HASH COLLISION EXPLOITATION
WITH FILES

A WORKSHOP BY

ANGE ALBERTINI

WITH THE HELP OF

MARC STEVENS

A.K.A.

LET'S PLAY

COLLTЯIS

Welcome

I'm here to teach, but also to **learn** from you.

This workshop is certainly not covering all perspectives or experience (yet?)

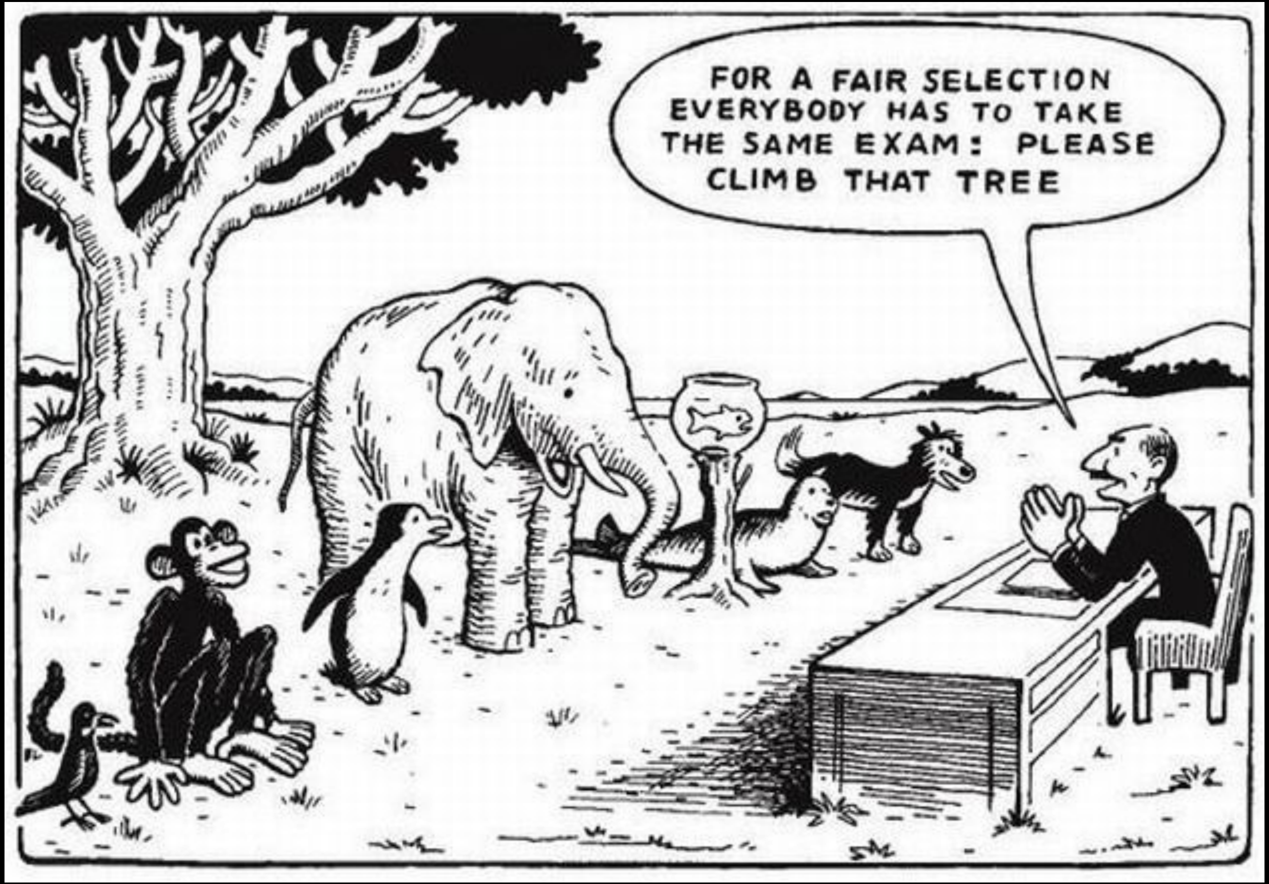
The slides are public and will be improved whenever needed.

Reach me at @angealbertini or ✉️ ange@corkami.com

with remarks, one-liners, suggestions...

Changelog:
- 2018/07/01 first public release - 3h version.

NO GATEKEEPING,
NO DOGMA,
NO CULT.



Everybody is a genius.
But if you judge a fish by its ability to climb a tree,
it will live its whole life believing that it is stupid.

not *Albert Einstein*

Table of Contents

WHAT'S A HASH FUNCTION? MD5, SHA1...

ALSO CALLED 'CHECKSUM'

RETURNS FROM ANY CONTENT A BIG FIXED-SIZE VALUE, ALWAYS DIFFERENT.

␣ → d41d8cd98f00b204e9800998ecf8427e

a → 0cc175b9c0f1b6a831c399e269772661

b → 92eb5ffee6ae2fec3ad71c777531578f

A → 7fc56270e7a70fa81a5935b72eacbe29

IMPOSSIBLE TO GUESS A CONTENT FROM ITS HASH VALUE.

? ← d41d8cd98f00b204e9800998ecf8427**f**

? ← d41d8cd98f00b204e9800998ecf8427**d**

If two contents have the same hash, they are (assumed to be) identical (if the hash is secure)

Hashes are used:

- to check passwords (compute input hash, compare with stored value)

  Confidential - do not share → `a59250af3300a8050106a67498a930f7`
  `p4ssw0rd` → `2a9d119df47ff993b662a8ef36f9ea20`

- to validate content integrity

  Downloading VLC 3.0.6 for Windows

  Thanks! Your download will start in few seconds...
  If not, click here. SHA-256 checksum: e75697cae485a9206a416aaa3b3eb18c9010056d1fcb53e3658be086c7080724

- to index files (ex: your pictures in the cloud)

...Unless there is a hash collision:

two different contents with the same hash result.

```
$ python
[...]
>>> crypt.crypt("5dUD&66", salt="br")
'brokenOz4KxMc'
>>> crypt.crypt("O!>',%$", salt="br")
'brokenOz4KxMc'
>>> crypt.crypt("O!>',%$", "br") == crypt.crypt("5dUD&66", "br")
True
>>>
```

This example uses the crypt(3) hash.

What's the extent of a hash collision?

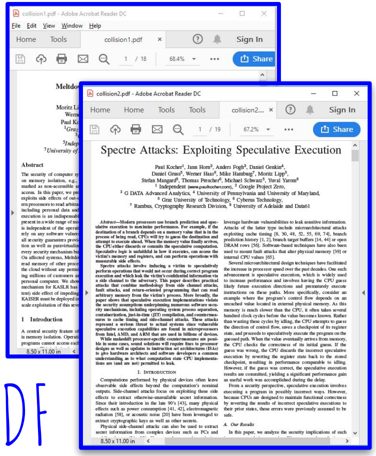
It's **impossible** to generate a file with predetermined hash
with MD5 or SHA1.

We can only generate two (or more) **different** files
that have the same hash.

With some file types, we can **instantly** generate files
that render the same way (via some tricks).

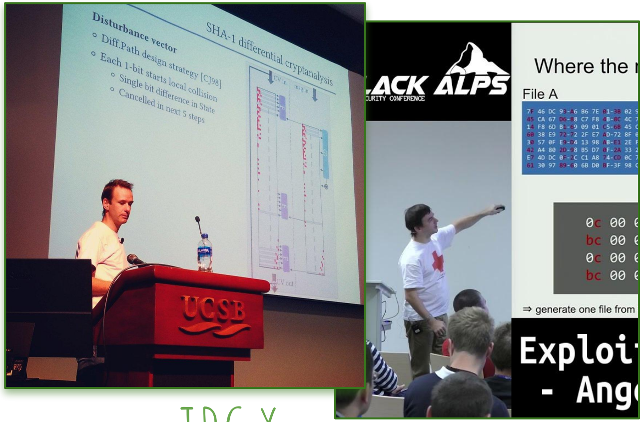
Results - 1/2

Instant MD5 collisions, with no recomputation

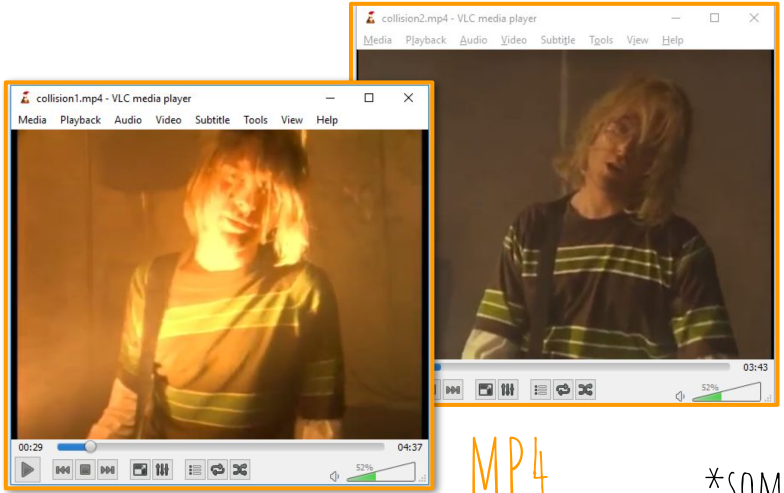


PDF

PNG*



JPG*



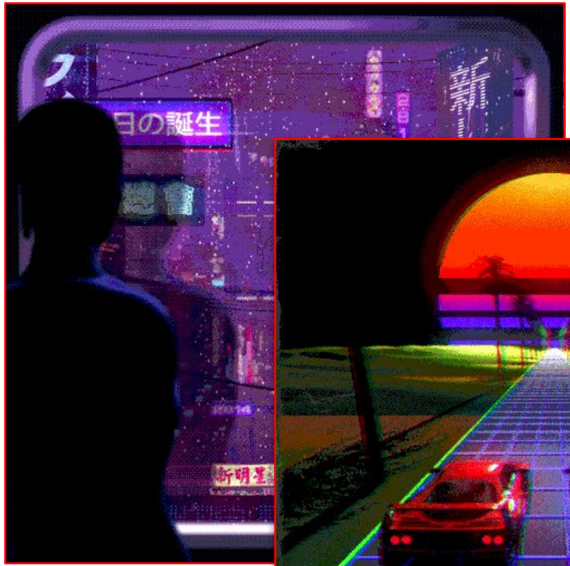
MP4

https://github.com/corkami/collisions

*some limitations

*SOME LIMITATIONS

RESULTS - 2/2



GIF*

PE

JP2

Just new collisions?

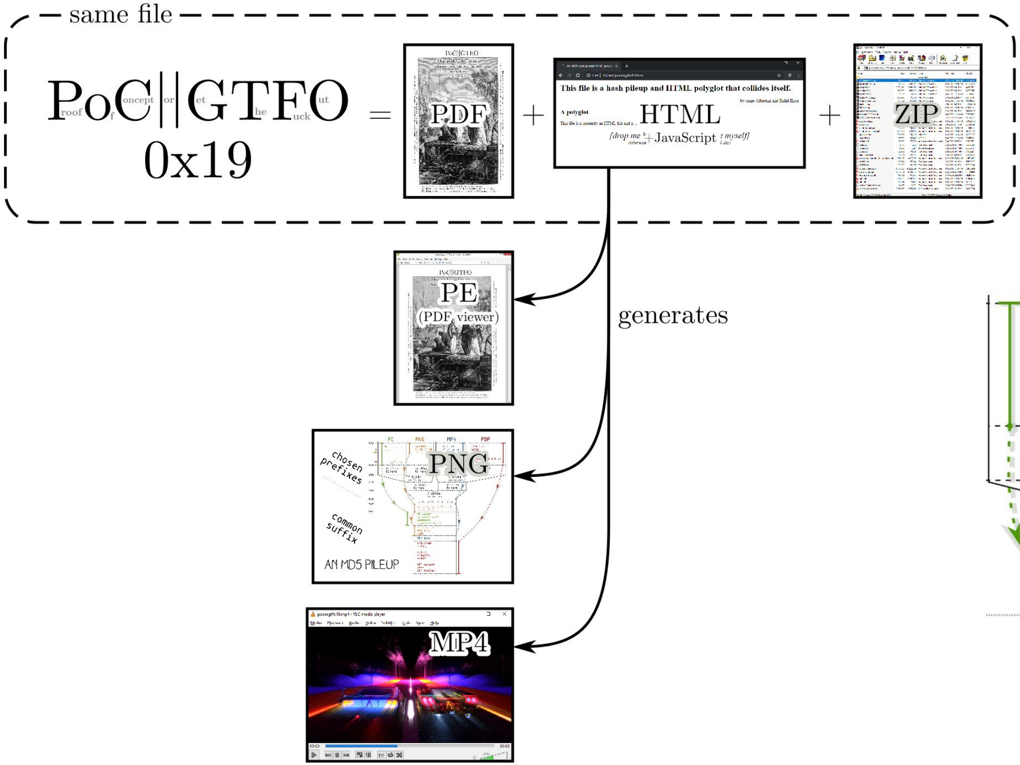
Instant, reusable and generic collisions:

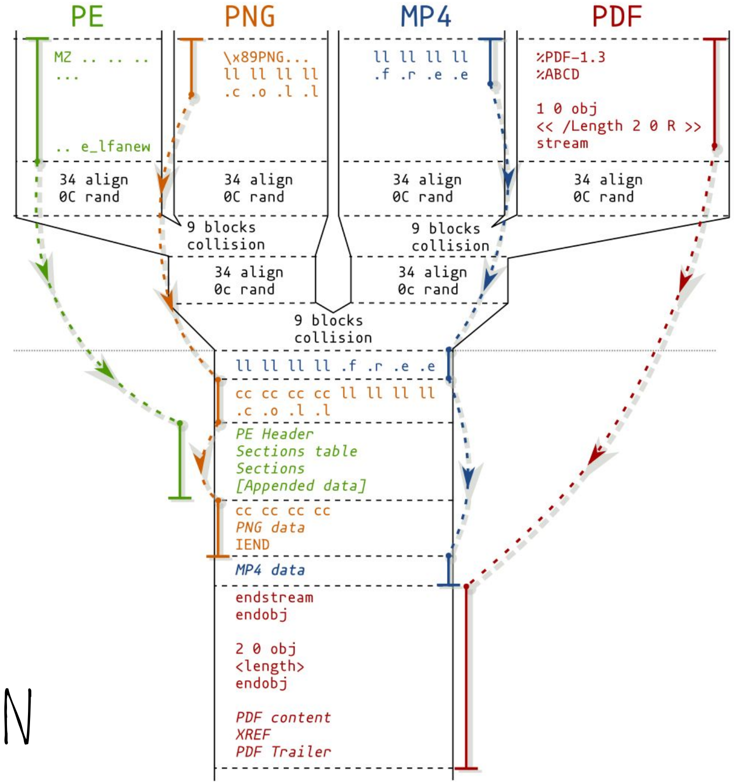
- Take any pair of files, run script, get colliding files.

Extreme case: the colliding PDFs are 100% standard.

From a parser perspective,

- the contents are unmodified: only the files' structures are.





Taking collisions to extremes:

Instant & generic PDF/PE/PNG/MP4 collision

Don't be fooled: shortcuts are necessary

Instant & generic collisions rely on attacks **and** file formats tricks.

Some formats have no suitable tricks.

-> no generic collisions for ELF, Mach-O, ZIP, TAR, Class.

These tricks will be re-usable with **future** collision attacks:

the same JPEG trick was re-used with 3 hash collisions (MD5, MalSHA1, SHA1)

GOALS

Goals of this workshop

Their impacts and limits,
without all the internal details

- Understand hash collisions attacks and their exploits
- Create your own exploits

Maybe you've heard of...

BEST
CRYPTO ATTACK

BEST
CRYPTO17 PAPER

INTERNATIONAL ASSOCIATION FOR CRYPTOLOGIC RESEARCH

THE SHATTERED ATTACK

A computation of the attack documented in Stevens13
using a JPG in a PDF exploit

WHAT THIS SLIDE DECK IS ABOUT

SHATTERED

Uses of hashes

- CHECK IF CONTENT HAS CHANGED: ✓ DO NOTHING ✗ REFRESH FILE [IF NEWER...]
- PROVIDE RANDOMIZATION: ✓ USER ID ✗ CRYPTO KEY

- MATCH A FILE TO A FILE/SET (WHITE/BLACKLISTING, INDEXING)

    ✗ IF THE SET IS USER-CONTROLLED

Use cases

A system uses MD5 to index/check integrity. Is it safe?

Collide a normal* file with a malicious one.

You can even do it on the fly!

To get instant collisions, a few hours of research and a few hours of computation is usually enough.

*rendering-wise, not structure-wise.

Having troubles to convince?

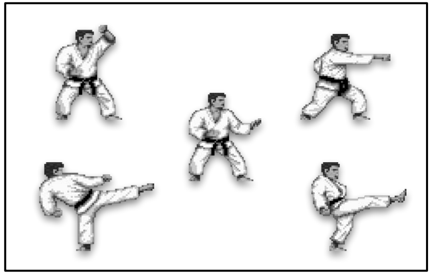
Let files do the talking.

Threats? theory...

Exploits PoCs? reality!

immediate threat

Theoretical attacks to put in practice



MD5 collisions: a good exercise

Hacking a file format == reading + manipulating + abusing parsers

Crafting a re-usable collision requires all these skills, and leaves an undeniable proof.

A re-usable MD5 collision is a good & impactful exercise:

If the collision is instant, the files work and have the same MD5,

it sets in stone your knowledge of that file format,

and you have a proof (of concept).

https://www.cwi.nl/system/files/PhD-Thesis-Marc-Stevens-Attacks-on-Hash-Functions-and-Applications.pdf

ATTACKS ON HASH FUNCTIONS

DON'T BE SCARED…

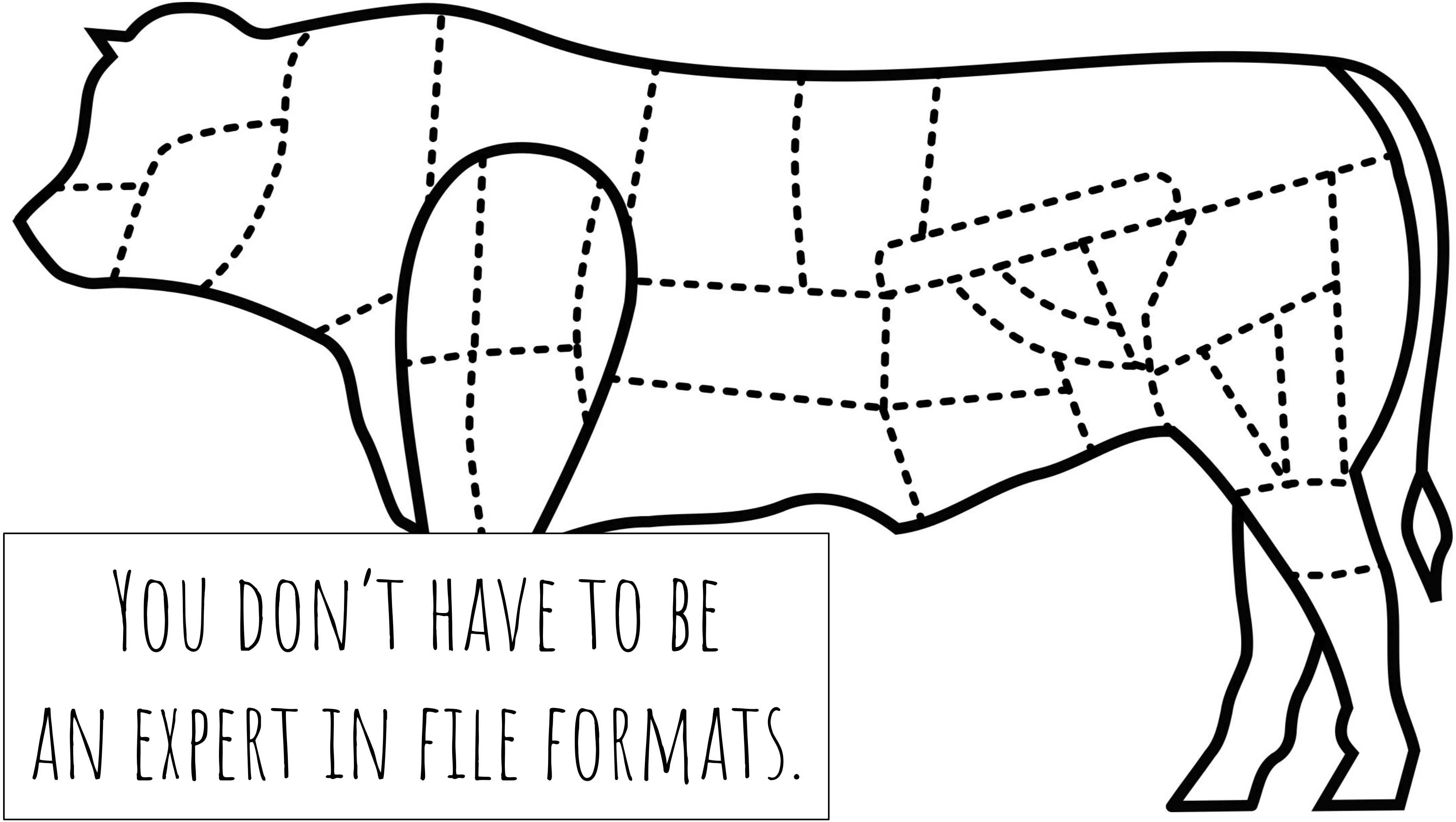
TALES FROM THE CRYPTO

You **don't** need to understand crypto$_{\text{graphy}}$ or maths...

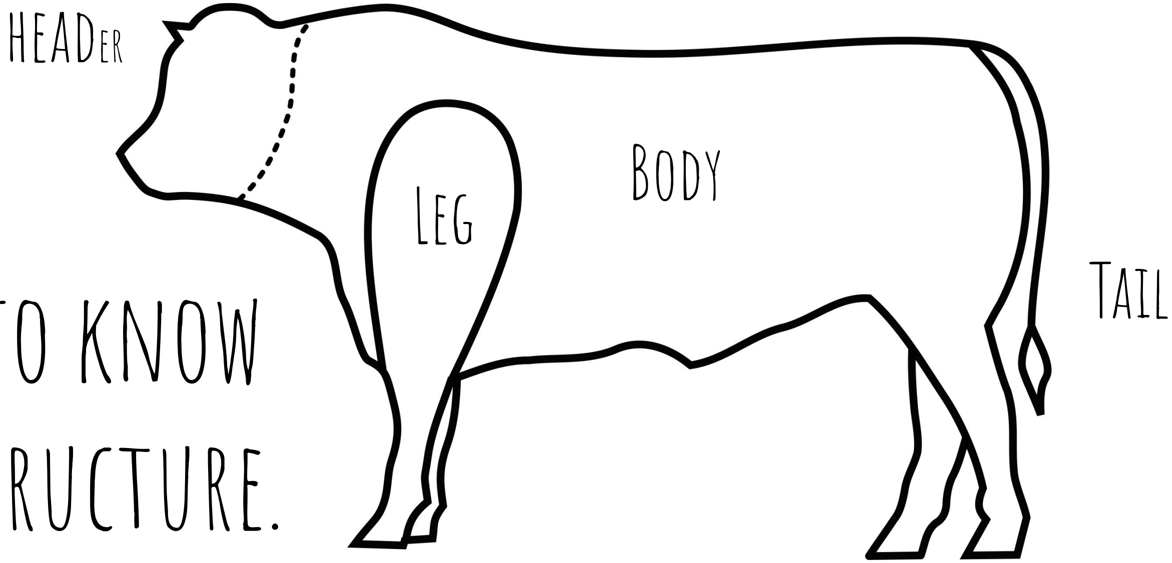
(to be honest, I don't either)

We'll just use existing attacks:

FastColl, UniColl, HashClash, Shattered - yes, that's all!



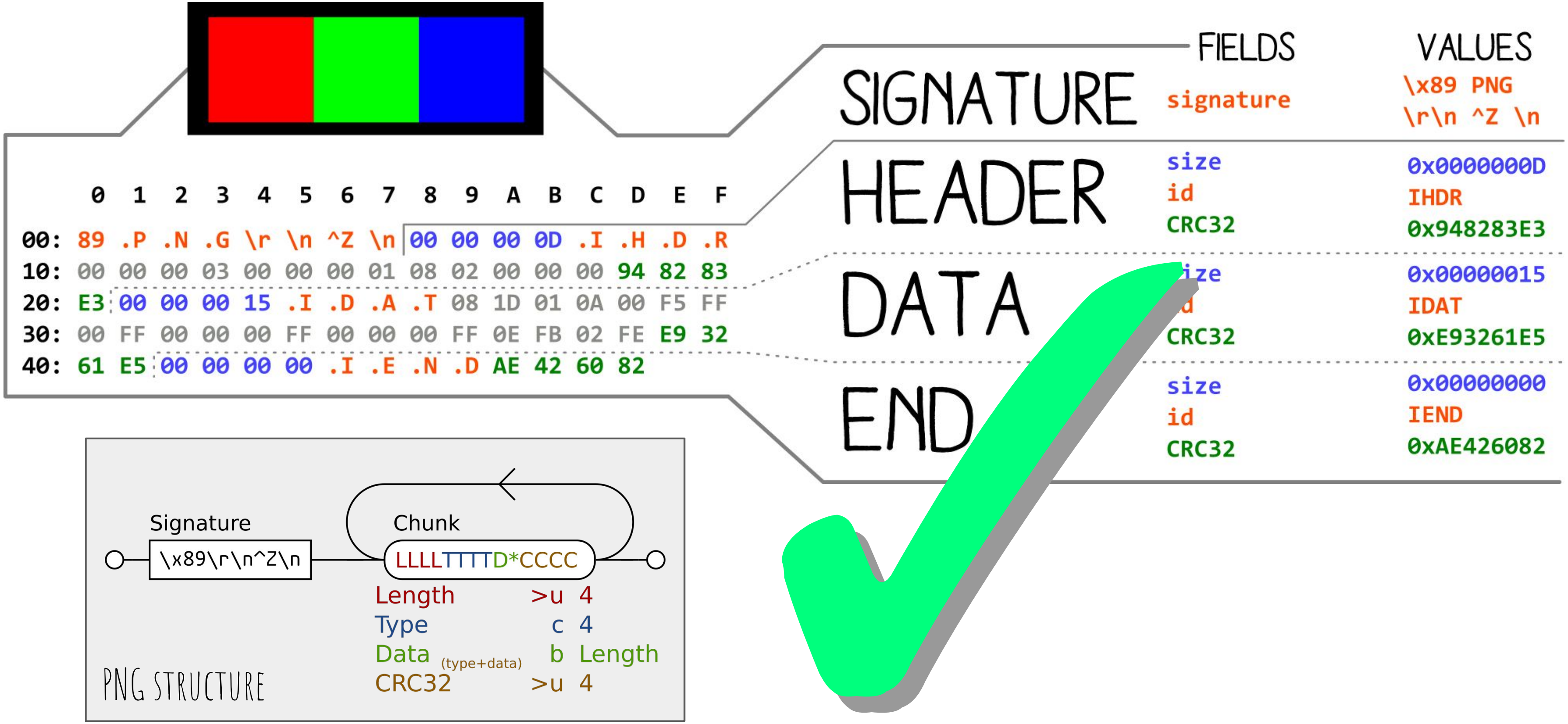
You don't have to be
an expert in file formats.



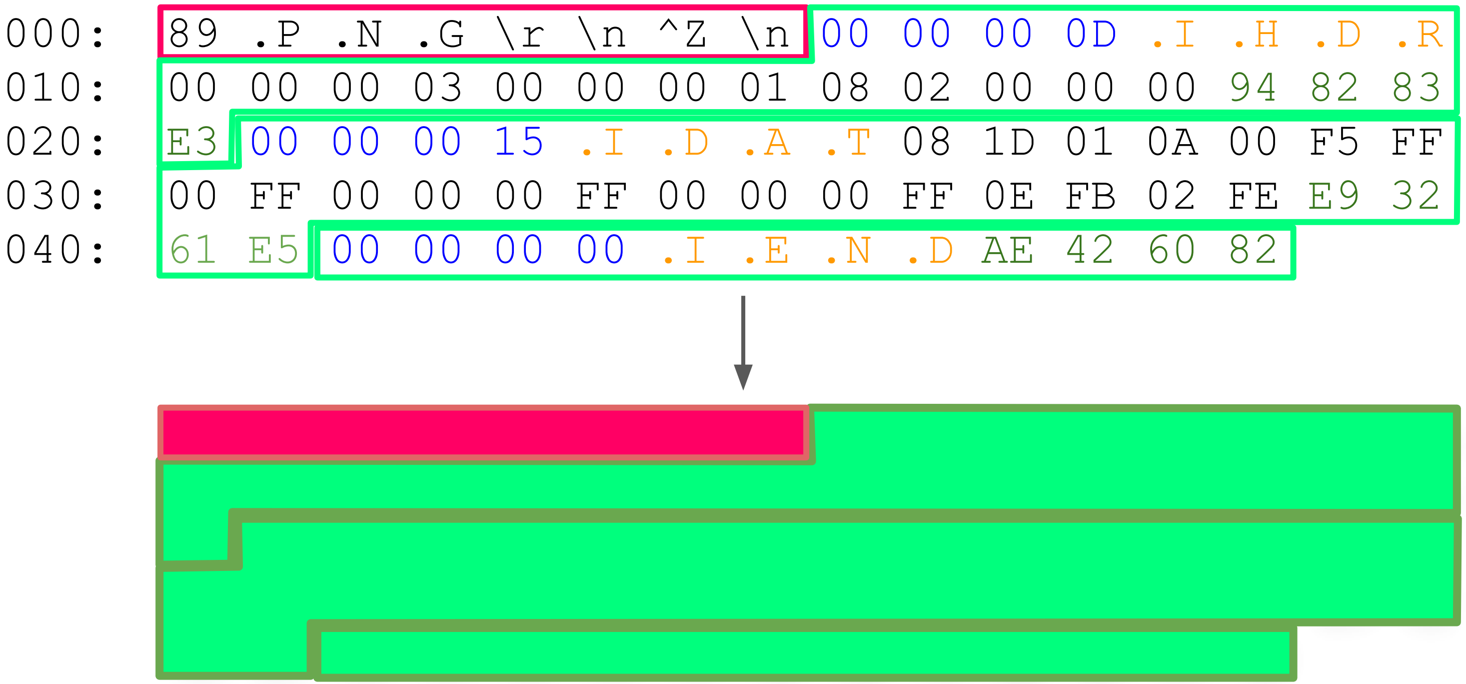
YOU JUST NEED TO KNOW THEIR OVERALL STRUCTURE.

(LESS COMPLEX THAN SOME LEGO MODELS)

EVEN THIS IS TOO MUCH!



YOU ONLY NEED TO UNDERSTAND THE HIGH LEVEL STRUCTURE (NOT THE WHOLE THING)



AND WE'LL IGNORE MOST CONTENTS, SO WE'LL JUST THINK IN BLOCKS.

To be honest

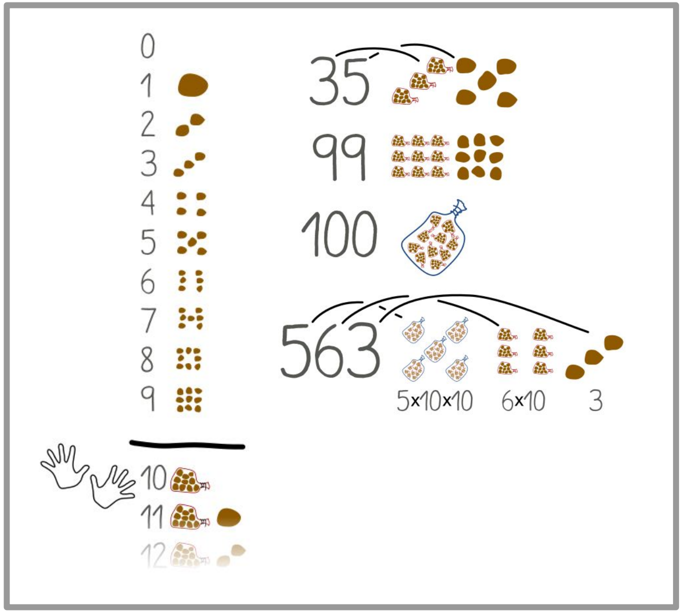
exploiting hash collisions

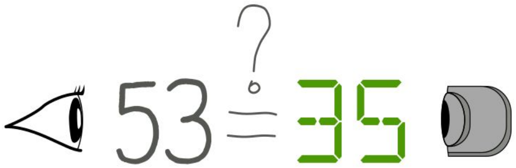
feels a bit like...


...PLAYING A PUZZLE GAME!
COLLTЯIS
Ange Albertini

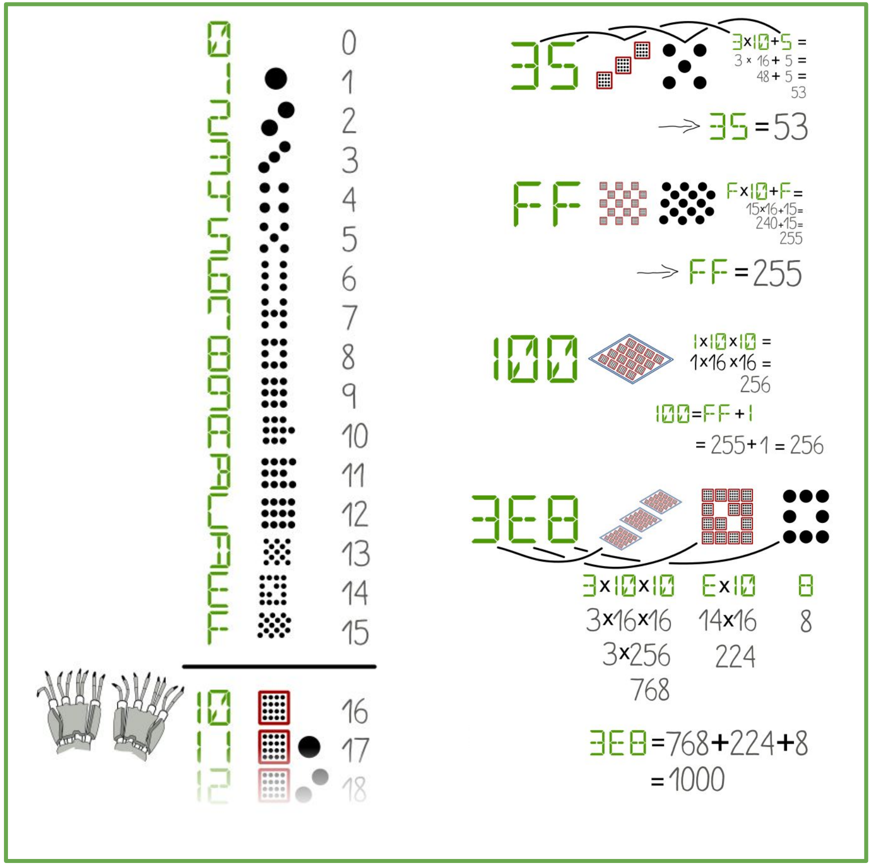
You just need to know
the rules of each block!

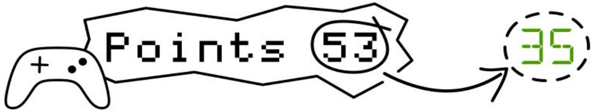

BASICS





YOU KNOW HEXADECIMAL?





YOU KNOW ASCII?

A STANDARD ENCODING:

CHARACTERS <=> VALUES

"A" <=> 0x41 = 65

"Z" <=> 0x5A = 90

"a" <=> 0x61 = 97

Hexadecimal

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F	
2-	SPACE	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	2-
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	3-
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	4-
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	5-
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	6-
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		7-
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F	

Decimal

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	
3-			SPACE	!	"	#	$	%	&	'	3-
4-	(	)	*	+	,	-	.	/	0	1	4-
5-	2	3	4	5	6	7	8	9	:	;	5-
6-	<	=	>	?	@	A	B	C	D	E	6-
7-	F	G	H	I	J	K	L	M	N	O	7-
8-	P	Q	R	S	T	U	V	W	X	Y	8-
9-	Z	[	\	]	^	_	`	a	b	c	9-
10-	d	e	f	g	h	i	j	k	l	m	10-
11-	n	o	p	q	r	s	t	u	v	w	11-
12-	x	y	z	{	|	}	~				12-
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	

You know endianness?

>

Big digits first

01 23 → 0x0123 = 291

Memory addresses

<

Little digits first

01 23 → 0x2301 = 8961

Memory addresses

Formats

PNG, JPG, MP4, Class

TIFF

ZIP, BMP, GZip

Executables: ARM (default), x86, x64

TIFF

Exists in both endianness

Start in the
top-left corner

+1

+0x10

```
000:  89 .P .N .G \r \n ^Z \n 00 00 00 0D .I .H .D .R
010:  00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
020:  E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
030:  00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
040:  61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

OFFSETS

CONTENTS

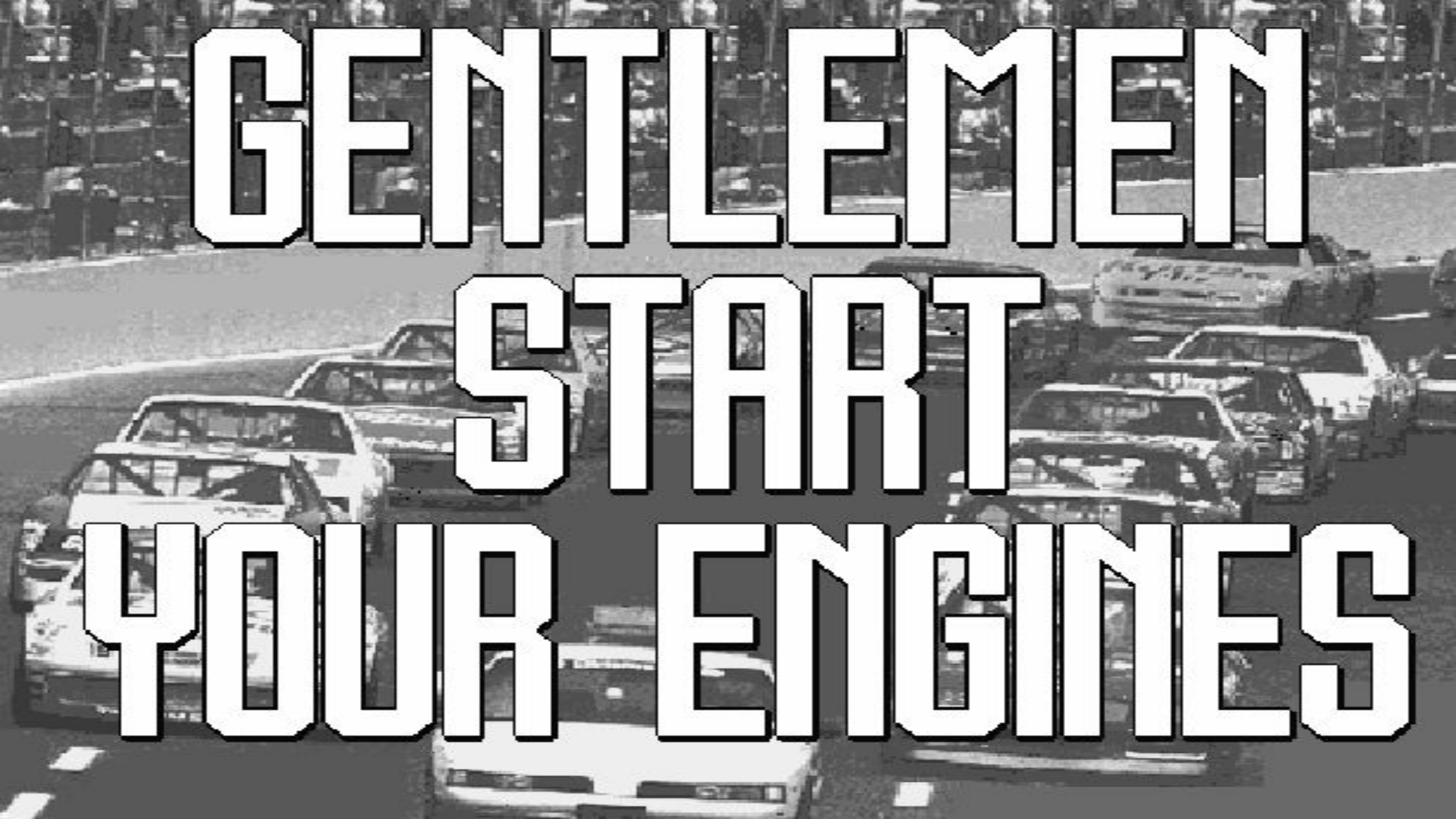
KAITAI
OKTETA
HEX FIEND
HXD
HIEW...

YOU KNOW HEXADECIMAL VIEWERS?

What are hash collisions in practice?

A computation that generates
    two distinct contents with the same hash.
We can set the **start** of these contents - we'll see why.

A hash collision generates a lot of randomness!
-> the final hash is **not** known in advance.


GENTLEMEN
START
YOUR ENGINES

1. HashClash installed

https://github.com/cr-marcstevens/hashclash

Cuda is **not** required

download source and compile

or

Download release binaries

2. A file format manipulation environment

Hex editor, assembly, scripting...

Whatever rocks your boat and you're familiar with.

3. A copy of Corkami/collisions

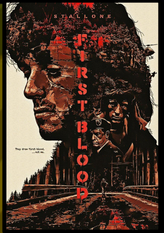
(recommended)

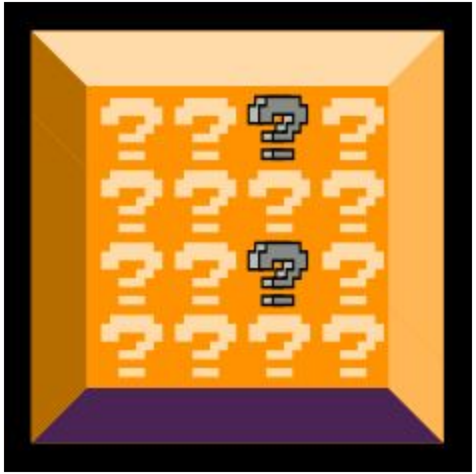
https://github.com/corkami/collisions

GET READY

Prerequisites

OUR FIRST COLLISION





The first block in our game:
An Identical Prefix Collision - FastColl

```
$ fastcoll -p empty
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'empty'
Using initial value: 0123456789abcdeffedcba9876543210

Generating first block: .
Generating second block: W.....
Running time: 0.343 s

$ _
```

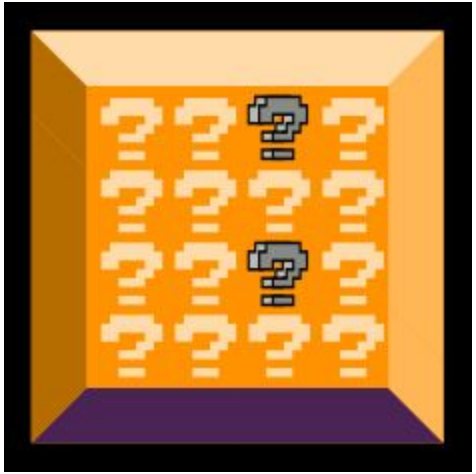
FROM NOTHING, GENERATE 2 FILES WITH THE SAME MD5.

Mission

```
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◄àiδ╠↨£ôO@δ3☻¡
50:  A4 09 2D FB-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E  ño-√§· ↔╤█↨═▌)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 {
```

```
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◄àiδ╠↨£ôO@δ3☻¡
50:  A4 09 2D 7B-15 FA 20 1D-D1 DB 17 CD-DD 29 59 1E  ño-{§· ↔╤█↨═▌)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 {
```

- TWO BLOCKS OF 64 BYTES
- TOTALLY RANDOM
- A FEW TINY DIFFERENCES



OUR FIRST HASH COLLISION

(YOUR COMPUTATION WILL BE DIFFERENT)

```
$ fastcoll -p empty
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'empty'
Using initial value: 0123456789abcdeffedcba9876543210

Generating first block: ........................
Generating second block: S10...............
Running time: 13.35 s

$ _
```

Mission

TRY AGAIN IN THE SAME CONDITIONS -> DIFFERENT COMPUTATION TIME.

```
00:  1D 92 56 C9-34 F6 C6 F2-C9 0C 97 90-AA 16 55 2A  ↔ÆV╔4÷╞≥╔♀ùÉ¬▬U*
10:  68 00 E7 44-8C 56 39 E8-47 A6 80 A6-4D B0 2B F2  h τDîV9ΦGªÇªM░+≥
20:  F6 12 D2 E6-D0 AC 13 2D-EF FF F0 DC-13 10 DE 72  ÷↕╥µ╨¼‼-∩ ≡▄‼►▐r
30:  32 99 B0 BB-C7 65 A6 66-73 10 56 FC-9C 5F 45 8B  2Ö░╗╟eªfs►Vⁿ£_Eï
40:  61 76 C9 56-3E DF 7E 28-DB AB DC 64-B4 9A 44 00  av╔V>▀~(█½▄d┤ÜD
50:  D3 4D BC 1E-80 1C B2 38-C9 B3 40 67-1A 60 A8 C6  ╙M╝▲Ç∟▓8╔│@g→`¿╞
60:  D3 BB 48 08-AF 04 30 16-B8 01 10 5B-92 14 F9 1C  ╙╗H◘»♦0▬╕☺►[Æ¶·∟
70:  3D 3C C6 AC-FF 2C FD AD-DB 2C 2C 4F-C1 06 9B 50  =<╞¼ ,²¡█,,O┴♠¢P
```

```
00:  1D 92 56 C9-34 F6 C6 F2-C9 0C 97 90-AA 16 55 2A  ↔ÆV╔4÷╞≥╔♀ùÉ¬▬U*
10:  68 00 E7 C4-8C 56 39 E8-47 A6 80 A6-4D B0 2B F2  h τ─îV9ΦGªÇªM░+≥
20:  F6 12 D2 E6-D0 AC 13 2D-EF FF F0 DC-13 90 DD 72  ÷↕╥µ╨¼‼-∩ ≡▄‼É▌r
30:  32 99 B0 BB-C7 65 A6 66-73 10 56 7C-9C 5F 45 8B  2Ö░╗╟eªfs►V|£_Eï
40:  61 76 C9 56-3E DF 7E 28-DB AB DC 64-B4 9A 44 00  av╔V>▀~(█½▄d┤ÜD
50:  D3 4D BC 9E-80 1C B2 38-C9 B3 40 67-1A 60 A8 C6  ╙M╝₧Ç∟▓8╔│@g→`¿╞
60:  D3 BB 48 08-AF 04 30 16-B8 01 10 5B-92 94 F9 1C  ╙╗H◘»♦0▬╕☺►[Æö·∟
70:  3D 3C C6 AC-FF 2C FD AD-DB 2C 2C CF-C1 06 9B 50  =<╞¼ ,²¡█,,╧┴♠¢P
```

- COMPLETELY DIFFERENT
- STILL RANDOM-LOOKING
-> LET'S IGNORE THE ASCII!

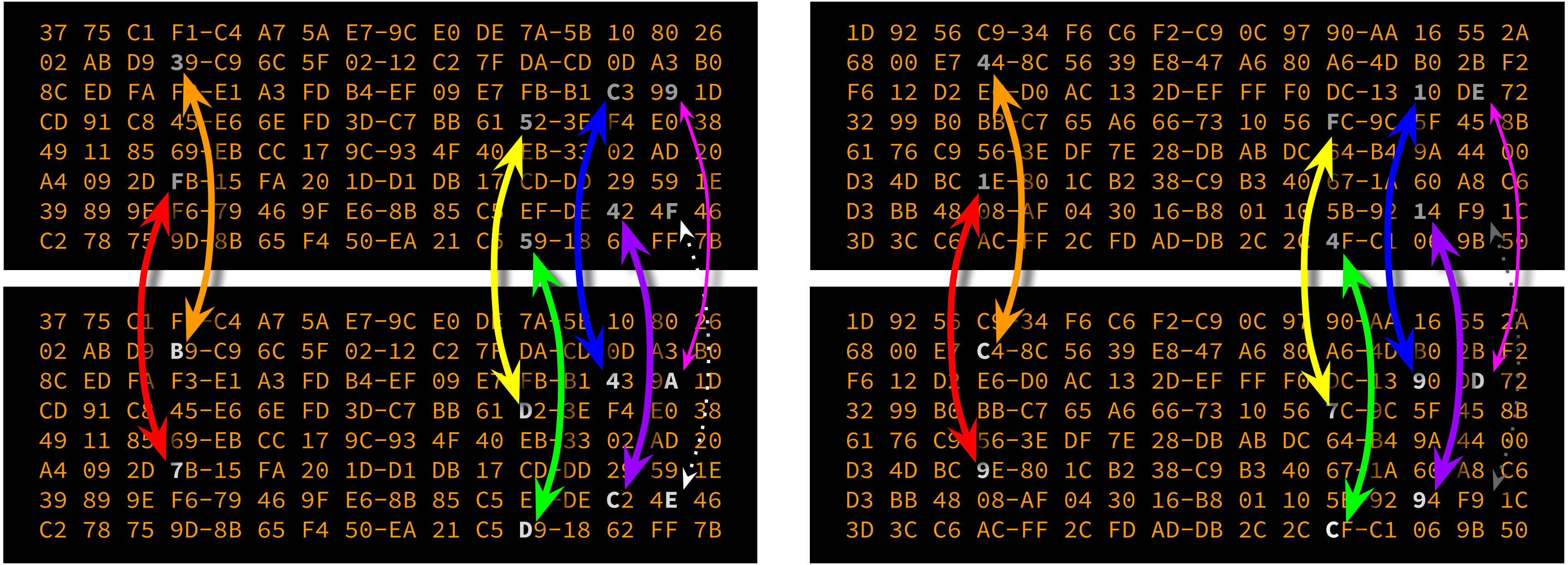
- DIFFERENCES AT THE SAME OFFSETS (THAT'S HOW IT WORKS)

OUR SECOND COLLIDING PAIR


A HASH COLLISION IS...
(IN THE CASE OF THESE MD5/SHA1 ATTACKS)
...A BIG PILE OF...
COMPUTED RANDOMNESS
WITH TINY DIFFERENCES.
REMINDER: THE FINAL HASH IS NOT KNOWN IN ADVANCE.

...AND THESE DIFFERENCES ARE ALWAYS AT THE SAME OFFSETS

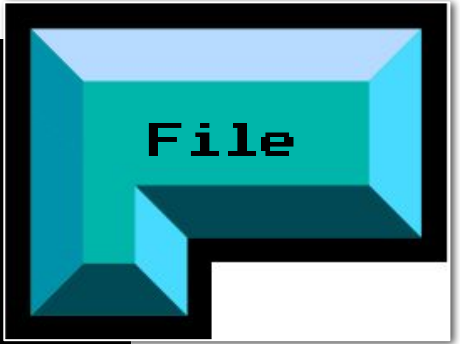
CHOSEN SPECIFICALLY BECAUSE OF WEAKNESSES IN THE HASH FUNCTION..



THE LAST ONES ARE SOMETIMES MISSING!

FOR MORE DETAILS, CHECK https://www.youtube.com/watch?v=iKE7DJd-PwU

```
$ fastcoll -p prefix
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'prefix'
Using initial value: 05ca8309f7b553d58845a18ab918a64c

Generating first block: ....
Generating second block: S10.........
Running time: 2.653 s

$
```




```
$ cat prefix
Here is a file with a few bytes
$
```

Now let's add an input - our prefix.

```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B F0 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 4B BF 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 08 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF D6 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 03 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 2A D2 5C 30 C0
```

```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B 70 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 CB BE 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 88 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF 56 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 83 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 AA D2 5C 30 C0
```

- Padded to 64 bytes
- Collision blocks appended
- Differences at the same relative offsets

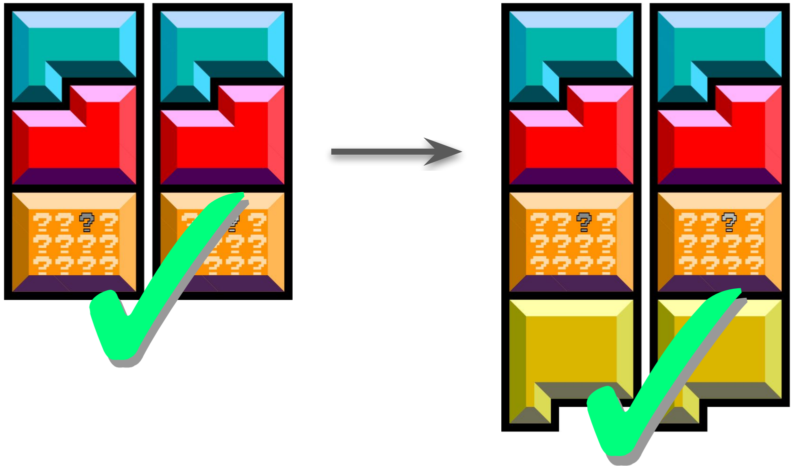
Similar blocks - added after padding to 64 bytes

Padded to 64 bytes?

Merkle–Damgård Construction
https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction

MD5, SHA1 work by processing 64 bytes block, from start to end.

- hash collisions always work with such alignment.
- Appending the same thing
  to two files with the same hash
  will give files with the same hash.



What can we do with this?

We can put whatever we want BEFORE and AFTER the collision.

We need the following from the target file format:

PADDING, for alignments

COLLISION BLOCKS' RANDOMNESS needs to be ignored

DIFFERENCES needs to be taken into account

APPENDED DATA needs to be ignored

```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B F0 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 4B BF 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 08 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF D6 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 03 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 2A D2 5C 30 C0
```

```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B 70 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 CB BE 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 88 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF 56 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 83 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 AA D2 5C 30 C0
```

Certificate

Hash collision

Computed your first FastColl

COLLTRIS

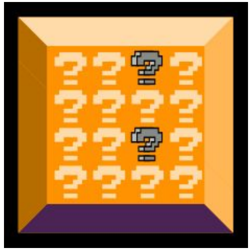
Ange Albertini

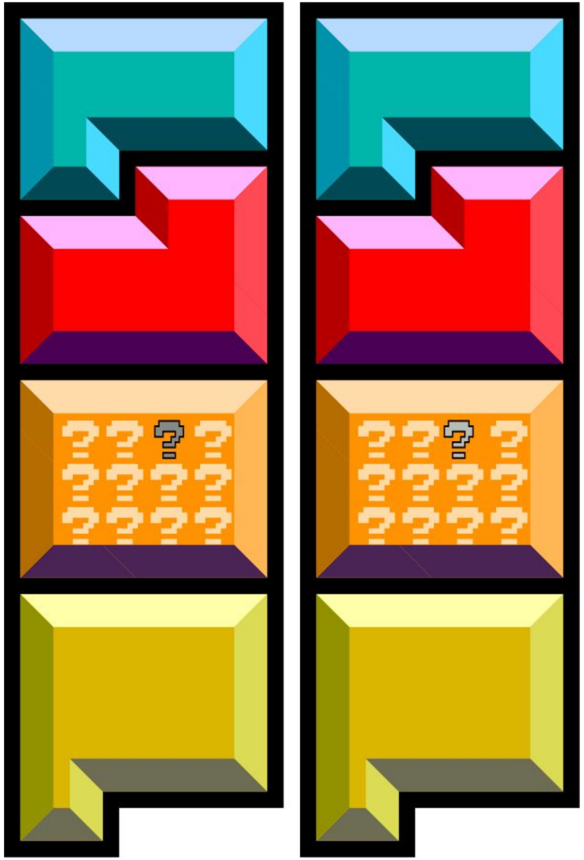
INSTRUCTOR


RECAP 1

Hash collision blocks

- start and end aligned to 64 bytes
  (via padding if needed)
- totally random
- tiny differences at fixed offsets

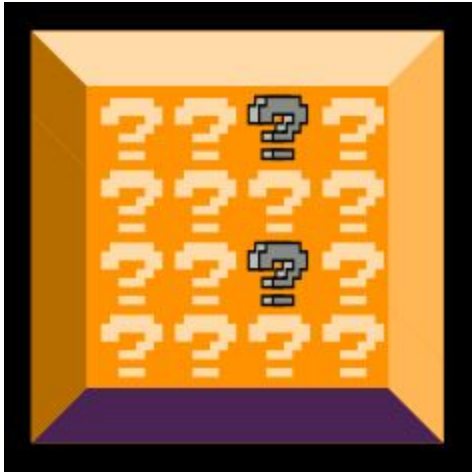


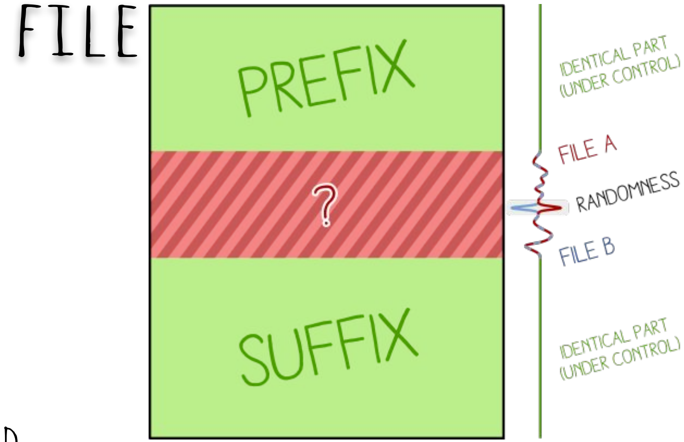


These properties are common to **all** the attacks on MD5 or SHA1.

An Identical Prefix hash Collision

- takes a single input
- prefix and suffix will be identical:
-> files almost identical
-> exploitation depends only on collision differences

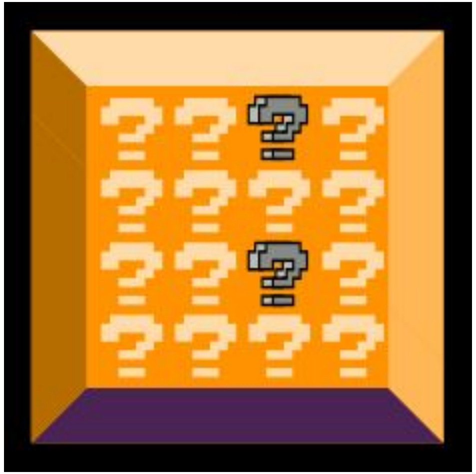




These properties are common to FastColl, Unicoll and Shattered.

FastColl

- TWO BLOCKS
- A FEW SECONDS
- IN THE MIDDLE (AWAY FROM START OR END)



-> HARD TO EXPLOIT!

THE FASTEST BUT ALSO THE MOST LIMITING.

```
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
```

What makes exploiting Fastcoll so difficult?

Every collision differences is surrounded by random data.

It's hard to declare a structure **and** its length in a single byte

such as a variable-length comment that we need for IPC exploitation.

(unless you just run some code to check the difference)

Adding some bruteforcing in the computation is an approach,

but it's a bit slow.

Bruteforcing is a solution

Extra constraint can be added manually inside FastColl source.

Cf PoCorGTFO 14:11

https://github.com/cr-marcstevens/hashclash/blob/master/src/md5fastcoll/block0.cpp#L101
https://github.com/angea/pocorgtfo#0x14

Thankfully there is UniColl.

```
// change q17 until conditions are met on q18, 
 unsigned counter = 0;
 while (counter < (1 << 7))
 {
   const uint32 q16 = Q[Qoff + 16];
   uint32 q17 = ((xrng64() & 0x3ffd7ff7) | (q16
   ++counter;

   uint32 q18 = GG(q17, q16, Q[Qoff + 15]) + tt
   q18 = RL(q18, 9); q18 += q17;
   if (0x00020000 != ((q18^q17)&0xa0020000))
     continue;

   uint32 q19 = GG(q18, q17, q16) + tt19;
   q19 = RL(q19, 14); q19 += q18;
   if (0x80000000 != (q19 & 0x80020000))
     continue;

   uint32 q20 = GG(q19, q18, q17) + tt20;
   q20 = RL(q20, 20); q20 += q19;
   if (0x00040000 != ((q20^q19) & 0x80040000))
     continue;

   block[1] = q17-q16; block[1] = RR(block[1], 
   uint32 q2 = block[1] + tt1; q2 = RL(q2, 12);
   block[5] = tt5 - q2;

   Q[Qoff + 2] = q2;
   Q[Qoff + 17] = q17;
   Q[Qoff + 18] = q18;
   Q[Qoff + 19] = q19;
   Q[Qoff + 20] = q20;
   MD5_REVERSE_STEP(2, 0x242070db, 17);
```

Instant computation doesn't give any instant exploitation.

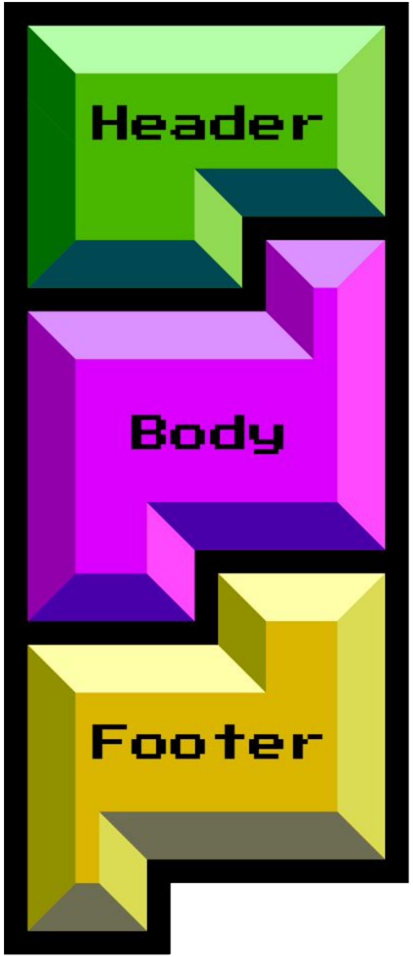
-> Instant collision relies on pre-computed prefix.

The general structure of file formats

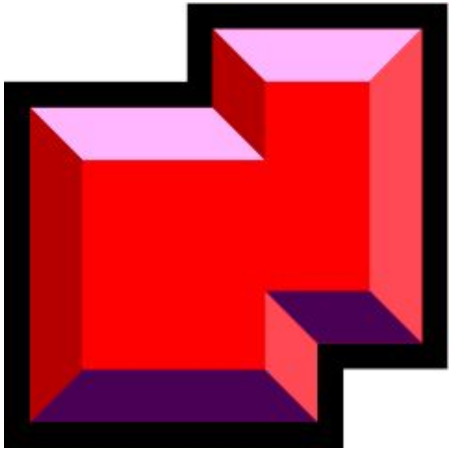
HEADER: at the start of the file.
It defines the file type, versions, and metadata.

BODY comes after. Made of several sub-elements.

FOOTER follows the body.
Indicates that the file is complete.
Any data is usually ignored after.



The "comment" block



Most format accept a comment block of some kind.

It usually can contain **anything** - not just text.

-> perfect to skip collision blocks or extra data

They can be inserted **several** times - they're just entirely skipped.

-> perfect for padding, collision blocks **and** extra data

They are usually length-defined:

-> give them a **variable** length via collision blocks **differences**.

```
HHHH
HHBB
BBBB
FFF
```

```
HHHH
HHCB
BBBB
BFFF
```

```
HHHH
HHCC
CCCC
CCBB
BBBB
FFF
```

```
HHHH
HHCB
CBBB
CCCC
CCCC
BCBF
FF??
??
```

These files are equivalent

(from a parser perspective).

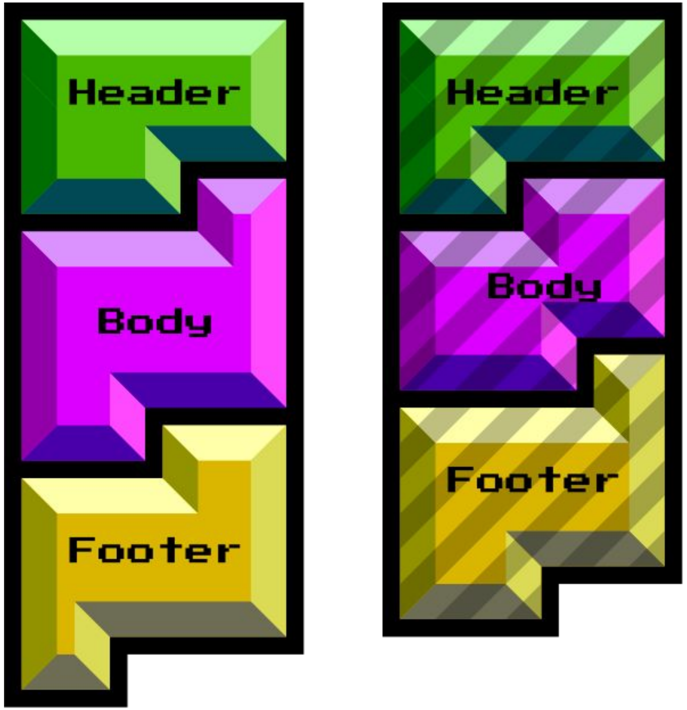
Same content, different structure.

Change from one to the other

is usually (very) easy.

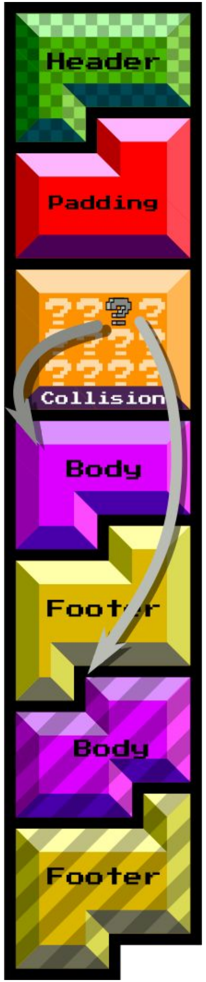
Take two files...

(of the same file type)

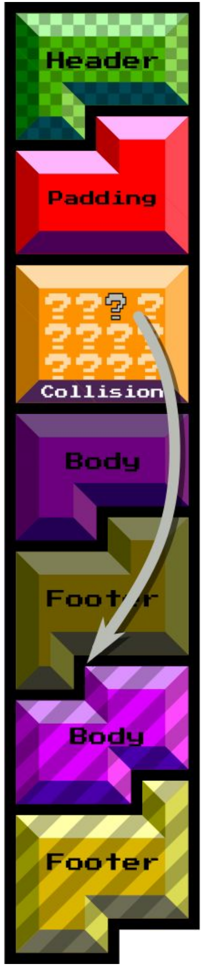


Create a super file combining both files' data

Both files' Body and Footer are kept original.
The header has to be a common ground.



FIND A WAY
TO MAKE THE COLLISION
WORK WITH THE FILE FORMAT.




NEXT
LEVEL

Now let's look at something else.

Poetry…?

Now we hash md5,
no enemy cares!
Only we gave
the shards.

…

A crypt~~ic~~o poem
https://github.com/Jurph/word-decrementer

Now we hath md5,

no enemy dares!

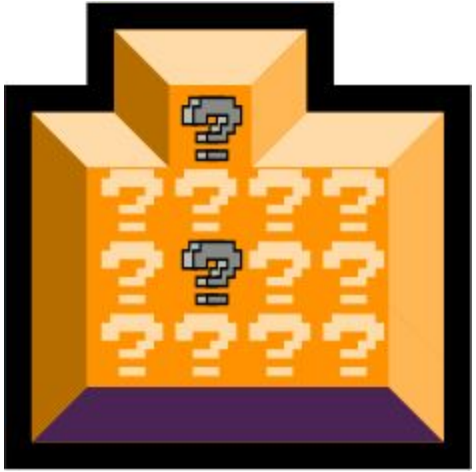
Only we have

the shares.

...

CHANGE THE 10TH LETTER OF EACH SENTENCE WITH THE NEXT ONE.

(LEADING SPACES ARE TOLERATED)



Our second block - another Identical Prefix Collision: UniColl.

```
$ ./poc_no.sh prefix
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

delta_m[2] = [!8!]
In-block prefix words: 5

Parsed path:
Q-3:|01100111 01000101 00100011 00000001|
[...]
Found collision!
2b3663b299b72c6b40d13ccd6c905a7d  collision1.bin
2b3663b299b72c6b40d13ccd6c905a7d  collision2.bin
[...]
$
```

WORDS OF 32 BITS

```
$ cat prefix
Here is my prefix!!\n
$
```

Mission

RUN ITS SCRIPT ON A PREFIX

https://github.com/cr-marcstevens/hashclash/releases
https://github.com/cr-marcstevens/hashclash/blob/master/scripts/poc_no.sh

```
00:  .H .e .r .e .  .i .s .  .m .y .  .p .r .e .f .i
10:  .x .! .! \n 85 33 77 E3 4E 2D B4 F7 33 52 CD 17
20:  63 F0 24 11 8E 42 EE 0D 6D 73 1D 18 FA BA 3F 2D
30:  53 C6 C3 9E 17 F6 86 5F 44 EB 71 C4 24 FB 67 10
40:  53 75 43 D7 3B 33 9A FE E7 B8 ED BD AE A8 07 B9
50:  F4 49 FA 94 34 01 54 DB BE 87 3C 39 AF CD A1 82
60:  C4 EA 3A F8 9B 7C BA D3 AC AF 3D 47 A1 03 0D 34
70:  7F FF 0C 58 92 BC 2B 8A A4 31 53 EE 2F 9B C1 F2
```

```
00:  .H .e .r .e .  .i .s .  .m .z .  .p .r .e .f .i
10:  .x .! .! \n 85 33 77 E3 4E 2D B4 F7 33 52 CD 17
20:  63 F0 24 11 8E 42 EE 0D 6D 73 1D 18 FA BA 3F 2D
30:  53 C6 C3 9E 17 F6 86 5F 44 EB 71 C4 24 FB 67 10
40:  53 75 43 D7 3B 33 9A FE E7 B7 ED BD AE A8 07 B9
50:  F4 49 FA 94 34 01 54 DB BE 87 3C 39 AF CD A1 82
60:  C4 EA 3A F8 9B 7C BA D3 AC AF 3D 47 A1 03 0D 34
70:  7F FF 0C 58 92 BC 2B 8A A4 31 53 EE 2F 9B C1 F2
```

Characteristics:

- Two blocks
- A few mins to compute

Important difference with FastColl:

- prefix as part of the collision blocks!!
- differences:
  10th char of prefix += 1
  10th char of 2nd block -= 1

Output of a UniColl computation

UniColl is an IPC with differences that you can easily predict

A true **Uni**corn of a **coll**ision

Rules of UniColl block prefix

- length multiple of 4
- first block starting at last chunk of 64 bytes.
- the longer the prefix in the block, the longer the computation
- maximum prefix in collision blocks:

  24 bytes - 9 minutes

```
$ cat prefix
Here is my long prefix!
$ time ./poc_no.sh prefix
MD5 differential path toolbox
[...]
Found collision!
6297bd824dc4a35ae83fe8e1bdceb9c2  collision1.bin
6297bd824dc4a35ae83fe8e1bdceb9c2  collision2.bin
[...]
user   9m12.321s
$
```

Sometimes, UniColl just… fail! Just retry then!

Happy ending

```
[...]
262144 9
370611 16
524288 19
Block 1: ./data/coll1_4205915269
53 75 43 d7 3b 33 9a fe e7 b7 ed bd ae a8 07 b9
f4 49 fa 94 34 01 54 db be 87 3c 39 af cd a1 82
c4 ea 3a f8 9b 7c ba d3 ac af 3d 47 a1 03 0d 34
7f ff 0c 58 92 bc 2b 8a a4 31 53 ee 2f 9b c1 f2
Block 2: ./data/coll2_4205915269
53 75 43 d7 3b 33 9a fe e7 b8 ed bd ae a8 07 b9
f4 49 fa 94 34 01 54 db be 87 3c 39 af cd a1 82
c4 ea 3a f8 9b 7c ba d3 ac af 3d 47 a1 03 0d 34
7f ff 0c 58 92 bc 2b 8a a4 31 53 ee 2f 9b c1 f2
Found collision!
2b3663b299b72c6b40d13ccd6c905a7d  collision1.bin
2b3663b299b72c6b40d13ccd6c905a7d  collision2.bin
```

```
[...]
65536 4
126153 8
131072 8
Block 1: ./data/coll1_2664753446
ed 3f f0 88 4c 9a fe 58 f7 68 48 1f 22 28 22 62
20 27 15 9e 1b da cf d4 df b7 7d e3 b4 a1 6c 33
26 2a 58 3e 50 ca c9 3f 84 37 52 65 37 b6 ac fb
9a f9 93 73 49 f9 df b7 48 84 29 c8 cb db 68 dc
Block 2: ./data/coll2_2664753446
ed 3f f0 88 4c 9a fe 58 f7 69 48 1f 22 28 22 62
20 27 15 9e 1b da cf d4 df b7 7d e3 b4 a1 6c 33
26 2a 58 3e 50 ca c9 3f 84 37 52 65 37 b6 ac fb
9a f9 93 73 49 f9 df b7 48 84 29 c8 cb db 68 dc
Found collision!
0b37822e3e06d0e69e2b12d5f742f6d6  collision1.bin
b7c77655f8a1d9b85c4ba7358939c9e4  collision2.bin
```

```
60   70   80   9
|----|----|----|
****************
0, totcond=10485
ta/bestpath.bin.
cat: 'data/coll1_*': No such file or directory
cat: 'data/coll2_*': No such file or directory
738994fa06fb97feec6de48887d6452d  collision1.bin
3170e138bd0606df43c72d8051ba6184  collision2.bin
```

Bad endings

UniColl

- Two blocks
- A few minutes
- In prefix

Slightly slower,
but easy to exploit.

```
00:  .H .e .r .e .  .i .s .  .m .y .  .p .r .e .f .i
10:  .x .! .! \n 85 33 77 E3 4E 2D B4 F7 33 52 CD 17
20:  63 F0 24 11 8E 42 EE 0D 6D 73 1D 18 FA BA 3F 2D
30:  53 C6 C3 9E 17 F6 86 5F 44 EB 71 C4 24 FB 67 10
40:  53 75 43 D7 3B 33 9A FE E7 B8 ED BD AE A8 07 B9
50:  F4 49 FA 94 34 01 54 DB BE 87 3C 39 AF CD A1 82
60:  C4 EA 3A F8 9B 7C BA D3 AC AF 3D 47 A1 03 0D 34
70:  7F FF 0C 58 92 BC 2B 8A A4 31 53 EE 2F 9B C1 F2
```

```
00:  .H .e .r .e .  .i .s .  .m .z .  .p .r .e .f .i
10:  .x .! .! \n 85 33 77 E3 4E 2D B4 F7 33 52 CD 17
20:  63 F0 24 11 8E 42 EE 0D 6D 73 1D 18 FA BA 3F 2D
30:  53 C6 C3 9E 17 F6 86 5F 44 EB 71 C4 24 FB 67 10
40:  53 75 43 D7 3B 33 9A FE E7 B7 ED BD AE A8 07 B9
50:  F4 49 FA 94 34 01 54 DB BE 87 3C 39 AF CD A1 82
60:  C4 EA 3A F8 9B 7C BA D3 AC AF 3D 47 A1 03 0D 34
70:  7F FF 0C 58 92 BC 2B 8A A4 31 53 EE 2F 9B C1 F2
```

Plan your exploit

Explore format specs for features you need.

Explore attacks: FastColl, UniColl [tree], HashClash

Plan your structure (pen & specs)

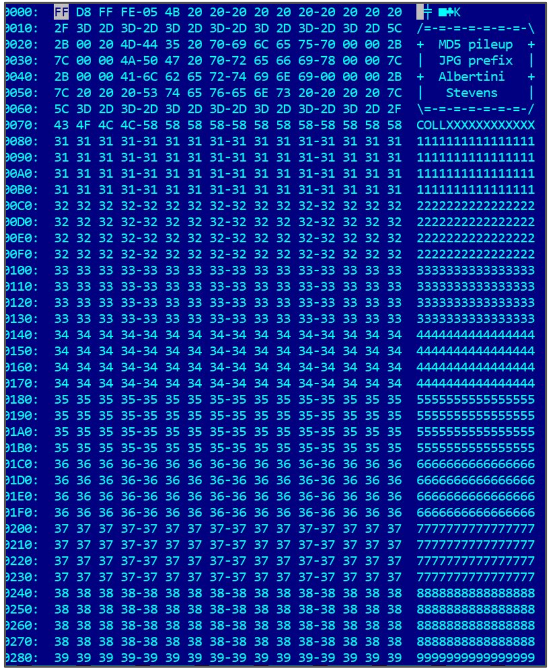
craft mockup files. Good to validate compatibility, CRCs...

Erase collision blocks to simulate colliding files.

Extract prefixes from mockups.

Run computations.

⇤ ⇥	Padding, for alignments
# & % ! @	collision blocks' randomness needs to be ignored
...?...	Differences need to be taken into account
... ?	Appended data needs to be ignored



Plan your generic exploit

Explore format landscape, tools and options.

Some minor tool option might be a perfect fit.

Ex: `mutool merge` w/ a dummy PDF is perfect for hacking

Understand compatibility in depth.

What makes exploiting UniColl so easy?

The first difference is surrounded by chosen text:

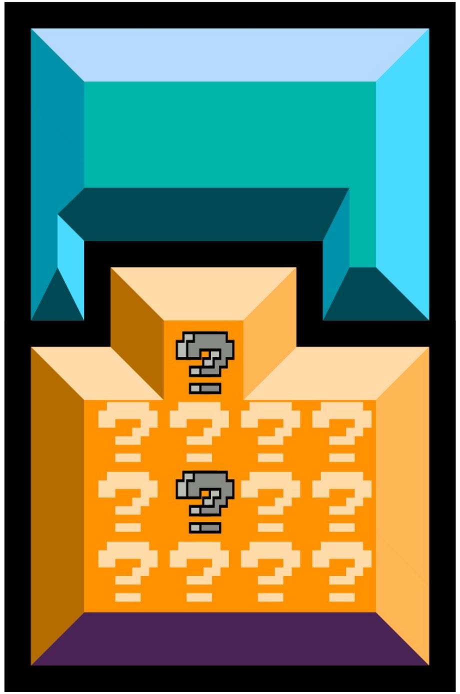
- no restrictions to declare a length before or after a type.

The difference is +1, which makes it trivial to plan the impact.

- i.e. one chunk will be exactly `0x100` longer than the other,
  - which is bigger than the collision block
  - but doesn't grow uncontrollably.

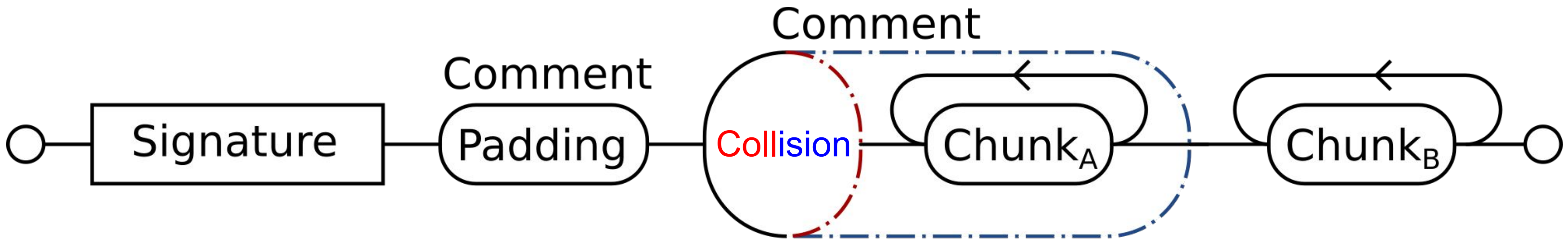
```
FastColl: 61 52 3E ⇔ 61 D2 3E
UniColl:  00 75 .D .A .T .A ⇔
          01 75 .D .A .T .A
```

Exploiting our first hash collision attack

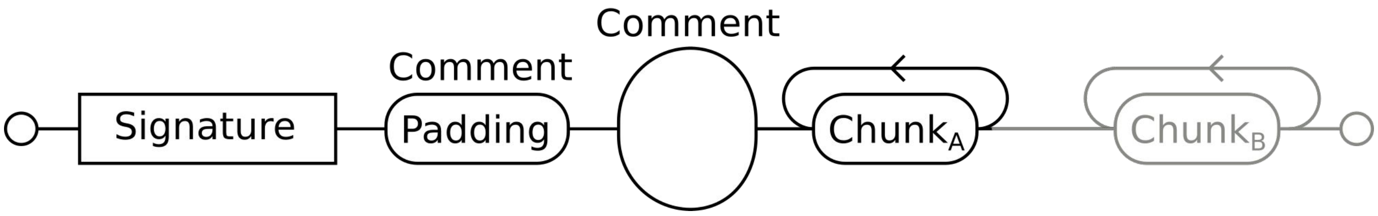


Layout of a generic file format exploitation

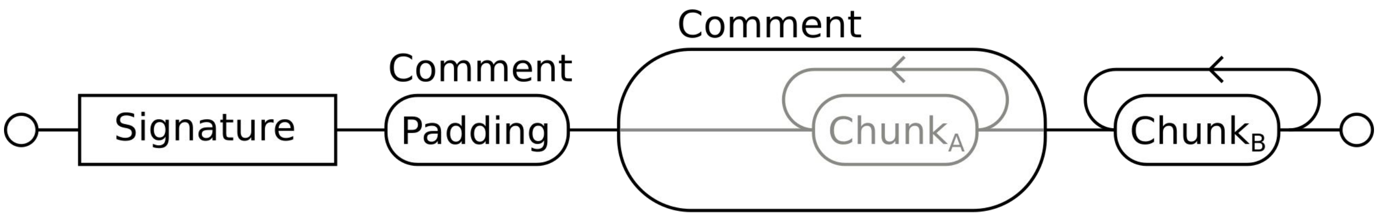
1. A fixed-length comment for padding.
2. A variable length comment at the start of collision blocks.
3. Using collision blocks to grow this comment over a first file's data, followed by a second's file data.



Case A (short comment)



Case B (long comment)

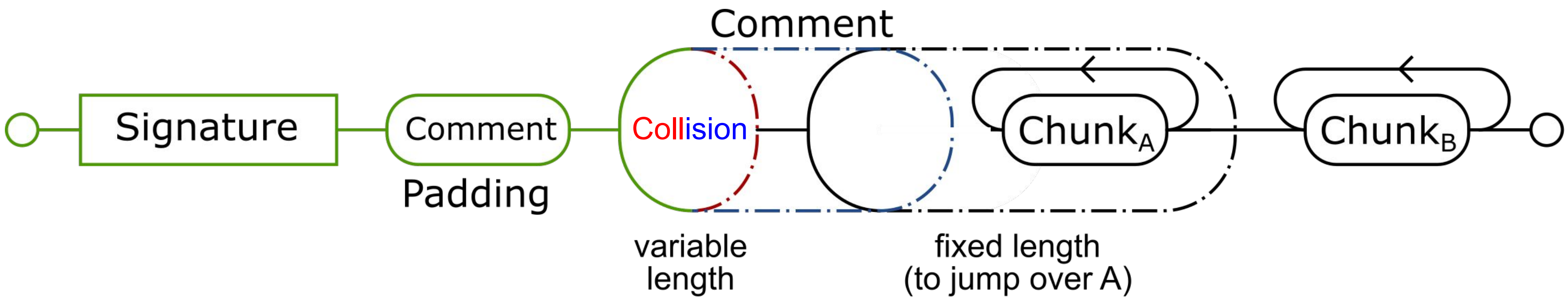


Making it generic?

The size of $\{\text{Chunk}_A\}$ is unknown in advance.

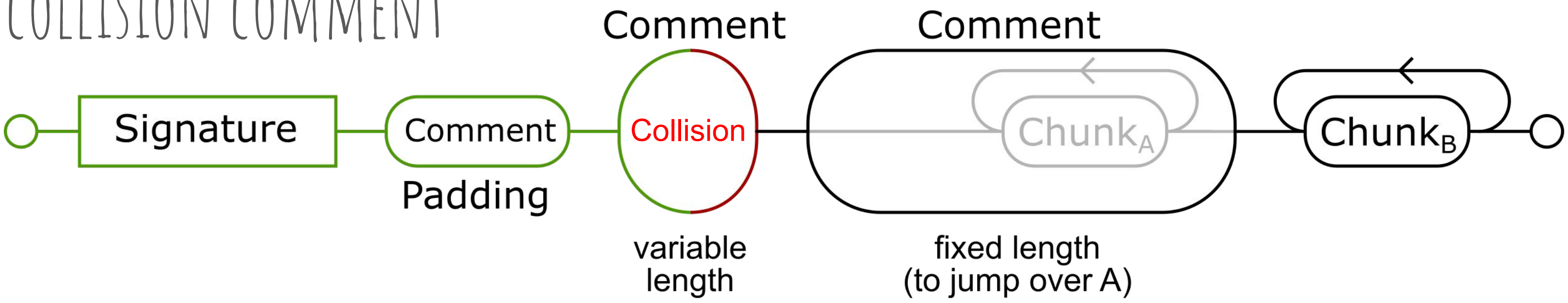
-> one extra comment to jump over these chunks

with its declaration switched on/off by the variable comment

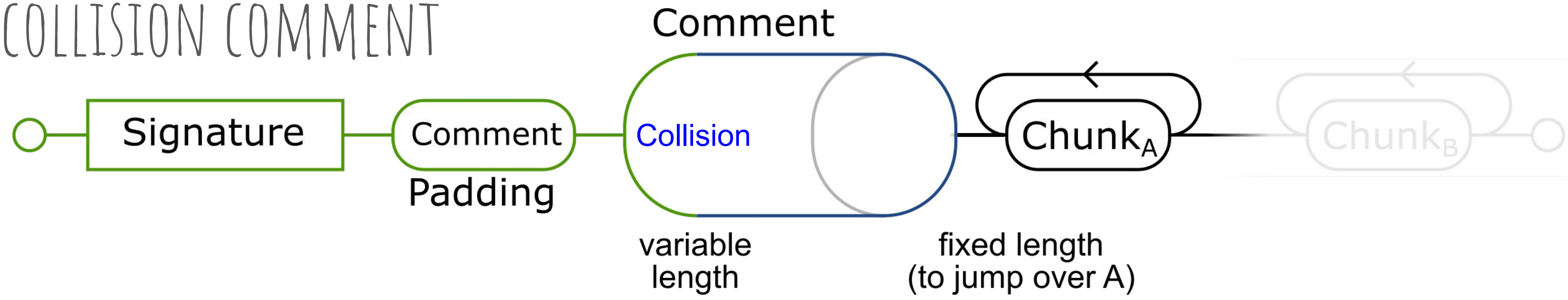


A chain of three comments

Short collision comment

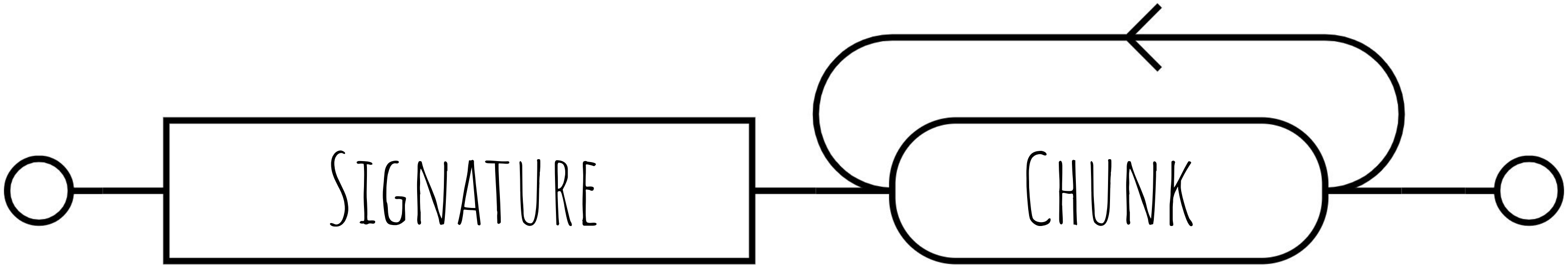


Long collision comment



The Portable Network Graphics format

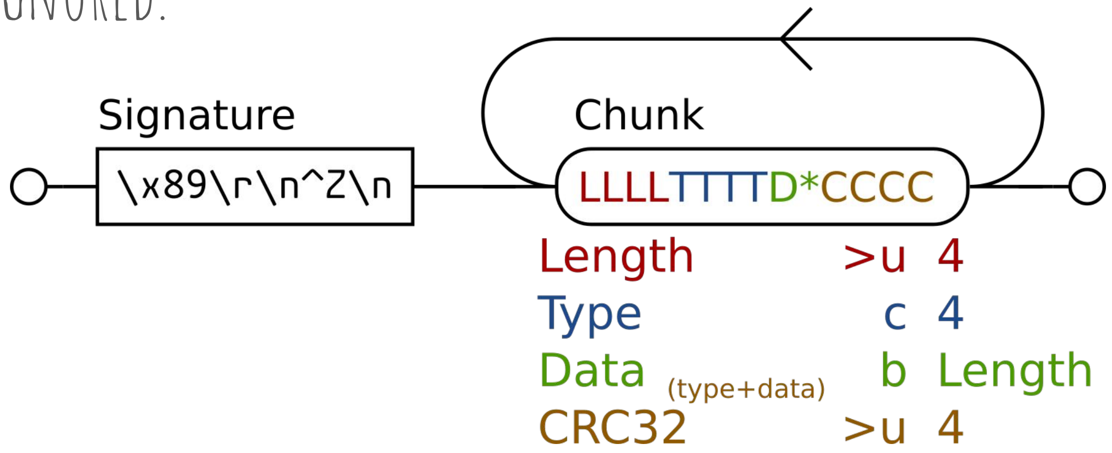
/ˌpiːɛnˈdʒiː/ PEE-EN-JEE
/pɪŋ/ PING



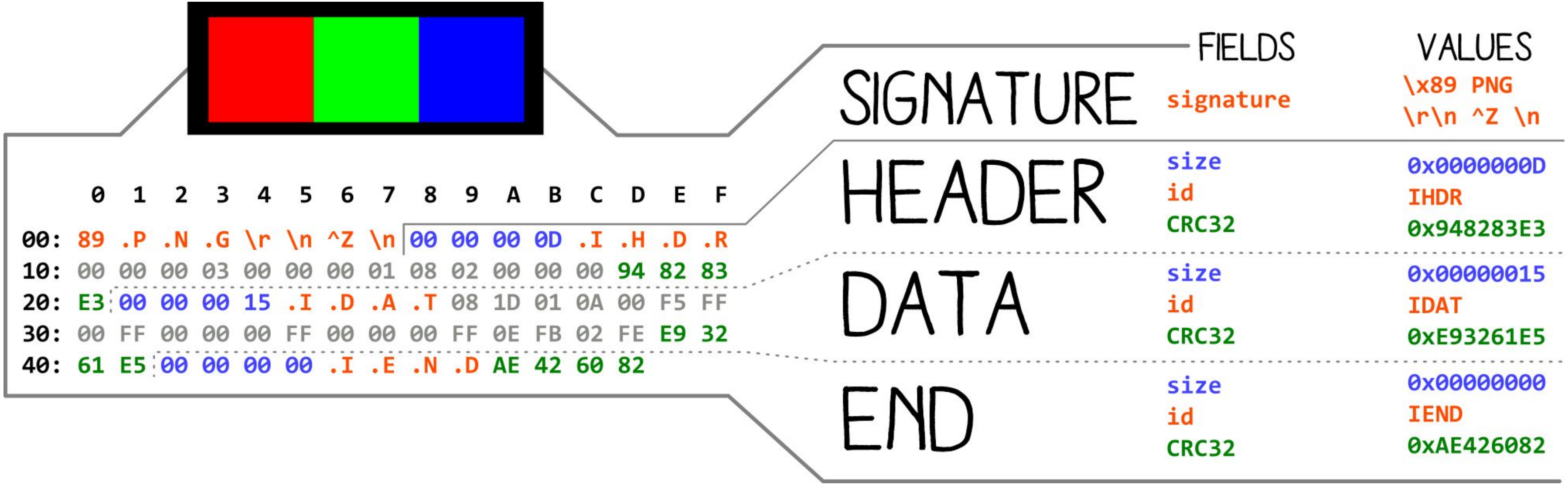
The most regular format:
a signature then a sequence of chunks.

The PNG format at chunk level

- Big endian length, on **4** bytes.
- A type, on **4** characters.
  Critical chunks: `IHDR/PLTE/IDAT/IEND`
  Lowercase starting types == ignored.
- CRCs are usually ignored



Overview of an actual PNG file



```python
import struct
import binascii

_MAGIC = "\x89PNG\x0d\x0a\x1a\x0a"

_crc32 = lambda d:(binascii.crc32(d) % 0x100000000)

def parse(f):
  assert f.read(8) == _MAGIC
  chunks = []
  while (True):
    l, = struct.unpack(">I", f.read(4))
    t = f.read(4)
    d = f.read(l)
    assert _crc32(t + d) == struct.unpack(">I", f.read(4))[0]
    chunks += [[t, d]]
    if t == "IEND":
      return chunks
  raise(BaseException("Invalid image"))
```

```python
def make(chunks):
  s = [_MAGIC]
  for t, d in chunks:
    s += [
      struct.pack(">I", len(d)),
      t,
      d,
      struct.pack(">I", _crc32(t + d))
    ]
  return "".join(s)
```

Trivial to parse or manipulate at chunk level.

```
0000: 89 .P .N .G \r \n ^Z \n 00 00 00 33 .a .N .G .E
0010: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0020: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0030: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0040: ?? ?? ?? ?? ?? ?? ?? 00-00 00 75 .m .A .R .C ??
0050: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0060: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0070: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0080: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0090: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00A0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00B0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00C0: ?? ?? ?? ?? ?? ?? ?? ?? XX XX XX XX .j .U .M .P
[...]
01C0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? .I .H .D .R
01D0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
[...]
????: ?? ?? ?? ?? ?? 00 00 00 00 .I .E .N .D AE 42 60
????: 82 ?? ?? ?? ?? 00 00 20 00 .I .H .D .R ?? ?? ??
[...]
????: ?? ?? ?? ?? ?? ?? ?? 00 00 00 00 .I .E .N .D AE
????: 42 60 82
```

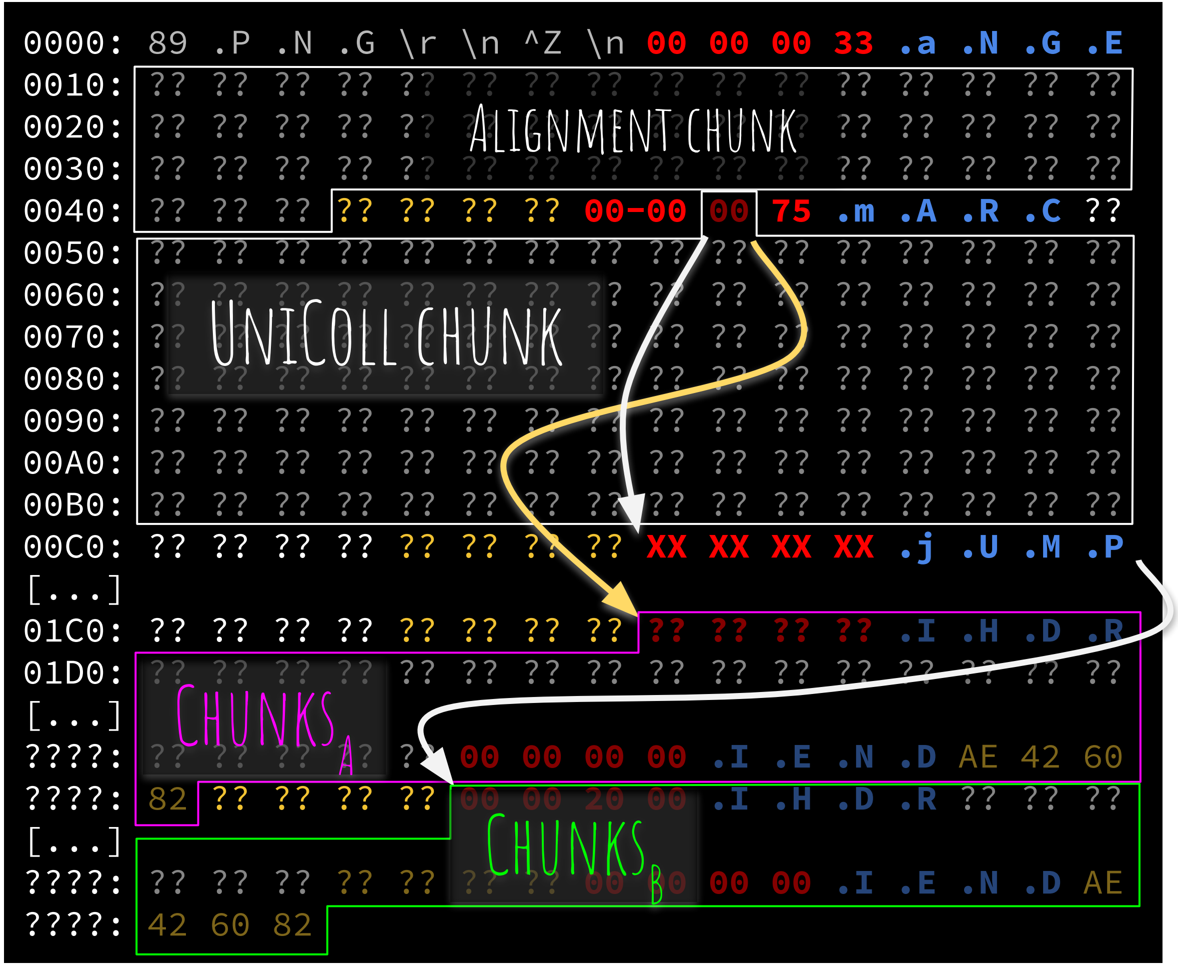
LENGTH

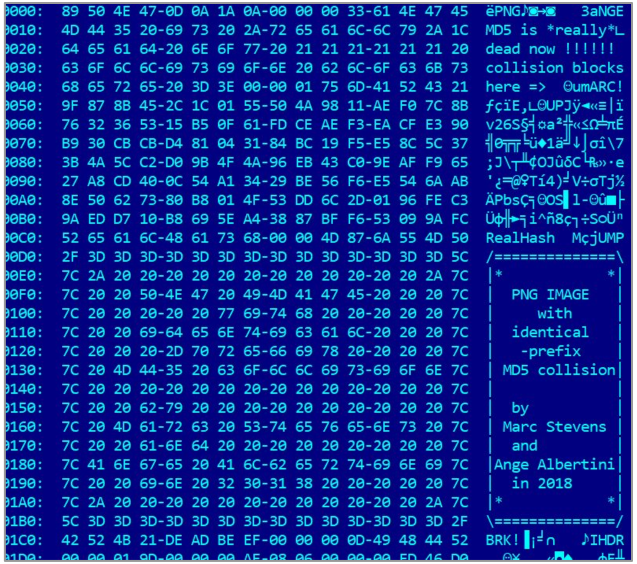
TYPE

CRC32

Mission

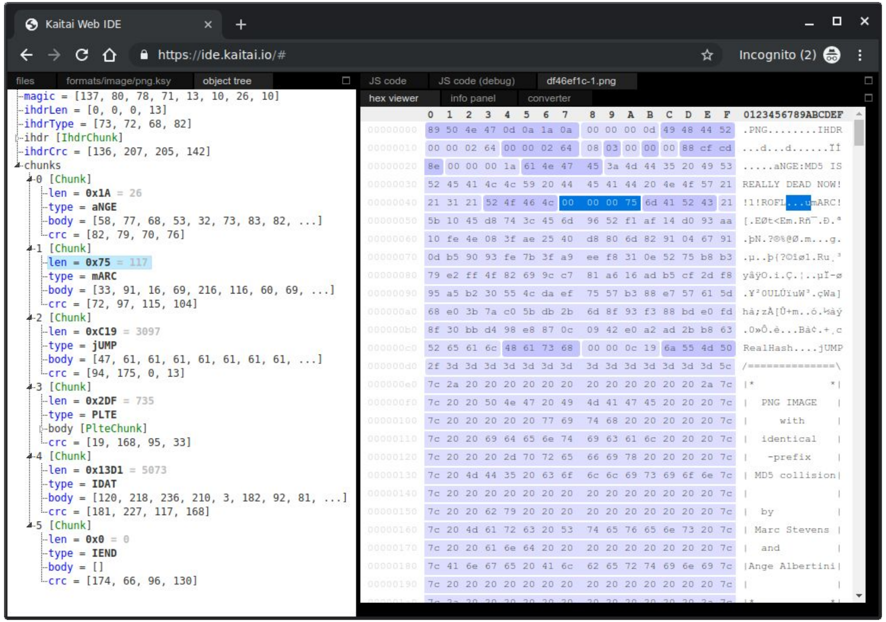
3 DUMMY CHUNKS: ALIGNMENT, COLLISION AND JUMP OVER (THE FIRST IMAGE) DATA

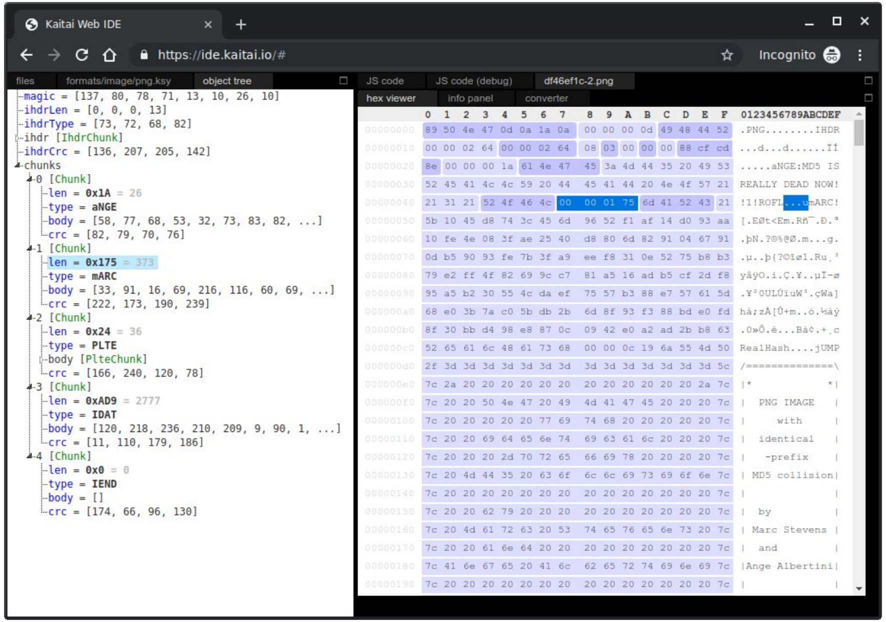




CF https://github.com/corkami/collisions/blob/master/scripts/png.py

Need to explore by yourself?





Open Kaitai IDE with the lightweight PoCs

https://ide.kaitai.io/ + https://github.com/corkami/collisions/blob/master/examples/free/README.md

Kaitai tricks

Only the high level structure is useful:

-> simpler grammar can be better.

Looser logic can be required:

Ex: **IHDR** chunk not in the first slot.

ICYDK You can directly edit the grammar in the IDE!

Simplified PNG grammar

```yaml
meta:
  id: png
  file-extension: png
  endian: be
seq:
  - id: magic
    contents: [137, 80, 78, 71, 13, 10, 26, 10]
  - id: chunks
    type: chunk
    repeat: until
    repeat-until: _.type == "IEND" or _io.eof
types:
  chunk:
    seq:
      - id: len
        type: u4
      - id: type
        type: str
        size: 4
        encoding: UTF-8
      - id: body
        size: len
      - id: crc
        size: 4
```

Know the format landscape

All PNG viewers seem to ignore CRCs.

Most PNG viewers tolerate starting w/ a dummy chunk.

-> generic collisions for any PNG pair

OS X (Safari, Preview) enforce an **IHDR** chunk first:

-> dimensions and colorspace are in the common prefix

Study the landscape helps to understand the scope of your exploit.

GitHub, Inc. [US] | https://github.com/file/file/blob/master/magic/Magdir/images#L440-L441

```
439
440  # Standard PNG image.
441  0       string          \x89PNG\x0d\x0a\x1a\x0a\x00\x00\x00\x0DIHDR     PNG image data
```

https://github.com/file/file/blob/master/magic/Magdir/images#L440-L441

Certificate

Collision exploit

Reusable PNG via UniColl

COLLTЯIS

Ange Albertini

INSTRUCTOR

BREAK

Back to exploiting FastColl...

It's just a matter of getting a format to comply with the single byte differences.

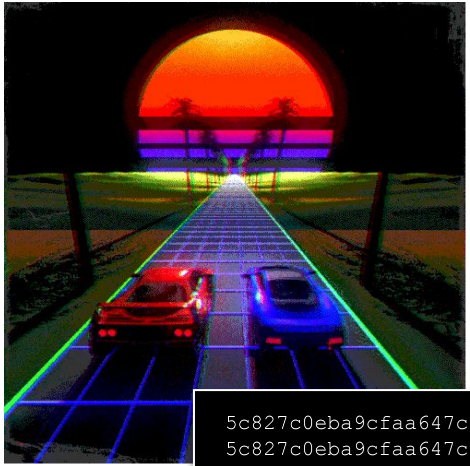
```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B F0 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 4B BF 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 08 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF D6 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 03 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 2A D2 5C 30 C0
```

```
00:  .H .e .r .e .  .i .s .  .a .  .f .i .l .e .  .w
10:  .i .t .h .  .a .  .f .e .w .  .b .y .t .e .s 00
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40:  CE 84 07 61 4B BA 7A 3D 3A EA 8A AA F8 EE 1D E5
50:  44 17 9B 70 0A E0 D2 64 21 E2 38 E1 94 18 0A F6
60:  93 D2 B5 E4 FC 2F 3A 32 4F 50 46 01 F1 CB BE 02
70:  23 EE EF BF 92 B5 7C 29 D9 C5 66 88 31 5E 7A 1D
80:  2F 5A 9C 5C 12 8E DF F2 85 17 5B DD 67 25 05 78
90:  13 F2 BF 56 64 59 F2 C8 8B C3 00 6F 8B 5F 88 C6
A0:  CB 3D 80 E4 9F 48 91 5E 34 06 D0 3A 8B 83 FB E0
B0:  ED 18 67 0F C8 3A C9 A1 E7 48 F6 AA D2 5C 30 C0
```

The Graphics Interchange Format

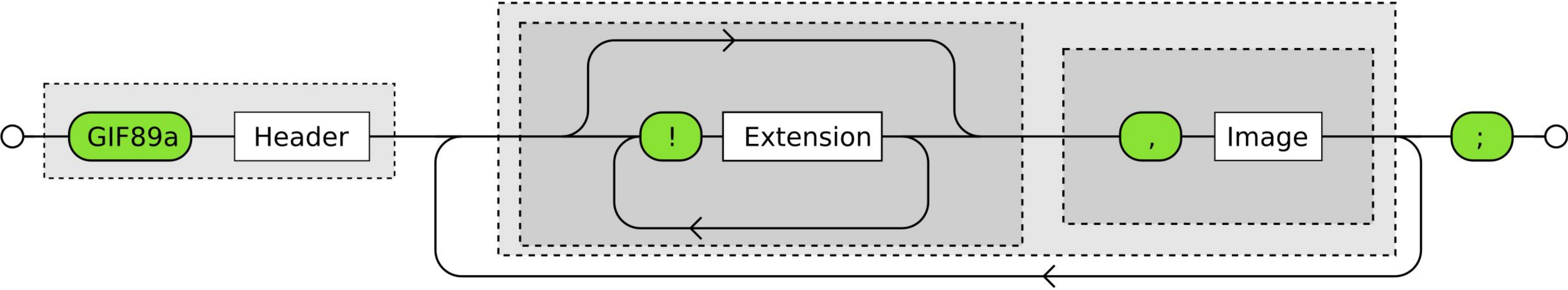
/dʒɪf/ JIF
/gɪf/ GHIF

- FastColl based exploit: instant computation
- same dimensions and palettes, single frame.
- First image displayed for 10 minutes (each image is a different frame).

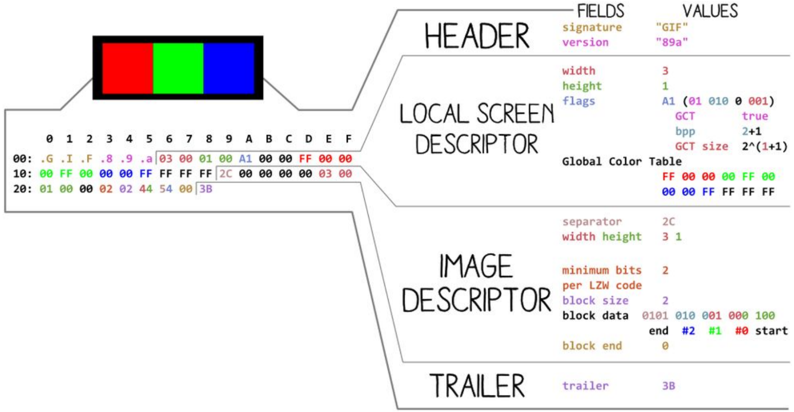



```
5c827c0eba9cfaa647c1a489bea77c60  *collision1.gif
5c827c0eba9cfaa647c1a489bea77c60  *collision2.gif
```

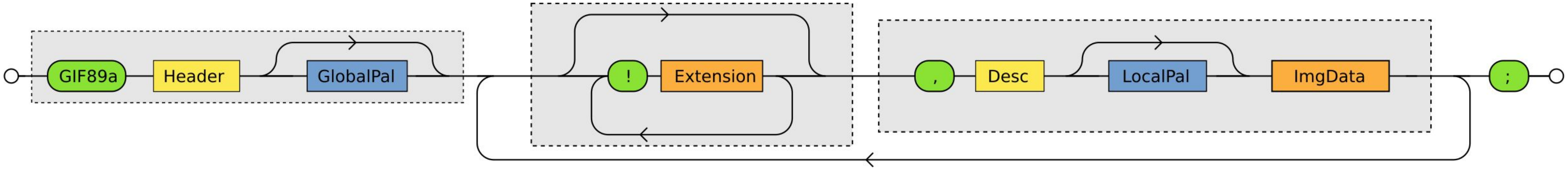
Overview of the Graphics Interchange Format



- Punctuation delimited: ! , ;
- A frame can be made of several images
- Header contains file palette & dimensions...
- Comments can only be later in the file, in extensions
-> no generic collisions for all GIFs

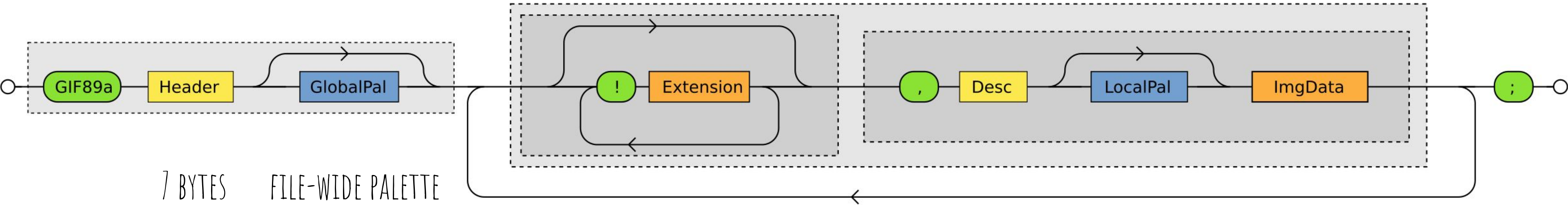


More details



- A header, with dimensions and optional global palette
- Sequence of optional extensions and image data.
  - comments are extensions.
- ImageData and Extension use the same SubBlocks structure.
- Global (file-wise) and local (image-wise) palettes can be too big.

Structure lengths



7 bytes

File-wide palette
(optional)
3-768 bytes

Examples of extension:
- Graphical Control
  defines delay between frames
- Comment
- Application
  Defines looping

10 bytes

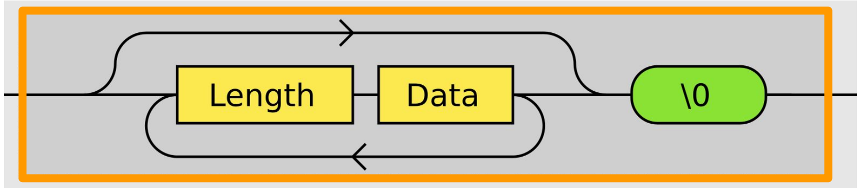
Image-wide palette
(optional)
3-768 bytes

Fixed size

SubBlocks-based

Variable-sized
(3-768 bytes)

GIF SubBlocks structures



Specific structure for comments **and** image data in GIF:

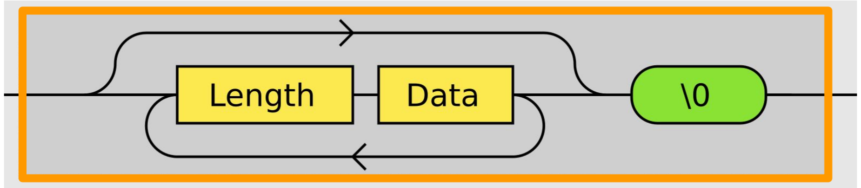
Cut in chunks of 255 bytes max, starting with their length, until **00**:

Examples of 2 equivalent comments:

```
07 .c .o .m .m .e .n .t
00
```

```
01 .c 04 .o .m .m .e 02 .n .t
00
```

GIF SUBBLOCKS IMPACT



- CAN'T JUMP OVER ANYTHING LONGER THAN 255 BYTES.
  -> **VERY** RESTRICTIVE

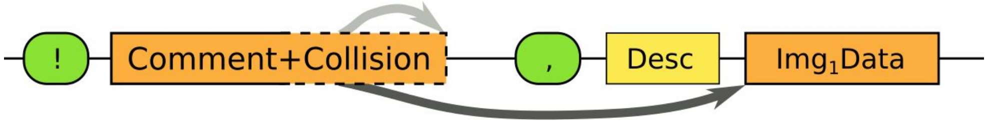
+ TURNS ANY NON-NULL BYTE INTO A FORWARD JUMP:
  GOOD FOR FASTCOLL

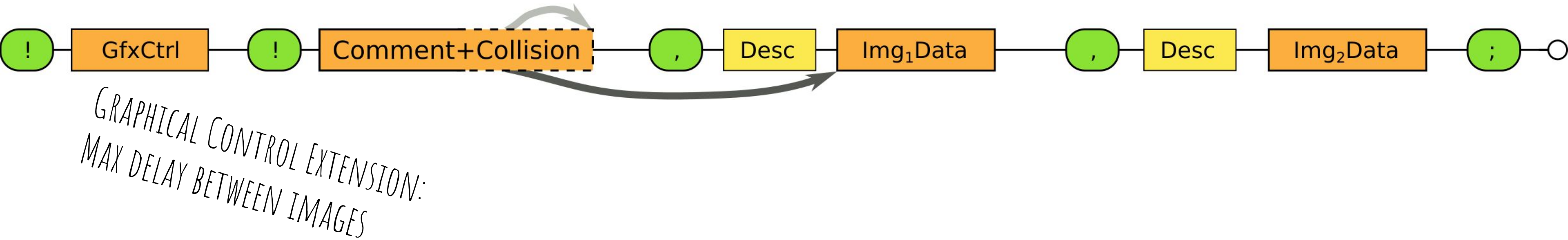
GIF data sled

Subblocks are common to image data and extensions (like comments):

- > extend comment to image data (turn pixels into comment)

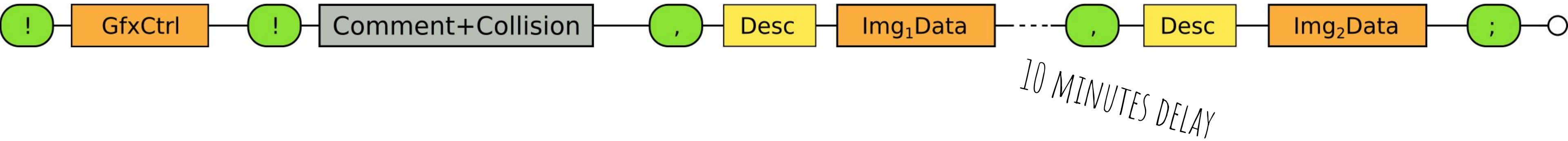
Reliable w/ minor overhead.





Long comment

Short comment



GIF comment manipulation via FastColl

Defines a comment

Chunk length: 0x7B

```
0330:   .. .. .. ..-.. .. .. ..-.. .. .. ..-.. 21 FE 7B
0340:   7B 07 80 42-FF 65 E4 4E-1F 99 A0 E8-4D BC 59 EB
0350:   E8 DA 58 4C-35 CF 2C 78-53 1E 79 D1-28 34 08 DA
0360:   B5 DB FF C6-80 0F 3A 46-EF 0F FB 1C-F9 71 E0 83
0370:   CC FB ED 70-D9 21 A5 7D-0A A1 10 B6-A7 C5 6D E0
0380:   71 82 1F FA-AC 77 A9 12-DD 8E F2 14-9D 64 5B F8
0390:   3D 66 C5 2E-D3 97 8F 2B-6E B9 AB 4B-4B 1D A7 C1
03A0:   A8 34 B5 2D-96 46 8A DE-A9 9C EF 18-6B 0C F8 08
03B0:   12 54 43 53-AD 8A 72 BA-19 83 66 B3-2F CF 85 FD
....    ...
```

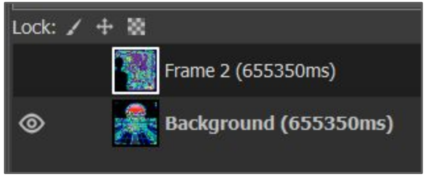
Chunk length: 0x33/0xB3

GIF

Special chunk structure -> single byte = "jump"

Same structure used for data -> can use it to jump over image A.

Just put a delay for image A to display long enough.




```
5c827c0eba9cfaa647c1a489bea77c60  *collision1.gif
5c827c0eba9cfaa647c1a489bea77c60  *collision2.gif
```

After 10 minutes,
the images are identical.

Common headers?

Headers include palette, dimensions:

-> use 2 images of same dimensions

-> normalize palette

Shortcut: merge them as 2 frames of the same animation, with a comment, no looping, and maximum delay.

GIF FastColl exploit

- combine 2 frames in a single animation with a comment.
- extend comment to align to 64 bytes
  with a jump to `0x7B` (last difference in the collision blocks)
- compute FastColl
- append images suffix
- adjust comments to:
  finish before first image: `.! F9`
  slide into first image data : `08 FE <high entropy>`

Actual example

```
000002F0:  FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF
00000300:  FF FF FF FF-FF FF FF FF-FF FF FF FF-FF 21 FE 2F                 !■/
00000310:  30 30 30 30-30 30 30 30-30 30 30 30-30 30 30 30  0000000000000000
00000320:  30 30 30 30-30 30 30 30-30 30 30 30-30 30 30 30  0000000000000000
00000330:  30 30 30 30-30 30 30 30-30 30 30 30-30 30 30 7B  000000000000000{
00000340:  A9 18 DA 09-0D C6 6E A1-EF 31 59 CF-95 85 EF 1D  ⌐↑┌○♪╞ní∩1Y╧òà∩↔
00000350:  46 E8 43 1C-9C C6 81 F6-FF 45 3C 65-71 2F 13 14  FΦC∟£╞ü÷ E<eq/‼¶
00000360:  D0 19 0E C9-82 C2 F1 5D-4B 26 DE FA-E1 A3 17 91  ╨↓♫╔é┬±]K&▐·ßú↨æ
00000370:  25 0F EA 39-6F A6 5D 4C-E2 D5 70 7F-59 DC 92 8C  %☼Ω9oª]LΓ╒p⌂Y▄Æî
00000380:  3E 3A F6 80-0C B0 58 97-C5 61 E2 F7-DD 55 E3 CB  >:÷Ç♀░Xù┼aΓ≈▌Uπ╦
00000390:  4B 57 C6 15-48 E9 70 1D-F6 2C 76 C7-4D 8B 45 9D  KW╞§HΘp↔÷,v╟MïE¥
000003A0:  5D F8 BA F9-BC 2E 6A 86-6A 43 15 A2-91 CC D2 BD  ]°║∙╝.jåjC§óæ╠╥╜
000003B0:  F2 C0 D5 EE-9F D4 F3 F7-6D 6B BA EA-FA C5 20 F7  ≥└╒εƒ╘≤≈mk║Ω·┼ ≈
000003C0:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
000003D0:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
000003E0:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
000003F0:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
00000400:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
00000410:  2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E-2E 2E 2E 2E  ................
00000420:  2E 2E 2E 2E-2E 2E 80 5F-5F 5F 5F 5F-5F 5F 5F 5F  ......Ç_________
00000430:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000440:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000450:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000460:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000470:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000480:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
00000490:  5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F-5F 5F 5F 5F  ________________
000004A0:  5F 5F 5F 5F-5F 5F 14 00-21 F9 04 00-FF FF FF 00  ______¶ !∙♦
000004B0:  2C 00 00 00-00 F4 01 F4-01 00 08 FE-00 59 09 BC  ,    ⌠☺⌠☺ ◘■ Y○╝
000004C0:  26 B0 00 41-82 03 0D 2A-5C C8 10 21-2B 87 05 09
```

COMMENT DECLARATION

ALIGNMENT

COLLISION BLOCKS

IMAGE

SUBBLOCKS

```
00000300:  [header palette ending................] .! FE 2F
00000310:  [comment for alignment.........................
...
00000330:  ..........................................  7B
00000340:  [collision block with its last difference.....
00000350:  at relative offset of 7B......................
...                                                 6
000003B0:  ...............................] EA [.........
000003C0:  [space to land to the shortest comment ........
000003D0:  its length will vary, but.....................
000003E0:  the longest comment will always be 0x80 longer.
...
00000420:  ...............] 80 [..........................
...
000004A0:  ...............] 14 00 .! F9 04 00 FF FF FF 00
000004B0:  2C 00 00 00 00 F4 01 F4 01 00 08 FE 00 59 09 BC
```

Cf https://github.com/corkami/collisions/blob/master/scripts/gif.py

Certificate

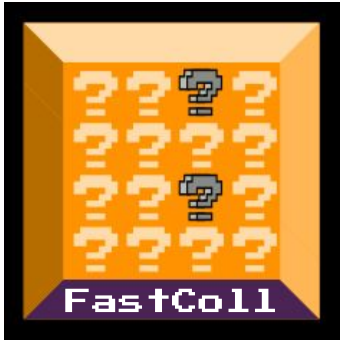
Collision exploit

Instant GIF collision via FastColl

Ange Albertini

INSTRUCTOR


RECAP 2



TWO BLOCKS

A FEW SECONDS

IN THE MIDDLE

(AWAY FROM START OR END)

TWO BLOCKS

A FEW MINUTES

IN PREFIX

THE TWO IDENTICAL PREFIX COLLISIONS AGAINST MD5

IPCs limitations

Some formats have hardcoded offsets, or don't tolerate early comments

Same prefix -> same file type

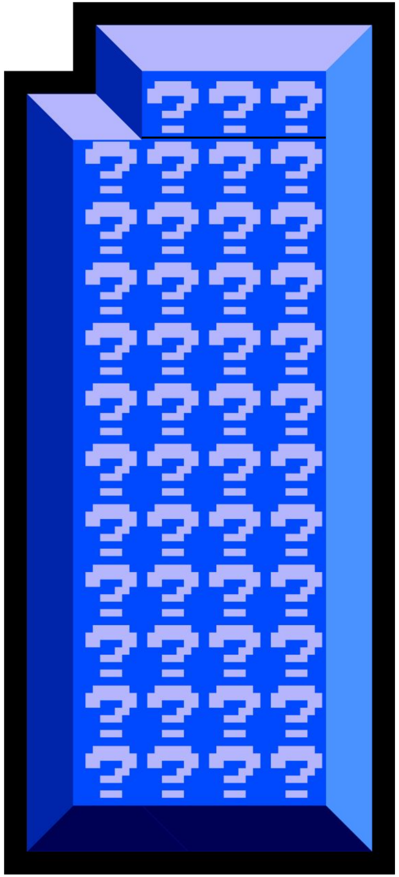
Same header -> same metadata

Enforced checksums prevent validity.

Only the length of a current structure level can be manipulated.

The ultimate attack

Chosen-Prefix Collisions



Our third block:
a Chosen Prefix hash Collision

A word of warning on CPC

Takes 72h.core hours to compute - if you're lucky.

requires simple monitoring and backtracking.

The fewer the collision blocks,

the longer to compute.

Launching a HashClash computation

Trivial: run `scripts/cpc.sh prefix1 prefix2`

Requires monitoring: `ls -latr | tail && date && tail screenlog.0`

If a step takes more than 1h, kill and backtrack at previous step:

```
pkill md5
scripts/cpc.sh prefix1 prefix2 <n-1>
```

Example of stalled HashClash computation:

```
[...]
-rwxr-xr-x  3 corkami corkami   4096 May 26 08:38 workdir4
-rw-r--r--  1 corkami corkami 552859 May 26 08:45 screenlog.0
Sun May 26 15:22:41 UTC 2019
25: Q14Q3m14tunnel   = 2
20: Q5m5tunnel       = 6
20: Q4m5tunnel       = 1
20: Q14m6Q3m5tunnel = 0
21: Q10m10tunnel    = 2
21: Q9m10tunnel      = 6
```

A 9-BLOCK CPC OF YES AND NO (DIFFERENCES ARE IRRELEVANT)

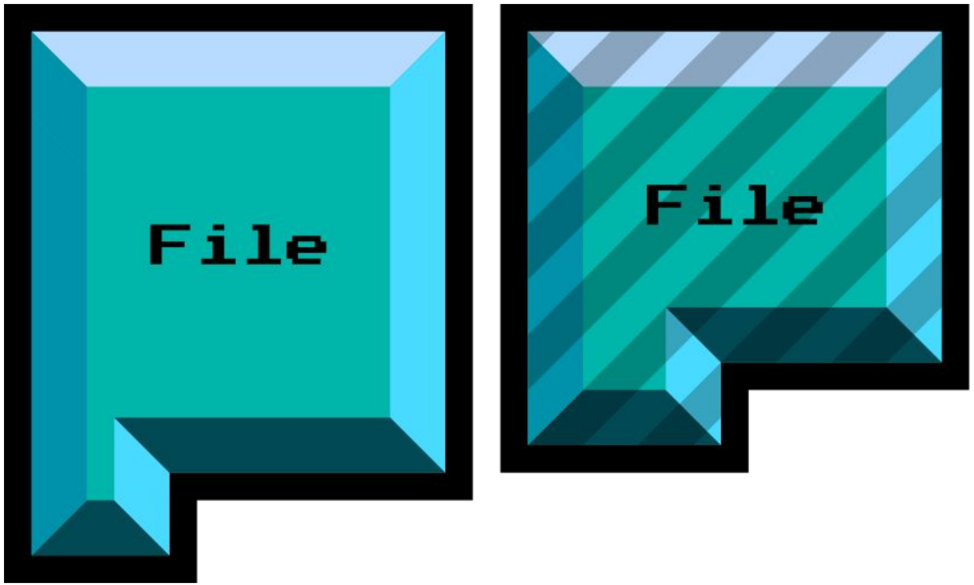
```
0000: .y .e .s 00-00 00 00 00-00 00 00 00-00 00 00 00
0010: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0020: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0030: 00 00 00 00-00 00 00 00-B7 46 38 09-8A 46 F1 7B
0040: F3 45 26 13-66 60 C8 01-B9 2A 75 25-5A 67 23 A6
0050: 92 3D EB 8D-B0 B7 57 F1-45 9F 22 95-BE C0 43 75
0060: 91 98 A2 D3-E0 FD 59 ED-D1 C5 FA 0B-79 65 97 4D
0070: B3 B3 E4 0C-11 0C 90 32-DE 4B A1 4B-B8 1B 5E C8
0080: 25 D3 8F 19-CD 10 43 07-D9 BB FF 8C-B7 5A 23 F9
0090: 4D D8 13 14-58 A3 35 97-C5 D1 D4 A9-9A E2 FD 1F
00A0: BA 78 40 00-C3 7E 93 B2-31 A3 6E 2D-34 6A 4A C9
00B0: 53 4E C0 45-36 1E C8 6A-56 98 E6 F0-57 1D 61 98
00C0: 13 FC FF CD-4D 83 A2 D2-BB B8 DC 04-2B E2 B8 83
00D0: DB 53 80 D7-3D E9 97 D3-23 5A 27 F9-98 9A E7 56
00E0: 7D 86 E4 35-1E B8 33 EE-EA 15 D1 81-BA 96 62 EC
00F0: 75 31 FB DA-4F AE 24 6F-67 D6 AF 10-96 29 FB C7
0100: A3 32 BB A9-EA D5 E4 AE-1F C2 FB 23-41 22 B2 E0
0110: 69 1E 29 20-6F 5B 20 1E-5E 3D 11 2F-3E 4D 9F 39
0120: 8B C9 5C 93-A5 EF A4 22-7D 9A 66 51-6E ED AD 70
0130: 32 90 D4 BD-67 92 38 9B-DC 15 0D BF-DC 71 72 27
0140: E0 5B 43 FA-44 59 E8 60-F7 63 7F F0-73 0A D4 BE
0150: 33 28 AA 99-2C 90 2D D0-01 58 E3 8F-58 50 30 99
0160: E8 60 DB 91-00 13 C9 1D-7A 61 9B 9A-5D 5E BD 71
0170: 23 1A D2 BD-A6 E0 38 66-0B 8C F5 99-56 79 63 D6
0180: 6E 5E D7 7E-C3 4E 9D 5F-65 23 C0 38-C9 55 5A A1
0190: E2 3C CA 78-58 4D B5 3B-04 45 C3 B4-44 C8 87 26
01A0: 02 60 F6 62-91 34 70 FE-C3 34 54 6D-76 07 7F 1A
01B0: 73 53 E6 0B-08 FB 82 80-AD 5F 22 15-18 69 B5 6E
01C0: BB 06 C3 A7-FF 39 15 52-BE FE D4 5C-D2 55 5A 71
01D0: EC E9 BC 1A-B7 BB 08 61-C5 3E E7 89-7C 93 03 FC
01E0: 1F 8A 9A D8-42 BF 6C 01-6A 39 26 84-74 58 E2 E4
01F0: 00 D4 67 7B-27 BD 93 6D-DF F0 10 4A-2B 00 7E 68
0200: 1D DE D5 8A-67 89 EA 52-0C 32 BD 30-A2 8C BE D0
0210: A7 35 BA C6-BB 7D 07 80-49 22 EF E5-10 B2 83 6D
0220: E6 18 6E E3-F0 52 E4 35-83 61 42 35-72 97 C5 8D
0230: 4F F7 93 68-5A 70 5F 5A-04 3A D5 42-C1 FA 0F E2
0240: AE 57 DB AF-F1 51 B8 B7-38 18 EF 2E-B8 A6 A9 2C
0250: 81 87 FA FE-B2 C4 DC 45-A3 64 91 6D-B8 6E F5 D1
0260: 4F 9C FA 62-3D 42 46 59-67 32 EC 99-DA 89 7A 88
0270: E7 AD E3 21-ED 3C 4B C0-4D 9F 83 3C-DC 7F B7 0A
```

Padding

Random buffer
(partial birthday attack bits)

Collision blocks

```
0000: .n .o 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0010: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0020: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0030: 00 00 00 00-00 00 00 00-19 71 E7 F7-09 72 FB 06
0040: F3 45 26 13-66 60 C8 01-B9 2A 75 25-5A 67 23 A6
0050: 92 3D EB 8D-B0 B7 57 F1-45 9F 22 95-BE C0 43 75
0060: 91 98 A2 D3-E0 FD 59 ED-D1 C5 FA 0B-79 65 97 51
0070: B3 B3 E4 0C-11 0C 90 32-DE 4B A1 4B-B8 1B 5E C8
0080: 25 D3 8F 19-CD 10 43 07-D9 BB FF 8C-B7 5A 23 F9
0090: 4D D8 13 14-58 A3 35 97-C5 D1 D4 A9-9A E2 FD 1F
00A0: BA 78 40 00-C3 7E 93 B2-31 A3 6E 2D-34 72 4A C9
00B0: 53 4E C0 45-36 1E C8 6A-56 98 E6 F0-57 1D 61 98
00C0: 13 FC FF CD-4D 83 A2 D2-BB B8 DC 04-2B E2 B8 83
00D0: DB 53 80 D7-3D E9 97 D3-23 5A 27 F9-98 9A E7 56
00E0: 7D 86 E4 35-1E B8 33 EE-EA 15 D1 81-FA 96 62 EC
00F0: 75 31 FB DA-4F AE 24 6F-67 D6 AF 10-96 29 FB C7
0100: A3 32 BB A9-EA D5 E4 AE-1F C2 FB 23-41 22 B2 E0
0110: 69 1E 29 20-6F 5B 20 1E-5E 3D 11 2F-3E 4D 9F 39
0120: 8B C9 5C 93-A5 EF A4 22-7D 9A 66 51-6E ED AF 70
0130: 32 90 D4 BD-67 92 38 9B-DC 15 0D BF-DC 71 72 27
0140: E0 5B 43 FA-44 59 E8 60-F7 63 7F F0-73 0A D4 BE
0150: 33 28 AA 99-2C 90 2D D0-01 58 E3 8F-58 50 30 99
0160: E8 60 DB 91-00 13 C9 1D-7A 61 9B 9A-5D 60 BD 71
0170: 23 1A D2 BD-A6 E0 38 66-0B 8C F5 99-56 79 63 D6
0180: 6E 5E D7 7E-C3 4E 9D 5F-65 23 C0 38-C9 55 5A A1
0190: E2 3C CA 78-58 4D B5 3B-04 45 C3 B4-44 C8 87 26
01A0: 02 60 F6 62-91 34 70 FE-C3 34 54 6D-76 07 FF 1A
01B0: 73 53 E6 0B-08 FB 82 80-AD 5F 22 15-18 69 B5 6E
01C0: BB 06 C3 A7-FF 39 15 52-BE FE D4 5C-D2 55 5A 71
01D0: EC E9 BC 1A-B7 BB 08 61-C5 3E E7 89-7C 93 03 FC
01E0: 1F 8A 9A D8-42 BF 6C 01-6A 39 26 84-6C 58 E2 E4
01F0: 00 D4 67 7B-27 BD 93 6D-DF F0 10 4A-2B 00 7E 68
0200: 1D DE D5 8A-67 89 EA 52-0C 32 BD 30-A2 8C BE D0
0210: A7 35 BA C6-BB 7D 07 80-49 22 EF E5-10 B2 83 6D
0220: E6 18 6E E3-F0 52 E4 35-83 61 42 35-72 97 CD 8D
0230: 4F F7 93 68-5A 70 5F 5A-04 3A D5 42-C1 FA 0F E2
0240: AE 57 DB AF-F1 51 B8 B7-38 18 EF 2E-B8 A6 A9 2C
0250: 81 87 FA FE-B2 C4 DC 45-A3 64 91 6D-B8 6E F5 D1
0260: 4F 9C FA 62-3D 42 46 59-67 32 EC 99-DA 89 7A 08
0270: E7 AD E3 21-ED 3C 4B C0-4D 9F 83 3C-DC 7F B7 0A
```



So, we have two files. Any pair of files.

What a CPC does

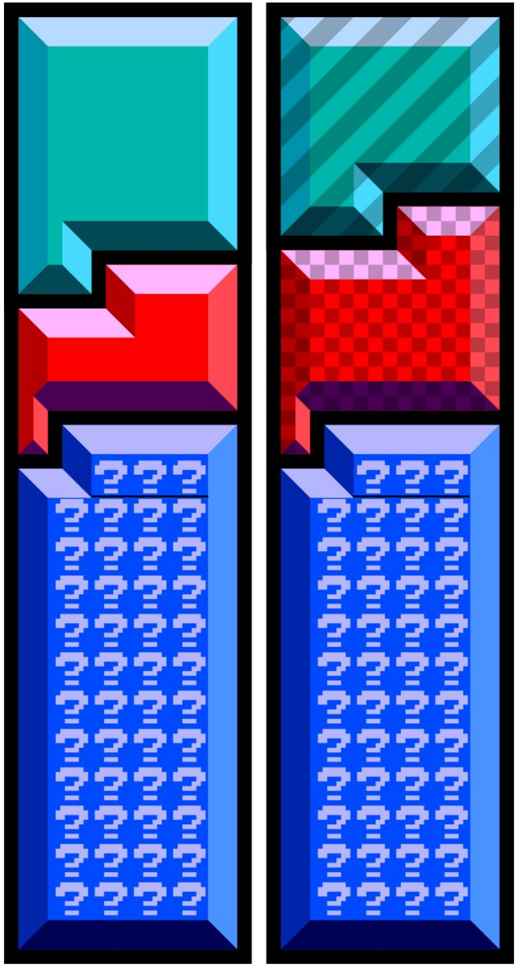
Pad both files to the same length.

We compute a collision,

that appends different blocks to both files.

Makes sense only if

both formats tolerate appended data.

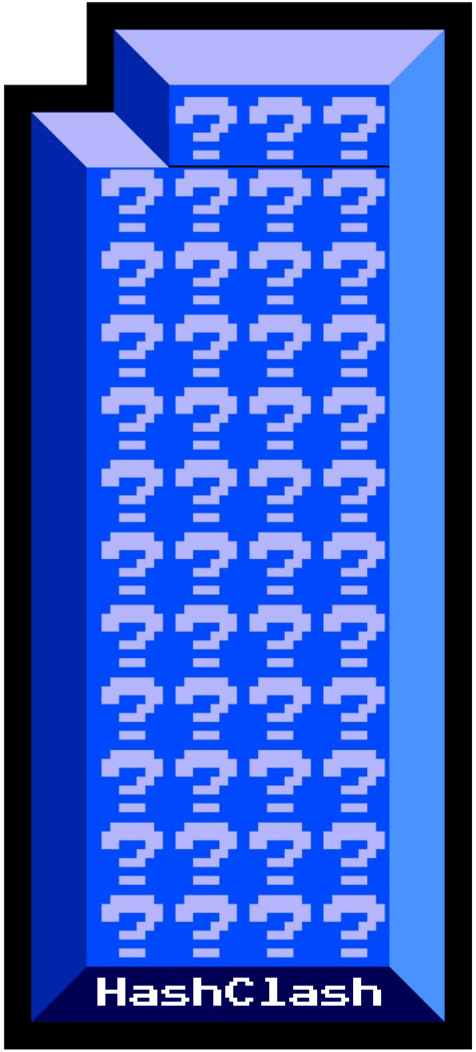


Differences are irrelevant

HashClash

7-9 blocks
A few hours
Irrelevant

Almighty, but slow
(requires some attention to compute)



Input: two arbitrary prefixes

Their content and length don't matter.

Shorter prefixes don't make anything faster.

Both are padded to the same size.

The last 12 bytes before the collision blocks are used for the attack.

- Different on both sides.

After, blocks of collision are appended (by default, 9 of them).

Number of blocks for CPC

CPC is made of 2 steps:

- an initial Birthday Search,
- then near collision computation for each block.

(which may require backtracking)

# blocks	complexity	attack publication
9	2^{39}	2009
3	2^{49}	2009
2	$\mathbf{2^{46}}$	**2019**
1	2^{53}	2009

Only the Birthday Search benefits from GPU.

The fewer the blocks, the more complex the BS: **400kH** for a single block CPC.

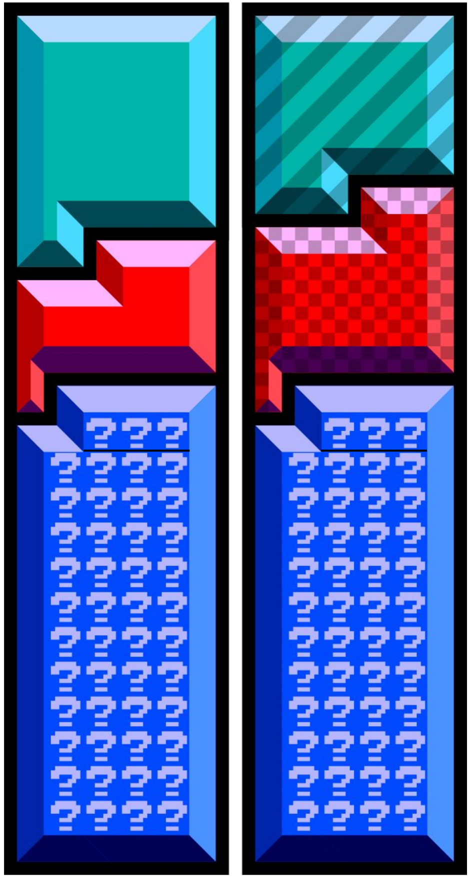
7-9 blocks is a good trade-off for desktop computation.

Impact of CPC

If two files formats tolerate appended data:

Compute collision. Done.

+ Straightforward
- Only works for a single pair.

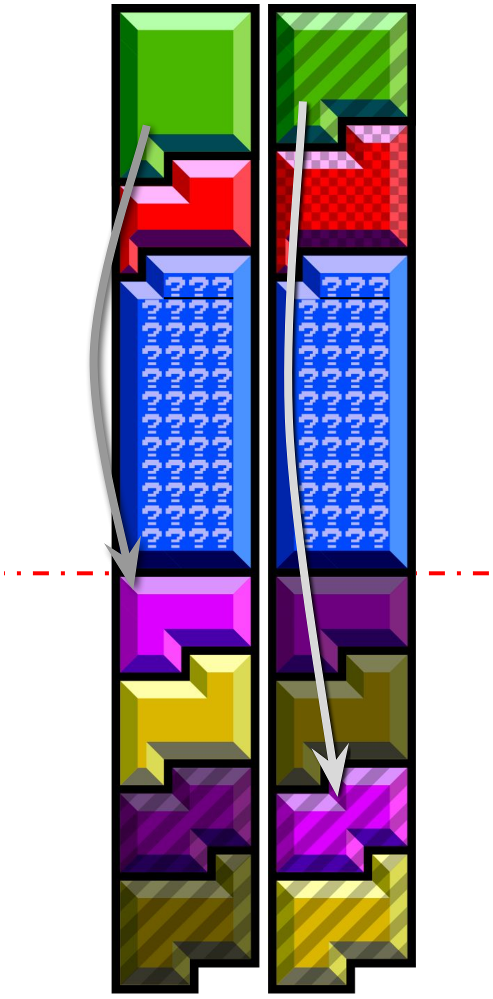


Using CPC as a prefix like an IPC

More computing than IPC, less restrictive.
Ex: PE format.

Do a CPC with headers rather than whole files.
Append body/footer of 2 files.

Enables mixing file types:
- valid/invalid files
- polyglot collisions

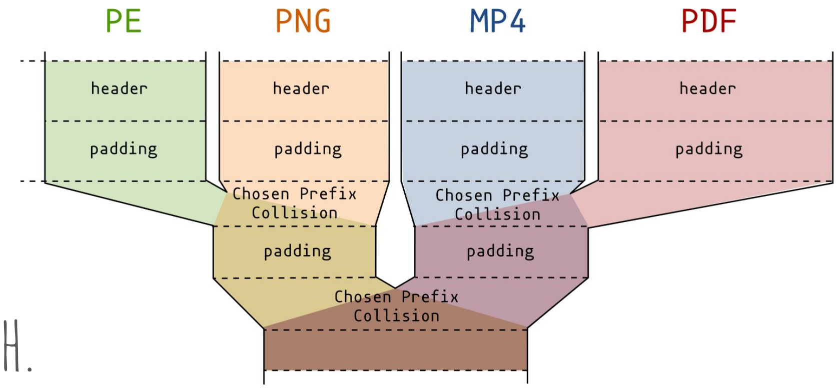


Chaining collisions

A collision makes

two different contents have the same hash.

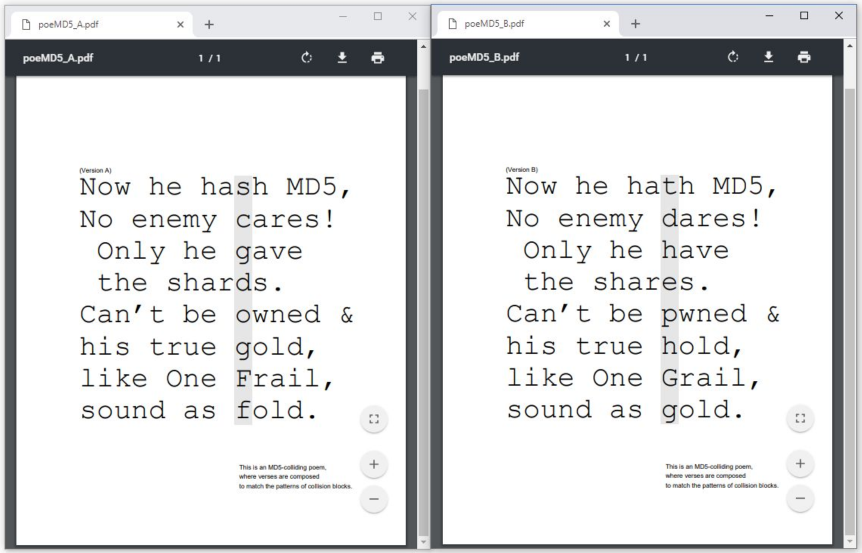
-> they can be chained like a tree.



Top nodes can be an IDC, others CPCs.

-> colliding more than 2 files

N collisions ⇒ N+1 colliding contents



The PoeMD5 chains 8 unicolls that are easy to flip manually.

https://github.com/corkami/collisions#pdf

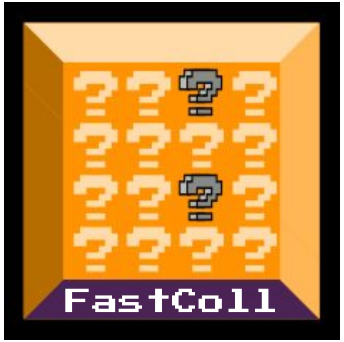

WRAP UP

MD5 is

~~a cryptographic hash~~

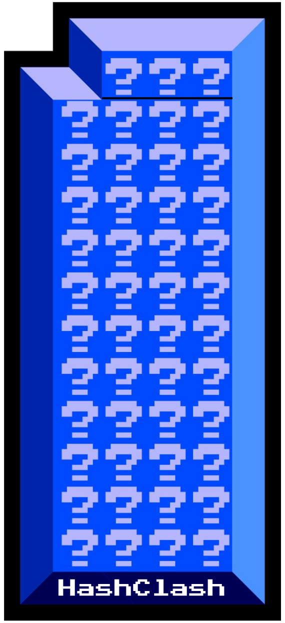
a toy function

...have fun!



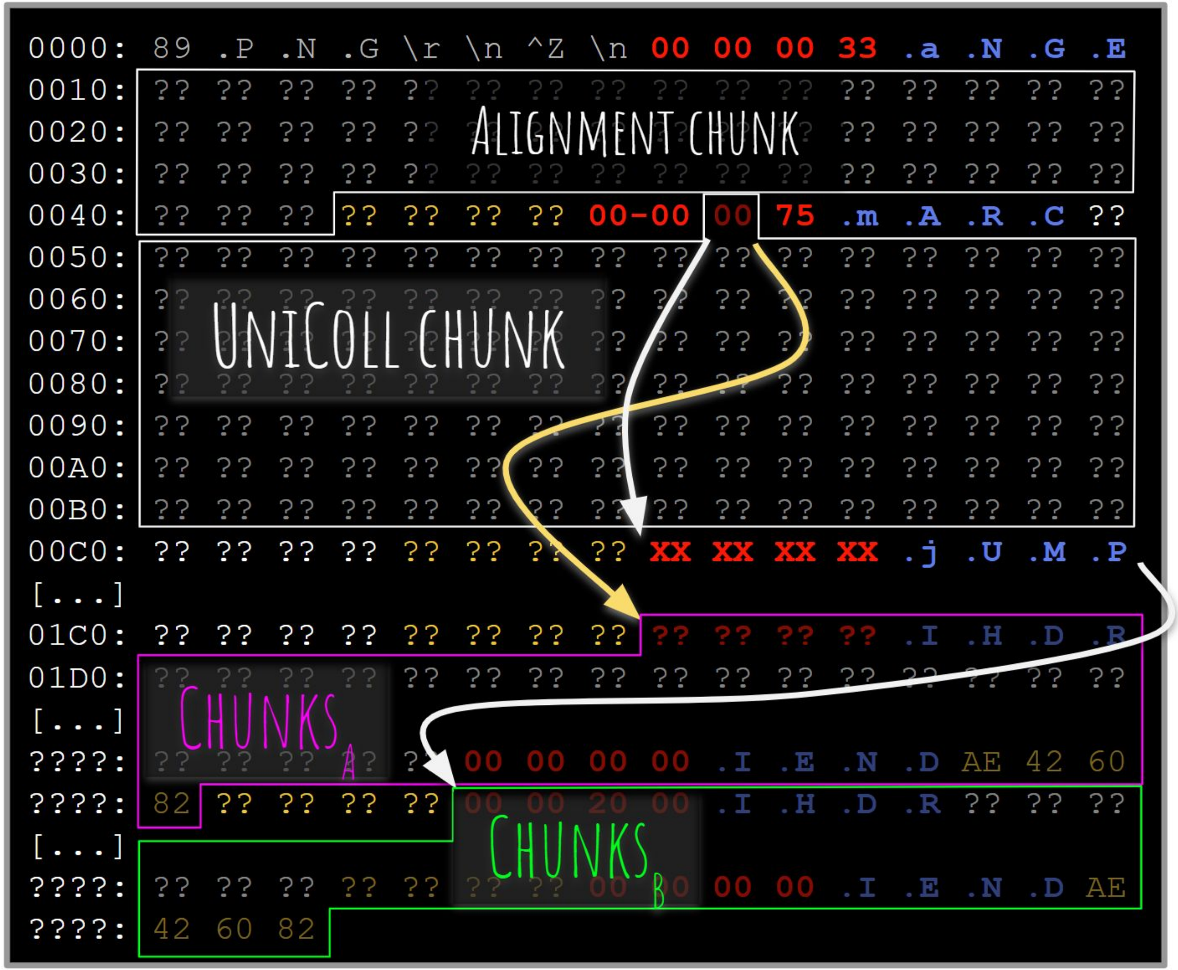
TWO BLOCKS
A FEW SECONDS
IN THE MIDDLE
(AWAY FROM START OR END)

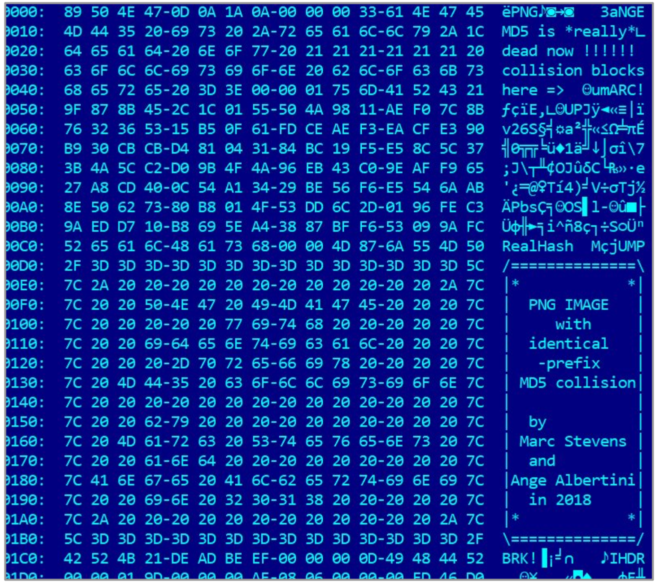
TWO BLOCKS
A FEW MINUTES
IN PREFIX



7-9 BLOCKS
A FEW HOURS
IRRELEVANT

ALL THE KNOWN (IMPLEMENTED) COLLISIONS ATTACKS ON MD5





A TYPICAL LAYOUT FOR AN INSTANT & GENERIC COLLISION VIA UNICOLL

Acknowledgments

They made this workshop possible:

Barbie Auglend, Christophe Brocas, Philippe Teuwen.

They made it better:

Jean-Philippe Aumasson, Nicolas Grégoire.

Thank you for making it this far!

Any feedback is welcome!

LET'S PLAY COLLTRIS

ANGE ALBERTINI

reverse engineering

VISUAL DOCUMENTATIONS

@angealbertini

ange@corkami.com

http://www.corkami.com

EXTRAS

* RESTRICTIONS

File format	Comment length	Generic collision	FastColl	UniColl	Shattered	HashClash
PDF	32	✓		✓ *		✓
JPG	16	✓ *		✓ *	✓ *	✓
PNG	32	✓ / ✗		✓ *		✓
MP4	32/64	✓ *		✓ *	✓ *	✓
PE	?	✓				✓
GIF	8	✗	✓ *			✓
ZIP	16	✗		✓ *		✓
ELF/TAR Mach-O/Class		✗				✓

UniColl attack patterns

PDF

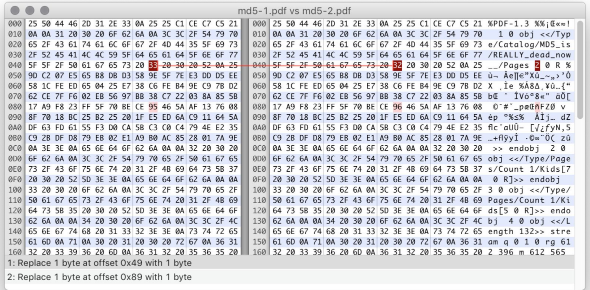
Merge documents, split **/Kids** in 2 part showing pages sets separately.

Declare a **/Catalog** objects that has its **/Pages** as object **2**.

```
0040: .. .. ./ .P .a .g .e .s .  .2 .  .0 . .R \n .%
```

The other file will have its pages referenced as object **3**.

```
0040: .. .. ./ .P .a .g .e .s .  .3 .  .0 . .R \n .%
```



More details @ https://github.com/corkami/collisions#pdf

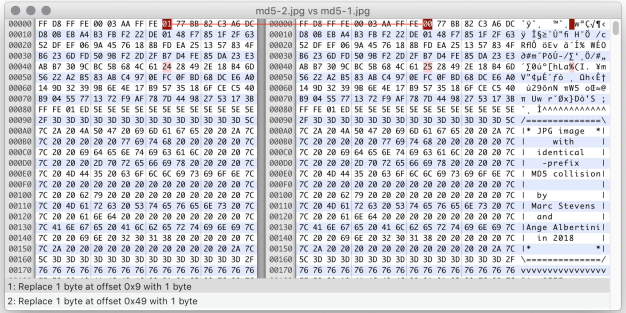
JPG

USE **FF FE** COMMENTS FOR ALIGNEMENT, THEN A COMMENT OF LENGTH 0x77

`0000: FF D8 FF FE-00 03 .. FF FE 00 77 .. .. .. .. ..`

THE OTHER FILE WILL HAVE A LONGER CHUNK OF 0x177.

`0000: FF D8 FF FE-00 03 .. FF FE 01 77 .. .. .. .. ..`



CF https://github.com/corkami/collisions#jpg

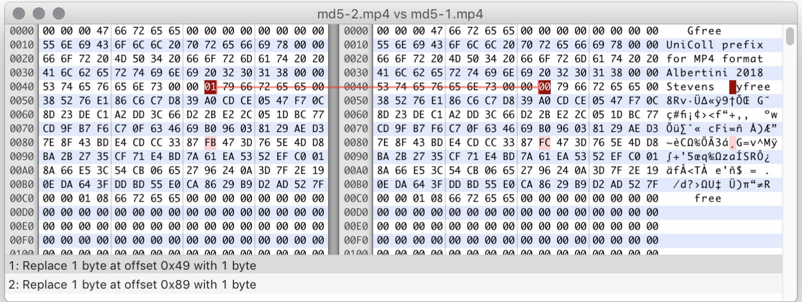
MP4

USE **FREE** ATOMS, FOR ALIGNEMENT THEN OF LENGTH `0x79`

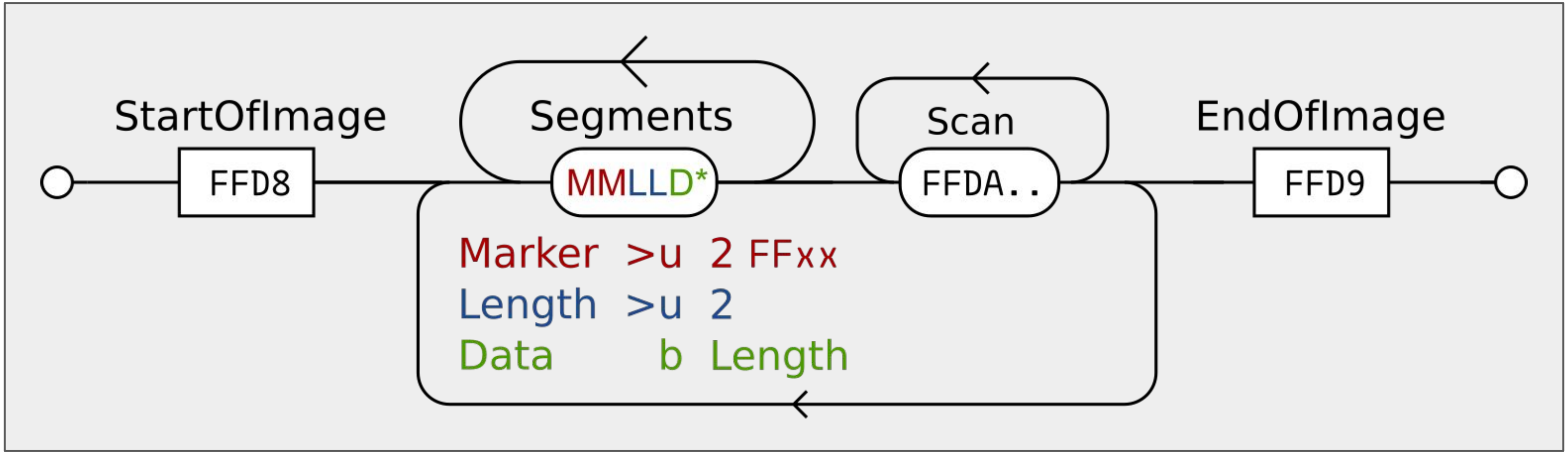
`0040: .. .. .. .. .. .. .. 00 00 00 79 .F .R .E .E ..`

THE OTHER FILE WILL HAVE A LONGER CHUNK OF `0x179`.

`0040: .. .. .. .. .. .. .. 00 00 01 79 .F .R .E .E ..`



More details @ https://github.com/corkami/collisions#mp4-and-others



COMMENT SEGMENT IN JPG: FF FE

SCANS CAN BE BIGGER THAN 64 KB -> SPLIT THEM VIA SAVING AS PROGRESSIVE

JPEG FILE STRUCTURE

Other UniColl variants (for completeness)

```
0000:  .U .n .i .C .o .l .l .  .2 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b 24 FA 3F 50 2F 7A B1 A7 04 DC 2F 39
0020:  07 E7 6F 33 B4 64 97 DD B1 95 8E F3 CB 60 18 B1
0030:  9F E9 DC B3 D8 03 FC 7C 52 40 8E 36 AF 0C 86 C7
0040:  8C 41 62 73 C9 B9 A7 EB 03 10 68 F0 5B 82 49 EE
0050:  B6 77 D5 50 E2 B8 D7 A2 61 16 78 B0 35 24 1B 2F
0060:  5A 83 E2 E0 49 4F B7 0D 7D 7C E7 3F CC B7 F3 72
0070:  8A 55 71 A0 B2 34 6C 0E 45 EE 04 60 ED 33 62 BC
```

- less predictable difference

+ 16 fixed bytes after the first difference

```
0000:  .U .n .i C3 .o .l .l .  .2 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b 24 FA 3F 50 2F 7A B1 27 04 DC 2F 39
0020:  07 E7 6F 33 B4 64 97 DD B1 95 8E F3 CB 60 18 B1
0030:  9F E9 DC B3 D8 03 FC 84 52 40 8E 36 AF 0C 86 C7
0040:  8C 41 62 F3 C9 B9 A7 EB 03 10 68 F0 5B 82 49 EE
0050:  B6 77 D5 50 E2 B8 D7 A2 61 16 78 30 35 24 1B 2F
0060:  5A 83 E2 E0 49 4F B7 0D 7D 7C E7 3F CC B7 F3 72
0070:  8A 55 71 A0 B2 34 6C 06 45 EE 04 60 ED 33 62 BC
```

Difference on the last byte

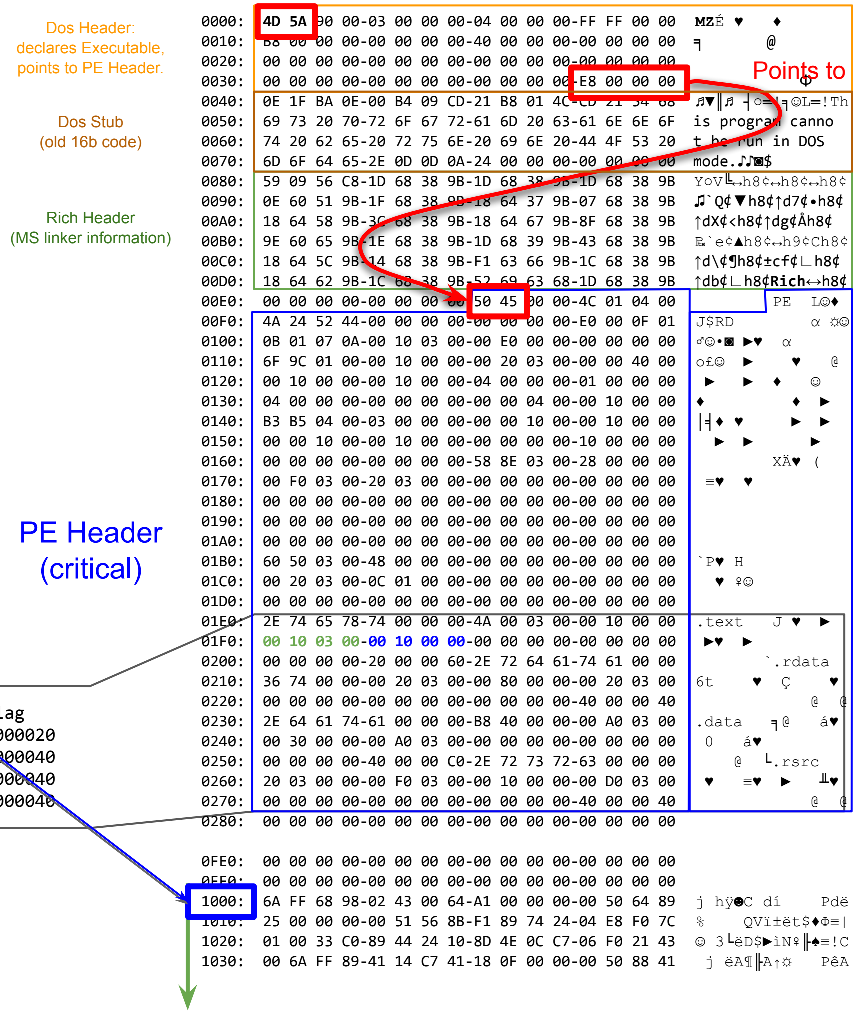
```
0000:  .U .n .i .C .o .l .l .  .3 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b EC D2 0C 56 2F 03 F6 66 D1 76 8F 87
0020:  FF E4 7B EC F3 31 0A 65 66 B5 BD 6D F5 2B FD 1E
0030:  4D 2D 99 37 0C B6 1B D5 63 94 DC 2E DB 97 F2 10
0040:  22 BA 25 C4 F6 F7 EC C6 D7 0E DB 5D 18 DF 90 F9
0050:  6A C5 2A 0A CC 88 3C 7F 6C AE 24 71 F9 BF 76 17
0060:  BE 60 AA DE 6F 0B 11 D0 52 E2 0E 85 BB 0B 8B 76
0070:  A1 18 87 03 D2 9D 39 80 79 10 50 3F BC 17 65 01
```

```
0000:  .U .n .i .C .o .l .l .  .3 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b EC D2 0C 56 2F 04 F6 66 D1 76 8F 87
0020:  FF E4 7B EC F3 31 0A E5 66 B5 BD 6D F5 2B FD 1E
0030:  4D 2D 99 37 0C B6 1B D5 63 94 DC 2E DB 97 F2 90
0040:  22 BA 25 C4 F6 F7 EC C6 D7 0E DB 5D 18 DF 90 F9
0050:  6A C5 2A 0A CC 88 3C 7F 6C AD 24 71 F9 BF 76 17
0060:  BE 60 AA DE 6F 0B 11 50 52 E2 0E 85 BB 0B 8B 76
0070:  A1 18 87 03 D2 9D 39 80 79 10 50 3F BC 17 65 81
```

PE collisions
via a CPC used like an IPC

Anatomy of a PE file

- DOS header points to PE header
- In between, DOS Stub (16 bit code), Rich header (MS Linker information)
- PE header contains all the critical information



Number	Name	VSize	Address	PSize	Offset	Flag
1	.text	0003004A	00001000	00031000	00001000	60000020
2	.rdata	00007436	00032000	00008000	00032000	40000040
3	.data	000040B8	0003A000	00003000	0003A000	C0000040
4	.rsrc	00000320	0003F000	00001000	0003D000	40000040

Abusing PE files

- DOS HEADER only contains 2 important fields, the rest is irrelevant
- Dos Stub and Rich header can be removed.
- PE HEADER can be moved further: just update POINTER.
- Sections can be moved further: just adjust offsets

only Magic and pointers are important

can be discarded

can be discarded

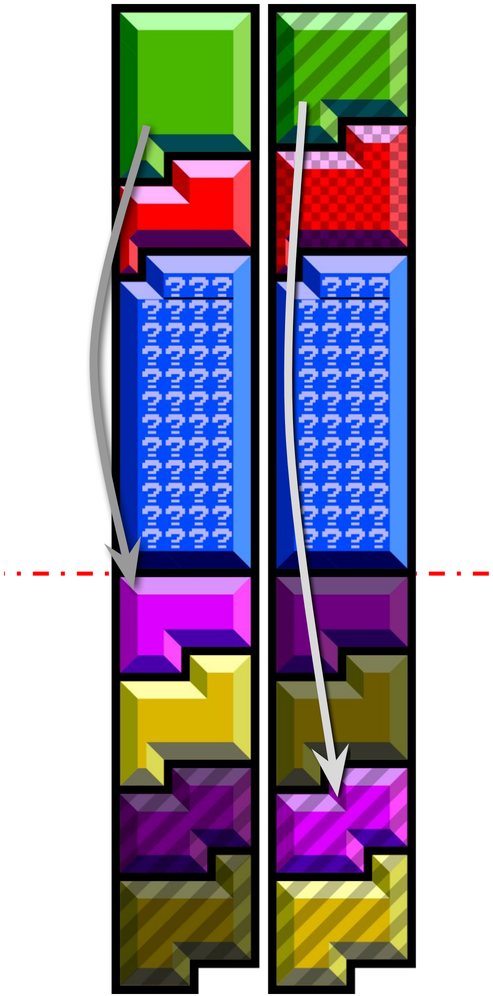
```
0000:  4D 5A 90 00-03 00 00 00-04 00 00 00-FF FF 00 00   MZÉ  ♥    ♦
0010:  B8 00 00 00-00 00 00 00-40 00 00 00-00 00 00 00   ╕        @
0020:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0030:  00 00 00 00-00 00 00 00-00 00 00 00-E8 00 00 00               Φ
0040:  0E 1F BA 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68
0050:  69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F   is program canno
0060:  74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20   t be run in DOS
0070:  6D 6F 64 65-2E 0D 0D 0A-24 00 00 00-00 00 00 00   mode.♪♪◙$
0080:  59 09 56 C8-1D 68 38 9B-1D 68 38 9B-1D 68 38 9B
0090:  0E 60 51 9B-1F 68 38 9B-18 64 37 9B-07 68 38 9B
00A0:  18 64 58 9B-3C 68 38 9B-18 64 67 9B-8F 68 38 9B
00B0:  9E 60 65 9B-1E 68 38 9B-1D 68 39 9B-43 68 38 9B
00C0:  18 64 5C 9B-14 68 38 9B-F1 63 66 9B-1C 68 38 9B
00D0:  18 64 62 9B-1C 68 38 9B-52 69 63 68-1D 68 38 9B              Rich
00E0:  00 00 00 00-00 00 00 00-50 45 00 00-4C 01 04 00           PE  L☺♦
00F0:  4A 24 52 44-00 00 00 00-00 00 00 00-E0 00 0F 01   J$RD        α ☼☺
0100:  0B 01 07 0A-00 10 03 00-00 E0 00 00-00 00 00 00
0110:  6F 9C 01 00-00 10 00 00-00 20 03 00-00 00 40 00
0120:  00 10 00 00-00 10 00 00-04 00 00 00-01 00 00 00
0130:  04 00 00 00-00 00 00 00-00 00 04 00-00 10 00 00
0140:  B3 B5 04 00-03 00 00 00-00 00 10 00-00 10 00 00
0150:  00 00 10 00-00 10 00 00-00 00 00 00-10 00 00 00
0160:  00 00 00 00-00 00 00 00-58 8E 03 00-28 00 00 00
0170:  00 F0 03 00-20 03 00 00-00 00 00 00-00 00 00 00
0180:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0190:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
01A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
01B0:  60 50 03 00-48 00 00 00-00 00 00 00-00 00 00 00   `P♥ H
01C0:  00 20 03 00-0C 01 00 00-00 00 00 00-00 00 00 00
01D0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
01E0:  2E 74 65 78-74 00 00 00-4A 00 03 00-00 10 00 00   .text   J ♥
01F0:  00 10 03 00-00 20 00 00-00 00 00 00-00 00 00 00
0200:  00 00 00 00-20 00 00 60-2E 72 64 61-74 61 00 00          `.rdata
0210:  36 74 00 00-00 20 03 00-00 80 00 00-00 30 03 00   6t
0220:  00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 40               @  @
0230:  2E 64 61 74-61 00 00 00-B8 40 00 00-00 A0 03 00   .data
0240:  00 30 00 00-00 B0 03 00-00 00 00 00-00 00 00 00    0
0250:  00 00 00 00-40 00 00 C0-2E 72 73 72-63 00 00 00       @  └.rsrc
0260:  20 03 00 00-00 F0 03 00-00 10 00 00-00 E0 03 00
0270:  00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 40               @  @
0280:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00

2000:  6A FF 68 98-02 43 00 64-A1 00 00 00-00 50 64 89
2010:  25 00 00 00-00 51 56 8B-F1 89 74 24-04 E8 F0 7C
2020:  01 00 33 C0-89 44 24 10-8D 4E 0C C7-06 F0 21 43
2030:  00 6A FF 89-41 14 C7 41-18 0F 00 00-00 50 88 41
```

Number	Name	VSize	Address	PSize	Offset	Flag
1	.text	0003004A	00001000	00031000	00002000	60000020
2	.rdata	00007436	00032000	00008000	00033000	40000040
3	.data	000040B8	0003A000	00003000	0003B000	C0000040
4	.rsrc	00000320	0003F000	00001000	0003E000	40000040

CPC-IPC exploitation for PE files

1. Craft 2 headers
2. Compute CPC (>3 hours)
3. Use both prefixes with 2 sets of data

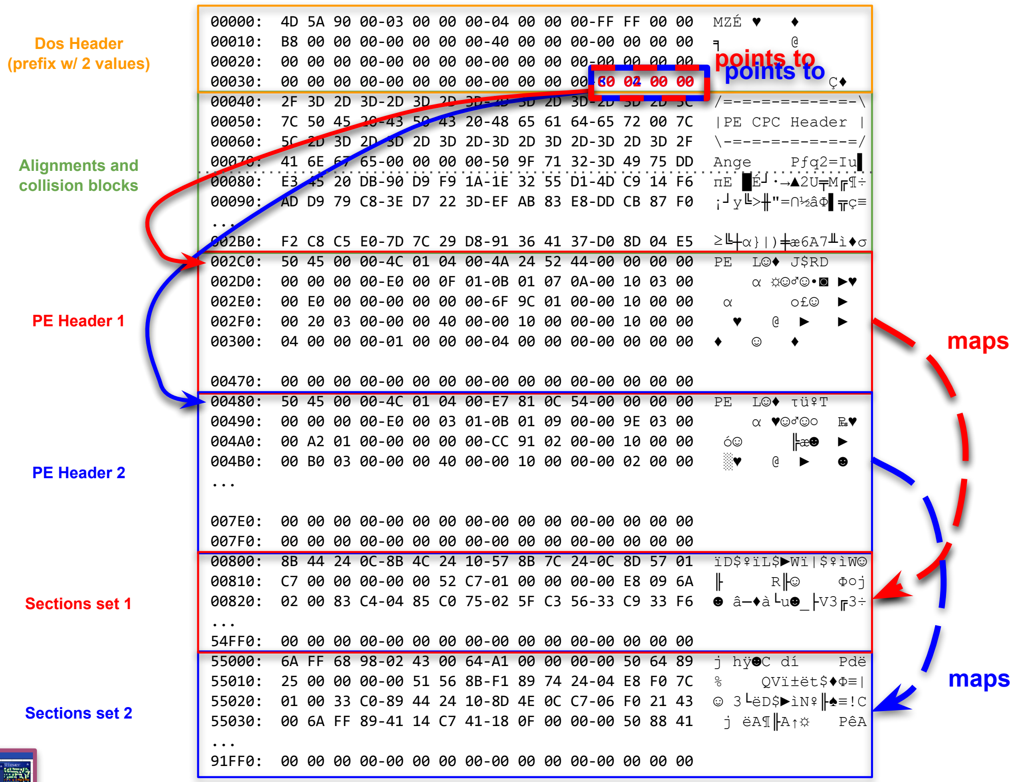
-> instant collision of any pair of PE files
(with no code modification)



Windows PE collisions

- DOS header is generic
  Pointers to 2 headers, over collision blocks.
- Dos Stub and Rich header are discarded
  to make place for collision blocks
- Two PE Headers that follow each other
- Both sections sets have adjusted offsets.

-> Reusable and instant PE collision



Shattered: a SHA-1 IPC

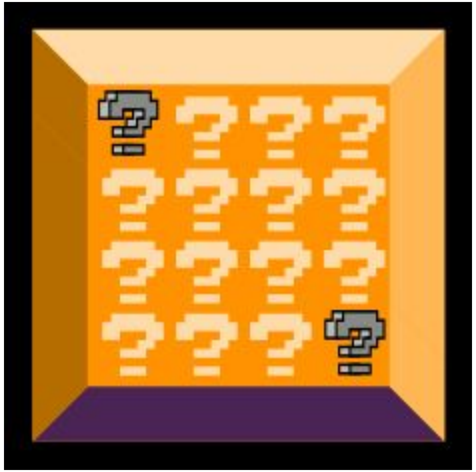
Shattered

- An IPC for Sha1
- Computed only once (?)
- Differences at start and end

-> "easy" to exploit

Official PoCs = JPGs in PDFs

PDFs embed JPGs natively



2 blocks

6K years

Start & end

Length / Type / Value <-> Type / Length / Value

Most formats declare lengths before type (**LTV**):

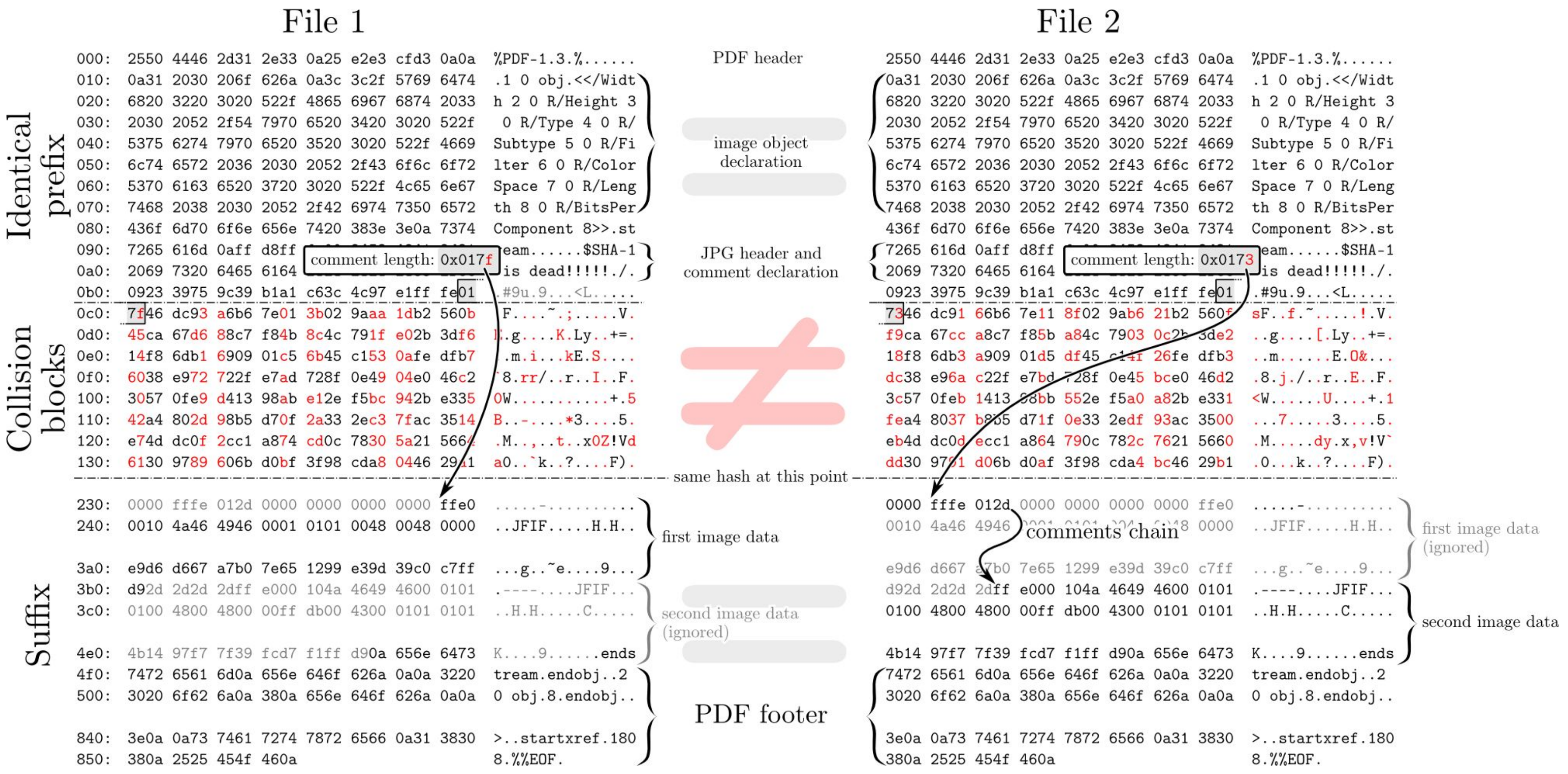
-> not good for hash collisions (type declaration is in random bytes)

JPG & MP4* are **TLV** & big endian -> exploitable w/ Shattered

Declare comment (`FF FE` for JPG, `free` for MP4)

then abuse length with collision difference.

*with 64b lengths



Shattered files layout

For more details about Shattered exploitation:

https://speakerdeck.com/ange/exploiting-hash-collisions (2017)

New in 2019

From Collisions to Chosen-Prefix Collisions Application to Full SHA-1

Gaëtan Leurent[1] and Thomas Peyrin[2,3]

SHA-1: faster practical CPC

(never computed yet)

MD5: More efficient CPC in 2 blocks.

(little impact)

Current hash collision complexity

SHA1

IPC

2^{65} 2017 Stevens (Shattered) The first collision for full SHA-1

CPC

2^{77} 2013 Stevens New collision attacks on SHA-1

2^{67} **2019 Leurent From Collisions to Chosen-Prefix Collisions**

MD5

IPC

2^{16} 2009 Stevens (FastColl) Short chosen-prefix collisions for MD5

CPC

2^{39}: 9 blocks 2009 Stevens (HashClash) Short chosen-prefix collisions for MD5

2^{53}: 1 block … …

2^{46}: 2 blocks **2019 Leurent From Collisions to Chosen-Prefix Collisions**