19:05 An MD5 Pileup Fit for Jake and Elwood

by Albertini and Stevens

This article is about applying known hash collisions to common file formats. It is *not* about new collisions—the most recent one we'll discuss was documented in 2012—but instead focuses on the byte patterns techniques that are exploited in the present attacks and are likely to continue being useful for future ones.

We'll extensively explore existing attacks, showing once again just how weak MD5 is (instant collisions of any of JPG, PNG, PDF, MP4, PE, *etc.*), and will also explore how the common file format features help the attacker construct colliding files. Indeed, the same file format tricks can be used on several hashes, as long as the collisions follow the same byte patterns. Compare, for instance, the JPEG tricks from PoC‖GTFO 14:10 and the malicious SHA1 collision from the SHAttered project.

Follow along and we'll learn the moves of the collision dance, the tricks of the trade for colliding different valid files so that they share a single hash. We'll begin by reviewing the available collision techniques, then show how real world files can be abused within the constraints of the available, practical block collisions.

State of the art

There are different ways in which we may want to construct colliding files, depending on whether we want to control the files' contents or the hashes themselves. The current status of known attacks—as of December 2018—is as follows:

Generating a file that matches a specific fixed hash is still impractical with MD5 and everything stronger. It is impractical even with MD2,[20] but can be done for simpler hashes such as Python's `crypt()`. The following example is thanks to Sven, (`@svblxyz`).

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

[20] `unzip pocorgtfo19.pdf thomsenmd2.pdf`

[21] `git clone https://github.com/nneonneo/sha1collider`

While we can't yet generate a file for an arbitrary MD5 hash, we can generate identical prefix collisions (FastColl, UniColl, SHAttered) and even chosen prefix collisions (HashClash). Because both hashed and file formats often run from beginning to end, these prefixes can be freely reused after generation to produce two arbitrary files that obey a specific file format (PNG, JPG, PE, *etc.*) with the same hash.

As an extreme example, making two different files with the same SHA1 took 6,500 core years, but now that those prefixes have been computed, we can instantly produce two different PDFs with the same SHA1 hash that show different pictures.[21]

Attacks

MD5 and SHA1 both operate on blocks of 64 bytes. If two content blocks A and B have the same hash, then appending (we'll write "+" for append) the same suffix C to both will retain equality of the total hash.

$$\text{hash}(A) = \text{hash}(B) \Rightarrow \text{hash}(A + C) = \text{hash}(B + C)$$

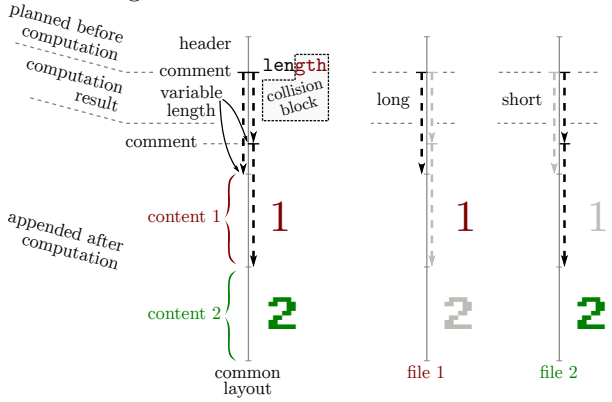
Collisions of files with fixed different prefixes work by inserting at a block boundary some number of computed *collision blocks* that depend on what came before in the file. These collision blocks are very random-looking with some minor differences, which follow a specific pattern for each attack. These tiny differences eventually get the hashes to converge to the same value after these blocks.

The key thing about file formats that makes this method work is that file formats also work top-down, and most of them work are interpreted as byte-level chunks. So the format requirements and the collision block insertion can be aligned to make valid format files with specific properties.

Inert 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 different content.

These comment chunks were typically not meant to be actual 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 shorter and the longer version of the chunk, another comment chunk is declared to jump over some content A. After this content A, we then simply append another content B. Since file formats usually define a terminator that make parsers stop when they reach it, A terminates parsing, so that the appended content B is ignored.



Typically, at least two comments are needed: one for block alignment, another to hide collision blocks. A third one may be needed to hide one file's contents, for reusable collisions.

The following common properties of file formats enable the construction of colliding files. These properties are not typically seen as weaknesses, but they can be detected or normalized out, making the attacker's task considerably harder:

- Dummy chunks that can be used as comments.
- Allowing more than one comment.
- Long comments. For example, lengths of 64b for MP4 and 32b for PNG make for trivial collisions, whereas 16b for JPG, 8b for GIF make for no generic collision for GIF, and limited ones for JPG.
- Storage arbitrary binary data in a comment, rather than just text or valid data.
- Allowing arbitrary data after the terminator.
- A lack of integrity checks. For example, CRC32 in PNGs are usually ignored, but would prevent PNG reusable collisions otherwise.
- Flat structure. For example, ASN.1 defines a parent structure with the length of all the enclosed substructures, which prevents these constructs: you'd need to abuse the length, but also the length of the parent. Note, however, that this feature of ASN.1 creates multiple sources of truth for the parsers, and puts the onus of checking that all these pieces of data agree on the parser itself. This is how you get Heartbleed.
- Allowing a comment to precede the header. This makes generic reusable collisions possible.

Now that we have the theory down, let's learn some moves.

Identical Prefix Collisions

Identical prefix files look almost identical. Their content have only a few bits of differences in the collisions blocks. All blocks before the collision are fixed and cannot be changed without recomputing the collision, while all blocks of the suffix are malleable and can altered so long as they stay equal to those in the colliding file.

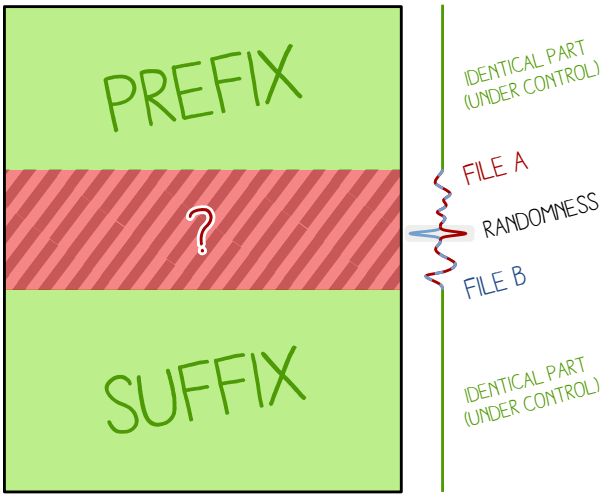
1. Define an arbitrary prefix. Its content and length don't matter.
2. Pad the prefix to the next 64-byte block.
3. Compute and append collision block(s) depending on the prefix. These blocks will look very random, with the specific differences predetermined by the attack.
4. After the block(s), the hash value is the same despite the file differences.
5. Add any arbitrary identical suffix as needed.

```
| Prefix         | = | Prefix         |
| :----:         |:-:| :----:         |
| Collision *A*  |!= | Collision *B*  |
| Suffix         | = | Suffix         |
```

Exploitation There are two classic ways of exploiting identical prefix collisions. The first is the *data exploit*: run code that checks for differences and displays one or the other. (This is typically trivial because differences are known in advance.) The second is the *structure exploit*, which we use a difference in the file structure, such as the length of a comment, to hide one content or show the other.

Here are two files with this structure, collided to show either A or B as determined by a switch in the collision.

Prefix	=	Prefix
Collision *A*	!=	Collision *B*
A	=	~~A~~
~~B~~	=	**B**



FastColl The final version of FastColl is from 2009. Here is its scorecard and a quick print of its difference mask, which describes which nybbles of the block might change and which must remain fixed.

Time:	a few seconds of computation
Space:	two blocks
Differences:	no control before, no control after.
exploitation:	hard

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

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

An example with an empty prefix:

```
   MD5: fe6c446ee3a831ee010f33ac9c1b602c
SHA256: c5dd2ef7c74cd2e80a0fd16f1dd6955c
        626b59def888be734219d48da6b9dbdd
00:  37 75 C1 F1-C4 A7 5A E7-9C E0 DE 7A-5B 10 80 26
10:  02 AB D9 39-C9 6C 5F 02-12 C2 7F DA-CD 0D A3 B0
20:  8C ED FA F3-E1 A3 FD B4-EF 09 E7 FB-B1 C3 99 1D
30:  CD 91 C8 45-E6 6E FD 3D-C7 BB 61 52-3E F4 E0 38
40:  49 11 85 69-EB CC 17 9C-93 4F 40 EB-33 02 AD 20
50:  A4 09 2D FB-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E
60:  39 89 9E F6-79 46 9F E6-8B 85 C5 EF-DE 42 4F 46
70:  C2 78 75 9D-8B 65 F4 50-EA 21 C5 59-18 62 FF 7B

00:  37 75 C1 F1-C4 A7 5A E7-9C E0 DE 7A-5B 10 80 26
10:  02 AB D9 B9-C9 6C 5F 02-12 C2 7F DA-CD 0D A3 B0
20:  8C ED FA F3-E1 A3 FD B4-EF 09 E7 FB-B1 43 9A 1D
30:  CD 91 C8 45-E6 6E FD 3D-C7 BB 61 D2-3E F4 E0 38
40:  49 11 85 69-EB CC 17 9C-93 4F 40 EB-33 02 AD 20
50:  A4 09 2D 7B-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E
60:  39 89 9E F6-79 46 9F E6-8B 85 C5 EF-DE C2 4E 46
70:  C2 78 75 9D-8B 65 F4 50-EA 21 C5 D9-18 62 FF 7B
   MD5: fe6c446ee3a831ee010f33ac9c1b602c
SHA256: e27cf3073c704d0665da42d597d4d201
        31013204eecb6372a5bd60aeddd5d670
```

You will find other examples, with an identical prefix in `fastcoll1.bin` and `fastcoll2.bin`. A variant of this is the single-block MD5 collision, but that takes five weeks of computation![22]

Unicoll This technique was documented in 2012 in Marc Stevens' Ph.D. thesis, "Attacks on Hash Functions and Applications."[23] The implementation from 2017 is on Github.[24]

UniColl lets you control a few bytes in the collision blocks, before and after the first difference. This makes it an identical-prefix collision with some controllable differences, the next best thing to 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 ninth byte. This makes it very useful in exploitation, as you can reason about the collision in your head: the ninth character of that sentence will be replaced with the next one. `0` is replaced by `1`, `a` is replaced by `b`, and so on.

Here are its scorecard and a map of differences.

[22] `https://marc-stevens.nl/research/md5-1block-collision/`
[23] `unzip pocorgtfo19.pdf stevensthesis.pdf`
[24] `git clone https://github.com/cr-marcstevens/hashclash && emacs hashclash/scripts/poc_no.sh`

Time:	a few minutes (depending on the number of bytes you want to control)
Space:	two blocks
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 ten bytes after the difference.

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

An example with $N = 1$ and 20 bytes of set text in the collision blocks:

```
UniColl 1 00:  55 6E 69 43-6F 6C 6C 20-31 20 70 72-65 66 69 78
  Prefix 10:  20 32 30 62-F5 48 34 B9-3B 1C 01 9F-C8 6B E6 44
         20:  FE F6 31 3A-63 DB 99 3E-77 4D C7 5A-6E B0 A6 88
         30:  04 05 FB 39-33 21 64 BF-0D A4 FE E2-A6 9D 83 36
         40:  4B 14 D7 F2-47 53 84 BA-12 2D 4F BB-83 78 6C 70
         50:  C6 EB 21 F2-F6 59 9A 85-14 73 04 DD-57 5F 40 3C
         60:  E1 3F B0 DB-E8 B4 AA B0-D5 56 22 AF-B9 04 26 FC
         70:  9F D2 0C 00-86 C8 ED DE-85 7F 03 7B-05 28 D7 0F

         00:  55 6E 69 43-6F 6C 6C 20-31 21 70 72-65 66 69 78
         10:  20 32 30 62-F5 48 34 B9-3B 1C 01 9F-C8 6B E6 44
         20:  FE F6 31 3A-63 DB 99 3E-77 4D C7 5A-6E B0 A6 88
         30:  04 05 FB 39-33 21 64 BF-0D A4 FE E2-A6 9D 83 36
         40:  4B 14 D7 F2-47 53 84 BA-12 2C 4F BB-83 78 6C 70
         50:  C6 EB 21 F2-F6 59 9A 85-14 73 04 DD-57 5F 40 3C
         60:  E1 3F B0 DB-E8 B4 AA B0-D5 56 22 AF-B9 04 26 FC
         70:  9F D2 0C 00-86 C8 ED DE-85 7F 03 7B-05 28 D7 0F
```

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

It was used in the Google CTF of 2018, where the frequency of a certificate serial changes and limitations on the lengths prevented the use of chosen prefix collisions.[25]

SHAttered (SHA1) Documented in 2013 by Marc Stevens,[26] computed in 2017.[27]

Time:	6500 years CPU and 110 years GPU
Space:	two blocks
Exploitation:	Medium. The differences are right at the start and at the end of the collision blocks. So no control before *and* after a length in the prefix/in the suffix: 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).

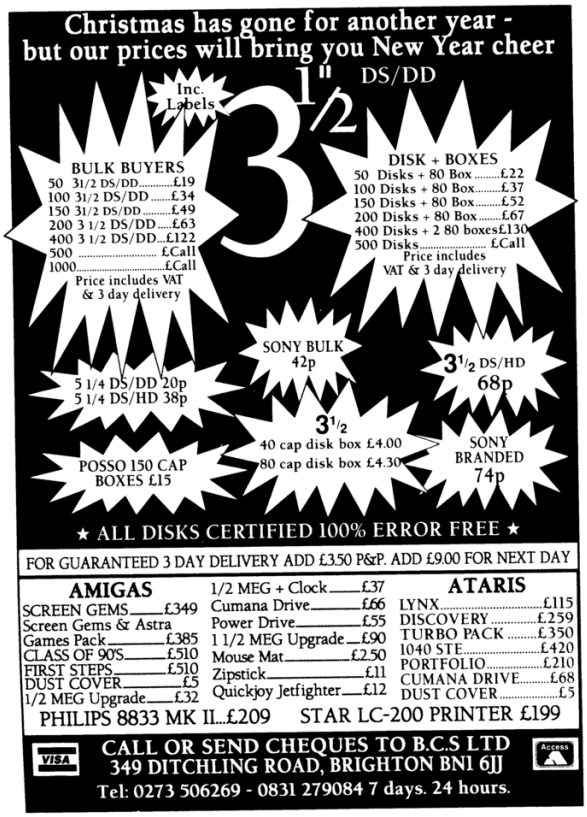
Differences:

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

The difference between collision blocks of each side is this Xor mask, with the practical collision shown in Figure 3.

```
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
```

`pocorgtfo18.pdf` uses the computed SHA1 prefixes, reusing the image directly from PDFLaTeX's source, but also checking the value of the prefixes via JavaScript in the HTML page. The file is a polyglot, valid as a ZIP, HTML, and PDF. (See PoC‖GTFO 18:10.)



[25] https://github.com/google/google-ctf/tree/master/2018/finals/crypto-hrefin
[26] https://marc-stevens.nl/research/papers/EC13-S.pdf
[27] http://shattered.io

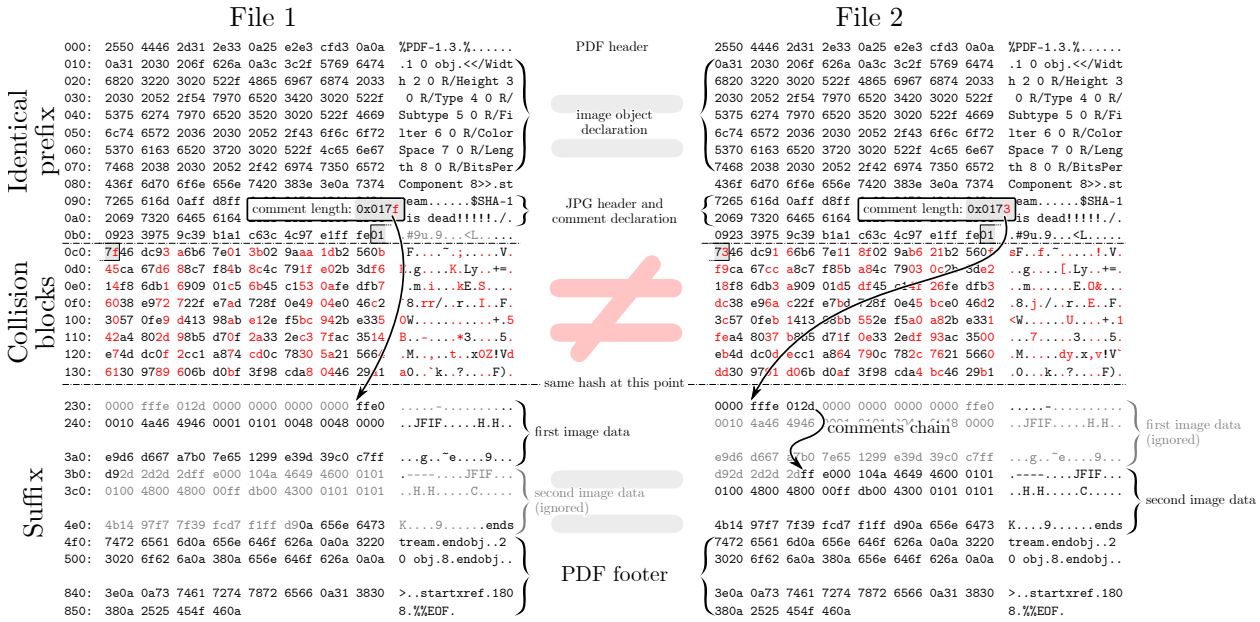
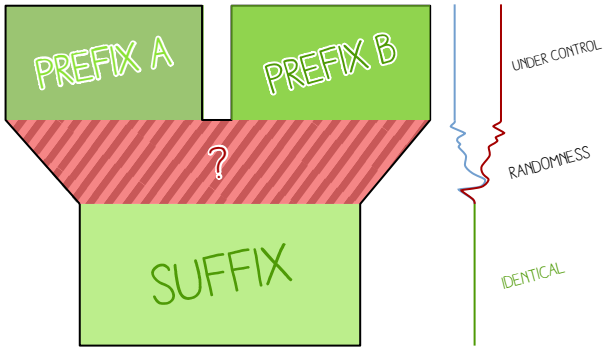


Figure 3. Shattered PoCs

Chosen-Prefix Collisions

Chosen prefix collisions allow us to collide any content, but they don't exist for SHA1 yet.

A	!=	B
Collision *A*	!=	Collision *B*



The steps are to first take two arbitrary prefixes, then to pad the shorter so that their lengths match. Both are then padded to the next block minus twelve bytes, and those twelve bytes are populated at random until a birthday search reveals a collision in the x near-collision blocks appended to the prefixes.

The fewer blocks, the longer the computation will take. While a single block took 400 kHours,[28] nine blocks took just seventy-two with HashClash.[29] Chosen prefix collisions are almighty, but they can take a very long time.

HashClash The final version of this technique appeared in 2009.[30] This collision of "yes" with "no" that is shown in Figure 4 took three hours on twenty-four cores. Note that this is a chosen prefix, and that these files have nothing in common for the first several bytes.

Attacks Summary

Hash	Name	Time	Prefix	Control
MD5	FastColl ('09)	2s	Identical	none
	UniColl ('12)	7–40m	Identical	4–10 bytes
	HashClash ('09)	72h	Chosen	none
SHA1	Shattered ('13)	6500yr	Identical	Prefix & Suffix

[28] `https://www.win.tue.nl/hashclash/SingleBlock/`
[29] `git clone https://github.com/cr-marcstevens/hashclash`
[30] `https://www.win.tue.nl/hashclash/ChosenPrefixCollisions/`

“yes” prefix:
Prefix, padding

```
000:  79 65 73 0A-3D 62 84 11-01 75 D3 4D-EB 80 93 DE
010:  31 C1 D9 30-45 FB BE 1E-71 F0 0A 63-75 A8 30 AA
020:  98 17 CA E3-A2 6B 8E 3D-44 A9 8F F2-0E 67 96 48
030:  97 25 A6 FB-00 00 00 00-49 08 09 33-F0 62 C4 E8
```

Collision blocks start

```
040:  D5 F1 54 CD-CA A1 42 90-7F 9D 3D 9A-67 C4 1B 0F
050:  04 9F 19 E8-92 C3 AA 19-43 31 1A DB-DA 96 01 54
060:  85 B5 9A 88-D8 A5 0E FB-CD 66 9A DA-4F 20 8A AA
070:  BA E3 9C F0-78 31 8F D1-14 5F 3E B9-0F 9F 3E 19
080:  09 9C BB A9-45 89 BA A8-03 E6 C0 31-A0 54 D6 26
090:  3F 80 4C 06-0F C7 D9 19-09 D3 DA 14-FD CB 39 84
0A0:  1F 0D 77 5F-55 AA 7A 07-4C 24 8B 13-0A 54 A2 BC
0B0:  C5 12 7D 4F-E0 5E F2 23-C5 07 61 E4-80 91 B2 13
0C0:  E7 79 07 2A-CF 1B 66 39-8C F0 8E 7E-75 25 22 1D
0D0:  A7 3B 49 4A-32 A4 3A 07-61 26 64 EA-6B 83 A2 8D
0E0:  BE A3 FF BE-4E 71 AE 18-E2 D0 86 4F-20 00 30 26
0F0:  0A 71 DE 1F-40 B4 F4 8F-9C 50 5C 78-DD CD 72 89
100:  BA D1 BF F9-96 80 E3 06-96 F3 B9 7C-77 2D EB 25
110:  1E 56 70 D7-14 1F 55 4D-EC 11 58 59-92 45 E1 33
120:  3E 0E A1 6E-FF D9 90 AD-F6 A0 AD 0E-C6 D6 88 12
130:  B8 74 F2 9E-DD 53 F7 88-19 73 85 39-AA 9B E0 8D
140:  82 BF 9C 5E-58 42 1E 3B-94 CF 5B 54-73 5F A8 4A
150:  FD 5B 64 CF-59 D1 96 74-14 B3 0C AF-11 1C F9 47
160:  C5 7A 2C F7-D5 24 F5 EB-BE 54 3E 12-B0 24 67 3F
170:  01 DD 95 76-8D 0D 58 FB-50 23 70 3A-BD ED BE AC
180:  B8 32 DB AE-E8 DC 3A 83-7A C8 D5 0F-08 90 1D 99
190:  2D 7D 17 34-4E A8 21 98-61 1A 65 DA-FC 9B A4 BA
1A0:  E1 42 2B 86-0C 94 2A F6-D6 A4 81 B5-2B 0B E9 37
1B0:  44 D2 E4 23-14 7C 16 B8-84 90 8B E0-A1 A7 BD 27
1C0:  C7 7E E6 17-1A 93 C5 EE-59 70 91 26-4E 9D C7 7C
1D0:  1D 3D AB F1-B4 F4 F1 D9-86 48 75 77-6E FE 98 84
1E0:  EF 3C 1C C7-16 5A 1F 83-60 EC 5C FE-CA 17 0C 74
1F0:  EB 8E 9D F6-90 A3 CD 08-65 D5 5A 4C-2E C6 BE 54
```

“no” prefix:
Prefix, padding

```
000:  6E 6F 0A E5-5F D0 83 01-9B 4D 55 06-61 AB 88 11
010:  8A FA 4D 34-B3 75 59 46-56 97 EF 6C-4A 07 90 CC
020:  FE 19 D7 CF-6F 92 03 9C-91 AA A5 DA-56 92 C1 04
030:  E6 4C 08 A3-00 00 00 00-8D B6 4E 47-FF AF 7A 3C
```

Collision blocks start

```
040:  D5 F1 54 CD-CA A1 42 90-7F 9D 3D 9A-67 C4 1B 0F
050:  04 9F 19 E8-92 C3 AA 19-43 31 1A DB-DA 96 01 54
060:  85 B5 9A 88-D8 A5 0E FB-CD 66 9A DA-4F 20 8A A9
070:  BA E3 9C F0-78 31 8F D1-14 5F 3E B9-0F 9F 3E 19
080:  09 9C BB A9-45 89 BA A8-03 E6 C0 31-A0 54 D6 26
090:  3F 80 4C 06-0F C7 D9 19-09 D3 DA 14-FD CB 39 84
0A0:  1F 0D 77 5F-55 AA 7A 07-4C 24 8B 13-0A 54 B2 BC
0B0:  C5 12 7D 4F-E0 5E F2 23-C5 07 61 E4-80 91 B2 13
0C0:  E7 79 07 2A-CF 1B 66 39-8C F0 8E 7E-75 25 22 1D
0D0:  A7 3B 49 4A-32 A4 3A 07-61 26 64 EA-6B 83 A2 8D
0E0:  BE A3 FF BE-4E 71 AE 18-E2 D0 86 4F-20 00 30 22
0F0:  0A 71 DE 1F-40 B4 F4 8F-9C 50 5C 78-DD CD 72 89
100:  BA D1 BF F9-96 80 E3 06-96 F3 B9 7C-77 2D EB 25
110:  1E 56 70 D7-14 1F 55 4D-EC 11 58 59-92 45 E1 33
120:  3E 0E A1 6E-FF D9 90 AD-F6 A0 AD 0E-CA D6 88 12
130:  B8 74 F2 9E-DD 53 F7 88-19 73 85 39-AA 9B E0 8D
140:  82 BF 9C 5E-58 42 1E 3B-94 CF 5B 54-73 5F A8 4A
150:  FD 5B 64 CF-59 D1 96 74-14 B3 0C AF-11 1C F9 47
160:  C5 7A 2C F7-D5 24 F5 EB-BE 54 3E 12-70 24 67 3F
170:  01 DD 95 76-8D 0D 58 FB-50 23 70 3A-BD ED BE AC
180:  B8 32 DB AE-E8 DC 3A 83-7A C8 D5 0F-08 90 1D 99
190:  2D 7D 17 34-4E A8 21 98-61 1A 65 DA-FC 9B A4 BA
1A0:  E1 42 2B 86-0C 94 2A F6-D6 A4 81 B5-2B 2B E9 37
1B0:  44 D2 E4 23-14 7C 16 B8-84 90 8B E0-A1 A7 BD 27
1C0:  C7 7E E6 17-1A 93 C5 EE-59 70 91 26-4E 9D C7 7C
1D0:  1D 3D AB F1-B4 F4 F1 D9-86 48 75 77-6E FE 98 84
1E0:  EF 3C 1C C7-16 5A 1F 83-60 EC 5C FE-CA 17 0C 54
1F0:  EB 8E 9D F6-90 A3 CD 08-65 D5 5A 4C-2E C6 BE 54
```

Figure 4. A Chosen Prefix Collision from HashClash

Exploitation

Identical prefix collisions are rather limited, but for all their versatility, chosen prefix collisions are a lot more time consuming to create.

Another approach is to craft reusable prefixes via either identical-prefix attack such as UniColl—or chosen prefix to overcome some limitations—but reuse that prefix pair in combinations with two payloads like a classic identical prefix attack.

Once a good prefix pair has been computed, we can instantly collide two source files. It's just a matter of massaging file data so that it fits both the file format specifications and the precomputed prefix requirements.

JPEG

The *Application* segment should in theory follow just after the *Start of Image* marker, but thankfully this isn't required in practice. The lets us make our collision generic, and the only limitation is the size of the smallest image.

A comment's length is stored in two bytes, limited to 65,536 bytes, which would be something like a 400×400 photo. To jump to another image, its *Entropy Coded Segment* needs to be split to scans which are smaller than this, either by storing the image progressively or by using `jpegtran` to apply custom scan sizes.

So an MD5 collision of two arbitrary JPGs is *instant*, and needs no chosen-prefix collision, just UniColl. See `jpg.py` for a handy script to collide photographs of your two authors to `collision*.jpg`.

PNG with a Comment First

The biggest limitation of PNG is that it uses CRC32 at the end of its chunks, which would prevent the use of collision blocks. But as a happy coincidence, most parsers ignore these checksums and we can as well!

The image meta data (dimensions, color space, *etc.*) are stored in the `IHDR` chunk, which should be right after the signature, before any potential comment. It would mean that we can only precompute collisions of images with the same metadata. However, that chunk can actually be located after a comment block for the vast majority of readers. 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 of four bytes, there's no need to modify the structure of either file. We can simply 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. The lengths will be `00 75` and `01 75`.

So an MD5 collision of two arbitrary PNG images is *instant*, with no prerequisite—no computation, just some minor file changes—and needs no chosen-prefix collision, just UniColl. See `png.py`, which collided these two logos from competing manufacturers.

PNG with IHDR First Most parsers of PNGs happily accept files that start with a chunk other than `IHDR`. However, some readers, notably Safari and Preview—do you known of any others, gentle reader?—do not tolerate it.

In this case, the image header and its properties (dimensions, color space) must be the first, before any collision blocks Both colliding files must then have the same properties.

Conveniently, UniColl is up to the task, and, of course, the computed prefix pair can be reused for any other pair of files with the same properties. The script `pngStd.py` will collide any pair of such files. It launches UniColl if needed to compute the prefix pair.

GIF

The GIF format is tricky for a number of reasons. It stores its metadata 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. Its comment chunk length is encoded by a single byte, so that the length of any comment chunk is capped at 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. This 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, although we can't have a generic prefix, we can at least collide any pair of GIF with 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, as we would in PNG. Nor can we jump over a big structure, as we would in JPG.

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

Yet there is another idea, which works generically with only a few limitations! It was suggested by Marc, and it's brilliant.

Note that the image data also follows the "<length, data>" sequence format. We can abuse this together with the GIF's animation feature. If the two GIFs we want to collide have no animations of their own, we only have to (1) normalize the palette, (2) set the first frame's duration to the maximum, and (3) draft a comment that jumps 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 find the next frame and display it.

So with some minor setup—only a few hundred bytes of overhead—we can sled over any GIF image and work around the 256 bytes limitation. Kudos to Marc for this nifty trick!

In the end, the current GIF limitations for instant MD5 collisions are that (1) it must have no animation, (2) the images must be normalized to a single palette,[31] (3) the images must the same dimensions, and (4) that after eleven minutes, both files will display the same final frame. Here are two MD5-colliding GIFs by KidMoGraph.

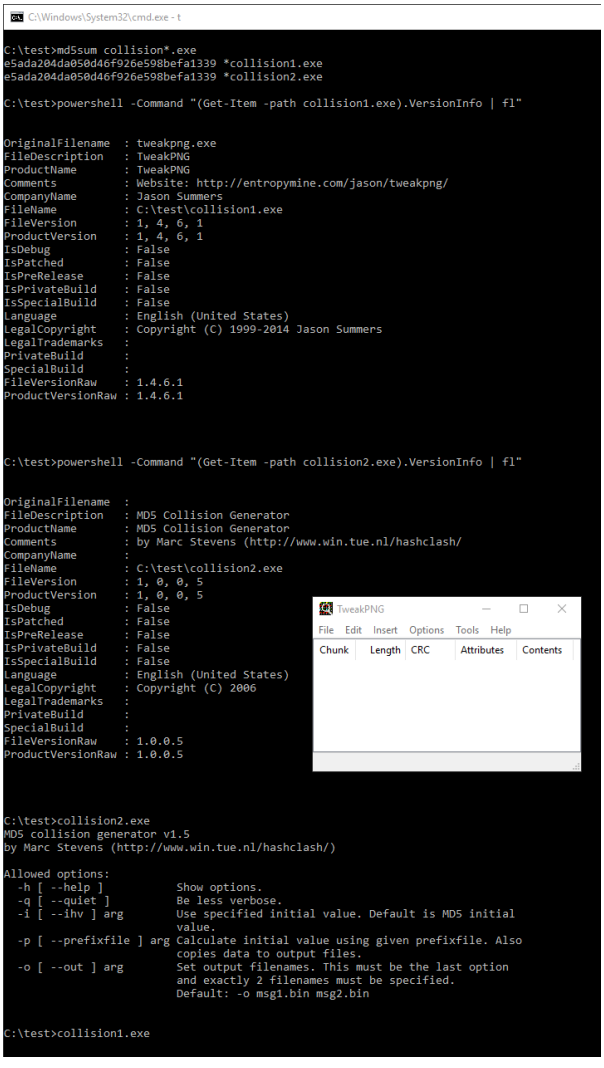
Portable Executable The Portable Executable has a peculiar structure, with a vestigial DOS header that points to a second structure, the PE header. This header must be at offset 0, and it has the fixed length of a full block, ending with a PE header pointer that is beyond UniColl's reach, so only a chosen prefix collision is useful in colliding PE files.

So the strategy is to move the PE header further into the file to leave room for a colliding block after the DOS header, then use chosen prefix collisions to fork a DOS header that points to two different PE offsets, with two different PE headers. These sections can follow each other, so long as you apply a delta to the offsets of the two section tables.

[31] `gifsicle -use-colormap web`

This means that it's possible to instantly collide any pair of PE executables—even if they use different subsystems or architectures! Although executables collisions are typically trivial via any loader, this kind of exploitation is transparent: the code is identical and loaded at the same address.

Attached you will find two colliding PEs: a GUI applicaton `tweakPNG.exe` (as `collision1.exe`) and a CLI application, `fastcoll.exe` (as `collision2.exe`). Windows never allows these two to meet, except in an MD5 collision! The script `pe.py` generates instant collisions of Windows Executables, sharing a hash but running different software.



The curious case of "Runtime R6002 - floating point not loaded" MSVC libraries check sections for permissions. This check can be patched out. Patch the following to set `eax` to 1 instead.[32]

```
1 C1E81F  shr   eax,01F
  F7D0    not   eax
3 83E001  and   eax,1
```

If you apply collisions on packed files, (such as UPX-ed files, to prevent specific PDF keywords like `endstream` from being visible in cleartext), the offsets will change, and this may cause the packer to fail to restore the right attributes. So you may want to patch out that code before UPX-ing the executable and colliding it.

MP4 and Others The MP4 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 Length is `0x0008`, covering an empty value and a four-byte type.

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 64-bit length, changing the atom to "Type Length Value", making it handily compatible with other collisions like SHAttered.[33]

Some atoms contain other atoms: in this case, they're called boxes. That's why this otherwise unnamed structure is called the "Atom/Box."

This Atom/Box format used in MP4 is actually a derivate of Apple's Quicktime, and it is used by many other formats including JP2, HEIF, and F4V. [34] The first atom's type is *usually* `ftyp`, which enables the parsers to differentiate the actual file format.

The format is quite permissive. To make a collision, just chain "free" atoms, abuse one's length with UniColl, then jump over the first payload.

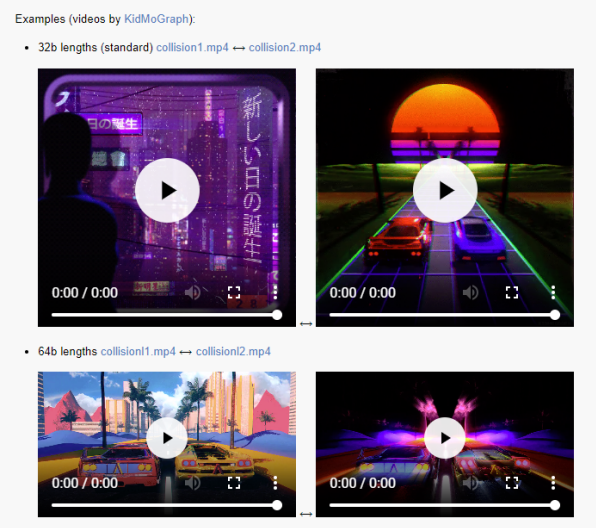
For MP4 files, the only thing to add is to adjust the `stco` (*Sample Table Chunk Offsets*) or the `co64` (its 64-bit equivalent) tables, since they are absolute offsets pointing to the `mdat` movie data. These rules are actually enforced, too!

[32] See the *manhunter.ru* article, "Runtime error r6002 floating point not loaded."

[33] *This, neighbors, is the kind of format cleverness that extracts its costs in bugs, blood, and meathooks. Avoid it when you design your own formats! —PML*

[34] See `http://www.ftyps.com/` for more.

The attached script `mp4.py` will instantly collide arbitrary video. As we already mentioned, it may be portable to other formats than MP4. The examples can be found in `collision1.mp4` and `collision2.mp4`.



Note that some viewers (OS X, Safari, Firefox) don't allow a file that starts with an Atom that is not `ftyp`. In this case, the prefix has to cover this, and it's not so generic. Besides that it's the same strategy as before, only limited to a single fixed file type.

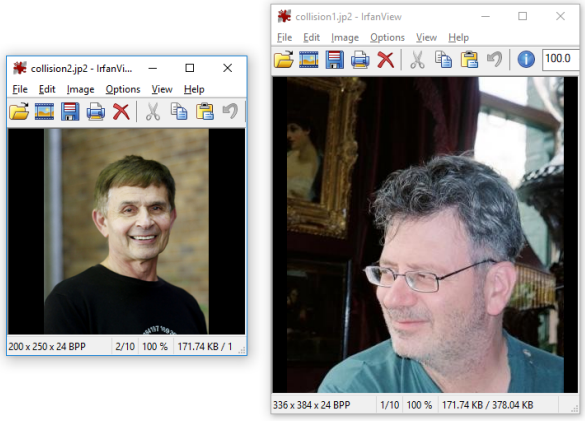
JPEG2000 JPEG2000 files usually start with the Atom/Box structure like MP4, followed by the last atom `jp2c` that typically ends the MP4 file (null length), then from this point on it follows the JFIF structure of a JPEG file (starting with `FF 4F` as a segment marker).

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

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

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

So `jp2.py` is even simpler! Enjoy the colliding JPEG2000 images of Oded Goldreich and Neal Koblitz: while we are all standing on the shoulders of giants, we might as well know their faces. (`collision1.jp2` and `collision2.jp2`)

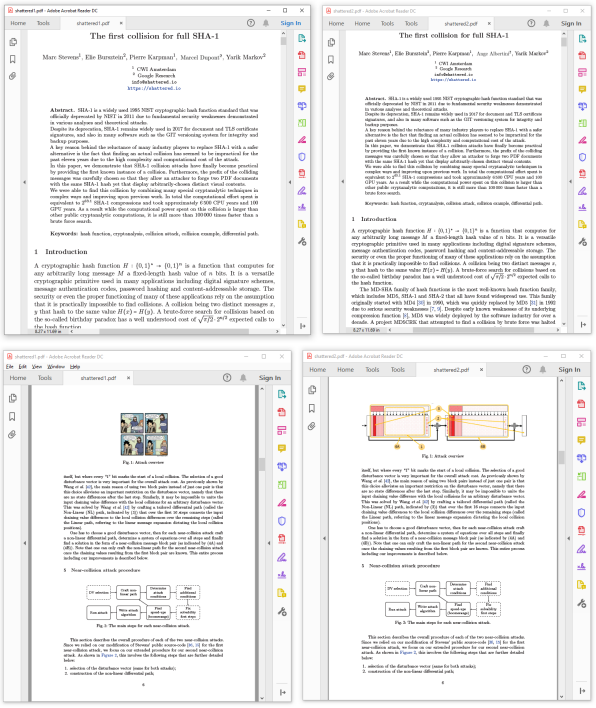


JPEGs in a PDF, as in SHAttered Unless this is your very first issue of this modest journal, neighbors, you probably agree that as a format, PDF is the king of polyglots, and arguably also of syntactic malleability and ambiguity. If however this is your first issue, then do spend a few moments looking up what formats the previous electronic issues doubled as besides being valid (or valid-at-the-time) PDF files—but be warned, it may turn you into a format syntax nerd or make you forever destroy your faith in signature-based security if you still have any.

Yet the SHAttered attack, which produced colliding PDF files of different contents, was not a PDF trick per se, but a JPG trick wrapped in a PDF. The collision of the PDFs is enabled by both of them containing a JPG-compressed object with crafted contents; the PDFs need to be totally identical otherwise.

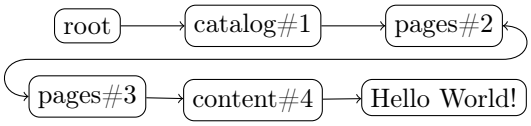
Note that the colliding documents can be totally normal, and can freely use the collision JPG anywhere in their displayed renderings, *e.g.*, on any page of multi-page documents.

The original examples from the SHAttered paper looked as follows, and are included in the examples as `shattered1.pdf` and `shattered2.pdf`.



When native resolution images are required, you can use a nifty trick to make a lossless JPEG! Just repeat each pixel across eight columns and eight rows in a greyscale image, as JPEG blurs across fundamental blocks that are 8×8.

PDF collisions with MD5 We can do MD5 collisions at the document level of PDF, with no restrictions at all on either file! Recall that PDF has a very different structure compared to other file formats, in that it uses object numbers and references to define a tree of objects. The interpretation of the whole document depends on the Root element, but there are many syntactically different tree structures that will be rendered identically.



For example, these two valid PDF files are equivalent to each other.

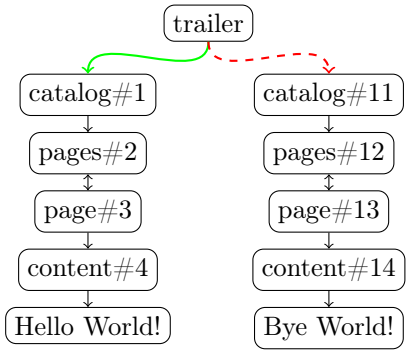
```
%PDF-1.
1 0 obj<</Pages 2 0 R>>endobj
2 0 obj<</Kids[3 0 R]/Count 1>>endobj
3 0 obj<</Parent 2 0 R>>endobj
trailer <</Root 1 0 R>>
```

```
%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>>
```

Some tricks then immediately suggest themselves, as storing unused objects in a PDF is happily tolerated. We can also skip object number, and there's even an official way to skip numbers in the trailing `XREF` table at the end of the document.

So storing two document trees in the same file is okay. We only need to make the root objects of the colliding documents to refer to the desired tree at will. To do this, we just take two documents, renumber their objects and references so that there is no overlap, and craft a collision so that the element number referenced as the Root object can be changed while keeping the same hash value. This trick is a perfect fit for UniColl with $N = 1$, so long as we adjust the `XREF` table accordingly.

This way, we can safely collide any pair of PDFs, no matter what their page numbers, dimensions, images, *etc.* might be.



PDF can store foreign data in two ways, as a *line comment* or as a *stream object*. In a line comment, the only forbidden characters are newlines (`\r` and `\n`). This can be used inside a dictionary object, *e.g.*, to modify an object reference, via UniColl. The following is a valid PDF object even though it contains binary collision blocks—just retry until you have no newline characters.

```
1 1 0 obj
  << /Type /Catalog /MD5_is /
    REALLY_dead_now__ /Pages 2 0 R
3 ...some ugly binary goes here...
  >>
5 endobj
```

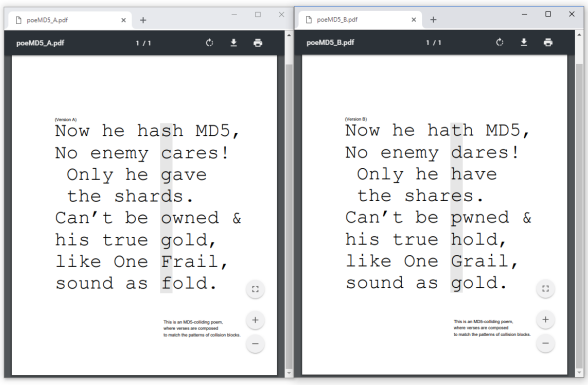
In a stream object, any data is possible, but since we're inside an object, we can't alter the rest of the PDF structure. So we need a Chosen Prefix collision to modify the structure outside the containing stream object.

The first case serves to highlight the beauty of UniColl, a collision where differences are predictable, so that you can write poetry in colliding data—thanks to Jurph![35]

Rather than modifying the structure of the document and fooling parsers, we'll just use collision blocks directly to produce differing texts, with alternate readings!

```
1          V                    V
   Now he hash MD5,     Now he hath MD5,
3  No enemy cares!      No enemy dares!
     Only he gave         Only he have
5    the shards.          the shares.
   Can't be owned &     Can't be pwned &
7  his true gold,       his true hold,
   like One Frail,      like One Grail,
9  sound as fold.       sound as gold.
             ^                    ^
```

You will find these colliding poems in `poeMD5_A.pdf` and `poeMD5_B.pdf`, a true cryptographic artistic creation!



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 the Root object point to different objects. Unlike SHAttered, this means instant collision of any arbitrary pair of PDFs, at document level.

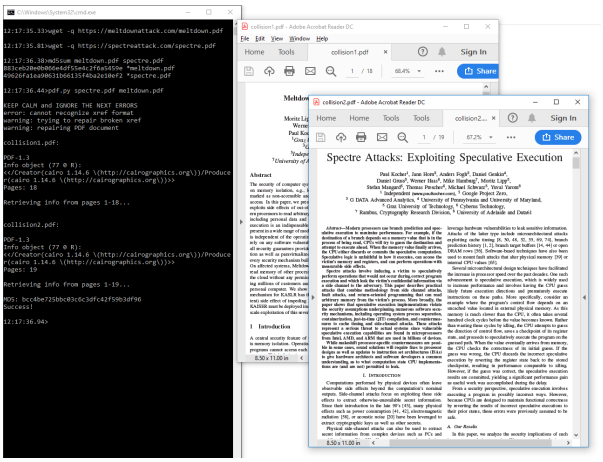
The MuPDF suite provides a useful trick: `mutool clean` output is reliably predictable, so it can be used to normalize PDFs as input and fix your merged PDF while keeping the important parts of the file unmodified. MuTool doesn't discard bogus key/values from PDF dictionaries unless asked, and keeps them in the same order, so using fake dictionary entries such as `/MD5_is /REALLY_dead_now__` is perfect for aligning things predictably without needing another kind of comments. However, `mutool` won't keep comments in dictionaries, so it won't support inline-comment tricks.

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

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

The script `pdf.py` takes less than a second (see `pdf.log`) to collide the two public PDF papers like Spectre and Meltdown (`collision1.pdf` and `collision2.pdf`.)

[35] `unzip pocorgtfo19.pdf word-decrementer.zip || git clone https://github.com/Jurph/word-decrementer`



Here's a 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.[36]

Studebaker

So Simple Anyone Can Run It

No need of a mechanic to take care of the Studebaker "20." No need of a chauffeur. You or your wife can run it as easily as an expert.

Simplicity of operation and control, added to the light running and easy riding qualities of Studebaker cars are the delight of their 75,000 owners.

The Studebaker (Flanders) "20" is equal in quality of material and workmanship to any car made, and its low price and low upkeep cost puts it within your reach.

We know the quality of our cars because every part is made in our own plants and guarantees to us what we guarantee to you. The Studebaker name, too, means service after you buy.

Ready for immediate delivery.

The $800 Studebaker (Flanders) "20"

Price, Standard Equipped, $800 f. o. b. Detroit. Equipped as above, with Top, Windshield, Prest-O-Lite Tank and Speedometer, $885.

Ask our dealer for the new Studebaker art catalogue or send to us for it

The Studebaker Corporation Detroit, Michigan

Bourbon Garage & Supply Co., Agts.

The previous techniques work with any pair of existing PDF files, but even better, you can compile colliding files with PDFLaTeX directly from TeX sources. You will nee PDFTeX's special operators for this.[37]

With these 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
    % (ending at 0xC0) are covered
    /0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    % with an extra char to be replaced by a return
    /0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
     0123456789ABCDEF0
  >>}

  % 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
     ]>>}

  % We 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`. PDFLaTeX has trouble generating reproducible builds across different version and distributions. You might even want to hook the time on execution to get the exact hash, if required.

Uncommon Strategies

Collision attacks are usually about two valid files of the same type with two different contents. However,

[36] See page 81 of Adobe's `PDF32000_2008.pdf`.

[37] `http://texdoc.net/texmf-dist/doc/pdftex/manual/pdftex-a.pdf`

we need not constrain ourselves to this scenario, so let's explore some weirder possibilities.

MultiColls: Multiple Collisions Chain

Nothing prevents us from chaining several collision blocks, and having *more than two* contents with the same hash value. This is the technique behind Hashquines, which show their own MD5 hash. PoC||GTFO 14 contained 609 FastColl collisions, to do just that through two file types in the same file.

Exploiting Ideas of Validity

A different strategy would be to interfere with file type recognition to prevent file scanners from seeing our files as corrupted. Overwriting the file's magic signature may be just enough, so long as both of our files, valid and invalid, get appended with another format that doesn't need to start at offset 0 (*e.g.*, archives such as ZIP, RAR, *etc.*). The scanner would then show 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.

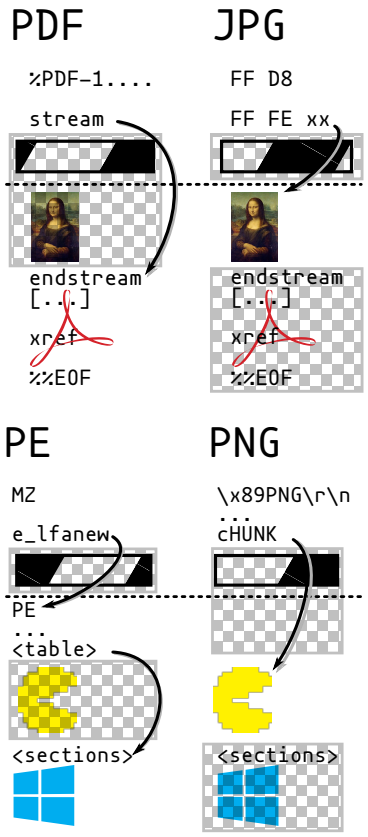
So although both files are technically valid ZIPs, most parsers will see different file types, since they tend to go with the first file type found *and* start scanning at offset 0.

PolyColls: Collisions of Different File Types

Assuming that whitelisting a file by its MD5 checksum takes precedence over other checks, we can use a collision to slip in an executable poison pill that collides with a whitelisted innocent file. For example, if an innocent `feelgood.jpg` gets whitelisted, we can then send an `evil.exe` that has the same MD5 but will be run by some internal system seeing it as cleared executable.

In these cases, a chosen prefix collision is required if both file formats need to start at offset 0.

Here are some examples of such *PolCcoll* layouts, a PDF/JPG collision polyglot and a PE/PNG polyglot.



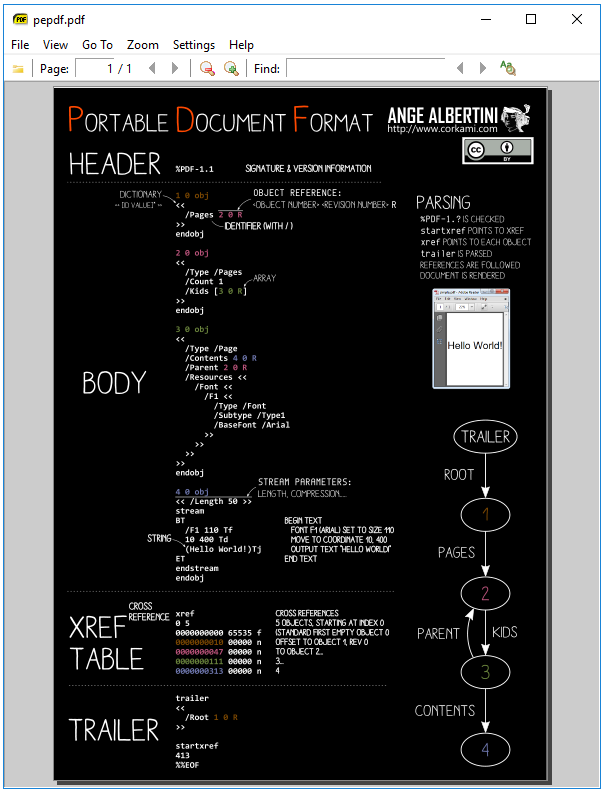
PE/JPEG Since a PE header is usually smaller than `0x500` bytes, it's a perfect fit for a JPG comment. We can begin with DOS/JPEG headers, then create a JPEG comment that jumps over the following PE header. We'll following this with a full JPG image, and then follow through with the rest of the PE specification.

Once again, the collision is instant. See `jpgpe.py` for a practical example that instantly combines `fastcoll.exe` and `marc.jpg`.

PDF/PE Merging a PDF with a dummy file via `mutool` is a good generic way to reorder objects and then get the first two objects discardable (dummy page and content). This is a perfect fit the trick of using a stream object as the PDF file's object with id `1 0` that references its actual length later on (after collision blocks) in the second object. Recall that it's perfectly legal for a stream object in a PDF file to specify its length indirectly, as a reference to another object that happens to contain a value of suitable type for the length.

The only problem is that `mutool` will always compute and inline the length, removing the length reference. This has to be re-inserted into the PDF instead of the computed value. Still, most references to `2 0 R` will be smaller than hardcoded lengths. Thankfully, this can be fixed without altering any object offset, so there's no need to patch the PDF file's XREF table.

The script `pdfpe.py` can, for instance, instantly collide a PDF viewer and a PDF document. See `pepdf.exe` and `pepdf.pdf`, in which a PDF viewer showing a PDF (itself showing a PDF) have the same MD5!



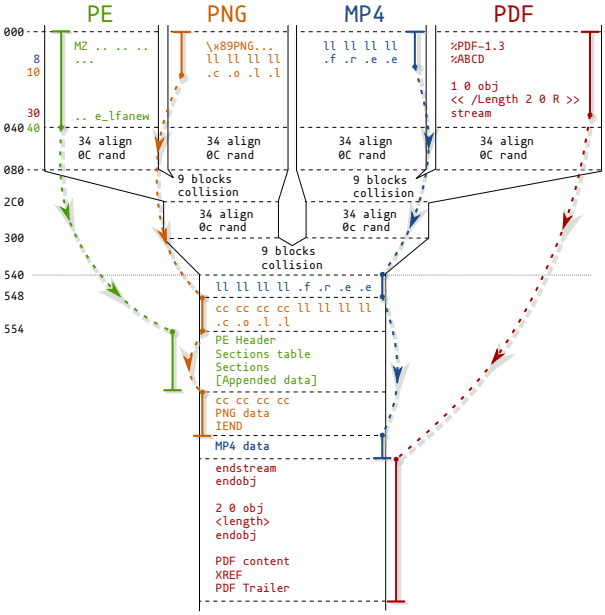
[38] `https://www.win.tue.nl/hashclash/Nostradamus/`

PDF/PNG Similarly, it's possible to collide an arbitrary PDF and PNG files with no restrictions on either side. This is instant, reusable, and generic. Check out `png-pdf.pdf` and `png-pdf.png`.

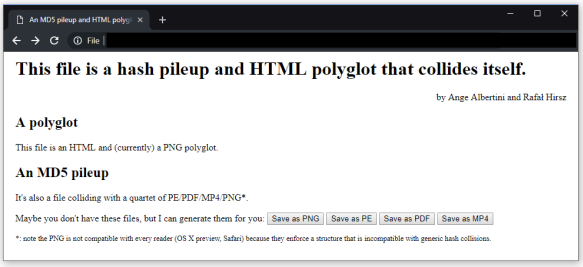
Pileups (Multi-Collisions) But why stop at colliding just two files? Cryptographic collisions are not limited to just two files! As demonstrated by the Nostradamus experiment[38] in 2008, chaining collisions makes it possible to collide more than two files. The first collisions can be either identical or chosen prefix, but all the following ones have to be chosen prefix collisions. You can call them multi-collisions, I prefer to call them *pileups*.

PE/PNG/MP4/PDF Combining all the previously acquired knowledge, I used three chosen prefix collisions to craft four different prefixes for different file types: document (PDF), video (MP4), executable (PE), and image (PNG) to produce this pileup.

This script is generic and instant, and it happily generated `pocorgtfo19.pdf`, `pocorgtfo19.png`, `pocorgtfo19.mp4`, and `pocorgtfo19.exe`.



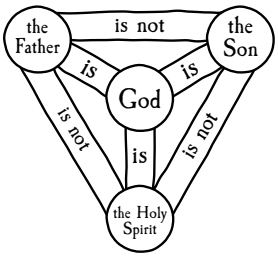
Since you may only distribute a single file and it's impossible to guess the other prefix values from it, a solution is to embed all prefixes of the collision in the JavaScript code and insert it in your PoCs, turning your files into HTML polyglots to easily share the related colliding files. (See `pocorgtfo19.html`.)



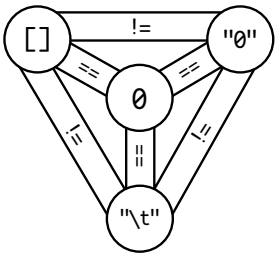
Gotta Collide 'em All! Another use of instant, reusable, and generic collisions would be to hide any file of a given type—say, PNG—behind dummy files or the same file every time. This is easy to do by just concatenating it to the same prefix after stripping the signature; you could even do that at a 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 two files, one of which contains a payload for MS 08-067, and collide them with the same PNG.

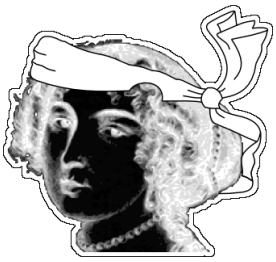


Trinity



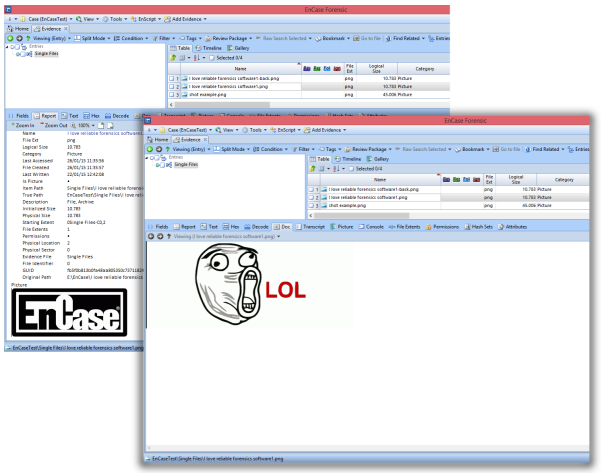
JavaScript

They now show the same dummy image, and they're absolutely identical until the second image at the file level! Their evil payload is now hidden behind identical-looking files with identical MD5 hashes!



Incriminating Files Another evil use case for collisions is to hide something incriminating inside something innocent, but desirable. A forensic evidence collection method that relies on comparing weak hashes would catch the innocent file, and you won't be able to prove that you didn't have the *other* file that shows incriminating content and hides innocent content.

Since forensic software typically focuses on quick parsing, not on detailed file analysis, this scenario is quite unsettlingly realistic. Here is an image showing different previews under different tabs of the EnCase forensic software:



Failures

Not all formats can have generic, reusable prefixes. 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.

ELF The ELF header is required at offset 0 and contains critical information such as whether the binary is 32-bit or 64-bit, its endianness, and its ABI version right at the beginning. This makes it impossible to have a universal prefix that could be followed by crafted collision blocks before these critical parameters that are specific to the original file.

Mach-O Mach-O doesn't even start with the same magic for 32 bits (`0xfeedface`) and 64 bits (`0xfeedfacf`). Soon after, there follow the number and the size of commands such as segment definitions, symtab, version, *etc.* Like ELF, easily reusable collisions are not possible for Mach-O files.

Java Class Files Right after the file magic and the version (which varies just enough to be troublesome), a Java class file contains the constant pool count, which is quite specific to each file. This precludes universal collisions for all files.

However, many files do share a common version and we can pad the shortest constant pool to the longest count. Specifically, we can first insert a *UTF8 literal* to align information, then declare another one with its length abused by the UniColl. This will require code manipulation, since all pool indexes will need to be shifted. Instant MD5 reusable collisions of Java Class should be possible, but they will require code analysis and modification.

TAR Tape Archives are a sequence of concatenated header and file contents, all aligned to 512 byte blocks. There is no central structure to the whole file, so there is no global header or comment of any kind to abuse.

One potential trick might be to start a dummy file of variable length, but the length is always at the same offset, which is not compatible with UniColl. This means that only chosen prefix collisions are practical for collided TAR files.

ZIP There's no generic reusable collision for ZIP either. However, it should be possible to collide two files in two core hours; that is, thirty-six times faster than a chosen prefix collision.

ZIP archives are a sandwich of at least three layers. First comes the files' content, a sequence of *Local File Header* structures, one per archived file or directory, then some index (a sequence of *Central Directory* entries), then a single structure that points to this index (the *End Of Central Directory*). The order of these layers is fixed and cannot be manipulated. Because of this required order, there's no generic prefix that could work for any collision.

However, we can explore some non-generic ways. Some parsers only heed the file content structure. That is not a correct way to parse a ZIP archive, and it can be abused.

Another approach could be to just merge the two archives we'd like to collide, with their merged layers, and to then use UniColl but with $N = 2$, which introduces a difference on the fourth byte, to kill the magic signature of the *End of Central Directory*.

This means one could collide two arbitrary ZIPs with a single UniColl and 24 bytes of a set prefix. In particular, a typical End of Central Directory, which is twenty-two bytes with an empty comment field, looks like this:

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

If we use this as our prefix (padding the prefix to 16 bits) for UniColl and $N = 2$, the difference is on the fourth byte, killing the magic `.P .K 05 06` by changing it predictably to `.P .K 05 86`. This is not generic at all, but it only takes hours, far less than the 72 of a chosen prefix collision.

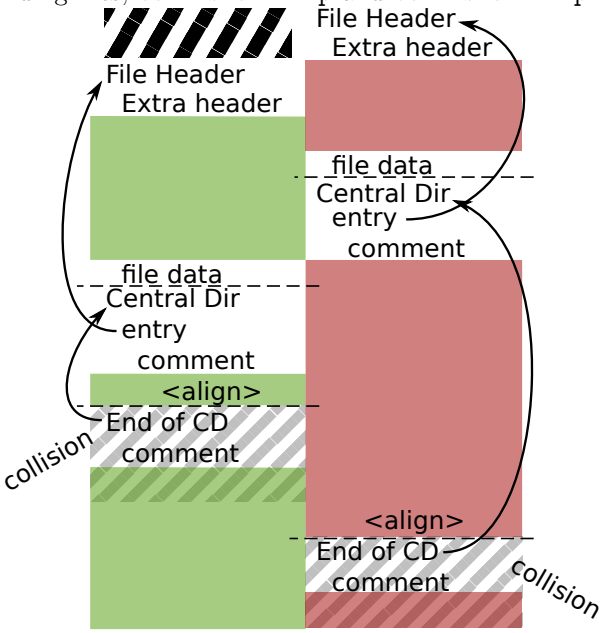
```
00:  504b 0506 0000 0000 0000 0000 0000 0000 PK..............
10:  0000 0000 0000 2121 eb66 cf9d db01 83bb ......!!.f......
20:  2888 4c41 e345 7d07 1634 5d4a 3b61 89a0 (.LA.E}..4]J;a..
30:  0029 94af 4168 2517 0bbc b841 cbf2 9587 .)..Ah%....A....
40:  e438 0043 6390 279d 7c9e a01e e476 4c36 .8.Cc.'.|....vL6
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..(
```

The problem is that some parsers still parse ZIP files from the beginning even though they should be parsed bottom-up. One way to make sure that both files are properly parsed is to chain two UniColl blocks, to enable and disable each *End of Central Directory*.

To prevent ZIP parsers from complaining about unused space, one can abuse *Extra Fields*, the file comments in *Central Directory*, and archive comments in the *End of Central Directory*. See `zip.asm` for the structure of a dual ZIP, which can host two different archive files.

After two UniColl computations, have two colliding files, `collision1.zip` and `collision2.zip`.



Summary

We will end with some handy observations, points which have been made earlier in this paper but might be worth further consideration.

- JPG has some limitations on data, which can be improved to some extent by manipulating the scans encoding.
- PDF with JPG is the initial implementation of the SHAttered attack, but it's simply a pure JPG trick in a PDF document rather than a complex abuse of the PDF structure as such.
- Safari requires PNGs to have their `IHDR` chunk in the first slot, before any collision blocks can be added. Doing so prevents a generic prefix, in which case the collision is limited to specific dimensions, color space, BPP, and interlacing.
- The Atom/Box formats such as MP4 may work with the same prefix for different subformats. Some subformats like JPEG2000 or HEIF require extra grooming, but the exploit strategy is the same—it's just that the collision is not possible between sub-formats, but only with a pair of prefixes for a specific subformat.
- Atom/Box is SHAttered-compatible only when using 64-bit lengths.
- For better compatibility, ZIP needs two UniColls for a complete archive, and these collisions depend on both files' contents.

Thanks to Philippe Teuwen for his extensive feedback on file formats in general, and to Rafał Hirsz for his continuing help with JavaScript.

Format	Generic?	FastColl	UniColl	Shattered	HashClash
PDF	Y		×		×
JPG	Y (1)		×	× (2)	×
PNG	Y/N (3)		×		×
MP4	Y (4)		×	× (5)	×
PE	Y				×
GIF	N	×			×
ZIP	N		× (6)		×
ELF	N				×
TAR	N				×
Mach-O	N				×
Class	N				×