ■Γª¥â6
40:  4B 14 D7 F2-47 53 84 BA-12 2D 4F BB-83 78 6C 70  K¶╫≥GSä║↕-O╗âxlp
50:  C6 EB 21 F2-F6 59 9A 85-14 73 04 DD-57 5F 40 3C  ╞δ!≥÷YÜà¶s♦▌W_@<
60:  E1 3F B0 DB-E8 B4 AA B0-D5 56 22 AF-B9 04 26 FC  ß?░█Φ┤¬░ FV"»╣♦&ⁿ
70:  9F D2 0C 00-86 C8 ED DE-85 7F 03 7B-05 28 D7 0F  ƒ╥♀ å╚φ▐à⌂♥{♣(╫☼
```

```
00:  55 6E 69 43-6F 6C 6C 20-31 21 70 72-65 66 69 78  UniColl 1!prefix
10:  20 32 30 62-F5 48 34 B9-3B 1C 01 9F-C8 6B E6 44   20bJH4╣;└☺ƒ╚kµD
20:  FE F6 31 3A-63 DB 99 3E-77 4D C7 5A-6E B0 A6 88  ■÷1:c█Ö>wM╟Zn░ªê
30:  04 05 FB 39-33 21 64 BF-0D A4 FE E2-A6 9D 83 36  ♦♣√93!d┐♪ñ■Γª¥â6
40:  4B 14 D7 F2-47 53 84 BA-12 2C 4F BB-83 78 6C 70  K¶╫≥GSä║↕,O╗âxlp
50:  C6 EB 21 F2-F6 59 9A 85-14 73 04 DD-57 5F 40 3C  ╞δ!≥÷YÜà¶s♦▌W_@<
60:  E1 3F B0 DB-E8 B4 AA B0-D5 56 22 AF-B9 04 26 FC  ß?░█Φ┤¬░ FV"»╣♦&ⁿ
70:  9F D2 0C 00-86 C8 ED DE-85 7F 03 7B-05 28 D7 0F  ƒ╥♀ å╚φ▐à⌂♥{♣(╫☼
```

UniColl has less control than chosen prefix, but it's much faster especially since it takes only 2 blocks.

It was used in the Google CTF 2018, where the frequency of a certificate serial changes and limitations on the lengths prevented the use of chosen prefix collisions.

Shattered (SHA1)

Documented in 2013, computed in 2017.

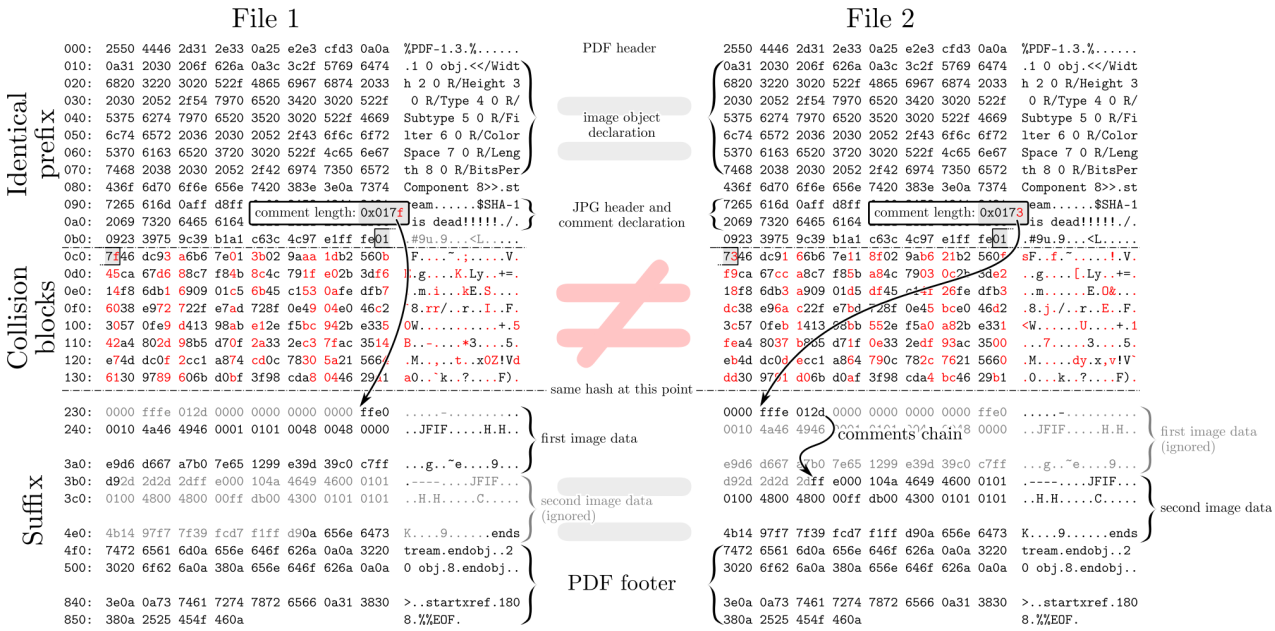
- time: 6500 years.CPU and 110 year.GPU
- space: 2 blocks
- differences:

```
.. .. .. DD ?? ?? ?? ??
or
?? ?? ?? DD .. .. .. ..
```

- exploitation: medium. The differences are right at the start of the collision blocks. So no control before and after the length: PNG stores its length before the chunk type, so it won't work. However it will work with JP2 files when they use the JFIF form (the same as JPG), and likely MP4 and other atom/box formats if you use long lengths on 64bits (in this case, they're placed *after* the atom type).

The difference between collision blocks of each side is this Xor mask:

```
0c 00 00 02 c0 00 00 10 b4 00 00 1c 3c 00 00 04
bc 00 00 1a 20 00 00 10 24 00 00 1c ec 00 00 14
0c 00 00 02 c0 00 00 10 b4 00 00 1c 2c 00 00 04
bc 00 00 18 b0 00 00 10 00 00 00 0c b8 00 00 10
```



Examples: PoC||GTFO 0x18 is using the computed SHA1 prefixes, re-using the image directly from PDFLaTeX source (see article 18:10), but also checking the value of the prefixes via JavaScript in the HTML page (the file is polyglot, ZIP HTML and PDF).

Chosen-prefix collisions

They allow to collide any content. They don't exist for SHA-1 yet.

$\mathscr{A}$	≠	$\mathscr{B}$
Collision *A*	≠	Collision *B*

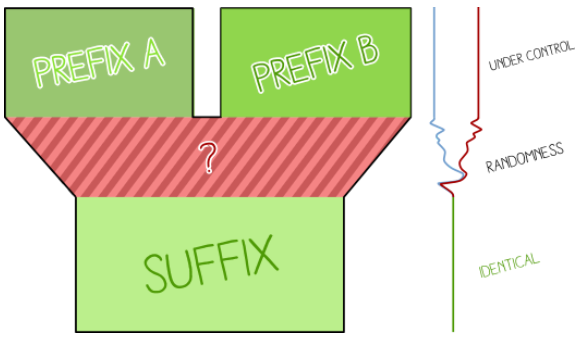
1. take 2 arbitrary prefixes
2. pad the shortest to be as long as the longest. both are padded to the next block - minus 12 bytes

- these 12 bytes of random data will be added on both sides to randomize the birthday search

3. X near-collision blocks will be computed and appended.

   The fewer blocks, the longer the computation.

   Ex: 400 kHours for 1 block. 72 hours.cores for 9 blocks with HashClash.



Chosen prefix collisions are almighty, but they can take a long time just for a pair of files.

HashClash (MD5)

Final version in 2009.

Examples: let's collide `yes` and `no`. It took 3 hours on 24 cores.

`yes` :

```
000:  79 65 73 0A-3D 62 84 11-01 75 D3 4D-EB 80 93 DE  yes◙=bä◄☺u╨MδÇô▌
010:  31 C1 D9 30-45 FB BE 1E-71 F0 0A 63-75 A8 30 AA  1┴┘0E√┘▲q≡◙cu¿0¬
020:  98 17 CA E3-A2 6B 8E 3D-44 A9 8F F2-0E 67 96 48  ÿ↨╨πókÄ=D⌐Å≥♫gûH
030:  97 25 A6 FB-00 00 00 00-49 08 09 33-F0 62 C4 E8  ù%ª√    I◘○3≡b─Φ

040:  D5 F1 54 CD-CA A1 42 90-7F 9D 3D 9A-67 C4 1B 0F  ╒±T═╩íBÉ⌂¥=Üg─←☼
050:  04 9F 19 E8-92 C3 AA 19-43 31 1A DB-DA 96 01 54  ♦ƒ↓ΦÆ├¬↓C1→█┌û©T
060:  85 B5 9A 88-D8 A5 0E FB-CD 66 9A DA-4F 20 8A AA  à╡Üê╪Ñ♫√=fÜ┌O è¬
070:  BA E3 9C F0-78 31 8F D1-14 5F 3E B9-0F 9F 3E 19  ║π£≡x1Å╤¶_>╣☼ƒ>↓

080:  09 9C BB A9-45 89 BA A8-03 E6 C0 31-A0 54 D6 26  ○£╗⌐Eë╢¿♥µ└1áT╓&
090:  3F 80 4C 06-0F C7 D9 19-09 D3 DA 14-FD CB 39 84  ?ÇL♠☼╟┘ ↓○╨┌¶²╤9ä
0A0:  1F 0D 77 5F-55 AA 7A 07-4C 24 8B 13-0A 54 A2 BC  ▼♪w_U¬z•L$ï‼◙Tó╝
0B0:  C5 12 7D 4F-E0 5E F2 23-C5 07 61 E4-80 91 B2 13  ┼↕}Oα^≥#┼•aΣÇæ▩‼

0C0:  E7 79 07 2A-CF 1B 66 39-8C F0 8E 7E-75 25 22 1D  τy•*╧←f9î≡Ä~u%"↔
0D0:  A7 3B 49 4A-32 A4 3A 07-61 26 64 EA-6B 83 A2 8D  º;IJ2ñ:•a&dΩkâóì
0E0:  BE A3 FF BE-4E 71 AE 18-E2 D0 86 4F-20 00 30 26  ╛ú ╛Nq«↑Γ╨åO  0&
0F0:  0A 71 DE 1F-40 B4 F4 8F-9C 50 5C 78-DD CD 72 89  ◙q▐▼@┤⌠Å£P\x▌=rë

100:  BA D1 BF F9-96 80 E3 06-96 F3 B9 7C-77 2D EB 25  ║╤┐•ûÇπ♠û≤╣|w-δ%
110:  1E 56 70 D7-14 1F 55 4D-EC 11 58 59-92 45 E1 33  ▲Vp╫¶▼UM∞◄XYÆEß3
120:  3E 0E A1 6E-FF D9 90 AD-F6 A0 AD 0E-C6 D6 88 12  >♫¡ín ┘Éi÷á¡♫╞╥ê↕
130:  B8 74 F2 9E-DD 53 F7 88-19 73 85 39-AA 9B E0 8D  ╕t≥₧▌S≈ê↓sà9¬¢αì

140:  82 BF 9C 5E-58 42 1E 3B-94 CF 5B 54-73 5F A8 4A  é┐£^XB▲;ö╧[Ts_¿J
150:  FD 5B 64 CF-59 D1 96 74-14 B3 0C AF-11 1C F9 47  ²[d╧Y╤ût¶│♀»◄└•G
160:  C5 7A 2C F7-D5 24 F5 EB-BE 54 3E 12-B0 24 67 3F  ┼z,≈╒$Jδ╛T>↕░$g?
170:  01 DD 95 76-8D 0D 58 FB-50 23 70 3A-BD ED BE AC  ☺▌òvì♪X√P#p:╜φ┘¼

180:  B8 32 DB AE-E8 DC 3A 83-7A C8 D5 0F-08 90 1D 99  ╕2█«Φ▄:âz╚╒☼◘É↔Õ
190:  2D 7D 17 34-4E A8 21 98-61 1A 65 DA-FC 9B A4 BA  -}↨4N¿!ÿa→e┌ⁿ¢ñ╢
1A0:  E1 42 2B 86-0C 94 2A F6-D6 A4 81 B5-2B 0B E9 37  ßB+å♀ö*÷╓ñü╡+♂07
1B0:  44 D2 E4 23-14 7C 16 B8-84 90 8B E0-A1 A7 BD 27  D╥Σ#¶│▬╕äÉïαí º╝'

1C0:  C7 7E E6 17-1A 93 C5 EE-59 70 91 26-4E 9D C7 7C  ╟~µ↨→ô┼εYpæ&N¥╟|
1D0:  1D 3D AB F1-B4 F4 F1 D9-86 48 75 77-6E FE 98 84  ↔=½±┤⌠±┘åHuwn■ÿä
1E0:  EF 3C 1C C7-16 5A 1F 83-60 EC 5C FE-CA 17 0C 74  ∩<└╟▬Z▼â`∞\■╨↨♀t
1F0:  EB 8E 9D F6-90 A3 CD 08-65 D5 5A 4C-2E C6 BE 54  δÄ¥÷Éú═◘eƒZL.╞╛T
```

no:

```
000:  6E 6F 0A E5-5F D0 83 01-9B 4D 55 06-61 AB 88 11  no◙σ_╨â☺¢MU♠a½ê◄
010:  8A FA 4D 34-B3 75 59 46-56 97 EF 6C-4A 07 90 CC  è·M4│uYFVùnlJ•É╠
020:  FE 19 D7 CF-6F 92 03 9C-91 AA A5 DA-56 92 C1 04  ■↓╫╧oÆ♥£æ¬Ñ┌VÆ┴♦
030:  E6 4C 08 A3-00 00 00 00-8D B6 4E 47-FF AF 7A 3C  µL◘ú    ì╢NG »z<

040:  D5 F1 54 CD-CA A1 42 90-7F 9D 3D 9A-67 C4 1B 0F  ╒±T═╩íBÉ⌂¥=Üg─←☼
050:  04 9F 19 E8-92 C3 AA 19-43 31 1A DB-DA 96 01 54  ♦ƒ↓ΦÆ├¬↓C1→█┌û©T
060:  85 B5 9A 88-D8 A5 0E FB-CD 66 9A DA-4F 20 8A A9  à╡Üê╪Ñ♫√=fÜ┌O è⌐
070:  BA E3 9C F0-78 31 8F D1-14 5F 3E B9-0F 9F 3E 19  ║π£≡x1Å╤¶_>╣☼ƒ>↓

080:  09 9C BB A9-45 89 BA A8-03 E6 C0 31-A0 54 D6 26  ○£╗⌐Eë╢¿♥µ└1áT╓&
090:  3F 80 4C 06-0F C7 D9 19-09 D3 DA 14-FD CB 39 84  ?ÇL♠☼╟┘ ↓○╨┌¶²╤9ä
0A0:  1F 0D 77 5F-55 AA 7A 07-4C 24 8B 13-0A 54 B2 BC  ▼♪w_U¬z•L$ï‼◙T▓╝
0B0:  C5 12 7D 4F-E0 5E F2 23-C5 07 61 E4-80 91 B2 13  ┼↕}Oα^≥#┼•aΣÇæ▩‼

0C0:  E7 79 07 2A-CF 1B 66 39-8C F0 8E 7E-75 25 22 1D  τy•*╧←f9î≡Ä~u%"↔
0D0:  A7 3B 49 4A-32 A4 3A 07-61 26 64 EA-6B 83 A2 8D  º;IJ2ñ:•a&dΩkâóì
0E0:  BE A3 FF BE-4E 71 AE 18-E2 D0 86 4F-20 00 30 22  ╛ú ╛Nq«↑Γ╨åO  0"
0F0:  0A 71 DE 1F-40 B4 F4 8F-9C 50 5C 78-DD CD 72 89  ◙q▐▼@┤⌠Å£P\x▌=rë

100:  BA D1 BF F9-96 80 E3 06-96 F3 B9 7C-77 2D EB 25  ║╤┐•ûÇπ♠û≤╣|w-δ%
110:  1E 56 70 D7-14 1F 55 4D-EC 11 58 59-92 45 E1 33  ▲Vp╫¶▼UM∞◄XYÆEß3
120:  3E 0E A1 6E-FF D9 90 AD-F6 A0 AD 0E-CA D6 88 12  >♫¡ín ┘Éi÷á¡♫╨╥ê↕
130:  B8 74 F2 9E-DD 53 F7 88-19 73 85 39-AA 9B E0 8D  ╕t≥₧▌S≈ê↓sà9¬¢αì

140:  82 BF 9C 5E-58 42 1E 3B-94 CF 5B 54-73 5F A8 4A  é┐£^XB▲;ö╧[Ts_¿J
150:  FD 5B 64 CF-59 D1 96 74-14 B3 0C AF-11 1C F9 47  ²[d╧Y╤ût¶│♀»◄└•G
160:  C5 7A 2C F7-D5 24 F5 EB-BE 54 3E 12-70 24 67 3F  ┼z,≈╒$Jδ╛T>↕p$g?
170:  01 DD 95 76-8D 0D 58 FB-50 23 70 3A-BD ED BE AC  ☺▌òvì♪X√P#p:╜φ┘¼

180:  B8 32 DB AE-E8 DC 3A 83-7A C8 D5 0F-08 90 1D 99  ╕2█«Φ▄:âz╚╒☼◘É↔Õ
190:  2D 7D 17 34-4E A8 21 98-61 1A 65 DA-FC 9B A4 BA  -}↨4N¿!ÿa→e┌ⁿ¢ñ╢
1A0:  E1 42 2B 86-0C 94 2A F6-D6 A4 81 B5-2B 2B E9 37  ßB+å♀ö*÷╓ñü╡++07
```

```
1B0:  44 D2 E4 23-14 7C 16 B8-84 90 8B E0-A1 A7 BD 27  D╥Σ#¶|▬┐äÉïαí º╝'

1C0:  C7 7E E6 17-1A 93 C5 EE-59 70 91 26-4E 9D C7 7C  ╟~µ↨→ô┼εYpæ&N¥╟|
1D0:  1D 3D AB F1-B4 F4 F1 D9-86 48 75 77-6E FE 98 84  ↔=½±┤⌠±┘åHuwn■ÿä
1E0:  EF 3C 1C C7-16 5A 1F 83-60 EC 5C FE-CA 17 0C 54  ∩<└╟▬Z▼â`∞\■╩↨♀T
1F0:  EB 8E 9D F6-90 A3 CD 08-65 D5 5A 4C-2E C6 BE 54  δÄ¥÷Éú═◘e╒ZL.╞╛T
```

Exploitations

Identical prefix collisions is usually seen as (very) limited, but chosen prefix is time consuming.

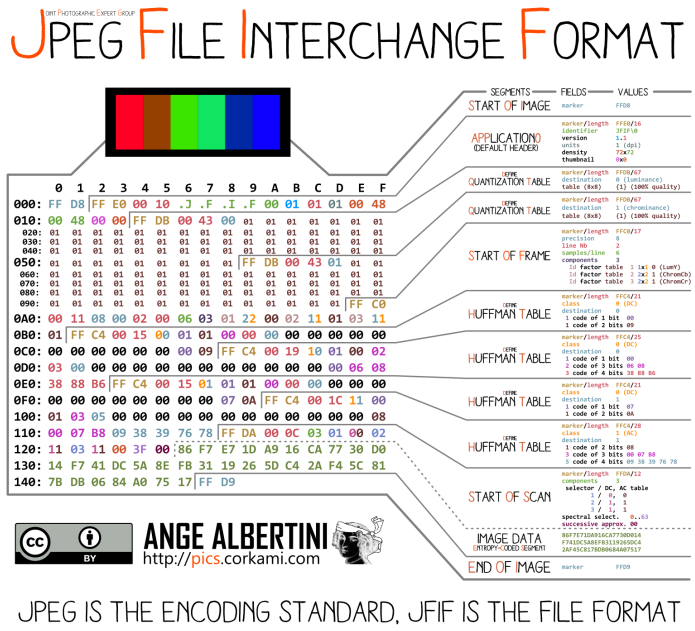
Another approach is to craft re-usable prefixes via either identical-prefix attack such as UniColl - or chosen prefix to overcome some limitations - but re-use that prefix pair in combinations with 2 payloads like a classic identical prefix attack.

Once the prefix pair has been computed, it makes colliding 2 contents instant: it's just a matter of massaging file data (according to specific file formats) so that it fits the file formats specifications and the precomputed prefix requirements.

Standard strategy

Classic collisions of 2 valid files with the same file type.

JPG



Theoretical limitations and workarounds:

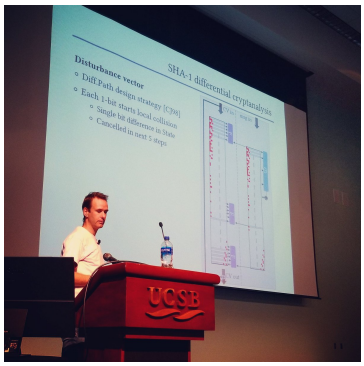
- the *Application* segment should in theory right after the *Start of Image* marker. In practice, this is not necessary, so our collision can be generic: the only limitation is the size of the smallest image.
- a comment's length is stored on 2 bytes, so it's limited to 65536 bytes. To jump over another image, its *Entropy Coded Segment* needs to be split to scans smaller than this, either by storing the image as progressive, either by using *JPEGTran* and custom scans definition.

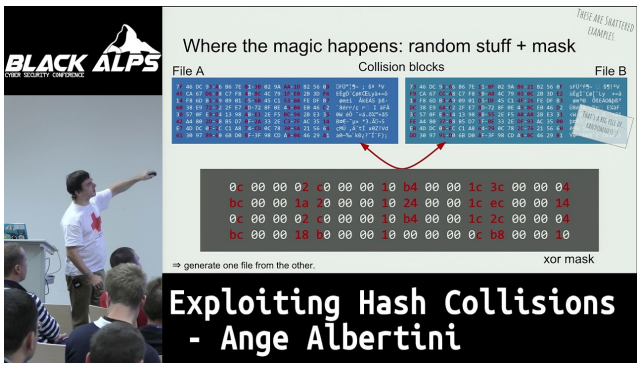
So an MD5 collision of 2 arbitrary JPGs is *instant*, and needs no chosen-prefix collision, just UniColl.

With the script:

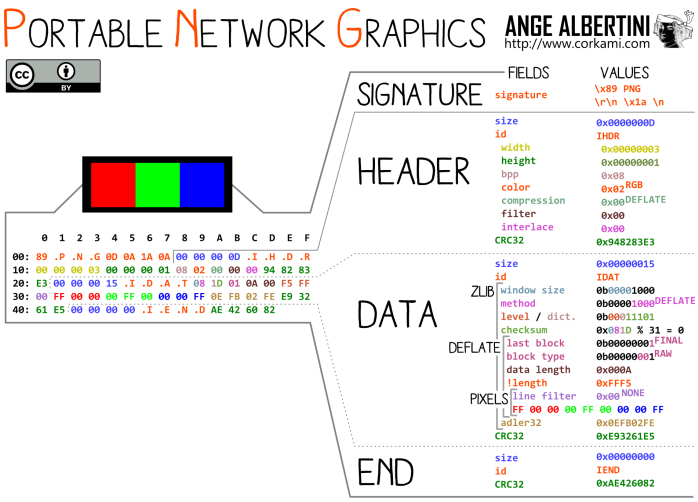
```
21:07:35.65>jpg.py Ange.jpg Marc.jpg

21:07:35.75>
```





PNG



Theoretical limitations and workarounds:

- PNG uses CRC32 at the end of its chunks, which would prevent the use of collision blocks, but in practice they're ignored.
- the image meta data (dimensions, color space...) are stored in the `IHDR` chunk, which should in theory be right after the signature (ie, before any potential comment), so it would mean that we can only precompute collisions of images with the same meta data. However, that chunk can actually be after a comment block, so we can put the collision data before the header, which enables to collide any pair of PNG with a single precomputation.

Since a PNG chunk has a length on 4 bytes, there's no need to modify the structure of either file: we can jump over a whole image in one go.

We can insert as many discarded chunks as we want, so we can add one for alignment, then one which length will be altered by a UniColl. so the length will be `00` `75` and `01` `75`.

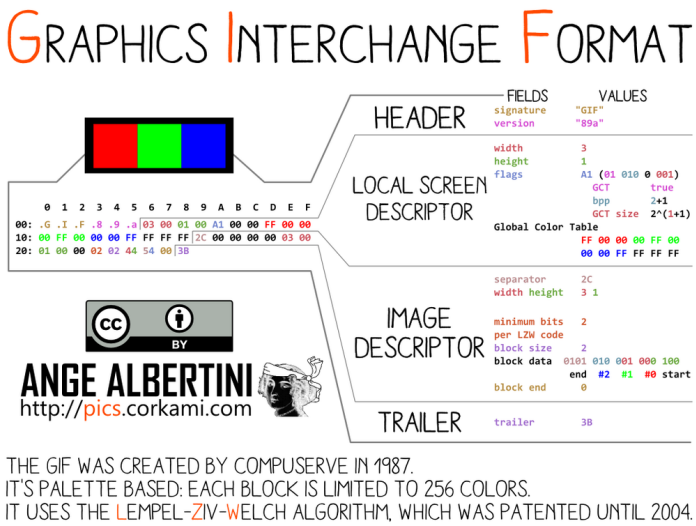
So an MD5 collision of 2 arbitrary PNG images is *instant*, with no prerequisite (no computation, just some minor file changes), and needs no chosen-prefix collision, just UniColl.

With the script:

```
19:27:04.79>png.py nintendo.png sega.png
```

```
19:27:04.87>
```

GIF



GIF is tricky:

- it stores its meta data in the header before any comment is possible, so there can't be a generic prefix for all GIF files.
- if the file has a global palette, it is also stored before a comment is possible too.
- its comment chunks are limited to a single byte in length, so a maximum of 256 bytes!

However, the comment chunks follow a peculiar structure: it's a chain of `<length:1>` `<data:length>` until a null length is defined. So it makes any non-null byte a valid 'jump forward'. Which makes it suitable to be used with FastColl, as shown in PoC||GTFO 14:11.

So at least, even if we can't have a generic prefix, we can collide any pair of GIF of same metadata (dimensions, palette) and we only need a second of FastColl to compute its prefix.

Now the problem is that we can't jump over a whole image like PNG or over a big structure like JPG.

A possible workaround is to massage the compressed data or to chunk the image in tiny areas like in the case of the GIF Hashquine, but this is not optimal.

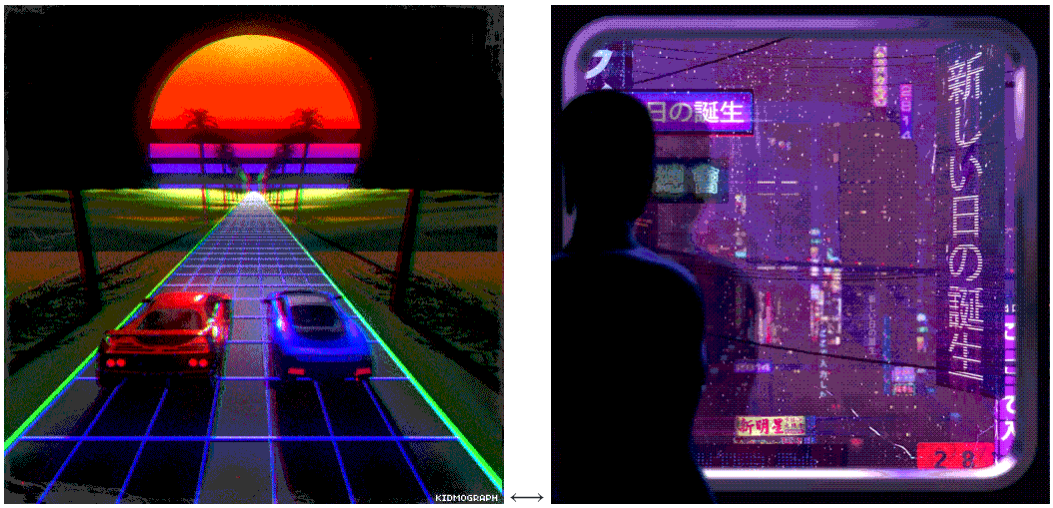
Another idea that works generically is that the image data is also stored using this `length data` sequence structure: so if we take 2 GIFs with no animation, we only have to:

- normalize the palette
- set the first frame duration to the maximum
- craft a comment that will jump to the start of the first frame data, so that the comment will sled over the image data as a comment, and end the same way: until a null length is encountered. Then the parser will meet the next frame, and display it.

With a minor setup (only a few hundred bytes of overhead), we can sled over any GIF image and work around the 256 bytes limitation. This idea was suggested by Marc, and it's brilliant!

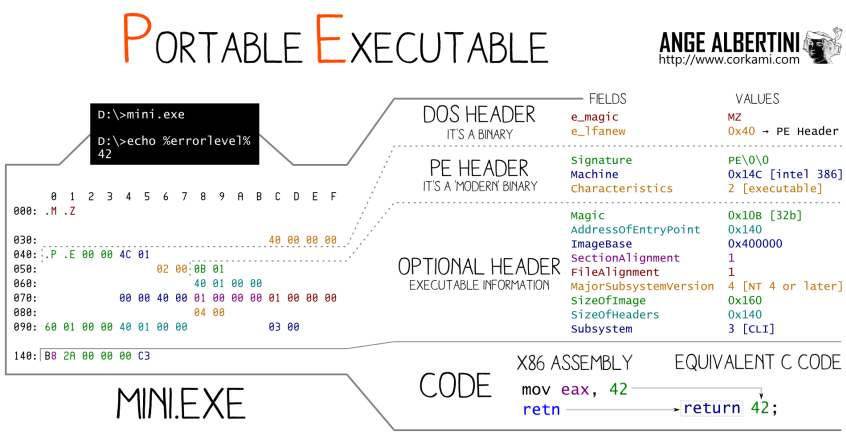
So in the end, the current GIF limitations for *instant* MD5 collisions are:

- no animation
- the images have to be normalized to the same palette - see `gifsicle --use-colormap web`
- the images have to be the same dimensions
- after 11 minutes, both files will show the same image



Pics by KidMoGraph

Portable Executable



The Portable Executable has a peculiar structure:

- the old DOS header is almost useless, and points to the next structure, the PE header. The DOS headers has no other role. DOS headers can be exchanged between executables.
- the DOS header has to be at offset 0, and has a fixed length of a full block, and the pointer is at the end of the structure, beyond UniColl's reach: so only Chosen Prefix collision is useful to collide PE files this way.
- The PE header and what follows defines the whole file.

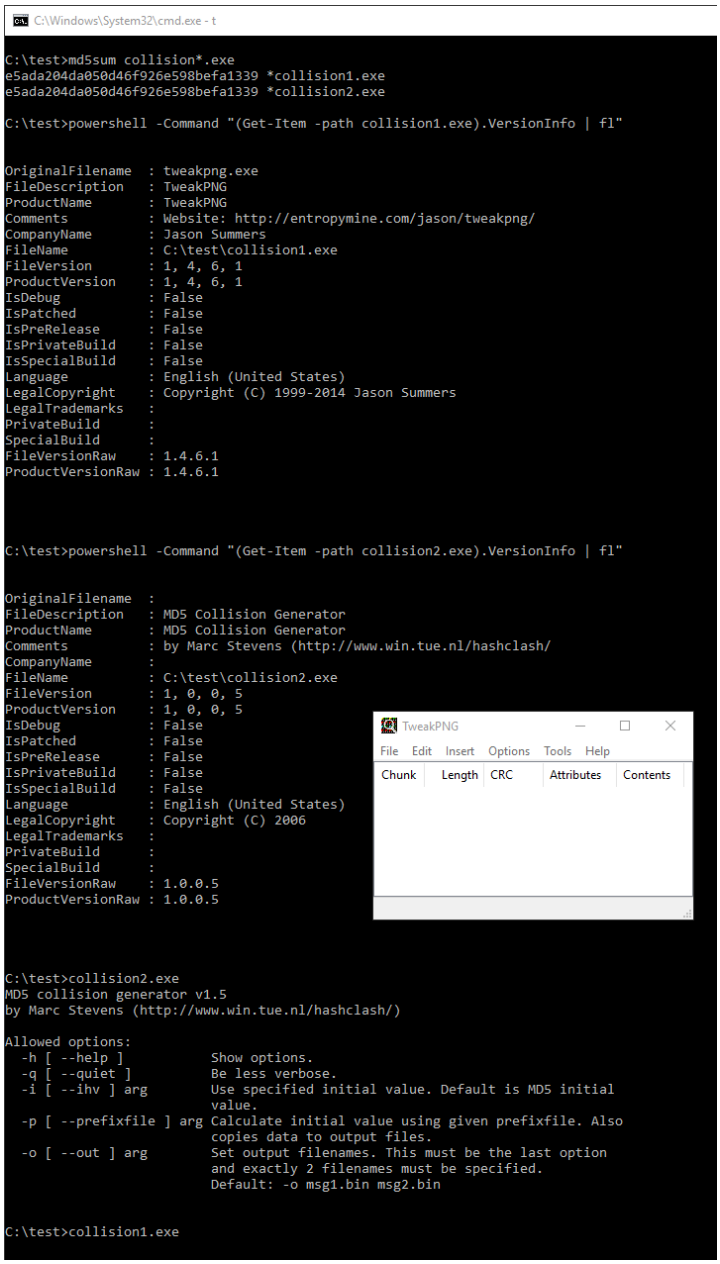
So the strategy is:

1. the PE header can be moved down to leave room for collision blocks after the DOS header.
2. The DOS header can be exploited (via chosen prefix collisions) to point to 2 different offsets, where 2 different PE headers will be moved.
3. The sections can be put next to each other, after the `DOS/Collisions/Header1/Header2` structure. You just need to apply a delta to the offsets of the 2 section tables.

This means that it's possible to instantly collide any pair of PE executables. Even if they use different subsystems or architecture.

While executables collisions is usually trivial via any loader, this kind of exploitation here is transparent: the code is identical and loaded at the same address.

Examples: tweakPNG.exe (GUI) ⟷ fastcoll.exe (CLI)



MP4 and others

This format's container is a sequence of `Length Type Value` chunks called Atoms. The length is a 32 bit big-endian and covers itself, the type and the value, so the minimum normal length is 8 (the type is a 4 ASCII characters string).

If the length is null, then the atom takes the rest of the file - such as `jp2c` atoms in JP2 files. If it's 1, then the Type is followed by a 64bit length, changing the atom to `Type Length Value`, making it compatible with other collisions like Shattered.

Some atoms contain other atoms: in this cases, they're called boxes. That's why this otherwise unnamed structure is called "atom/box".

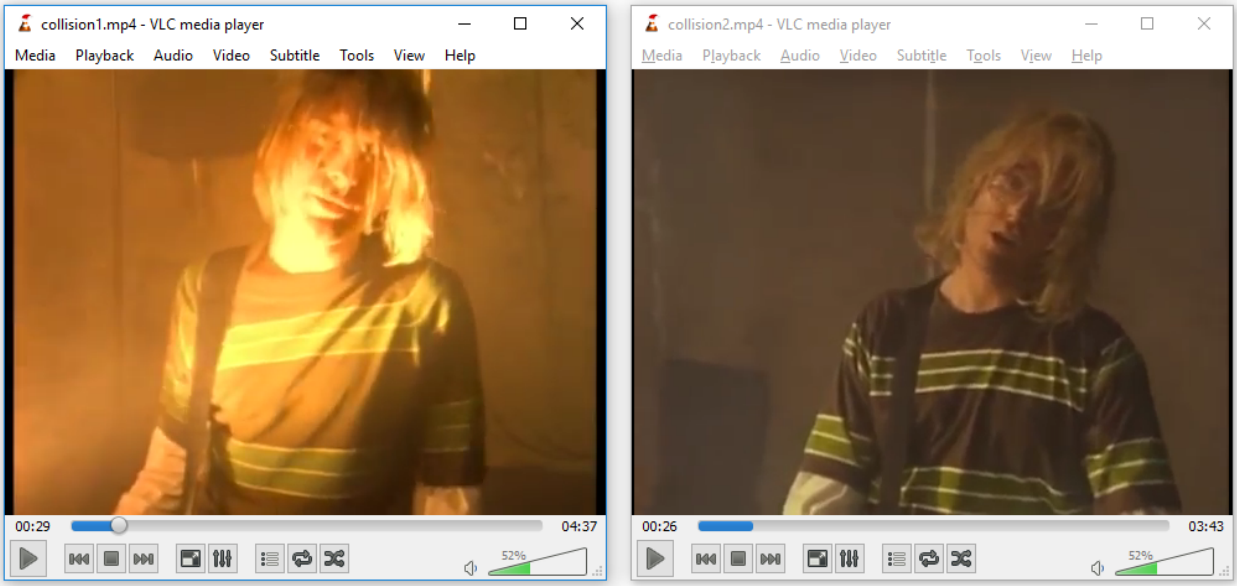
This "atom/box" format used in MP4 is actually a derivate of Apple Quicktime, and is used by many other formats (JP2, HEIF, F4V).

The first atom type is *usually* `ftyp`, which enables to differentiate the actual file format.

The format is quite permissive: just chain `free` atoms, abuse one's length with UniColl, then jump over the first payload.

For MP4 files, the only thing to add is to adjust the `stco` (Sample Table - Chunk Offsets) or `co64` (the 64 bit equivalent) tables, since they are absolute(!) offsets pointing to the `mdat` movie data - and they are actually enforced!

This gives a script that instantly collides any arbitrary video - and as mentioned, it may work on other format than MP4.



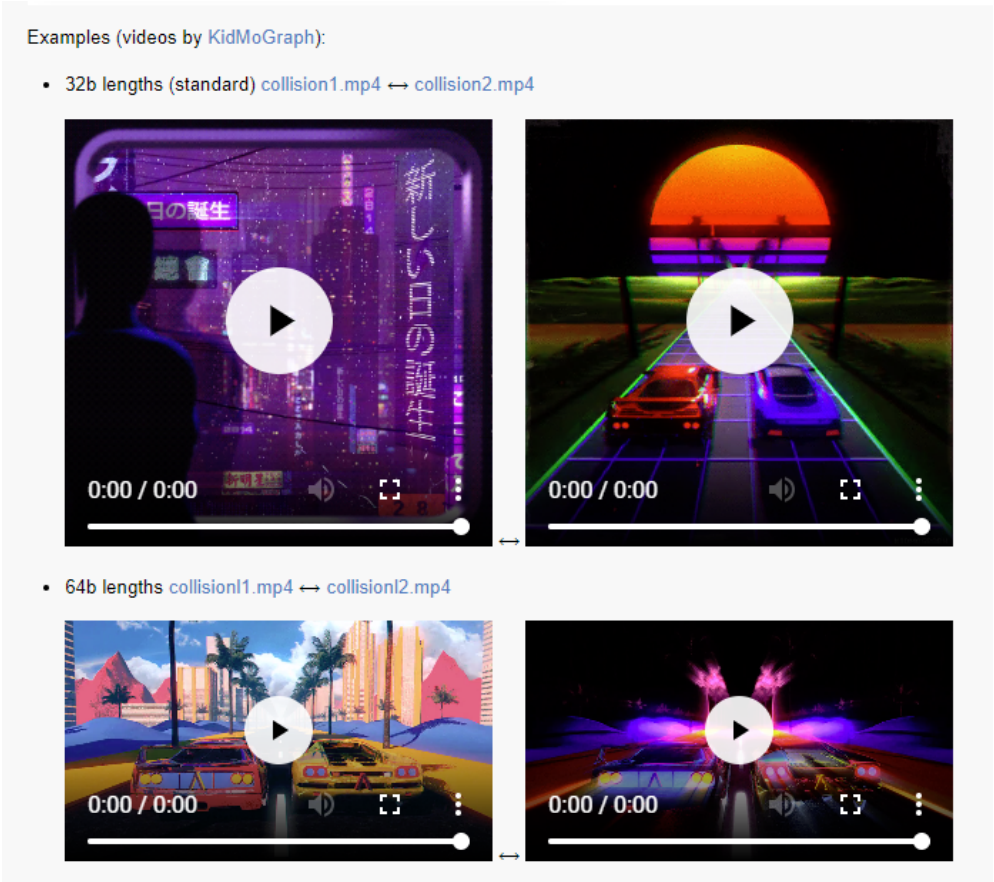
Examples (videos by KidMoGraph):

- 32b lengths (standard) collision1.mp4 ⟷ collision2.mp4

  🏭 ⟷ 🚞

- 64b lengths collisionl1.mp4 ⟷ collisionl2.mp4

  ☀️ ⟷ 🌙



JPEG2000

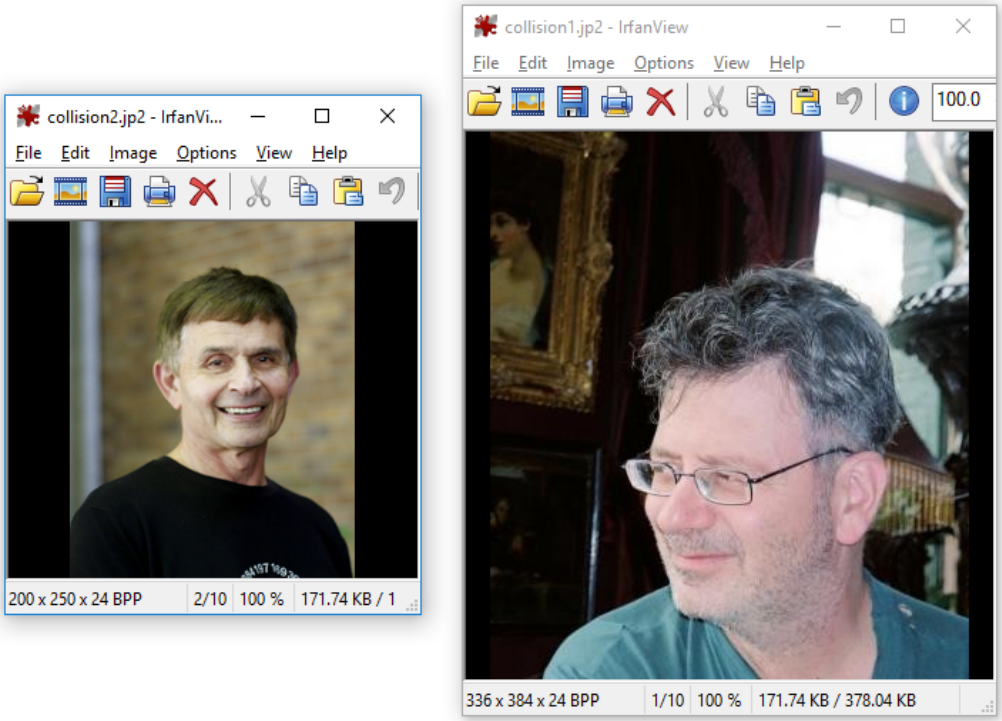
JPEG2000 files usually start with the Atom/Box structure like MP4, then the last atom `jp2c` is typically until the end of the file (null length), then from this point on it follows the JFIF structure, like JPEG (starting with `FF 4F` as a segment marker).

The pure-JFIF form is also tolerated, in which case collision is like JPEG: Shattered-compatible, but with comments limited to 64Kb.

On the other hand, if you manipulate JPEG2000 files with the Atom/Box, you don't have this limitation.

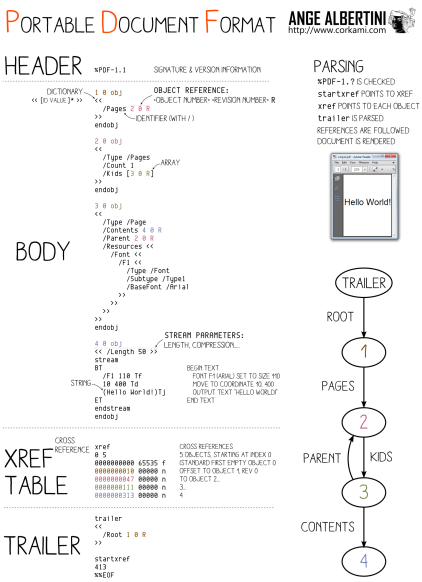
As mentioned before, if you're trying to collide this structure and if there are more restriction - for example starting with a `free` atom is not tolerated by some format - then you can compute another UniColl prefix pairs specific to this format: JPEG2000 seems to enforce a `'jP '` atom first before the usual `ftyp`, but besides, that's the only restriction: there's no need to relocate anything.

So the resulting script is even simpler!



Examples: collision1.jp2 ⟷ collision2.jp2

PDF



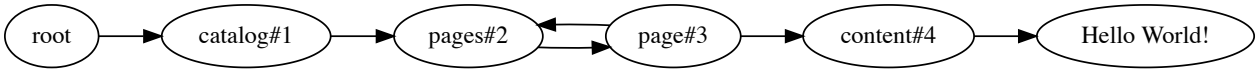
collision

Shattered exploitation was not a PDF trick, but a JPG trick in a PDF.

It only enabled a PDF to contain a JPG-compressed object that could have 2 different contents. Both PDFs needed to be totally identical beside.

With MD5 (and other collision patterns), we can do PDF collisions at document level, with no restrictions at all on either file!

PDF has a very different structure from other file formats. It uses object numbers and references to define a tree. The whole document depends on the Root element.



This (valid) PDF

```
%PDF-1.
1 0 obj<</Pages 2 0 R>>endobj
2 0 obj<</Kids[3 0 R]/Count 1>>endobj
3 0 obj<</Parent 2 0 R>>endobj
trailer <</Root 1 0 R>>
```

is equivalent to:

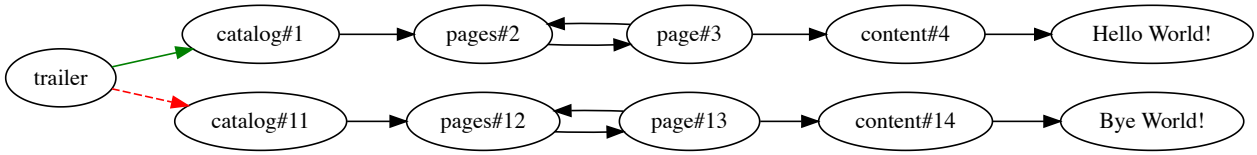
```
%PDF-1.
11 0 obj<</Pages 12 0 R>>endobj
12 0 obj<</Kids[13 0 R]/Count 1>>endobj
13 0 obj<</Parent 12 0 R>>endobj
trailer <</Root 11 0 R>>
```

Tricks:

- Storing unused objects in a PDF is tolerated.
- Skipping any object numbers is also OK. There's even an official way to skip numbers in the `XREF` table.

So storing 2 document trees in the same file is OK. We just need to make the root object refer to either root object of both documents.

So we just need to take 2 documents, renumber objects and references so that there is no overlap, craft a collision so that the element number referenced as Root object can be changed while keeping the same hash value, which is a perfect fit for UniColl with `N=1`, and adjust the `XREF` table accordingly.



This way, we can safely collide any pair of PDFs, no matter the page numbers, dimensions, images...

comments

PDF can store foreign data in two ways:

- as a line comment, in which the only forbidden characters are newline (`\r` and `\n`). This can be used inside a dictionary object, to modify for example an object reference, via UniColl. So this is a valid PDF object even if it contains binary collision blocks - just retry until you have no newline characters:

  ```
  1 0 obj
  << /Type /Catalog /MD5_is /REALLY_dead_now__ /Pages 2 0 R
  %¥┬•σe┐▉╚X‰_~π▌ ┌εXL■φe♦%τ8├■[...]p╛╬ûFZ»‼v◘Åp↑╝%▓% ▼σφj╔◄dZ■c²aU≤╨╙[├└─yNΓ5╔+■╪yδ☻ß⌐⁘¼à(☺z‰
  >>
  endobj
  ```

- as a stream object, in which case any data is possible, but since we're inside an object, we can't alter the whole PDF structure, so it requires a chosen prefix collision to modify the structure outside the containing stream object.

colliding text

The first case makes it possible to highlight the beauty of UniColl, a collision where differences are predictable, so you can write poetry over colliding data - thanks Jurph!

Rather than modifying the structure of the document and fool parsers, we'll just use collision blocks directly to produce directly text, with alternate reading!

A true cryptographic artistic creation :)

- poeMD5 A

```
         V
Now he hash MD5,
No enemy cares!
 Only he gave
 the shards.
Can’t be owned &
his true gold,
like One Frail,
sound as fold.
         ^
```

- poeMD5 B

```
         V
Now he hath MD5,
No enemy dares!
 Only he have
 the shares.
Can’t be pwned &
his true hold,
like One Grail,
sound as gold.
         ^
```

(Note I screwed up with Adobe compatibility, but that's my fault, not UniColl's)

colliding document structure

Whether you use UniColl as inline comment or Chosen Prefix in a dummy stream object, the strategy is similar: shuffle objects numbers around, then make Root object point to different objects, so unlike Shattered, this means instant collision of any arbitrary pair of PDF, at document level.

A useful trick is that `mutool clean` output is reliably predictable, so it can be used to normalize PDFs as input, and fix your merged PDF while keeping the important parts of the file unmodified. MuTool doesn't discard bogus key/values - unless asked, and keep them in the same order, so using fake dictionary entries such as `/MD5_is /REALLY_dead_now__` is perfect to align things predictably without needing another kind of comments. However it won't keep comments in dictionaries (so no inline-comment trick)

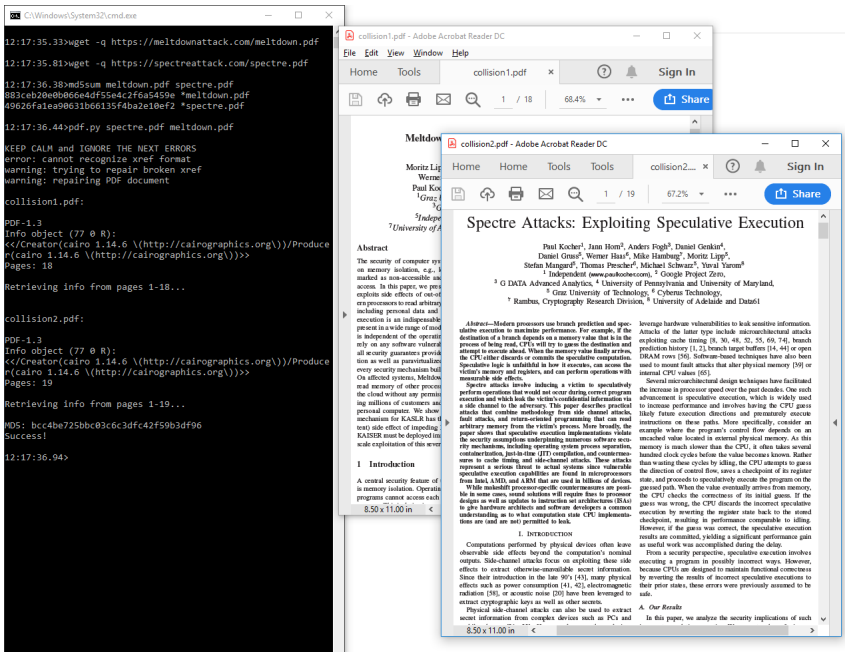
An easy way to do the object-shuffling operation without hassle is just to merge both PDF files via `mutool merge` then split the `/Pages` object in 2.

To make room for this object, just merge in front of the 2 documents a dummy PDF.

Optionally, create a fake reference to the dangling array to prevent garbage collection from deleting the second set of pages.

Example: with this script, it takes less than a second to collide the 2 public PDF papers like Spectre and Meltdown:

Examples: spectre.pdf ⟷ meltdown.pdf



Possible extension: chain UniColl blocks to also keep pairs of the various non-critical objects that can be referenced in the Root object - such as `Outlines`, `Names`, `AcroForm` and Additional Actions (`AA`) - in the original source files.

in PDFLaTeX

The previous techniques work with just a pair of PDF files, but it's also possible to do it directly from TeX sources via specific PDFTeX operators.

You can define objects directly - including dummy key and values for alignments - and define empty objects to reserve some object slots by including this at the very start of your TeX sources:

```
% set PDF version low to prevent stream XREF
\pdfminorversion=3

\begingroup

  % disable compression to keep alignments
  \pdfcompresslevel=0\relax

  \immediate
  \pdfobj{<<
    /Type /Catalog

    % cool alignment padding
    /MD5_is /REALLY_dead_now__

    % the first reference number should be on offset 0x49, so 2 will be changed to 3 by UniColl
    /Pages 2 0 R

    % now padding so that the collision blocks (ends at 0xC0) are covered
    /0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    % with an extra character to be replaced by a return char
    /0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
  >>}

  % the original catalog of the shifted doc
  \immediate\pdfobj{<</Type/Pages/Count 1/Kids[8 0 R]>>}

  % the original catalog of the host doc
  \immediate\pdfobj{<</Type/Pages/Count 1/Kids[33 0 R]>>}

  % now we need to reserve PDF Objects so that there is no overlap
  \newcount\objcount

  % the host size (+3 for spare object slots) - 1
  % putting a higher margin will just work, and XREF can have huge gaps
  \objcount=25
  \loop
    \message{\the\objcount}
    \advance \objcount -1
```

```
  \immediate\pdfobj{<<>>} % just an empty object

  \ifnum \objcount>0
  \repeat

\endgroup
```

Don't forget to normalize PDFLaTeX output - with `mutool` for example - if needed: PDFLaTeX is hard to get reproducible builds across distributions - you may even want to hook the time on execution to get the exact hash if required.

Uncommon strategies

Collisions are usually about 2 valid files of the same type.

MultiColls: multiple collisions chain

Nothing prevents to chain several collision blocks, and have more than 2 contents with the same hash value. An example of that are Hashquines - that shows their own MD5 value. The PoCGTFO 14 file contains 609 FastColl collisions, to do that through 2 file types in the same file.

Validity

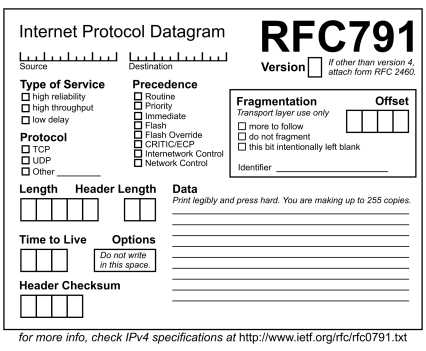
A different strategy would be to kill the file type to bypass scanning as a corrupted file. Just overwriting the magic signature will be enough. Appending both files (as valid or invalid) with a format that doesn't need to be at offset 0 (archive, like ZIP/RAR/...) would reveal another file type.

This enables polyglot collisions without using a Chosen prefix collision:

1. use UniColl to enable or disable a magic signature, for example a PNG:
2. append a ZIP archive

While technically both files are a valid ZIP, since most parser return the first file type found and they start scanning at offset 0, they will see a different file type.

Examples:



⟷ invalid

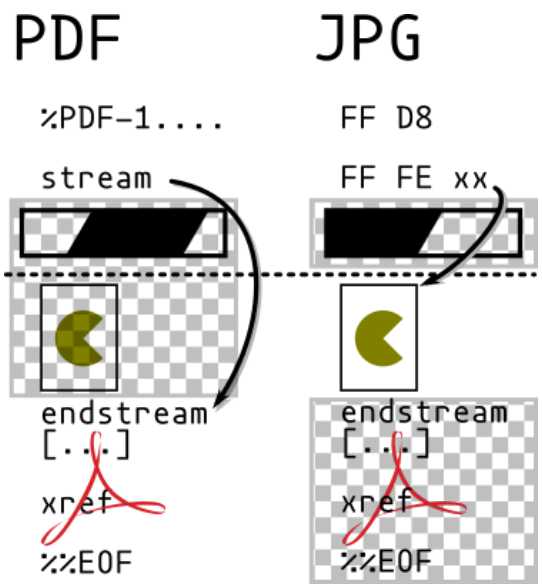
PolyColls: collisions of different file types

It's also possible to have both side of a collision with different types to lower suspicion:

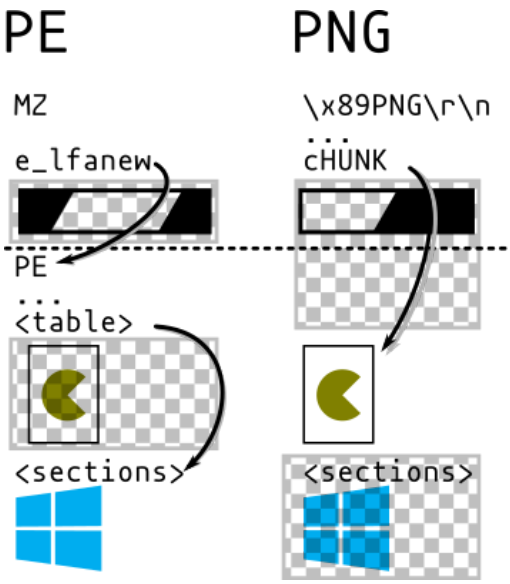
Attack scenario:

1. send `holiday.jpg`
2. get it whitelisted
3. send `evil.exe`, which has the same MD5.

Some examples of polycoll layouts:



PDF/JPG polycoll



PE/PNG polycoll

Portable Executable - JPG

Since a PE header is usually smaller than 0x500 bytes, it's a perfect fit for a JPG comment:

1. start with DOS/JPG headers
2. JPEG-comment jumps over PE Header
3. Put the full JPG image
4. Put the whole PE specifications

Once again, the collision is instant.

Examples: fastcoll.exe ⟷ Marc.jpg

PDF - PNG

Similarly, it's possible to collide for example arbitrary PDF and PNG files with no restriction on either side. This is instant, re-usable and generic.

Examples: Hello.pdf ⟷ 1x1.png

Use cases

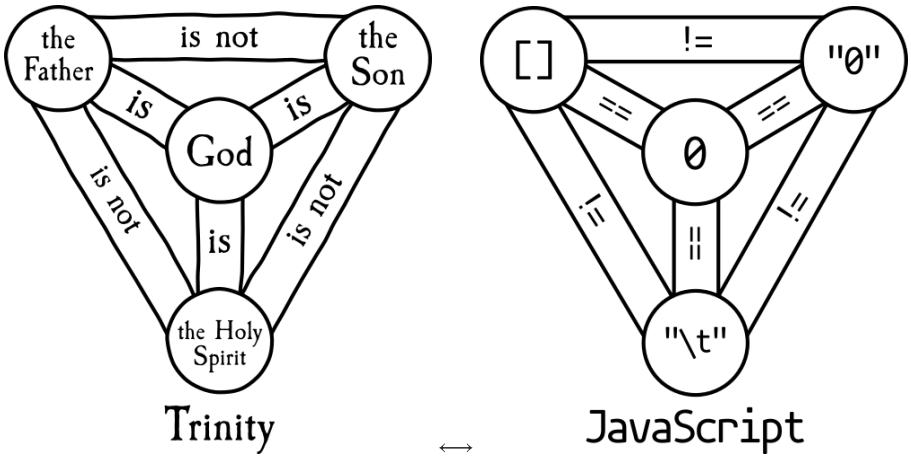
Better discard MD5 altogether, because file introspection is just too time-consuming and too risky!

Gotta collide 'em all!

Another use of instant, re-usable and generic collisions would be to hide any file of a given type - say PNG - behind dummy files (or the same file every time) - which is actually just by concatenating it to the same prefix after stripping the signature - you could even do that at library level!

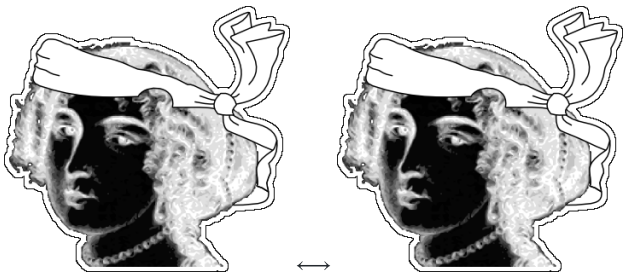
From a strict parsing perspective, all your files will show the same content, and the evil images would be revealed as a file with the same MD5 as previously collected.

Let's take 2 files:



and collide them with the same PNG.

They now show the same dummy image, and they're absolutely identical until the 2nd image at file level!

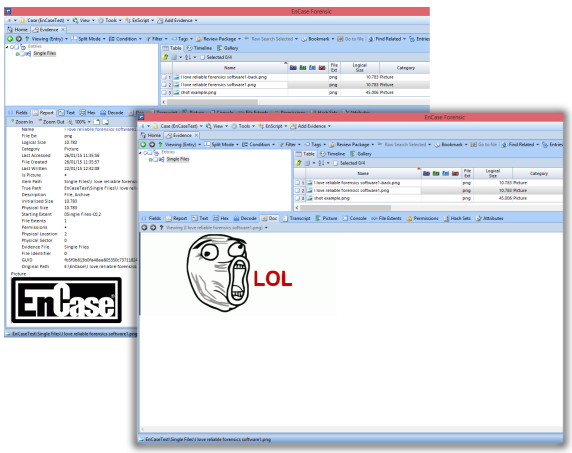


Their evil payload is hidden behind a file with the same MD5 respectively.

Incriminating files

Another use case for collisions is to hide something incriminating inside something innocent, but desirable: if the only thing to collect evidence is comparing weak hashes, then you can't deny that you don't have the other file (showing incriminating content but hiding innocent content).

Softwares typically focus on (quick) parsing, not on detailed file analysis.



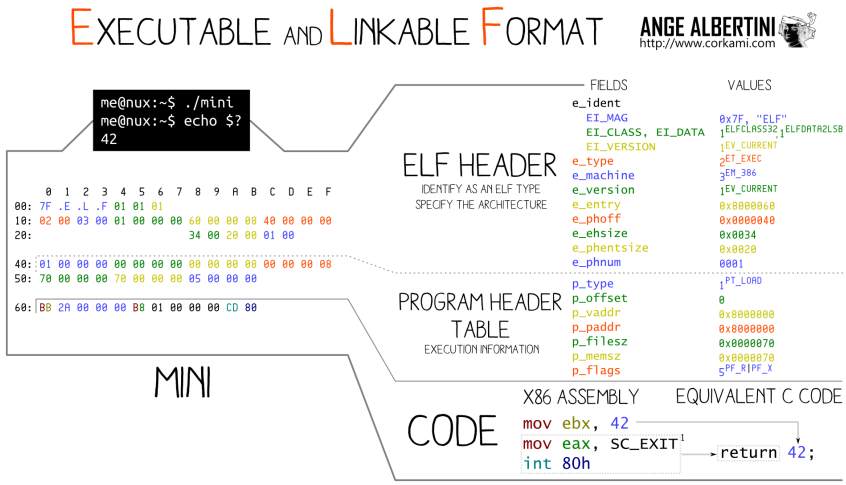
an image showing different previews under different tabs of EnCase Forensic

Failures

Not all formats can have generic prefixes that can be re-used: if some kind of data holder can't be inserted between the magic signature and the standard headers that are critical and specific to each file, then generic collisions are not possible.

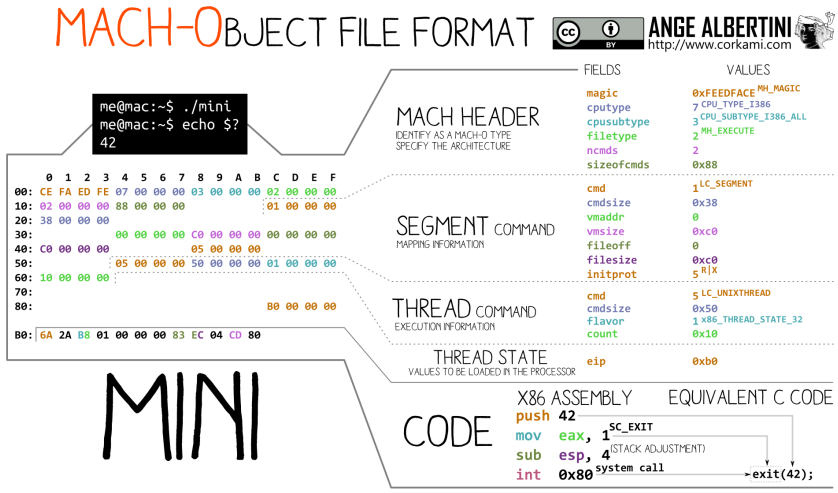
Of course, one might still turn the old files into a new one, and even use code to branch out to 2 different payloads, but it's more like porting payloads than colliding file structure.

ELF



The ELF header is required at offset 0 and contains critical information such as 32b/64b, endianness and ABI right from the beginning, so it's impossible to have a universal prefix then collision blocks before critical parameters that are specific to the original file.

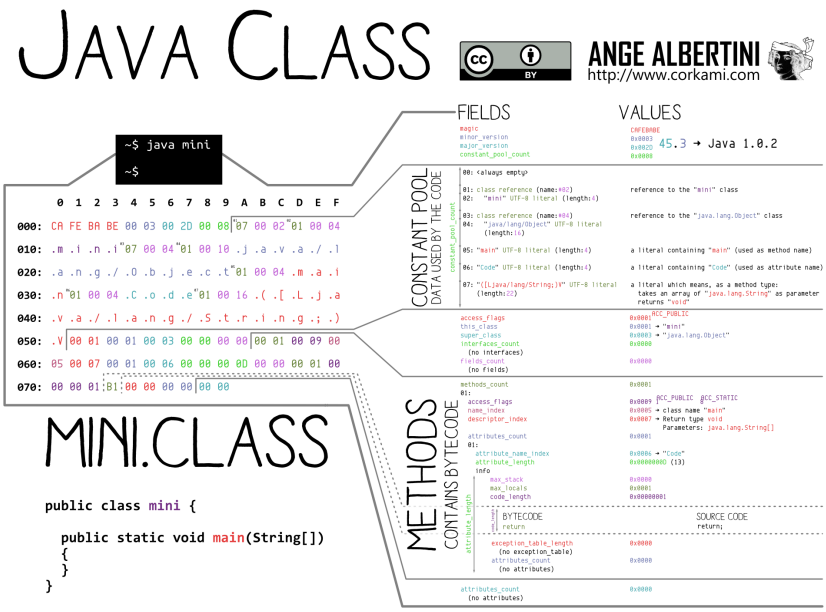
Mach-O



Mach-O don't even start with the same magic for 32b (`feedface`) and 64b (`feedfacf`). Soon after, there is the number and size of commands (such as segment definition, symtab, version,...).

Like ELF, re-usable collisions are not possible.

Java Class



Right from the start magic are located the versions (which can be troublesome) but the constant pool count which is quite specific to each file, so no universal collisions for all files.

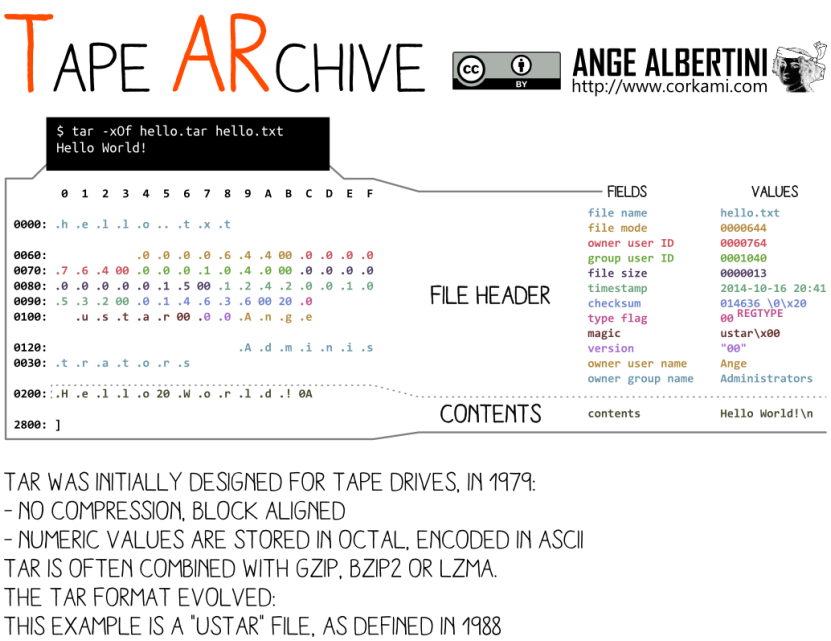
However, many files still have a common version and we can pad the shortest constant pool to the longuest count. First, insert a *UTF8 literal* to align information, then declare another one with its length abused by a UniColl (the length is stored on 16 bytes as big endian).

However this will require code manipulation since all pool indexes will be shifted.

Instant MD5 re-usable collisions of Java Class should be possible, but require code analysis and modification.

TAR

TL;DR No re-usable collision for TAR files, no other strategy than Chosen Prefix.



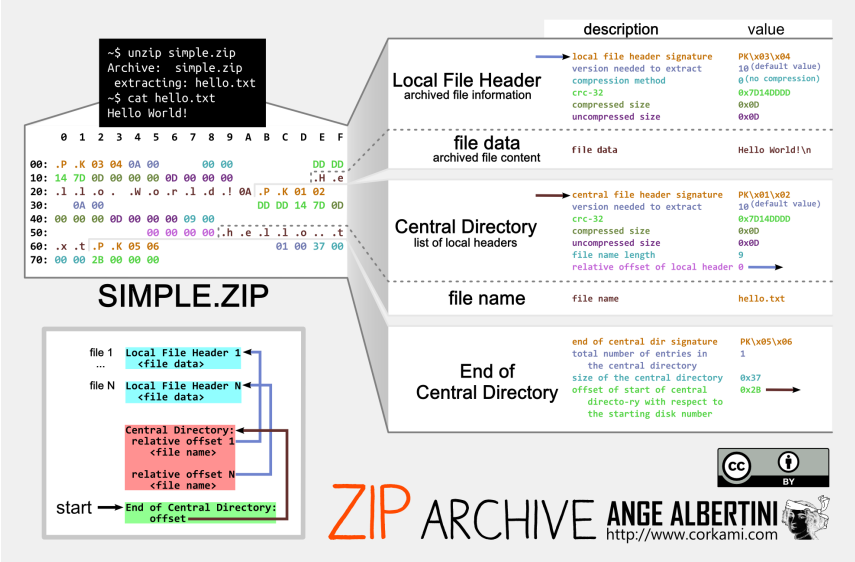
Tape Archives are a sequence of concatenated header and file contents, all aligned to 512 bytes.

There's no central structure to the whole file. So no global header or comment of any kind to abuse.

A trick would be to start a dummy file of variable length, but the length is always at the same offset, which is not compatible with UniColl, which means only Chosen Prefix collisions is useful here.

ZIP

TL;DR There's no generic re-usable collision for ZIP. It should be possible to collide 2 files in 2h.core (36 times faster than Chosen Prefix)



ZIP archives are a sandwich of 3 layers (at least). First comes the files' content (sequence of `Local File Header` structures, one per archived file or directory), then some index (again, a sequence of `Central Directory`), then a single structure that points to this index (`End Of Central Directory`).

The order of these layers can't be moved around. Some parser only need the file content's structure, but that's not a correct way to parse and it can be abused.

Because of this required order, there's no generic prefix that could help for any collision.

non generic approach

Another approach could be to just merge both archives, with their merged layers, and using UniColl - but with N=2, which introduces a difference on the 4th byte - to kill the magic signature of the `End of Central Directory`.

This means one could collide 2 arbitrary ZIP with a single UniColl and 24 bytes of set prefix.

A typical End of Central Directory, which is 22 bytes if the comment is empty:

```
00: 504b 0506 0000 0000 0000 0000 0000 0000  PK..............
10: 0000 0000 0000                           ......
```

If we use this as prefix (padd the prefix to 16 bits) for UniColl and `N=2`, the difference is on the 4th byte, killing the magic `.P .K 05 06` by changing it predictably to `.P .K 05 86`

```
00: 504b 0506 0000 0000 0000 0000 0000 0000  PK..............
10: 0000 0000 0000 2121 eb66 cf9d db01 83bb  ......!!.f......
20: 2888 4c41 e345 7d07 1634 5d4a 3b61 89a0  (.LA.E}..4]J;a..
30: 0029 94af 4168 2517 0bbc b841 cbf2 9587  .)..Ah%....A....
40: e438 0043 6390 279d 7c9e a01e e476 4c36  .8.Cc.'.|....vL6
50: 527f b1f4 653e d866 f98d 7278 5324 0bd5  R...e>.f..rxS$..
60: b31d ef6d d5d6 1163 5a2e a8a5 21bf eab4  ...m...cZ...!...
70: c59c 028e a913 f6b7 0036 c93f 5092 a628  .........6.?P..(
```

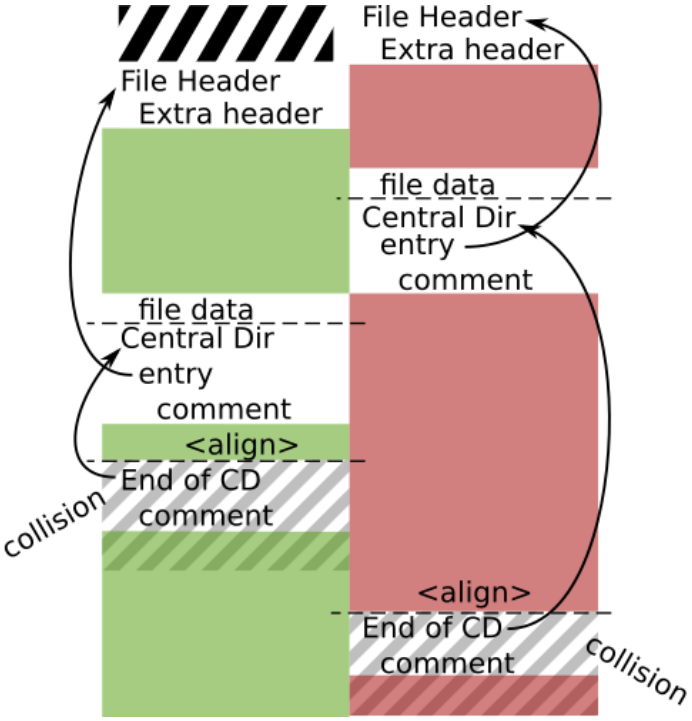
```
00: 504b 0586 0000 0000 0000 0000 0000 0000  PK..............
10: 0000 0000 0000 2121 eb66 cf1d db01 83bb  ......!!.f......
20: 2888 4c41 e345 7d07 1634 5d4a 3b61 89a0  (.LA.E}..4]J;a..
30: 0029 94af 4168 251f 0bbc b841 cbf2 9587  .)..Ah%....A....
40: e438 00c3 6390 279d 7c9e a01e e476 4c36  .8..c.'.|....vL6
50: 527f b1f4 653e d866 f98d 72f8 5324 0bd5  R...e>.f..r.S$..
60: b31d ef6d d5d6 1163 5a2e a8a5 21bf eab4  ...m...cZ...!...
70: c59c 028e a913 f6af 0036 c93f 5092 a628  .........6.?P..(
```

This is not generic at all, but much faster than Chosen-Prefix collision:

```
real 12m23.993s
user 112m24.072s
sys 2m0.194s
```

A problem is that some parsers still parse ZIP files upside-down even if they should be parsed bottom-up: a way to make sure that both files are properly parsed is to chain 2 UniColl blocks, to enable/disable each `End of Central Directory`.

To prevent ZIP parsers from complaining about unused space, one can abuse `Extra Fields`, file comments in `Central Directory` and archive comments in `End of Central Directory`.

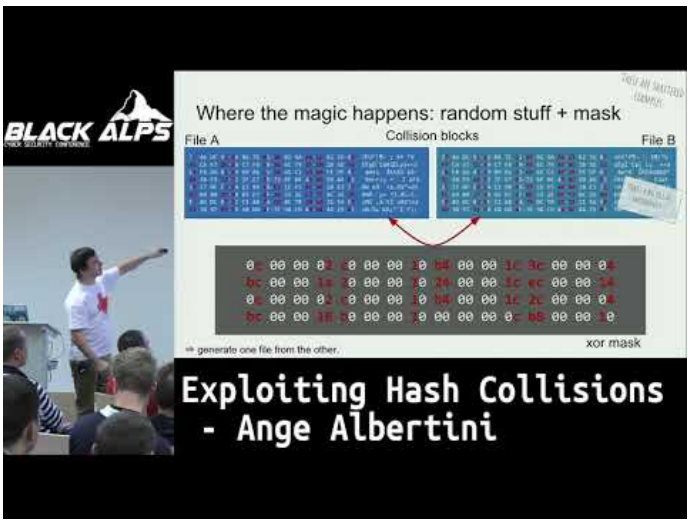


Example: here is an assembly source that describes the structure of a dual ZIP, that can host 2 different archive files.

After 2 unicoll computations, it gives the 2 colliding files: collision1.zip ⟷ collision2.zip

Presentations

Exploiting Hash Collisions (2017):

- slides
- video 

Conclusion

Kill MD5!

Unless you actively check for malformations or collisions blocks in files, don't use MD5!

It's not a cryptographic hash, it's a toy function!