2019/12/03
229 slides

HASH COLLISION EXPLOITATION

WITH FILES

A WORKSHOP BY

ANGE ALBERTINI

WITH THE HELP OF

MARC STEVENS

A.K.A.

Sessions:
2019/07/02 150p @ Pass The Salt
2019/07/24 199p @ Google
2019/08/19 208p @ Google
2019/10/23 222p @ Hack.lu
2019/11/07 225p @ Black Alps
2019/12/03 229p @ Google

Welcome

I made this deck to share my knowledge, but also to **learn** from you.
The slides are public and have been improved several times whenever needed.

It may not cover all perspectives or answer all questions,
so feel free to
Reach me at @angealbertini or ✉ ange@corkami.com
with questions, one-liners, suggestions...

This desk's URL is:
https://speakerdeck.com/ange/colltris
Make sure you have the latest version.

color blind ?
LMK if you're having any troubles.

TL;DR (for experts)

Hash collisions existed for more than a decade and are often misunderstood:

- exageration: "MD5 is broken, just don't use it!"
- understatement: "It always takes hours!"

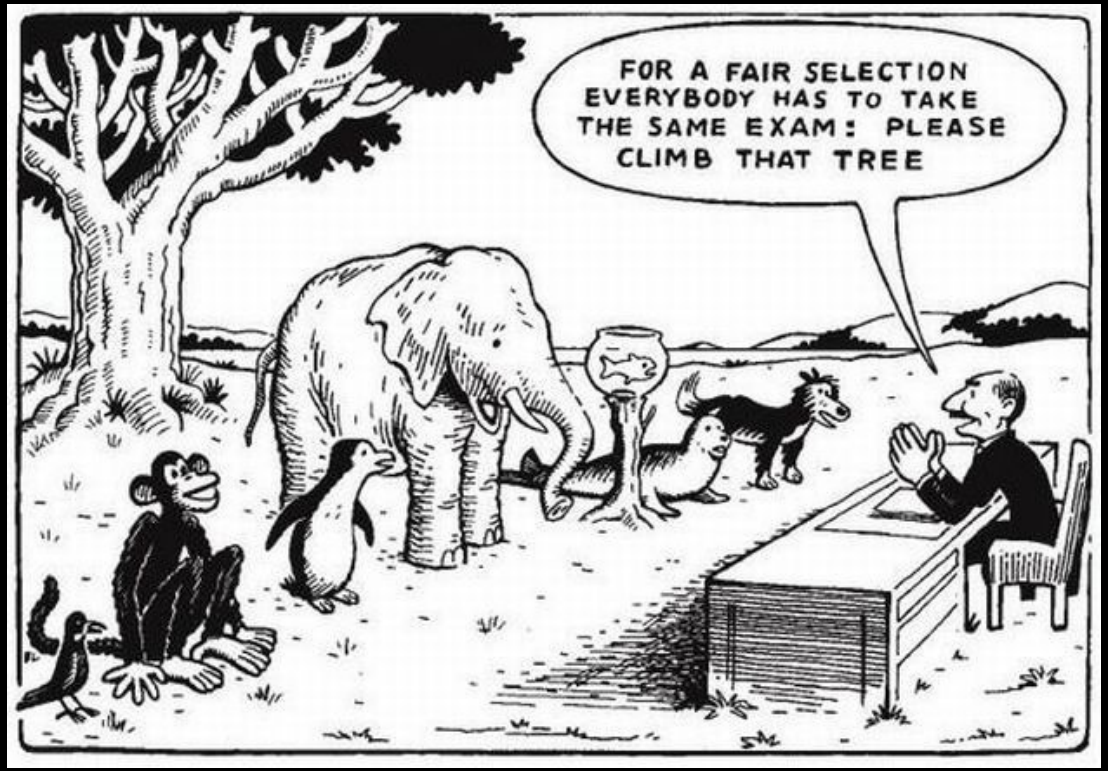
Generating colliding files can be sped up (from hours to instant)
via tricks specific to some file formats: which ones, and why?

This workshop aims to clarify these topics (and avoid the crypto).

Please be considerate.

No gatekeeping, no dogma, no cult.

Don't show off, share knowledge.
It's just better for everyone.



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*

Knowledge points checklist (✓ / ✗)

- [ ] Hash collisions attacks: FastColl/UniColl/HashClash/Shattered.
- [ ] File formats abuses: shuffling, parasites, polyglots.
- [ ] Hash functions: MD5/SHA1, blocks, length extension.
- [ ] File formats: magic, header, body, chunks, footer.
- [ ] Hexadecimal / ASCII / hex viewer / endianness.

Contents

Introduction

- Goals
- Basics
- Prerequisites

Main

- $Coll_1$: FastColl
- File formats basics
- $Coll_2$: UniColl
  - $Exploit_1$: PNG
- $Coll_3$: HashClash*
  - $Exploit_2$: PE
- $Coll_4$: Shattered

Final

- Wrap up
- Extras
  - $Exploit_3$: GIF

There are **only 4** existing collisions attacks!

*HashClash is actually the name of the whole project.
But HashClashCPC is too long :)

About the author

- Reversing since the late 80's
- Author of Corkami
- 6 years at PoC or GTFO*

- Occasional drawer, singer

Professionally

- 13 years of malware analysis
- 1 year of security research

My license plate is a CPU,
my phone case is a PDF doc,
my resume is a PDF/SNES/Genesis rom.

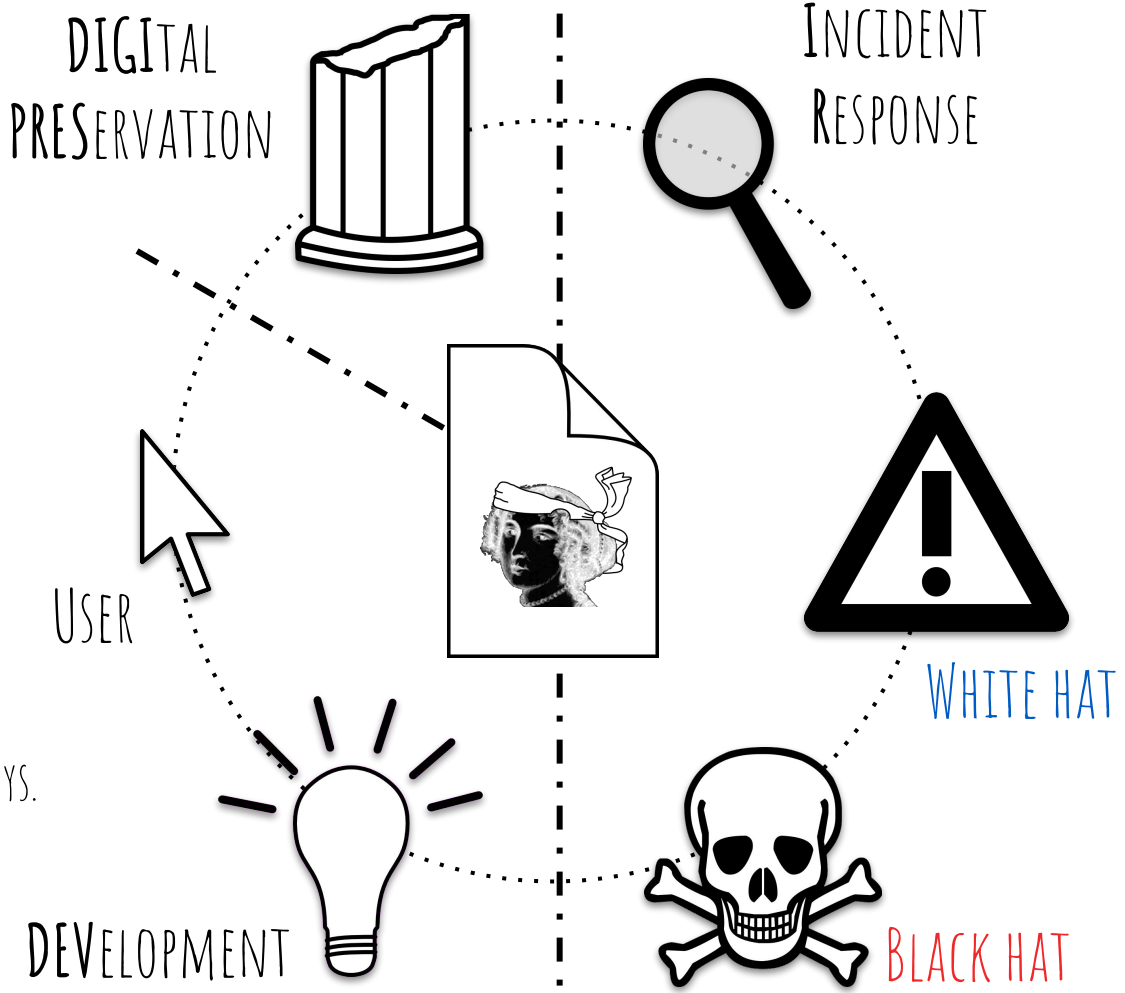
There are various

(with a few things in common)

communities around file formats

...and I'm interested in **all** of them.

My life is about file formats - they're my toys.



A gentle introduction to...

Hash functions

(only 5 slides)

What's a hash function? MD5, SHA1...

Also called 'checksum'

Returns from any content a big fixed-size value, always different.

␣ → d41d8cd98f00b204e9800998ecf8427**e**

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

- 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'
>>> _
```

This example uses the crypt(3) hash.

Hash collision != password cracking

Password cracking (HashCat, John the Ripper):

finds a **string** that matches a value, a hash.

Hash collision (HashClash, Shattered):

Make $\text{Content}_{\text{Good}}$ and $\text{Content}_{\text{Bad}}$ with the same hash.

What's the extent of a hash collision?

It's **impossible** to generate a file with predetermined hash (pre-image attack)
with MD5 or SHA1. But Maraca and Snefru were broken.

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

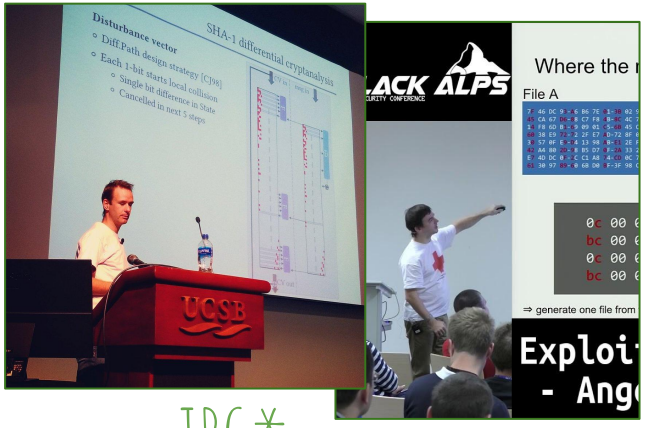
With some file types, we can **instantly** generate files
that render the same way (via some tricks).

Cf Lessons from the history of attacks on secure hash-functions

Results - 1/2

Instant MD5 collisions, with no recomputation

https://github.com/corkami/collisions

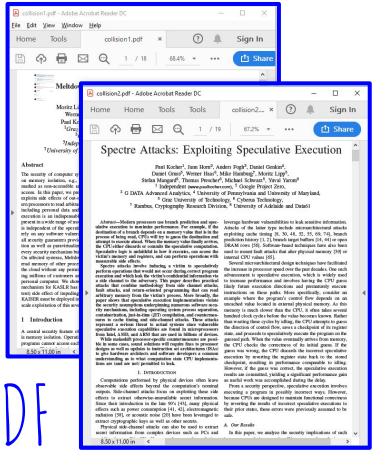


JPG*

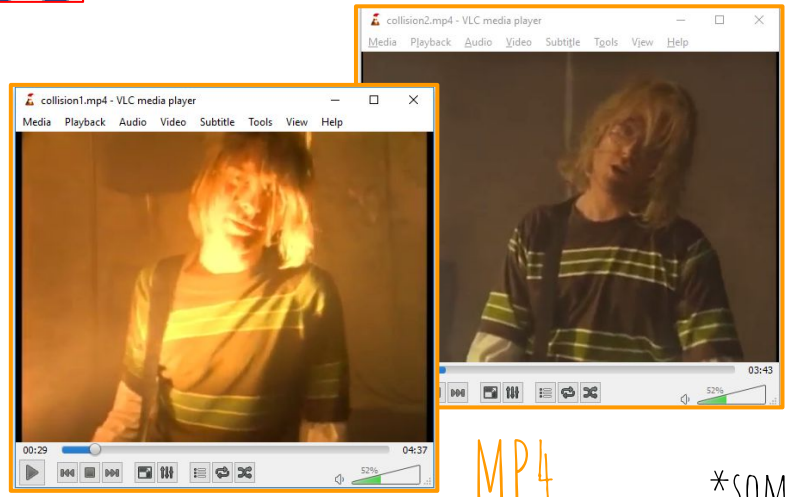
Sega

NINTENDO

PNG*



PDF



MP4

*some limitations

* WITH SOME LIMITATIONS

RESULTS - 2/2

GIF*

PE

JP2

Just new collisions?

```
$ time ./png.py yes.png no.png

real    0m0.039s
user    0m0.025s
sys     0m0.017s
$ md5sum collision*.png
7af5775114be02b9b2594418a68a4cb8  collision1.png
7af5775114be02b9b2594418a68a4cb8  collision2.png
$
```

Instant, reusable and generic collisions:

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

Ex: script -> output recording

In some cases (PDFs), the colliding files are 100% standard:

From a parser perspective,

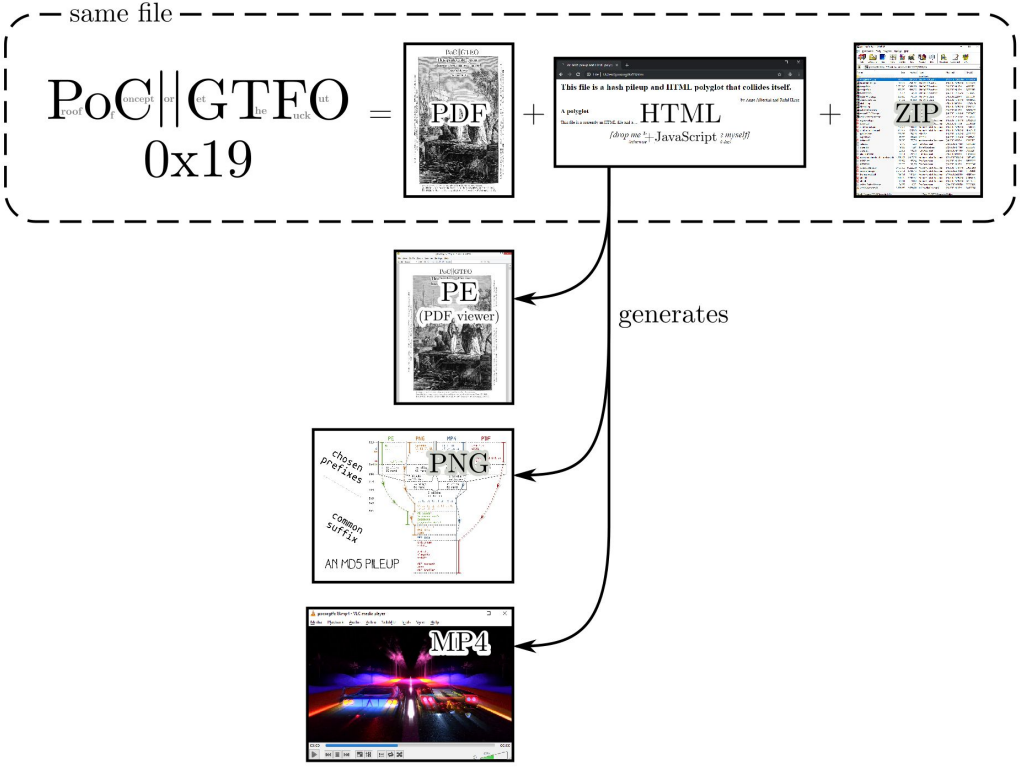
the contents are unmodified: only the files' structures are.

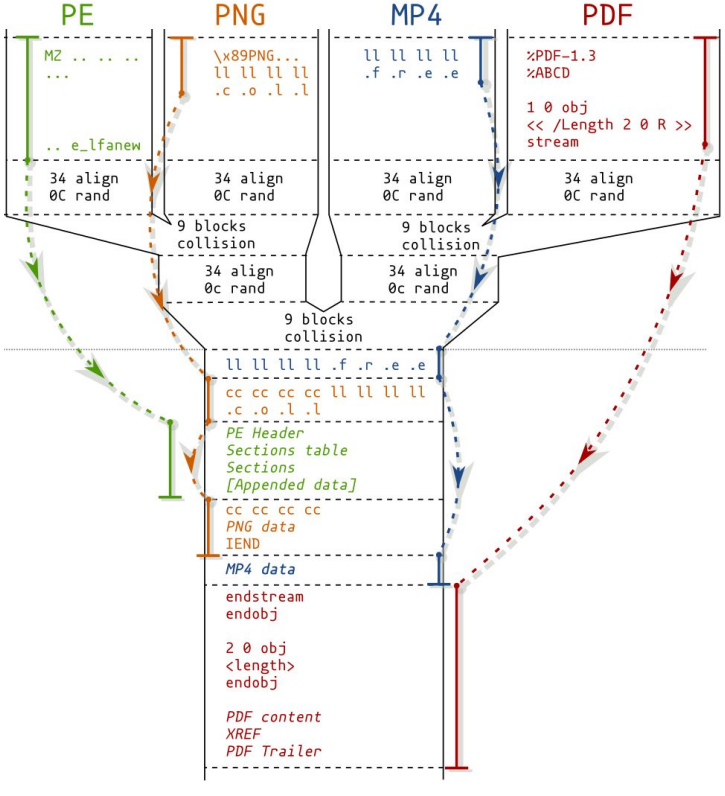
Demystifying long-lasting myths

Hash collisions are usually perceived to apply only to:

1. a pair of files
2. of the same file type

3. Colliding files are expected to be very different.



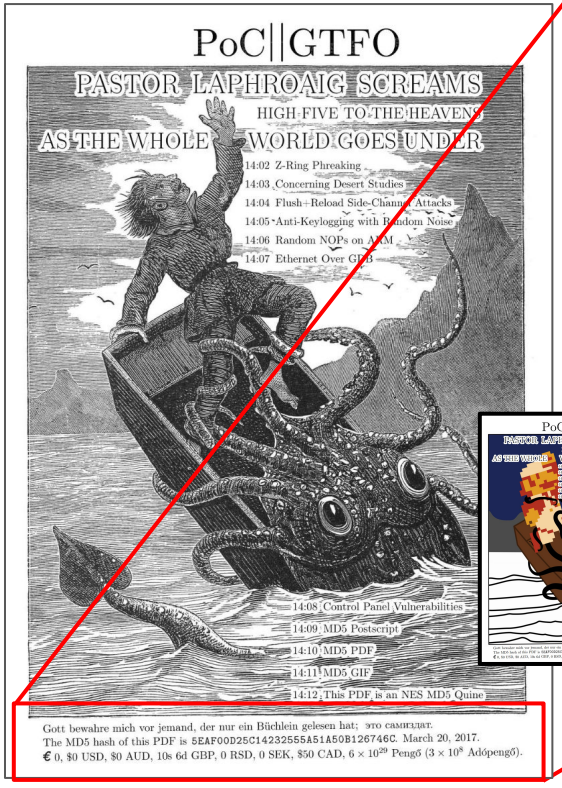


INSTANT & GENERIC PDF/PE/PNG/MP4 COLLISION

A MULTI-TYPE QUARTET OF AN EXECUTABLE, IMAGE, VIDEO, DOCUMENT.

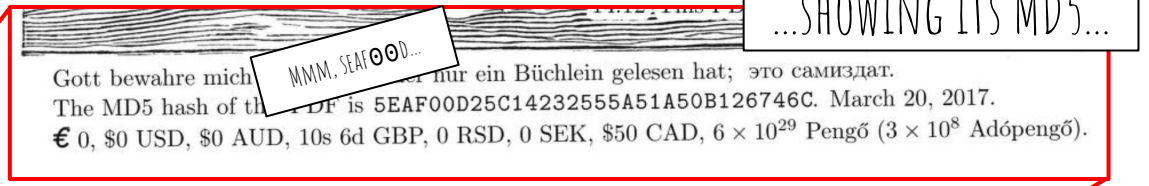
https://github.com/angea/pocorgtfo/blob/master/README.md#0x19

A 60 page LaTeX-generated PDF...



<= alternate cover but same MD5!

...showing its MD5...



...also a NES rom...

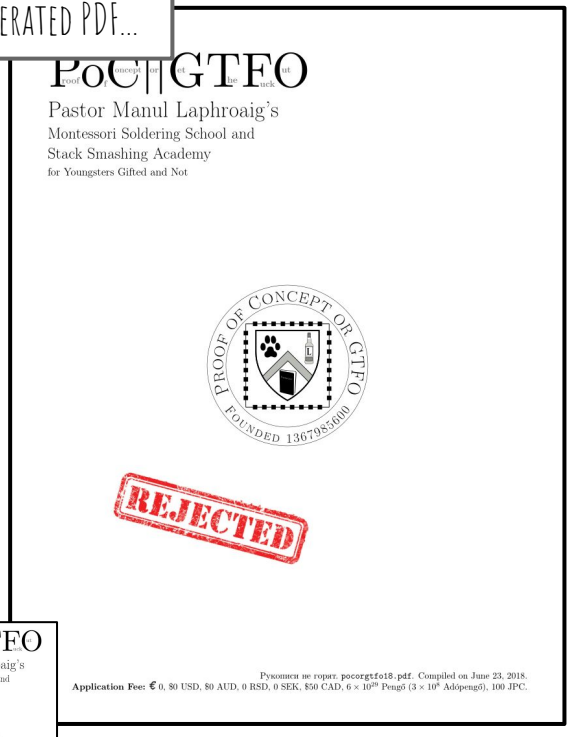
Tiny change (text), same MD5

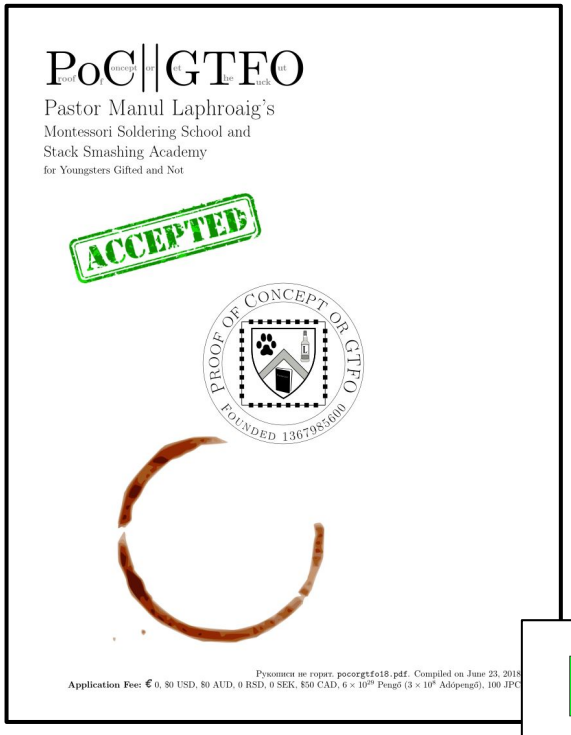
...showing the same MD5!

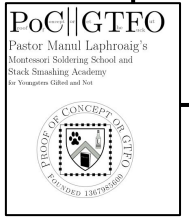
https://github.com/angea/pocorgtfo/blob/master/README.md#0x14

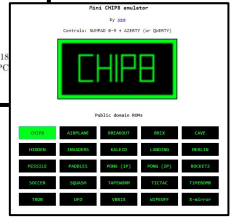
609 FastColls in the file!

A 64 page LaTeX-generated PDF...









Two covers via a "dual-content" JPG

and 2 payloads via HTML polyglot

Tiny change (background image), same SHA1

https://github.com/angea/pocorgtfo/blob/master/README.md#0x18 (howto)

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 reusable with **future** collision attacks:

the same JPEG trick was reused with 3 hash collisions (MD5, MalSHA1, SHA1)


GOALS

Goals of this workshop

- file formats structures and manipulations
- Understand hash collisions attacks and their exploits
- Create your own exploits

their impacts and limits,
without all the internal details

A system that you study indexes files with MD5: what can you do?

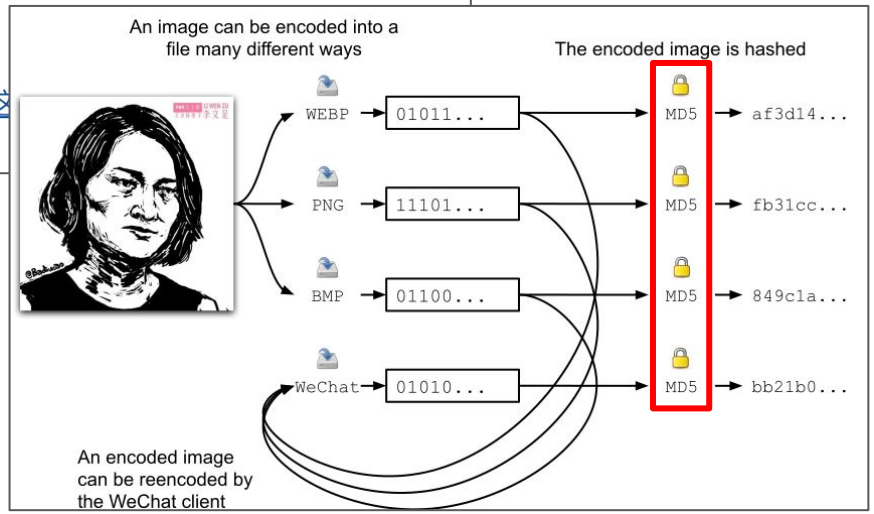
"No one uses MD5 anymore!" ?

https://citizenlab.ca/2019/07/cant-picture-this-2-an-analysis-of-wechats-realtime-image-filtering-in-chats/

(Can't) Picture This 2

An Analysis of WeChat's Realtime Image Filtering in Chats

By Jeffrey Knockel and Ruohan Xiong July 15, 2019 点击这



You may have heard of...

Crypto attack pwnie

Best CRYPTO17 paper

The Shattered attack

Nominated for Péter Szőr award

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

What this slide deck is about
(more details about shattered here)

Official paper
Presentations:
- Marc (crypto) video
- Pierre (computation) video / slides
- Elie (high level) video / slides
- Ange (file formats) video / slides

Uses of hashes

- check if contents have 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 computation is usually enough.

*rendering-wise, not structure-wise.

Having troubles to convince?

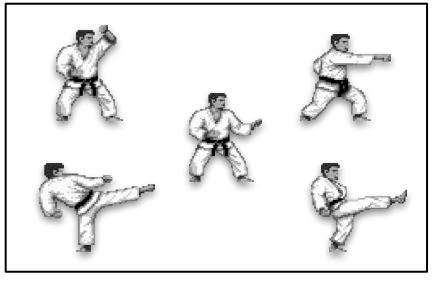
Let files do the talking.

Threats? theory...

Exploits PoCs? reality!

immediate threat

Theoretical attacks to put in practice



MD5 collisions: a good hacking challenge

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).

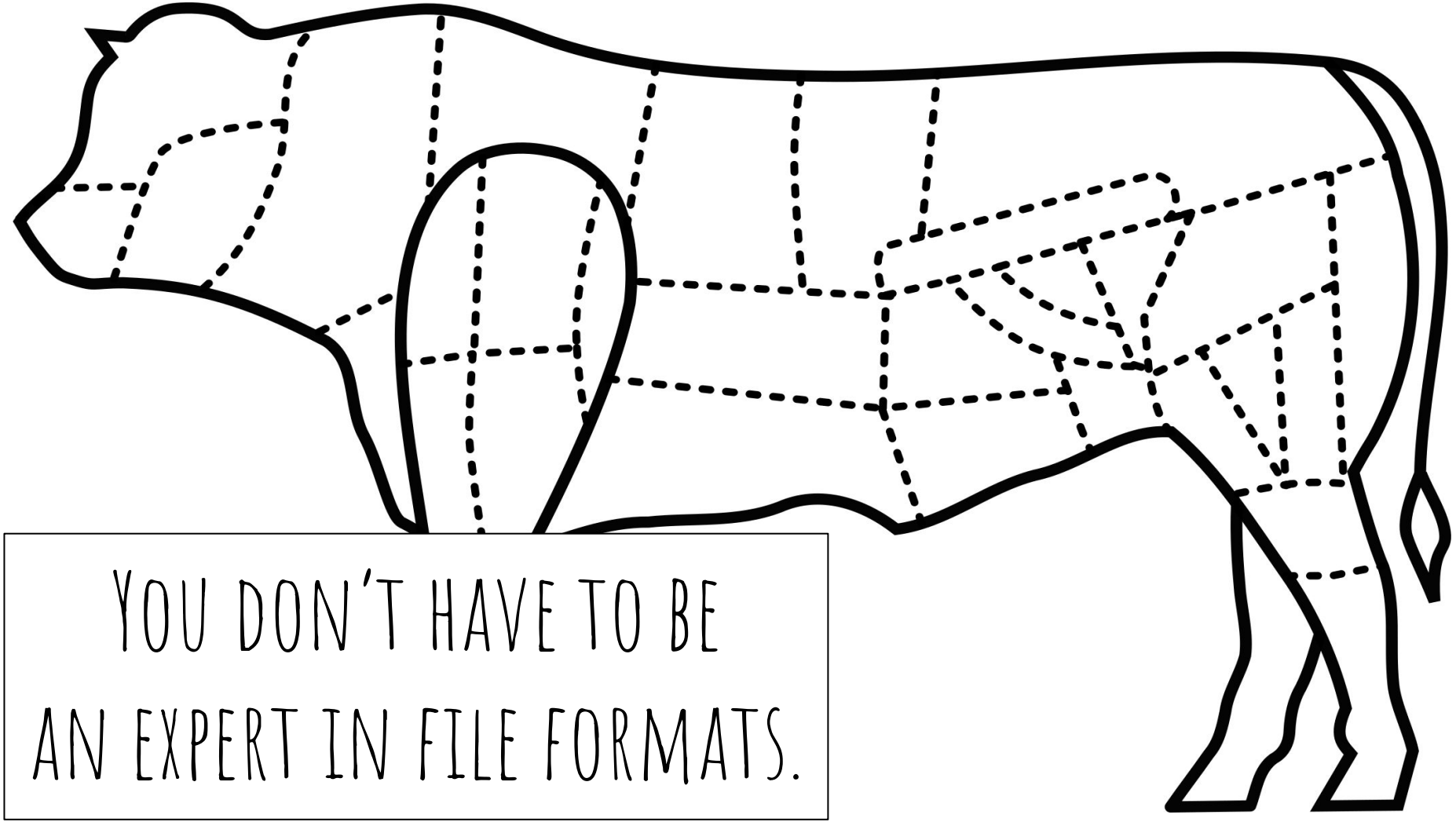
DON'T BE SCARED...

You **don't** need to understand crypto*graphy* or maths...

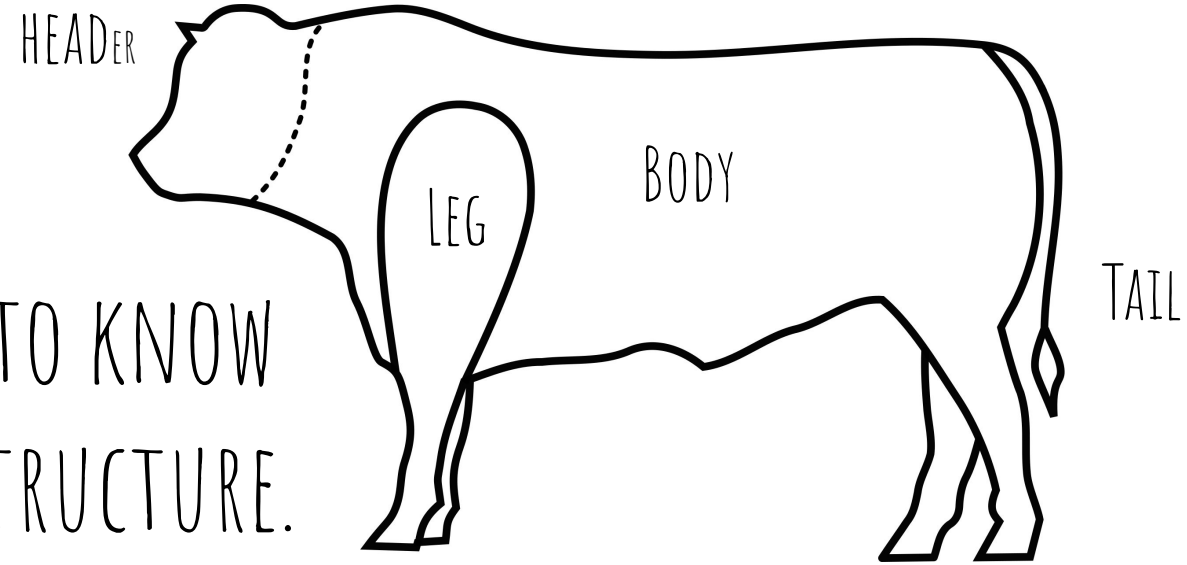
(to be honest, I don't either)

We'll just use existing attacks:

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

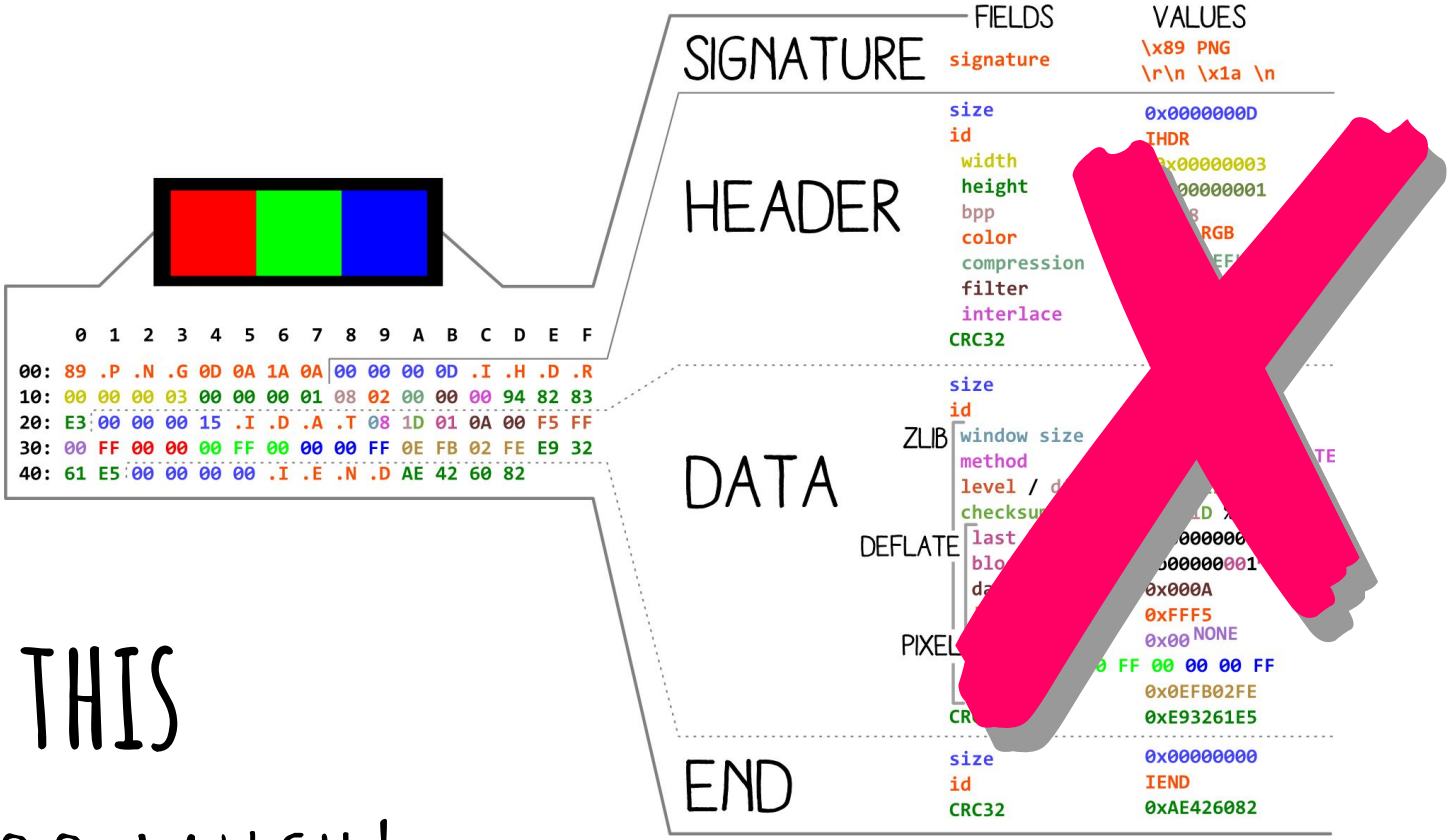


You don't have to be an expert in file formats.



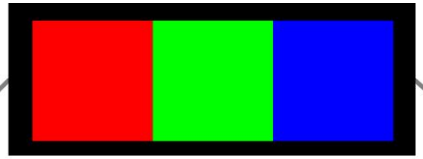
You just need to know their overall structure.

(less complex than some lego models)



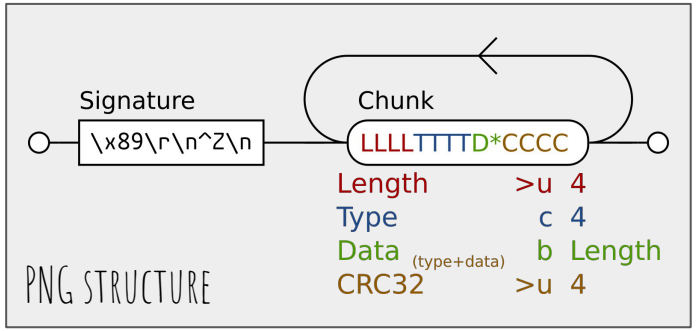
EVEN **THIS**

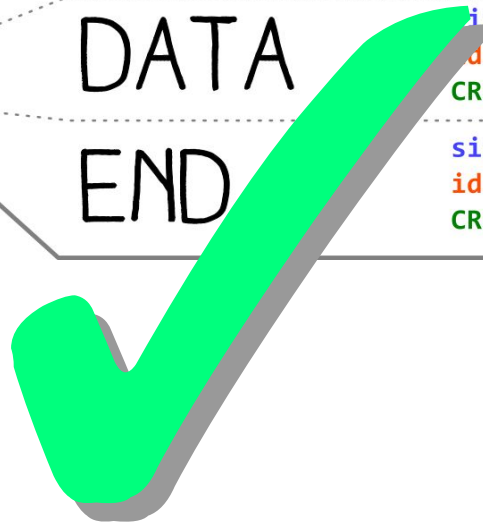
IS TOO MUCH!



```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 89 .P .N .G \r \n ^Z \n 00 00 00 0D .I .H .D .R
10: 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
20: E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
30: 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
40: 61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

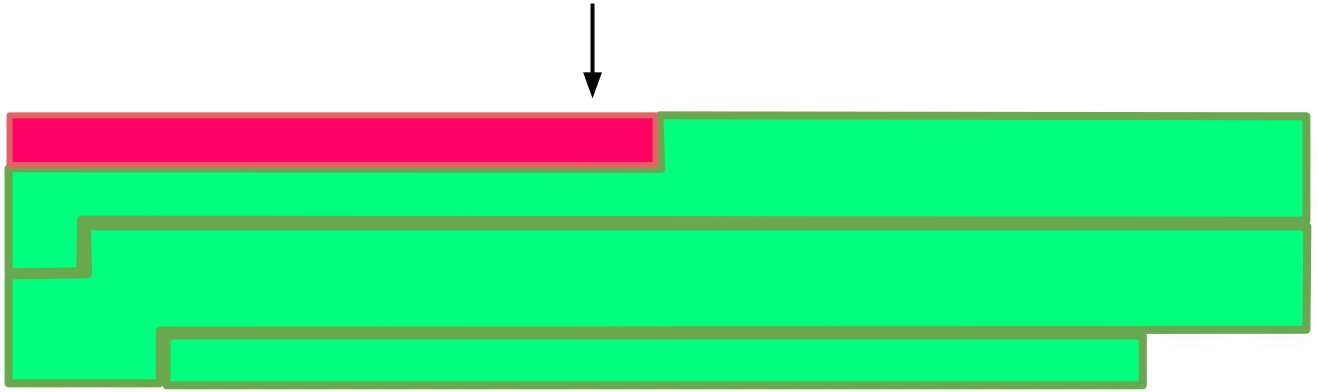
	FIELDS	VALUES
SIGNATURE	signature	\x89 PNG \r\n ^Z \n
HEADER	size	0x0000000D
	id	IHDR
	CRC32	0x948283E3
DATA	size	0x00000015
	id	IDAT
	CRC32	0xE93261E5
END	size	0x00000000
	id	IEND
	CRC32	0xAE426082





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

```
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
```



And we'll ignore most contents, so we'll just think in blocks.

To be honest

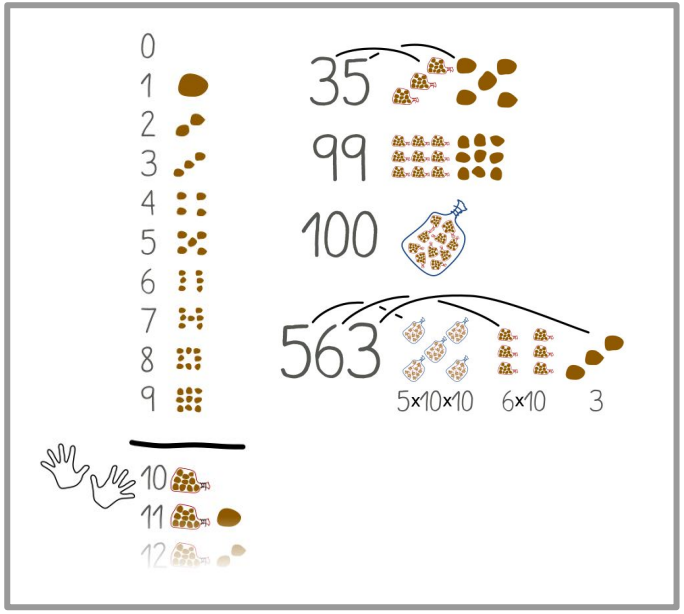
Exploiting hash collisions feels a bit like...

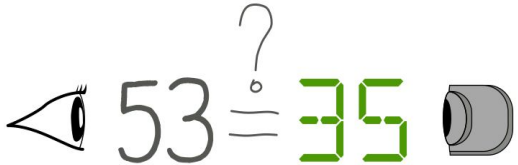

...PLAYING A PUZZLE GAME!
COLLTRIS
Ange Albertini

You just need to know the rules of each block!

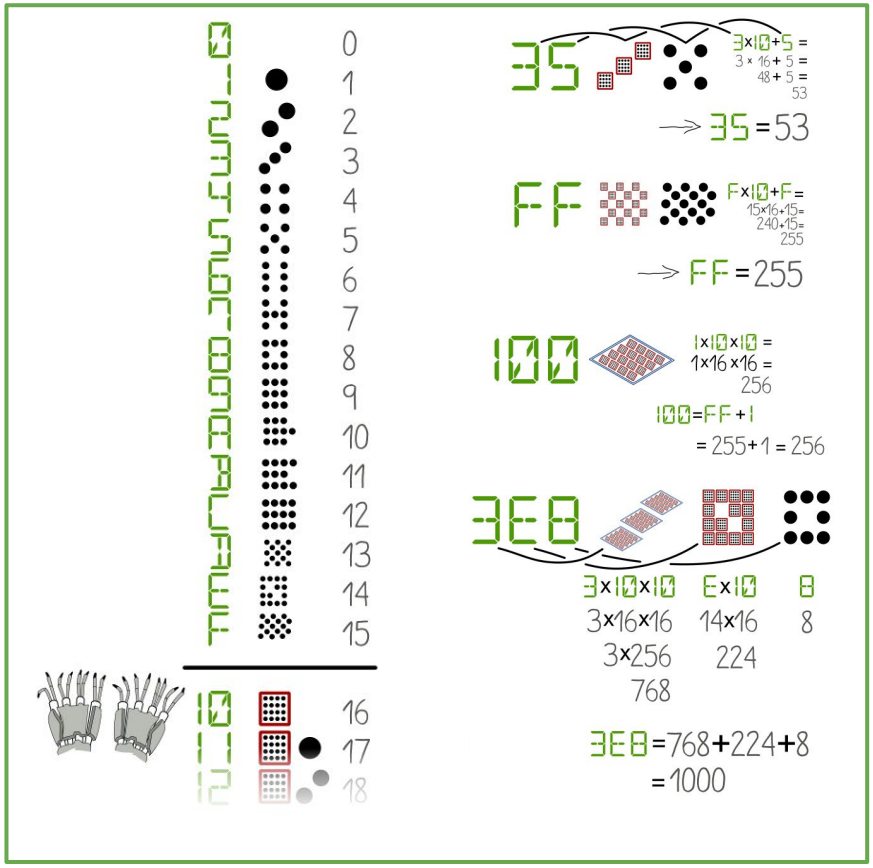
For a simpler introduction to the topic, check this. slides / video

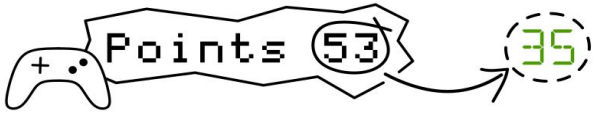

BASICS





You know hexadecimal?





You know ASCII?

A standard encoding: characters ⟨=⟩ values

"A" ⟨=⟩ 0x41 = 65

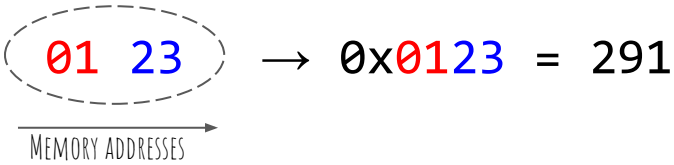
"Z" ⟨=⟩ 0x5A = 90

"a" ⟨=⟩ 0x61 = 97

You know endianness?

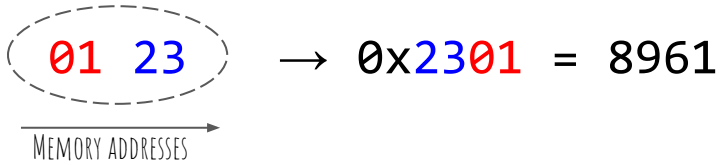
-> Big digits first

Reading like humans.



<- Little digits first

Reading like many processors.



Formats

PNG, JPG, MP4, Class

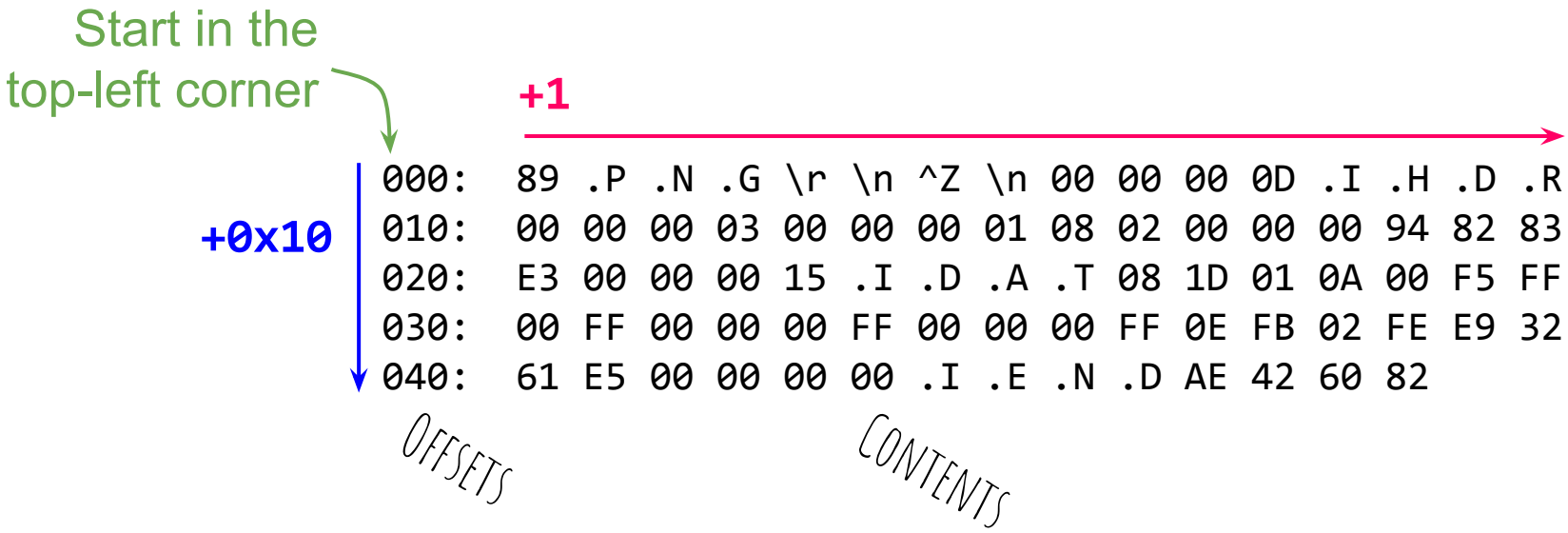
TIFF

Exists in both endianness

ZIP, BMP, GZip

Executables: ARM (default), x86, x64

TIFF



You know hexadecimal viewers?

Note: for this workshop, wrapping at 0x10 / 16 bytes is **important**.

Recommended hex tools

Web	Kaitai
Linux	xxd/Hexedit/Okteta/Bvi/Dhex
Mac	Hex Fiend
Win	HxD/Hiew

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
GET
READY
Prerequisites

Prerequisites 1/2

To run your own computations:

A few warnings will happen but it's OK!:

```
src/sha1attackgenerator/collfind.cpp:1266:64: warning:
[-Wshift-overflow=]
  if (((Q20bu+(m15add<<20))&Q20mask)==Q20val && (((Q21b
```

Check that the executables are there!

```
~/git/hashclash/bin$ ls md5*
md5_birthdaysearch    md5_diffpathconnect  md5_diffpathhelper
md5_diffpathbackward  md5_diffpathforward  md5_fastcoll
```

HashClash, not HashCat.

HashClash

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

CUDA is **NOT** required

Download source and compile, or download release binaries

May require **psmisc** and **autoconf-archive**

Can't compile? Computation too slow?

Compiling and computing can be troublesome.
In case, all the computed examples of the slides are available:
https://github.com/corkami/collisions/tree/master/workshop/prefixes

So you can skip that step and focus on file manipulation (if you prefer).

OTOH you may want to at least try to run FastColl: it's instant and never fails.

Even works with Wine w/ Windows binaries:
cf5600ab

```
corkami:~$ wine ~/fastcoll_v1.0.0.5.exe
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Allowed options:
  -h [ --help ]           Show options.
```

Prerequisites 2/2

A file format manipulation environment

Hex editor, assembly, scripting...

Whatever rocks your boat and you're familiar with.

Recommended hex tools

Web	Kaitai
Linux	xxd/Hexedit/Okteta/Bvi/Dhex
Mac	Hex Fiend
Win	HxD/Hiew

A copy of Corkami/collisions

(contains materials for this workshop)

https://github.com/corkami/collisions

A copy of these slides (for readability if needed)

https://speakerdeck.com/ange/colltris

If you're not familiar with hex viewing:

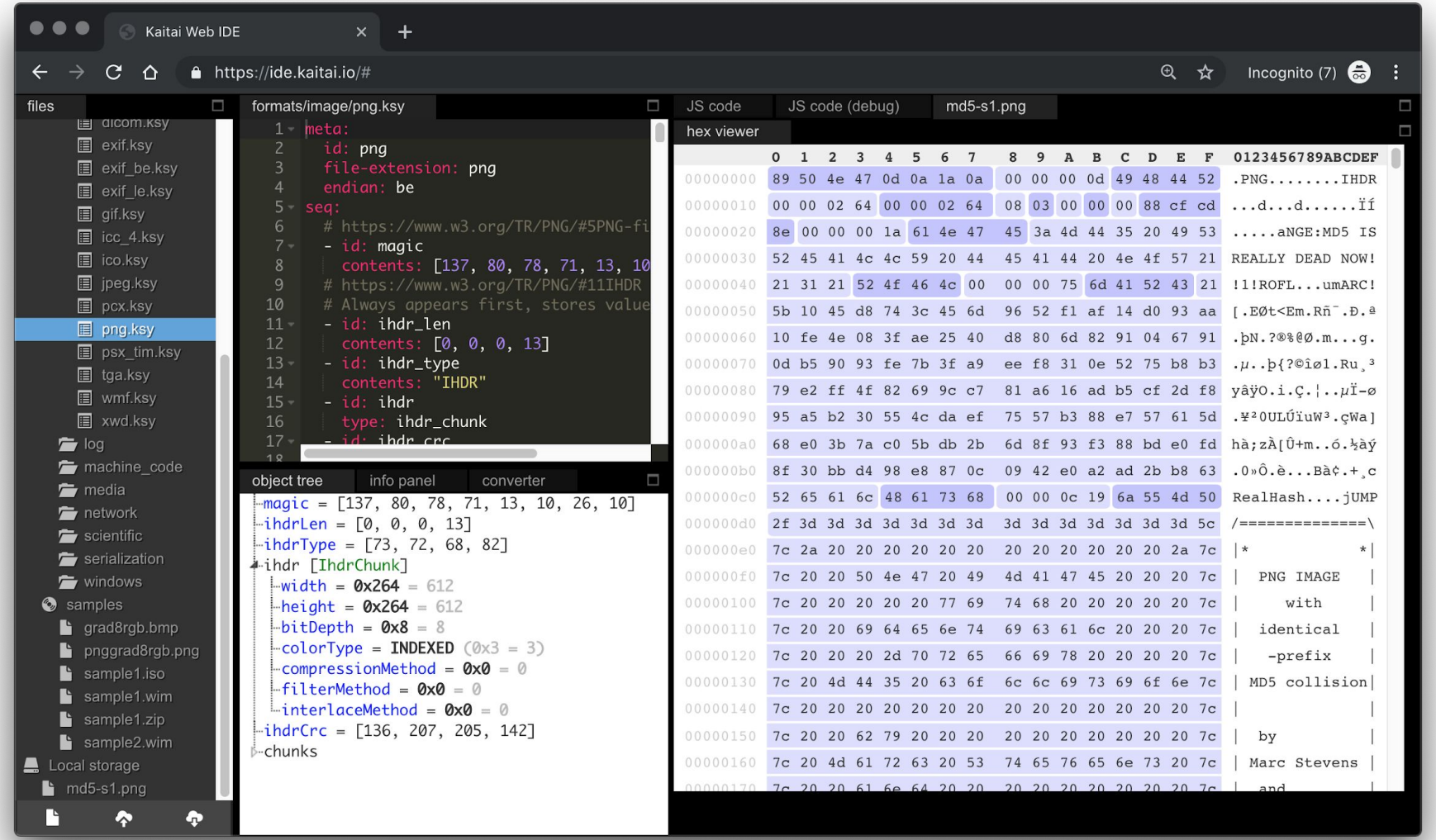
Kaitai 101

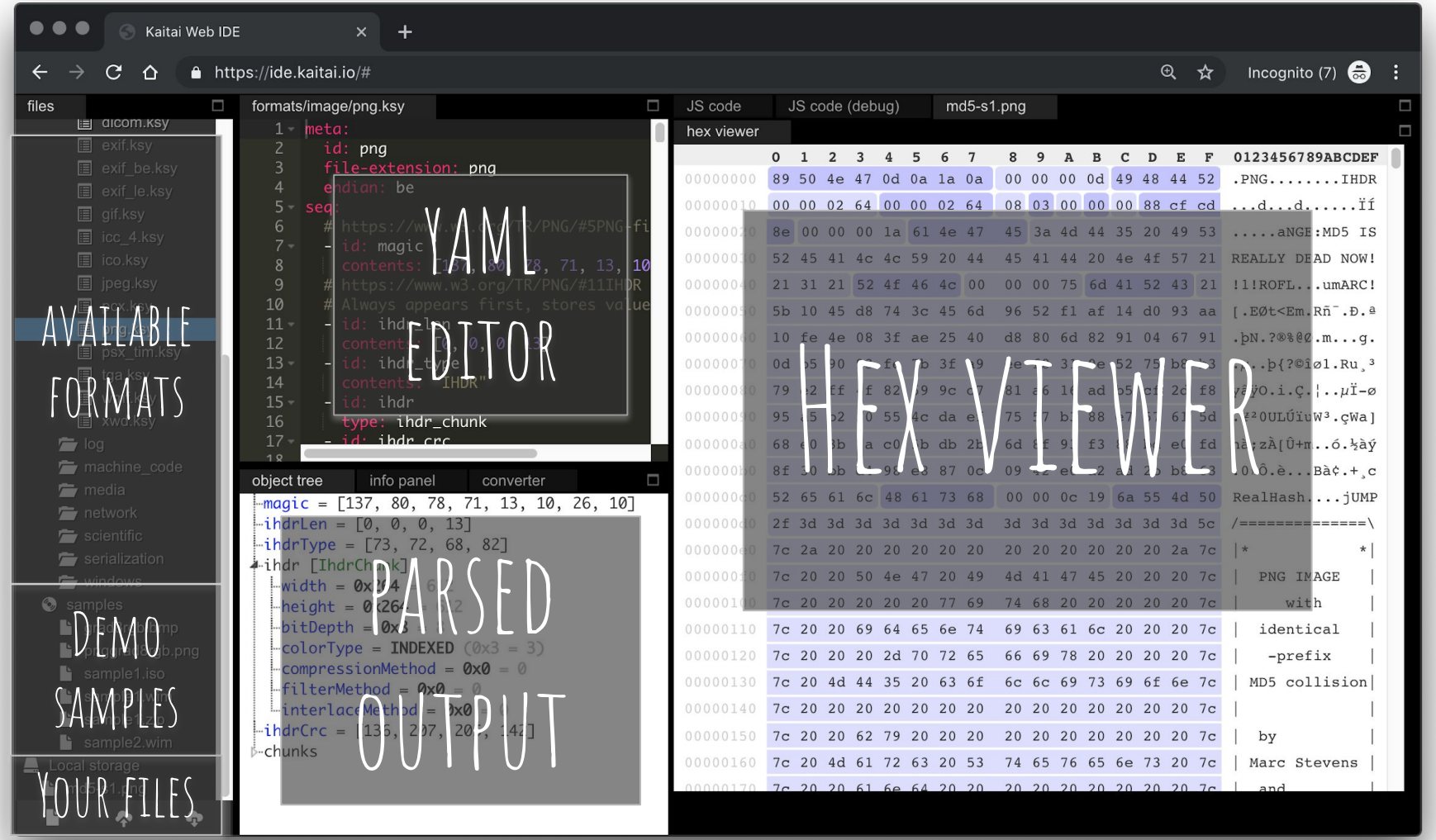
Kaitai is a pretty awesome tool
with a great online **viewer**
(no modifications of the files are allowed)
https://ide.kaitai.io

Kaitai in a nutshell

- Uses YAML-based parsers. Many formats are already supported.
- Drop your file on the IDE, select the file format.
- if you modify the Yaml source (for ex: to remove unneeded details),
  the modified version is saved locally and reusable/downloadable.

Drawback: instantly fails on invalid files
(but you can fix that directly in the GUI).


Kaitai Web IDE
https://ide.kaitai.io/#
Incognito (7)
files
dicom.ksy
exif.ksy
exif_be.ksy
exif_le.ksy
gif.ksy
icc_4.ksy
ico.ksy
jpeg.ksy
pcx.ksy
png.ksy
psx_tim.ksy
tga.ksy
wmf.ksy
xwd.ksy
log
machine_code
media
network
scientific
serialization
windows
samples
grad8rgb.bmp
pnggrad8rgb.png
sample1.iso
sample1.wim
sample1.zip
sample2.wim
Local storage
md5-s1.png
formats/image/png.ksy
meta:
id: png
file-extension: png
endian: be
seq:
https://www.w3.org/TR/PNG/#5PNG-fi
- id: magic
contents: [137, 80, 78, 71, 13, 10
https://www.w3.org/TR/PNG/#11IHDR
Always appears first, stores value
- id: ihdr_len
contents: [0, 0, 0, 13]
- id: ihdr_type
contents: "IHDR"
- id: ihdr
type: ihdr_chunk
- id: ihdr_crc
object tree
info panel
converter
magic = [137, 80, 78, 71, 13, 10, 26, 10]
ihdrLen = [0, 0, 0, 13]
ihdrType = [73, 72, 68, 82]
ihdr [IhdrChunk]
width = 0x264 = 612
height = 0x264 = 612
bitDepth = 0x8 = 8
colorType = INDEXED (0x3 = 3)
compressionMethod = 0x0 = 0
filterMethod = 0x0 = 0
interlaceMethod = 0x0 = 0
ihdrCrc = [136, 207, 205, 142]
chunks
JS code
JS code (debug)
md5-s1.png
hex viewer
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
00000000 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 .PNG........IHDR
00000010 00 00 02 64 00 00 02 64 08 03 00 00 00 88 cf cd ...d...d......ÏÍ
00000020 8e 00 00 00 1a 61 4e 47 45 3a 4d 44 35 20 49 53 .....aNGE:MD5 IS
00000030 52 45 41 4c 4c 59 20 44 45 41 44 20 4e 4f 57 21 REALLY DEAD NOW!
00000040 21 31 21 52 4f 46 4c 00 00 00 75 6d 41 52 43 21 !1!ROFL...umARC!
00000050 5b 10 45 d8 74 3c 45 6d 96 52 f1 af 14 d0 93 aa [.EØt<Em.Rñ¯.Ð.ª
00000060 10 fe 4e 08 3f ae 25 40 d8 80 6d 82 91 04 67 91 .þN.?®%@Ø.m...g.
00000070 0d b5 90 93 fe 7b 3f a9 ee f8 31 0e 52 75 b8 b3 .µ..þ{?©îø1.Ru¸³
00000080 79 e2 ff 4f 82 69 9c c7 81 a6 16 ad b5 cf 2d f8 yâÿO.i.Ç.¦..µÏ-ø
00000090 95 a5 b2 30 55 4c da ef 75 57 b3 88 e7 57 61 5d .¥²0ULÚïuW³.çWa]
000000a0 68 e0 3b 7a c0 5b db 2b 6d 8f 93 f3 88 bd e0 fd hà;zÀ[Û+m..ó.½àý
000000b0 8f 30 bb d4 98 e8 87 0c 09 42 e0 a2 ad 2b b8 63 .0»Ô.è...Bà¢.+¸c
000000c0 52 65 61 6c 48 61 73 68 00 00 0c 19 6a 55 4d 50 RealHash....jUMP
000000d0 2f 3d 3d 3d 3d 3d 3d 3d 3d 3d 3d 3d 3d 3d 3d 5c /==============\
000000e0 7c 2a 20 20 20 20 20 20 20 20 20 20 20 20 2a 7c |*   *|
000000f0 7c 20 20 50 4e 47 20 49 4d 41 47 45 20 20 20 7c | PNG IMAGE |
00000100 7c 20 20 20 20 20 77 69 74 68 20 20 20 20 20 7c | with |
00000110 7c 20 20 69 64 65 6e 74 69 63 61 6c 20 20 20 7c | identical |
00000120 7c 20 20 20 2d 70 72 65 66 69 78 20 20 20 20 7c | -prefix |
00000130 7c 20 4d 44 35 20 63 6f 6c 6c 69 73 69 6f 6e 7c | MD5 collision|
00000140 7c 20 20 20 20 20 20 20 20 20 20 20 20 20 20 7c | |
00000150 7c 20 20 62 79 20 20 20 20 20 20 20 20 20 20 7c | by |
00000160 7c 20 4d 61 72 63 20 53 74 65 76 65 6e 73 20 7c | Marc Stevens |
00000170 7c 20 20 61 6e 64 20 20 20 20 20 20 20 20 20 7c | and |


Kaitai Web IDE
https://ide.kaitai.io/#
Incognito (7)
files
formats/image/png.ksy
JS code
JS code (debug)
md5-s1.png
hex viewer
AVAILABLE FORMATS
YAML EDITOR
HEX VIEWER
DEMO SAMPLES
PARSED OUTPUT
YOUR FILES
object tree
info panel
converter
Local storage

If you're not familiar with format manipulations

- Skim through some specs. Get an idea of the high level structure.
- Look for possible shortcuts that a parser might take, such as:
  - Secondary structures might be absent or corrupted -> parasitizing
  - Structures order might not be enforced -> shuffling
- Check standard open-source implementations.

And don't worry, all the reference scripts are on the Corkami GitHub: https://github.com/corkami/collisions

Efficient diffing

```
01-fastcoll-1.bin
0000 0000: 37 75 C1 F1 C4 A7 5A E7  9C E0 DE 7A 5B 10 80 26  7u....Z. ...z[..&
0000 0010: 02 AB D9 39 C9 6C 5F 02  12 C2 7F DA CD 0D A3 B0  ...9.l_. ........
0000 0020: 8C ED FA F3 E1 A3 FD B4  EF 09 E7 FB B1 C3 99 1D  ........ ........
0000 0030: CD 91 C8 45 E6 6E FD 3D  C7 BB 61 52 3E F4 E0 38  ...E.n.= ..aR>..8
0000 0040: 49 11 85 69 EB CC 17 9C  93 4F 40 EB 33 02 AD 20  I..i.... .O@.3..
0000 0050: A4 09 2D FB 15 FA 20 1D  D1 DB 17 CD DD 29 59 1E  ..-... . .....)Y.
0000 0060: 39 89 9E F6 79 46 9F E6  8B 85 C5 EF DE 42 4F 46  9...yF.. .....BOF
0000 0070: C2 78 75 9D 8B 65 F4 50  EA 21 C5 59 18 62 FF 7B  .xu..e.P .!.Y.b.{

01-fastcoll-2.bin
0000 0000: 37 75 C1 F1 C4 A7 5A E7  9C E0 DE 7A 5B 10 80 26  7u....Z. ...z[..&
0000 0010: 02 AB D9 B9 C9 6C 5F 02  12 C2 7F DA CD 0D A3 B0  .....l_. ........
0000 0020: 8C ED FA F3 E1 A3 FD B4  EF 09 E7 FB B1 43 9A 1D  ........ .....C..
                    C7 BB 61 D2 3E F4 E0 38  ...E.n.= ..a.>..8
                    93 4F 40 EB 33 02 AD 20  I..i.... .O@.3..
                    D1 DB 17 CD DD 29 59 1E  ..-{.. . .....)Y.
                    8B 85 C5 EF DE C2 4E 46  9...yF.. ......NF
                    EA 21 C5 D9 18 62 FF 7B  .xu..e.P .!...b.{
next difference  ESC quit  T mo
goto position      Q quit  B mo
```

VBinDiff

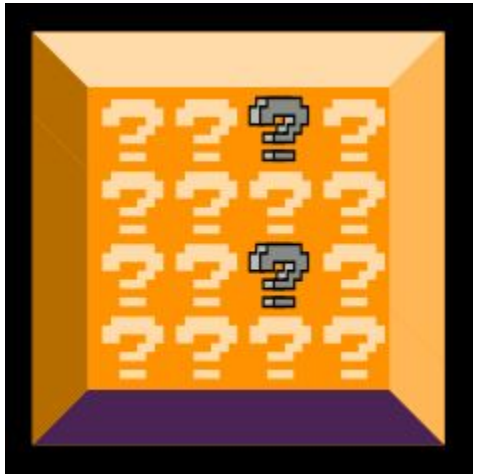
```
$ diff <(xxd 01-fastcoll-1.bin) <(xxd 01-fastcoll-2.bin)
2,4c2,4
< 00000010: 02ab d939 c96c 5f02 12c2 7fda cd0d a3b0  ...9.l_.........
< 00000020: 8ced faf3 e1a3 fdb4 ef09 e7fb b1c3 991d  ................
< 00000030: cd91 c845 e66e fd3d c7bb 6152 3ef4 e038  ...E.n.=..aR>..8
---
> 00000010: 02ab d9b9 c96c 5f02 12c2 7fda cd0d a3b0  .....l_.........
> 00000020: 8ced faf3 e1a3 fdb4 ef09 e7fb b143 9a1d  .............C..
> 00000030: cd91 c845 e66e fd3d c7bb 61d2 3ef4 e038  ...E.n.=..a.>..8
6,8c6,8
< 00000050: a409 2dfb 15fa 201d d1db 17cd dd29 591e  ..-... ......)Y.
< 00000060: 3989 9ef6 7946 9fe6 8b85 c5ef de42 4f46  9...yF.......BOF
< 00000070: c278 759d 8b65 f450 ea21 c559 1862 ff7b  .xu..e.P.!.Y.b.{
---
> 00000050: a409 2d7b 15fa 201d d1db 17cd dd29 591e  ..-{.. ......)Y.
> 00000060: 3989 9ef6 7946 9fe6 8b85 c5ef dec2 4e46  9...yF........NF
> 00000070: c278 759d 8b65 f450 ea21 c5d9 1862 ff7b  .xu..e.P.!...b.{
```

Radiff (from Radare)

https://r2wiki.readthedocs.io/en/latest/tools/radiff2/

```
~/git/corkami/collisions/workshop/prefixes$ radiff2 -x 10*
  offset     0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF   0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000! 6e6f00000000000000000000000000000 no.............  79657300000000000000000000000000 yes.............
0x00000010  00000000000000000000000000000000 ................  00000000000000000000000000000000 ................
0x00000020  00000000000000000000000000000000 ................  00000000000000000000000000000000 ................
0x00000030! 00000000000000001971e7f70972fb06 .........q...r..  0000000000000000b74638098a46f17b .........F8..F.{
0x00000040  f34526136660c801b92a75255a6723a6 .E&.f`...*u%Zg#.  f34526136660c801b92a75255a6723a6 .E&.f`...*u%Zg#.
0x00000050  923deb8db0b757f1459f2295bec04375 .=....W.E."...Cu  923deb8db0b757f1459f2295bec04375 .=....W.E."...Cu
0x00000060! 9198a2d3e0fd59edd1c5fa0b79659751 ......Y.....ye.Q  9198a2d3e0fd59edd1c5fa0b7965974d ......Y.....ye.M
0x00000070  b3b3e40c110c9032de4ba14bb81b5ec8 .......2.K.K..^.  b3b3e40c110c9032de4ba14bb81b5ec8 .......2.K.K..^.
0x00000080  25d38f19cd104307d9bbff8cb75a23f9 %.....C.......Z#.  25d38f19cd104307d9bbff8cb75a23f9 %.....C.......Z#.
...
0x000000a0! ba784000c37e93b231a36e2d34724ac9 .x@..~..1.n-4rJ.  ba784000c37e93b231a36e2d346a4ac9 .x@..~..1.n-4jJ.
0x000000b0  534ec045361ec86a5698e6f0571d6198 SN.E6..jV...W.a.  534ec045361ec86a5698e6f0571d6198 SN.E6..jV...W.a.
0x000000c0  13fcffcd4d83a2d2bbb8dc042be2b883 ....M.......+...  13fcffcd4d83a2d2bbb8dc042be2b883 ....M.......+...
...
0x000000e0! 7d86e4351eb833eeea15d181fa9662ec }..5..3.......b.  7d86e4351eb833eeea15d181ba9662ec }..5..3.......b.
0x000000f0  7531fbda4fae246f67d6af109629fbc7 u1..O.$og....)..  7531fbda4fae246f67d6af109629fbc7 u1..O.$og....)..
0x00000100  a332bba9ead5e4ae1fc2fb234122b2e0 .2.........#A"..  a332bba9ead5e4ae1fc2fb234122b2e0 .2.........#A"..
...
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
...
0x00000220! e6186ee3f052e4358361423572 97cd8d ..n..R.5.aB5r...  e6186ee3f052e43583614235729 7cd8d ..n..R.5.aB5r...
0x00000230  4ff793685a705f5a043ad542c1fa0fe2 O..hZp_Z.:.B....  4ff793685a705f5a043ad542c1fa0fe2 O..hZp_Z.:.B....
0x00000240  ae57dbaff151b8b73818ef2eb8a6a92c .W...Q..8......,  ae57dbaff151b8b73818ef2eb8a6a92c .W...Q..8......,
...
0x00000260! 4f9cfa623d4246596732ec99da897a08 O..b=BFYg2....z.  4f9cfa623d4246596732ec99da897a88 O..b=BFYg2....z.
0x00000270  e7ade321ed3c4bc04d9f833cdc7fb70a ...!.<K.M..<....  e7ade321ed3c4bc04d9f833cdc7fb70a ...!.<K.M..<....
```

OUR FIRST COLLISION



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

WARNING

Collision computing is a very random process

Many possible problems across the different attacks:

- variable computing time (Ex: with FastColl from 0.3 to 13s on the same machine!)
- different results from each execution.
- final result might just not collide.
- computation might be stalled.

-> restart/backtracking might be required.

https://github.com/cr-marcstevens/hashclash/blob/master/src/md5fastcoll/main.cpp#L106

```
seed32_1 = uint32(time(NULL));
```

https://www.cwi.nl/system/files/PhD-Thesis-Marc-Stevens-Attacks-on-Hash-Functions-and-Applications.pdf#page=110

Algorithm 6-2 Collision finding algorithm.

Given a full differential path $\mathfrak{q}_{-3}, \ldots, \mathfrak{q}_{64}$ consisting of only direct and bac
and the set $\mathcal{T}_1, \ldots, \mathcal{T}_8$ of tunnels from Table 6-3, perform the following s

1. Determine for all tunnels for which bits b the extra bitconditions a
   3 can be met. For each possible case, apply compatible bitcondi
   extra bitconditions and change the bitconditions $\mathfrak{q}_t[b]$ of the chang
   in the first round from '.' to '0'.
2. Perform the steps below until a collision block has been found.
3. Select $Q_1, Q_2, Q_{13}, \ldots, Q_{16}$ such that $\mathfrak{q}_1, \mathfrak{q}_2, \mathfrak{q}_{13}, \ldots, \mathfrak{q}_{16}$ hold.
4. Compute m_1, Q_{17}.
5. If $\mathfrak{q}_{17}$ holds and the rotation for $t = 16$ is successful, then procee
6. Store the set $\mathcal{Z}$ of all pairs (Q_1, Q_2) meeting $\mathfrak{q}_1, \mathfrak{q}_2$ that do not c
   bits of Q_2 involved in $\mathfrak{q}_3$.
7. For all $Q_3, \ldots, Q_7$ meeting $\mathfrak{q}_3, \ldots, \mathfrak{q}_7$ do:
8. Compute m_6, Q_{18}.
9. If $\mathfrak{q}_{18}$ holds and the rotation for $t = 17$ is successful, then proc
10. For all $Q_8, \ldots, Q_{12}$ meeting $\mathfrak{q}_8, \ldots, \mathfrak{q}_{12}$ do:
11. Compute m_{11}, Q_{19}.
12. If $\mathfrak{q}_{19}$ holds and the rotation for $t = 18$ is successful, then pro

MISSION START!

EXAMPLE

```
bin$ md5_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

bin$ _
```

```
bin$ touch empty
bin$ du -b empty
0       empty
bin$ _
```

Mission

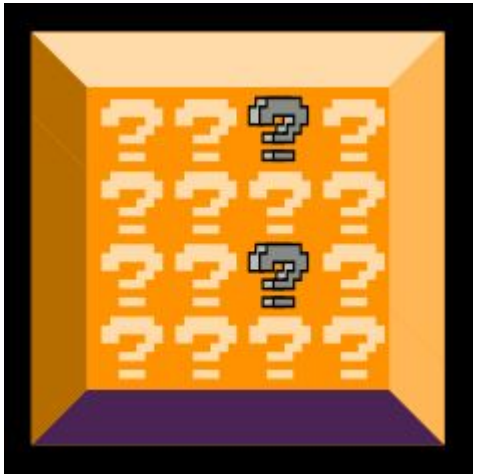
From nothing, generate 2 files with the same MD5.

Create an empty file, run FastColl on it (recording).

```
00:  37 75 C1 F1-C4 A7 5A E7-9C E0 DE 7A-5B 10 80 26  7u┴±─°Zτ£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┤∩στ√▒├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┴±─°Zτ£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┤∩στ√▒C╓↔
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

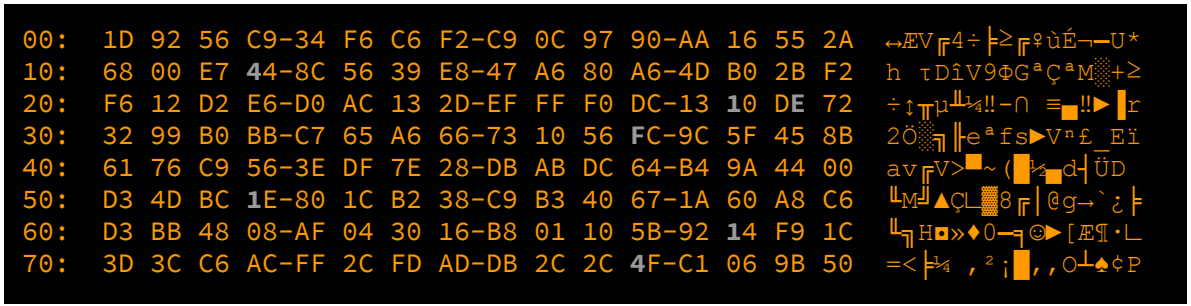


Our first hash collision

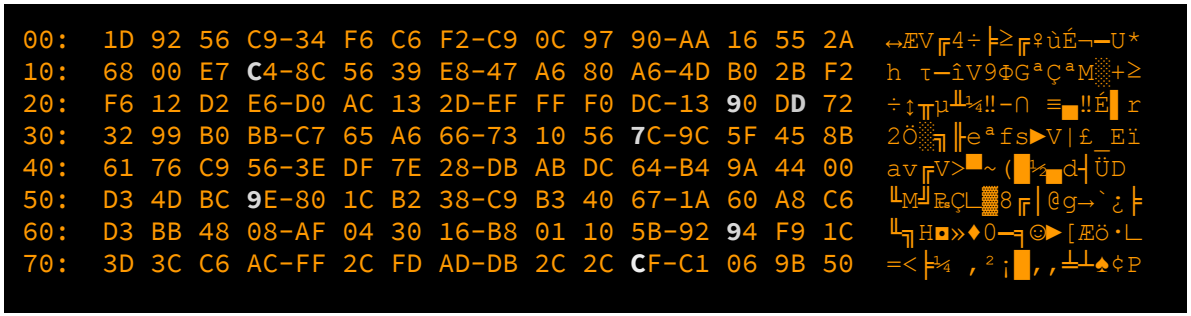
(Your computation will be different)

```
bin$ md5_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

bin$ _
```

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╝▲ÇL▒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╝₧ÇL▒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...

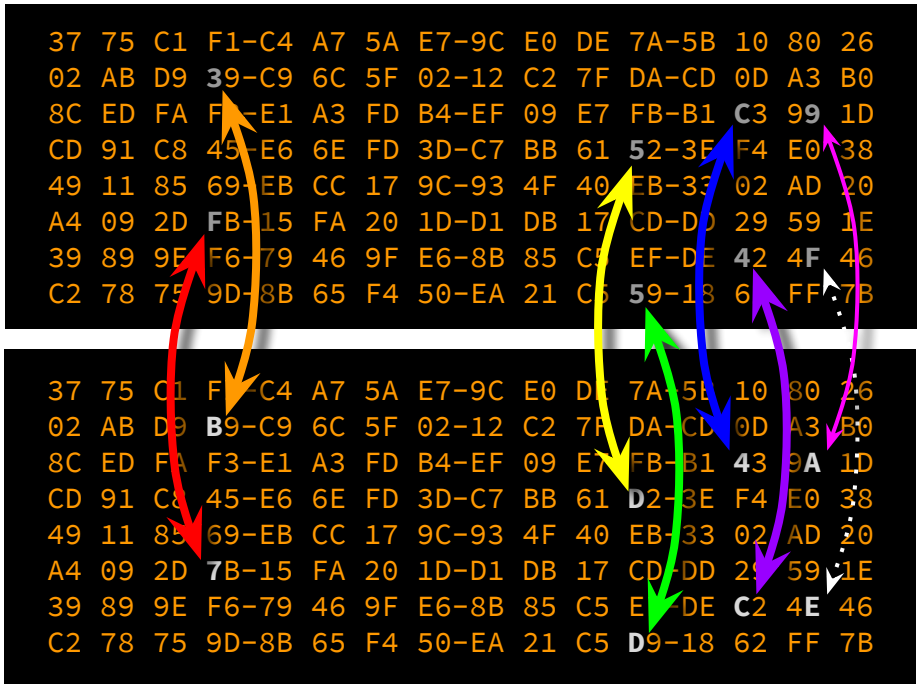
computed randomness

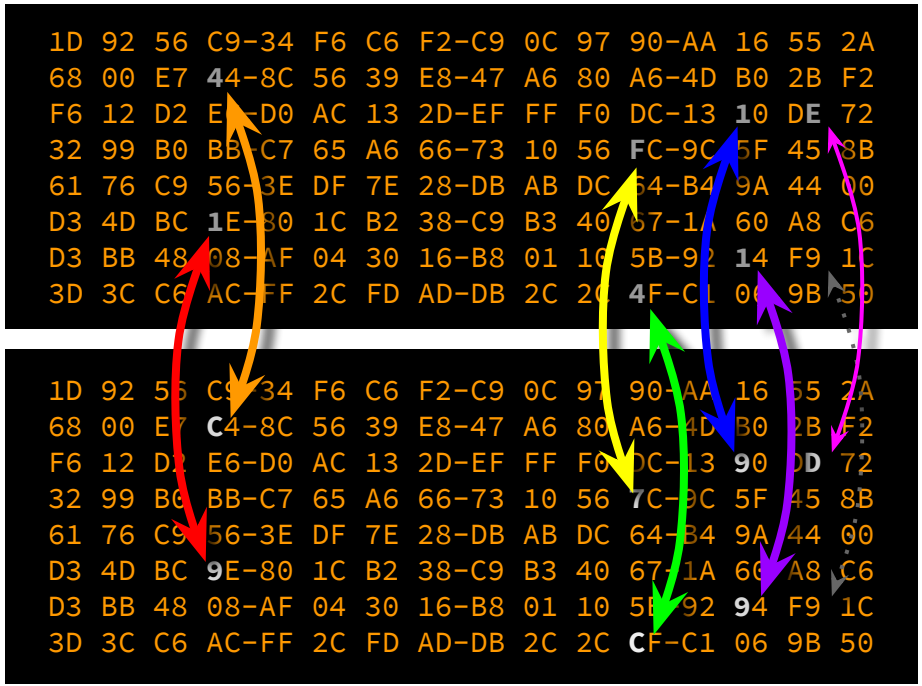
with tiny differences.

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

```
bin$ md5_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

bin$ _
```


```
bin$ cat prefix
Here is a file with a few bytes
bin$ du -b prefix
31      prefix
bin$ _
```

Now let's add an input - our prefix.

The content and length of the prefix is not important here (recording).

```
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

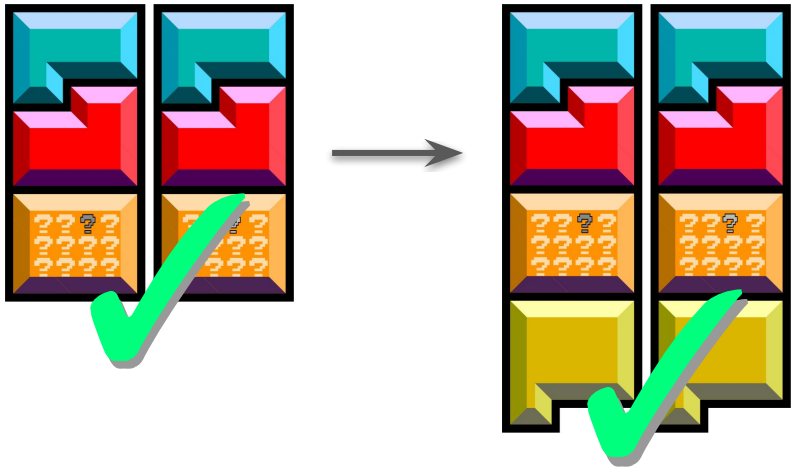
Padded to 64 bytes?

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

-> Appending the same thing *at block boundaries*

to two files with the same hash

will give files with the same hash.



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

Length extension attack
https://en.wikipedia.org/wiki/Length_extension_attack

Similarities of all current collision attacks

All current hash collisions attacks work with such alignment:
padding, then adding (at block boundaries) a number of blocks.

-> Via these attacks:

1- Every pair with the same hash will have the **same length**.

2- The end of the files is either **identical** (suffix),
Or high entropy, very similar and aligned to 64 bytes
(no suffix, just collision blocks).

From now on...

We always work with 64 bytes blocks.

It's really a block game!

Double-check your hex viewer wrapping!
(some viewers adjust their wrapping when you resize the window)

Certificate (easy)

Hash collision

Computed your first FastColl

Ange Albertini

INSTRUCTOR


RECAP

Step 1/4 : the prefix (optional)

We define the start of the file.

The collision computation will depend on that.

The prefix can be empty.

Its content and size make **no** difference at all.

Step 2/4 : the padding (if needed)

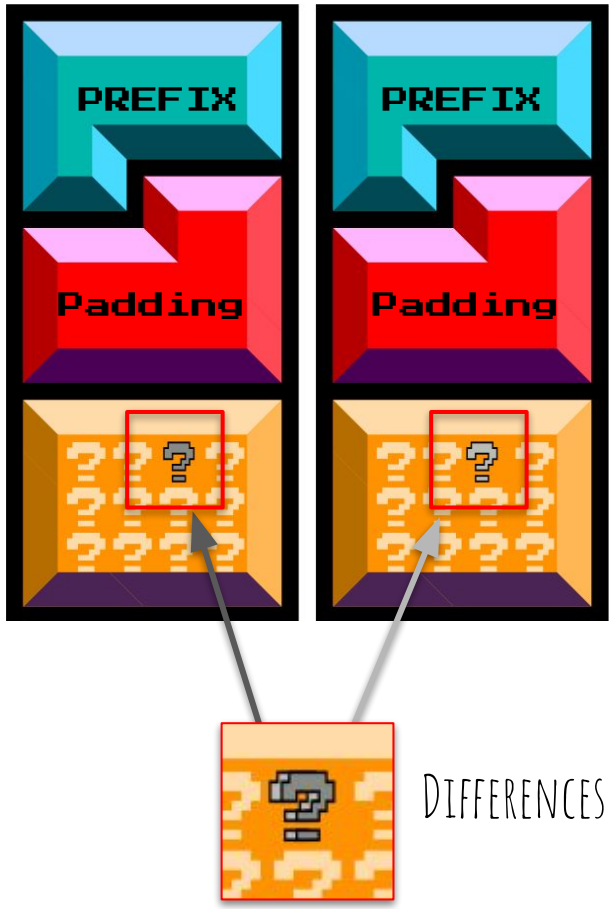
We add some data to the prefix
  to get a rounded size (a multiple of 64).
Its content doesn't matter, only its length:
  it just aligns things.

Step 3/4 : the collision blocks

We compute a pair of blocks full of randomness
with tiny differences.

Despite the differences,
the hash of both files is the same.

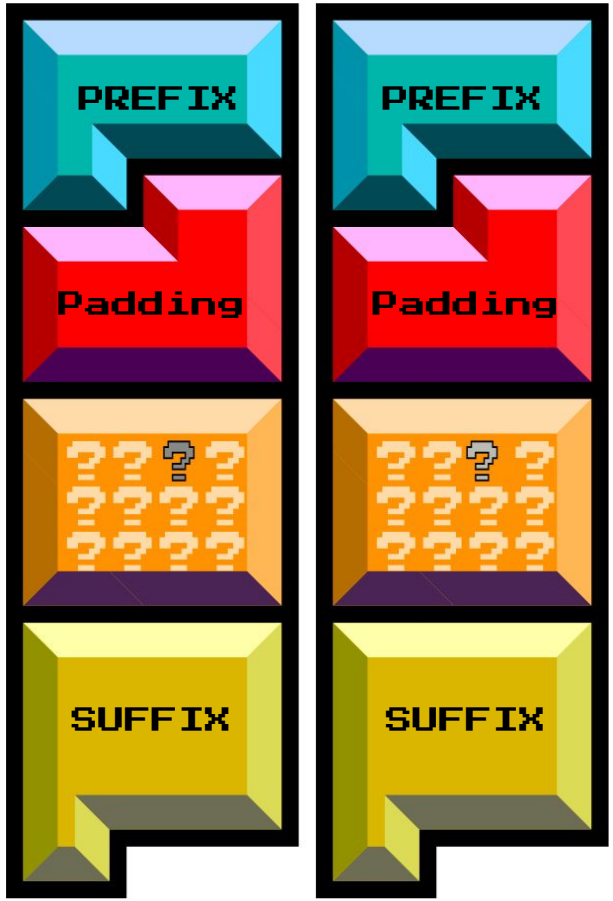
These collision blocks only work
for that exact prefix.



Step 4/4 : the suffix

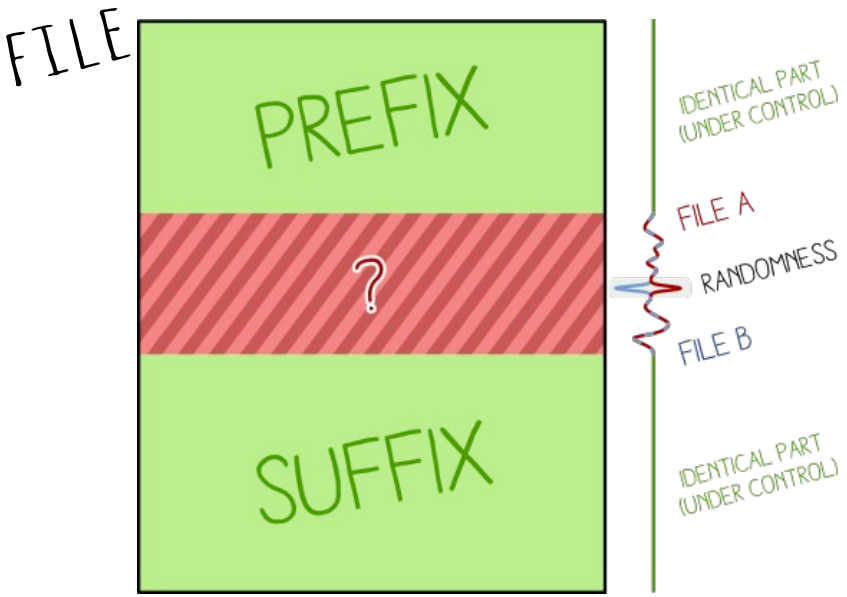
You can add anything to both sides (not required).

The hash value will remain the same.



An Identical Prefix hash Collision

- takes a single input
- prefix and suffix will be identical:
-> files almost identical
-> exploitation depends only on collision differences
-> two contents coexist in the same file.



These properties are common to FastColl, Unicoll and Shattered.

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

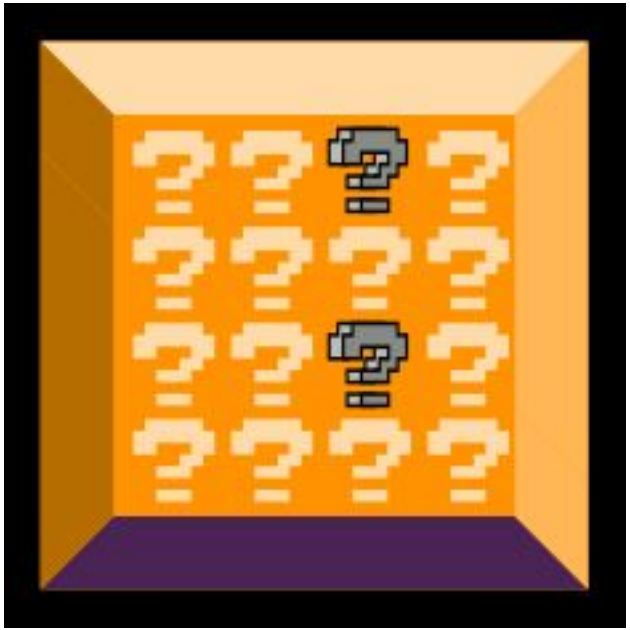
… 🛑 ? Several contents can co-exist (usually appended data)

```
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
```

Hash collision blocks

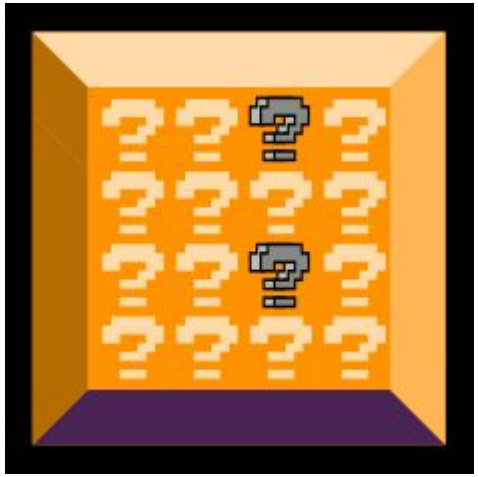
- start and end aligned to 64 bytes
  (via padding if needed)
- totally random
- tiny differences at fixed offsets
  We can't change these offsets.
  (they depend on the hash function).



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

FastColl

two blocks
a few seconds
in the middle
(away from start or end)



-> hard to exploit!

The fastest,
but 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 usually impossible to declare
a structure **and** its length in a single byte.
(such as a variable-length comment).

Cheating workaround:
some code (executable, javascript) checks
the difference in the collision block.

Workaround: bruteforcing

(Bruteforcing the collision block output, not the hash value)

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 and its unique powers.

```
99    // change q17 until conditions are met on q18, 
100    unsigned counter = 0;
101    while (counter < (1 << 7))
102    {
103      const uint32 q16 = Q[Qoff + 16];
104      uint32 q17 = ((xrng64() & 0x3ffd7ff7) | (q16
105      ++counter;
106
107      uint32 q18 = GG(q17, q16, Q[Qoff + 15]) + tt
108      q18 = RL(q18, 9); q18 += q17;
109      if (0x00020000 != ((q18^q17)&0xa0020000))
110        continue;
111
112      uint32 q19 = GG(q18, q17, q16) + tt19;
113      q19 = RL(q19, 14); q19 += q18;
114      if (0x80000000 != (q19 & 0x80020000))
115        continue;
116
117      uint32 q20 = GG(q19, q18, q17) + tt20;
118      q20 = RL(q20, 20); q20 += q19;
119      if (0x00040000 != ((q20^q19) & 0x80040000))
120        continue;
121
122      block[1] = q17-q16; block[1] = RR(block[1], 
123      uint32 q2 = block[1] + tt1; q2 = RL(q2, 12);
124      block[5] = tt5 - q2;
125
126      Q[Qoff + 2] = q2;
127      Q[Qoff + 17] = q17;
128      Q[Qoff + 18] = q18;
129      Q[Qoff + 19] = q19;
130      Q[Qoff + 20] = q20;
131      MD5_REVERSE_STEP(2, 0x242070db, 17);
```

RECAP

Instant computation doesn't give any instant exploitation.

-> Instant exploitation relies on pre-computed collisions **and** file format tricks.

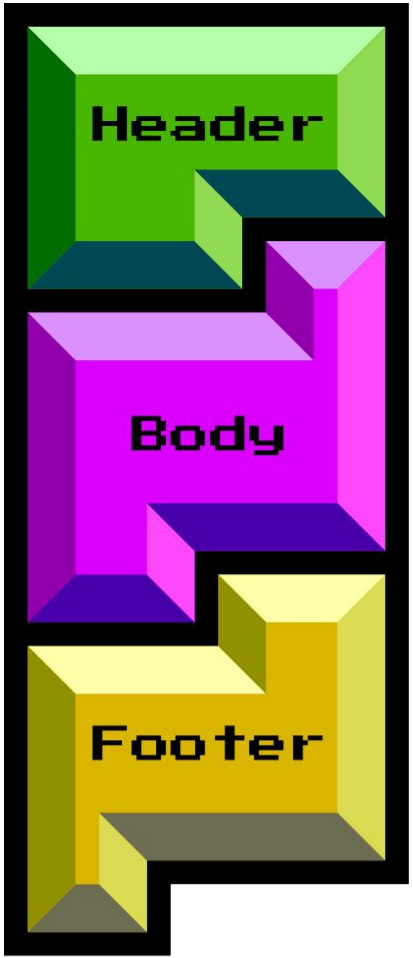
Basics of (most) File formats

A typical general structure

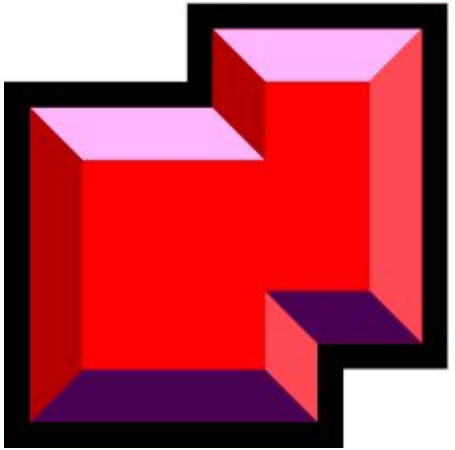
header: required at the start of the file.
It defines the file type, versions, metadata...

body comes after the header.
It's made of several chunks that may be moved around.

footer follows the body.
It indicates that the file is complete.
Parsers ignore any following data.



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**.

Robust parsers are 'detectives'

Most files have all body chunks in a perfect and optimal arrangement,
But parsers must be robust against weird structures or incomplete implementations.

-> In practice, they just parse chunks and collect data on the way.
And when a footer [or EOF] is reached,
they check if they have enough data to render something.
If not the file is considered invalid.

Even if most files are perfectly structured,
robust parsers behave more like detectives.

Most files are perfectly structured

They were generated
by one of the standard libraries,
in normal conditions,
and with typical requirements.

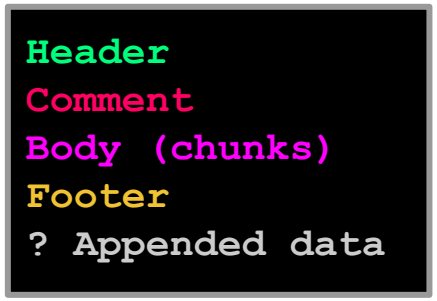
~~Corner cases~~

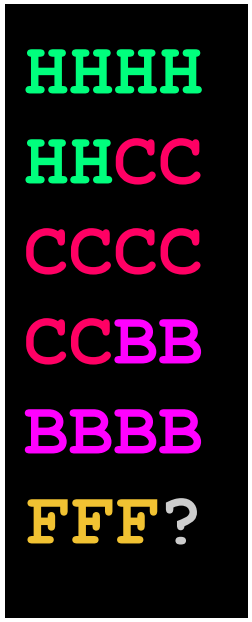
Abusing 'detective' parsers

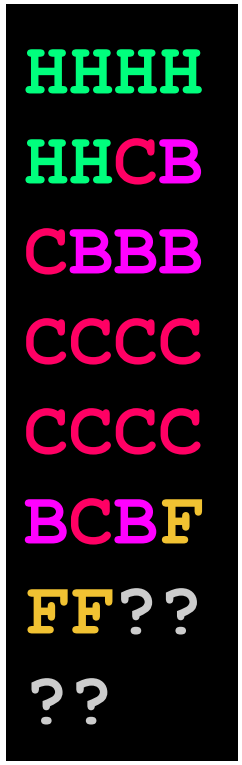
Since parsers behave like detectives,
we can move pieces of a file around, split them, hide them among rubbles.

For a detective,
an escape room and a simple list of hints are equivalent.

For most parsers,
the presence/completeness/order of the files structures may not matter.







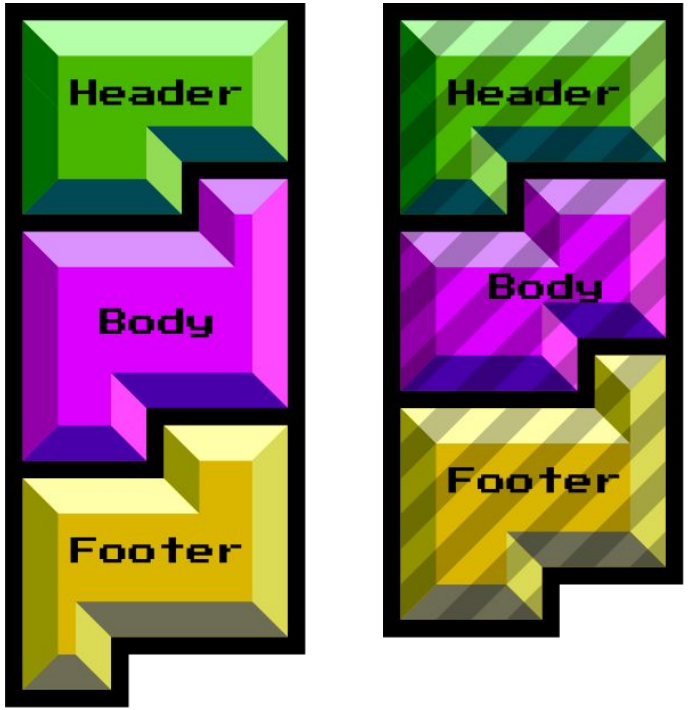
These files are equivalent (from a parser perspective).

Same content, different structure.

Turning one into another is usually (very) easy.

Take two files...

(of the same file type)



Plan a special common header.

Same images dimensions? Color space?
Remove some features.
Flatten content.

...

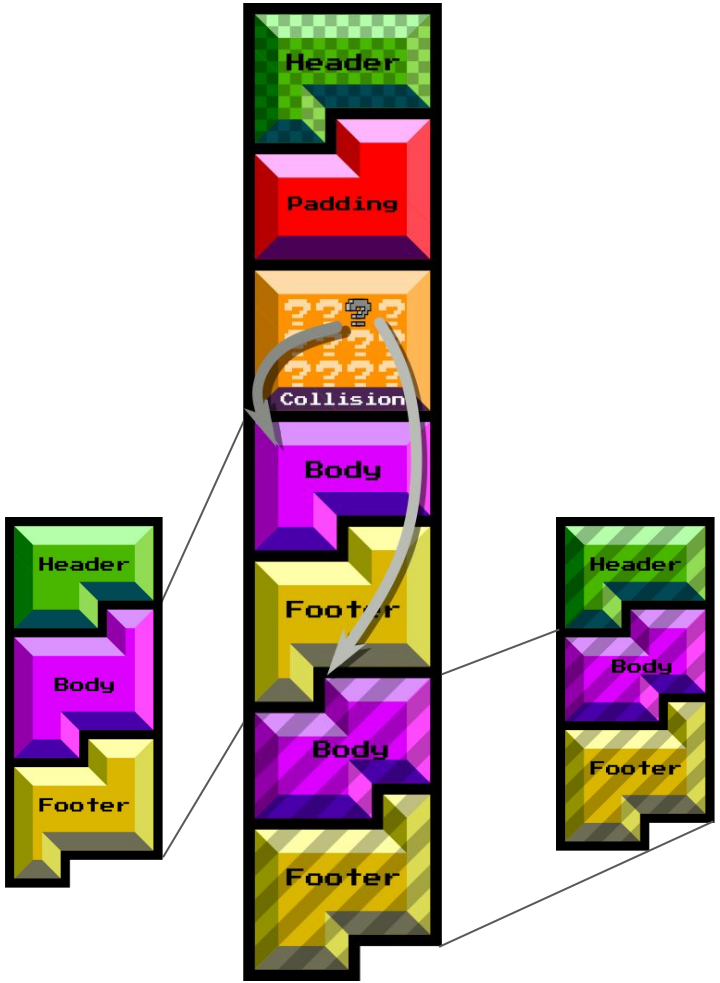
Compute the collision for this header.

Padding and randomness with tiny differences.
These differences follow some patterns
that will be abused.
Margin errors have to be mitigated.

Create a super file combining both files' data

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

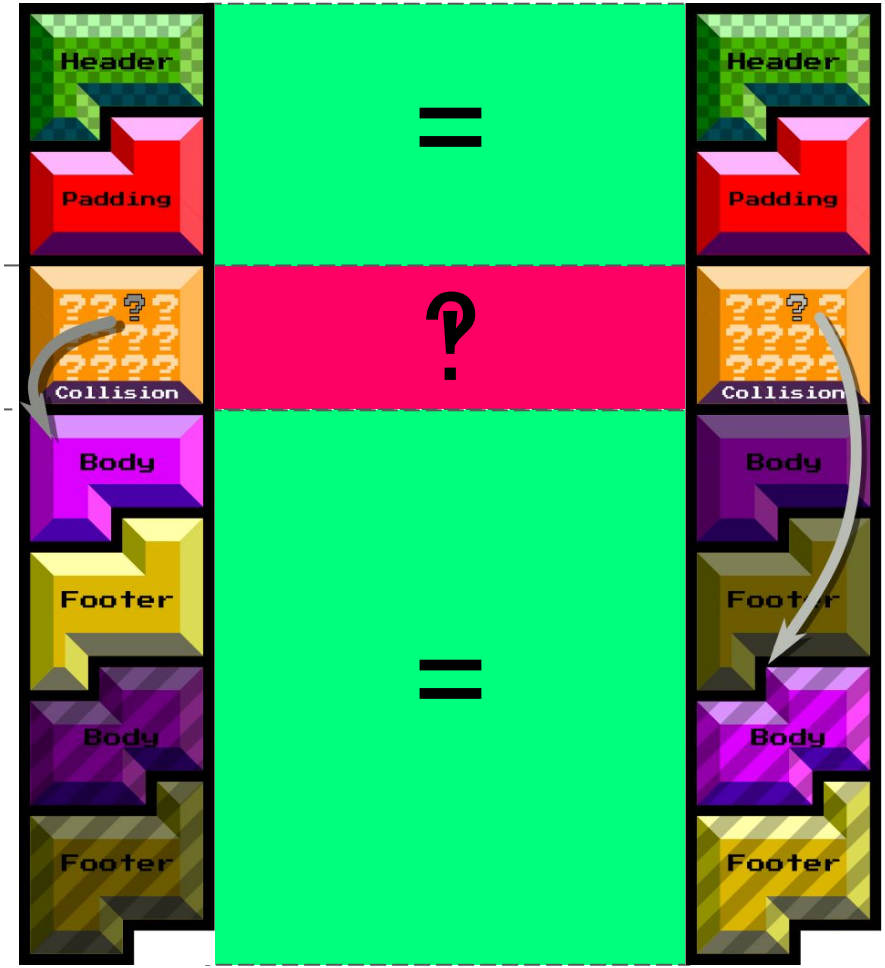
The collision blocks differences will act like a switch to enable one content or the other.



PREFIX

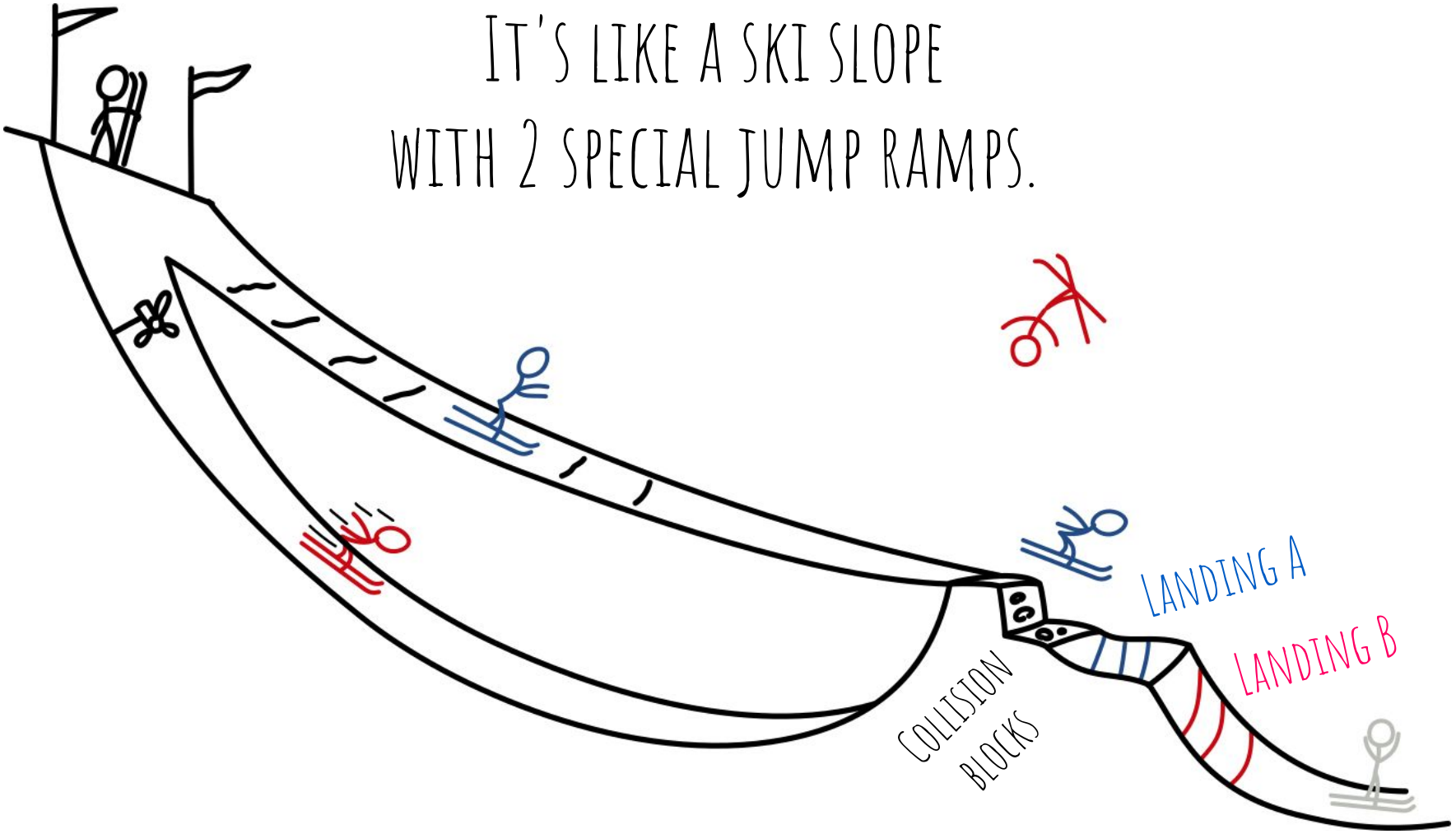
COLLISION

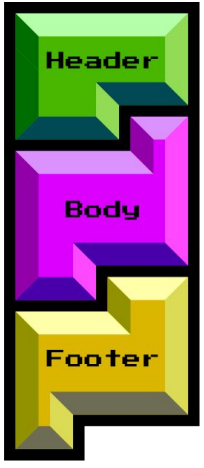
SUFFIX



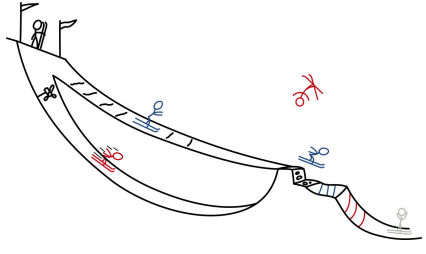
It's an IPC, so their hash value is the same.

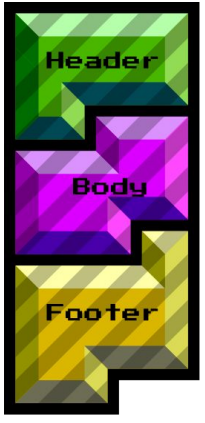
It's like a ski slope with 2 special jump ramps.





Each colliding file will render as one of the original pair.






A VERY SPECIAL
COLLISION

Now let's look at something different.

Poetry…?

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

...

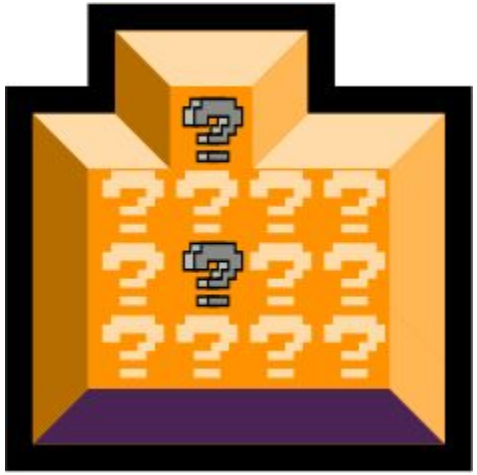
A cryptico poem
https://github.com/Jurph/word-decrementer

Now we hath md5,
no enemy dares!
Only we have
the shares.

...

Increment the 10th letter of each sentence.

(leading spaces are tolerated)



Our second block - another Identical Prefix Collision: UniColl.

```
scripts$ ./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
[...]
scripts$ _
```

WORDS OF 32 BITS

```
scripts$ cat prefix
Here is my prefix!!\n
scripts$ du -b prefix
20      prefix
scripts$ _
```

USE EXACTLY THIS PREFIX FOR YOUR FIRST TRY!

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

Sometimes, unicoll just... fails! Just re-run it then!

Happy end

```
[...]
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
```

Dead ends

```
[...]
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
```

```
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 minutes to compute

Important difference with FastColl:

- prefix as a part of the collision blocks (!!)
  -> **no padding**

- differences:
  10th char of prefix += 1 (!!)
  10th char of 2nd block -= 1

Output of a UniColl computation

A true UNIcorn of a COLLision

A hybrid IPC where:

- you can define the data around the first difference.
- you can set the first difference:

    your text AND your text +1

No other collision does that.

Why +1 on the 10th character?

- because crypto
  (due to specific MD5 properties)
- no, you can't change it as you like.
- the other working cases are not
  as easy to exploit.

Other working cases:

https://www.cwi.nl/system/files/PhD-Thesis-Marc-Stevens-Attacks-on-Hash-Functions-and-Applications.pdf#page=200

- $\delta B = \pm(\delta m_{11} = 2^{15}, \delta m_4 = \delta m_{14} = 2^{31})$ [WY05]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_2 = 2^8, \delta m_{11} = 2^{15}, \delta m_4 = \delta m_{14} = 2^{31})$ [SSA+09b]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_{11} = 2^b)$ for $b \in \{0, \ldots, 30\}$ [SLdW07c]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_{11} = 2^{31})$ [SLdW07c]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{10} = 2^{31})$ [XF10]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_8 = 2^{31})$ [XLF08]: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_6 = 2^8, \delta m_9 = \delta m_{15} = 2^{31})$ [XFL08]: $i = 37$, $\delta WS_{37} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_9 = 2^{27}, \delta m_2 = \delta m_{12} = 2^{31})$ [VJBT08]: $i = 37$, $\delta WS_{37} \in \{0, \underline{2^{31}}\}$.
- $\delta B = \pm(\delta m_4 = 2^{20}, \delta m_7 = \delta m_{13} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_2 = 2^8)$: $i = 37$, $\delta WS_{37} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{11} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31}, \delta m_8 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_2 = 2^8, \delta m_{14} = 2^{31})$: $i = 37$, $\delta WS_{37} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_4 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_{14} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_4 = 2^{25})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{25})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_{14} = 2^{16})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_4 = 2^{20})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_6 = 2^8)$: $i = 50$, $\delta WS_{50} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_9 = 2^{27})$: $i = 50$, $\delta WS_{50} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_9 = 2^{27})$: $i = 37$, $\delta WS_{37} \in \{0, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31}, \delta m_{11} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{31}, \delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{25}, \delta m_{13} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{0, \underline{2^{31}}\}$.

```
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

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
```

With N=2:

- less predictable difference

+ 16 fixed bytes after the first difference

With N=3:

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
```

Other implemented variants:

cf https://github.com/cr-marcstevens/hashclash/blob/master/scripts/poc_no.sh#L29-L44

N=1: "--diffm2 9" [by default]

N=2: "--diffm13 28 --diffm0 32 --diffm6 32"

N=3: "--diffm6 9 --diffm9 32 --diffm15 32"

Prefix

```
00: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40: .h .e .r .e .  .i .s .  .m .y .  .p .r .e .f .i
50: .x .! .! 0a
```

```
00: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40: .h .e .r .e .  .i .s .  .m .y .  .p .r .e .f .i
50: .x .! .! 0a a4 8e d8 3f ae 42 a5 6b 47 e1 b4 72
60: 7a 86 27 96 60 3a e6 9a 8a 37 7d 2f 8e ac a6 ad
70: fd 56 ff d8 23 59 1c 81 da 57 1c 84 ee f5 17 07
80: 39 f9 b5 e5 d8 a6 c4 02 89 df e2 c0 82 1e f8 fa
90: 1e c3 c4 3e 77 17 12 98 d6 78 ed 80 dc 4f 83 86
a0: 21 68 77 44 e2 dc 81 c8 69 33 eb 95 3a 60 08 a0
b0: 05 37 f7 cc 0b b1 ee 94 76 0c af da 18 8b c2 57
```

```
00: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40: .h .e .r .e .  .i .s .  .m .z .  .p .r .e .f .i
50: .x .! .! 0a a4 8e d8 3f ae 42 a5 6b 47 e1 b4 72
60: 7a 86 27 96 60 3a e6 9a 8a 37 7d 2f 8e ac a6 ad
70: fd 56 ff d8 23 59 1c 81 da 57 1c 84 ee f5 17 07
80: 39 f9 b5 e5 d8 a6 c4 02 89 de e2 c0 82 1e f8 fa
90: 1e c3 c4 3e 77 17 12 98 d6 78 ed 80 dc 4f 83 86
a0: 21 68 77 44 e2 dc 81 c8 69 33 eb 95 3a 60 08 a0
b0: 05 37 f7 cc 0b b1 ee 94 76 0c af da 18 8b c2 57
```

collision blocks

The prefix can be any size. Only the last block is special (remember: no padding)

2nd rule of UniColl's last prefix block

Length multiple of 4 (otherwise it won't work as intended)

```
$ md5sum collision*
47bc2a72c2885bcc624342fe8d3756fd  collision1.bin
478ccf32058daf9d1fa44fd78959ed50  collision2.bin
$ xxd prefix
00000000: 7070 7070                                pppp
$ xxd collision1.bin | head -1
00000000: 7070 7070 2c65 975b e9de 9dd9 98dc 7cd4  pppp,e.[......|.
```

If its length is a multiple of 4, the end of the prefix's last block is in the collision block.

If not, the collision block doesn't start with the prefix' content.

```
$ md5sum collision*
43a22967d7d6343da5c50da277f63f62  collision1.bin
43a22967d7d6343da5c50da277f63f62  collision2.bin
$ xxd prefix
00000000: 70                                       p
$ xxd collision1.bin | head -1
00000000: c830 2859 684c dc50 cbba 8f0f 8fa7 48ef  .0(YhL.P......H.
```

3rd rule of UniColl's last prefix block

If that part is longer than 20 bytes,

it may never collide.

-> increase delays in the script.

```
$ diff poc_no.sh long_poc_no.sh
11c11
< data=200000
---
> data=1000000
56c56
<       sleep 10
---
>       sleep 120
```

Standard version

```
000: .H .e .r .e .  .i .s .  .m .y .  .l .o .n .g .
010: .p .r .e .f .i .x .! \n 17 ef 03 3a 3b 0b d8 ba
020: 11 0c 27 fe 71 8c ec 39 ab 48 97 fb 81 8e 7a 50
030: 4c 5d 44 73 05 08 f7 ff 16 06 44 db ea 1c 8b 50
040: 70 f4 66 04 c8 23 16 b0 83 99 e5 76 09 92 13 f0
050: 3e 3e d2 bb d9 fb 3e 00 78 64
060: aa e1 03 6b 77 61 46 3b 84 d4
070: 23 b9 e2 26 fd 35 6c d0 e4 56

000: .H .e .r .e .  .i .s .  .m .z
010: .p .r .e .f .i .x .! \n 17 ef
020: 11 0c 27 fe 71 8c ec 39 ab 48
030: 4c 5d 44 73 05 08 f7 ff 16 06
                16 b0 83 98
                3e 00 78 64
                46 3b 84 d4
                6c d0 e4 56
```

24 bytes:

```
real    2m21.203s
user    33m3.508s
sys     0m55.864s
```

```
000: .H .e .r .e .  .i .s .  .m .y .  .l .o .n .g .
010: .p .r .e .f .i .x .! .0 .1 .2 .3 0a a5 4d 0a a2
020: 4a 9e 0e b1 20 bc df 89 bf ed d3 eb bb e2 e9 6f
030: e9 fa 24 39 02 42 4f 8b 77 77 8c 50 0f ff a7 86
040: b4 7a e4 de 00 27 f8 c7 ad ef b1 2d 25 35 f7 9f
050: fa 01 64 3c 06 8a 5b 66 ad 46 fa 5f 11 ea 91 c6
060: d3 70 11 45 57 dd 8a 0f b8 af 4d 0d d2 d2 40 43
070: 48 c0 22 4d 67 b9 b2 5b 0a 98 ff e0 7f d3 7b 6d

000: .H .e .r .e .  .i .s .  .m .z .  .l .o .n .g .
010: .p .r .e .f .i .x .! .0 .1 .2 .3 0a a5 4d 0a a2
020: 4a 9e 0e b1 20 bc df 89 bf ed d3 eb bb e2 e9 6f
030: e9 fa 24 39 02 42 4f 8b
040: b4 7a e4 de 00 27 f8 c7
050: fa 01 64 3c 06 8a 5b 66
060: d3 70 11 45 57 dd 8a 0f
070: 48 c0 22 4d 67 b9 b2 5b
```

28 bytes:

```
real    13m21.452s
user    227m35.496s
sys     3m27.896s
```

Slower version

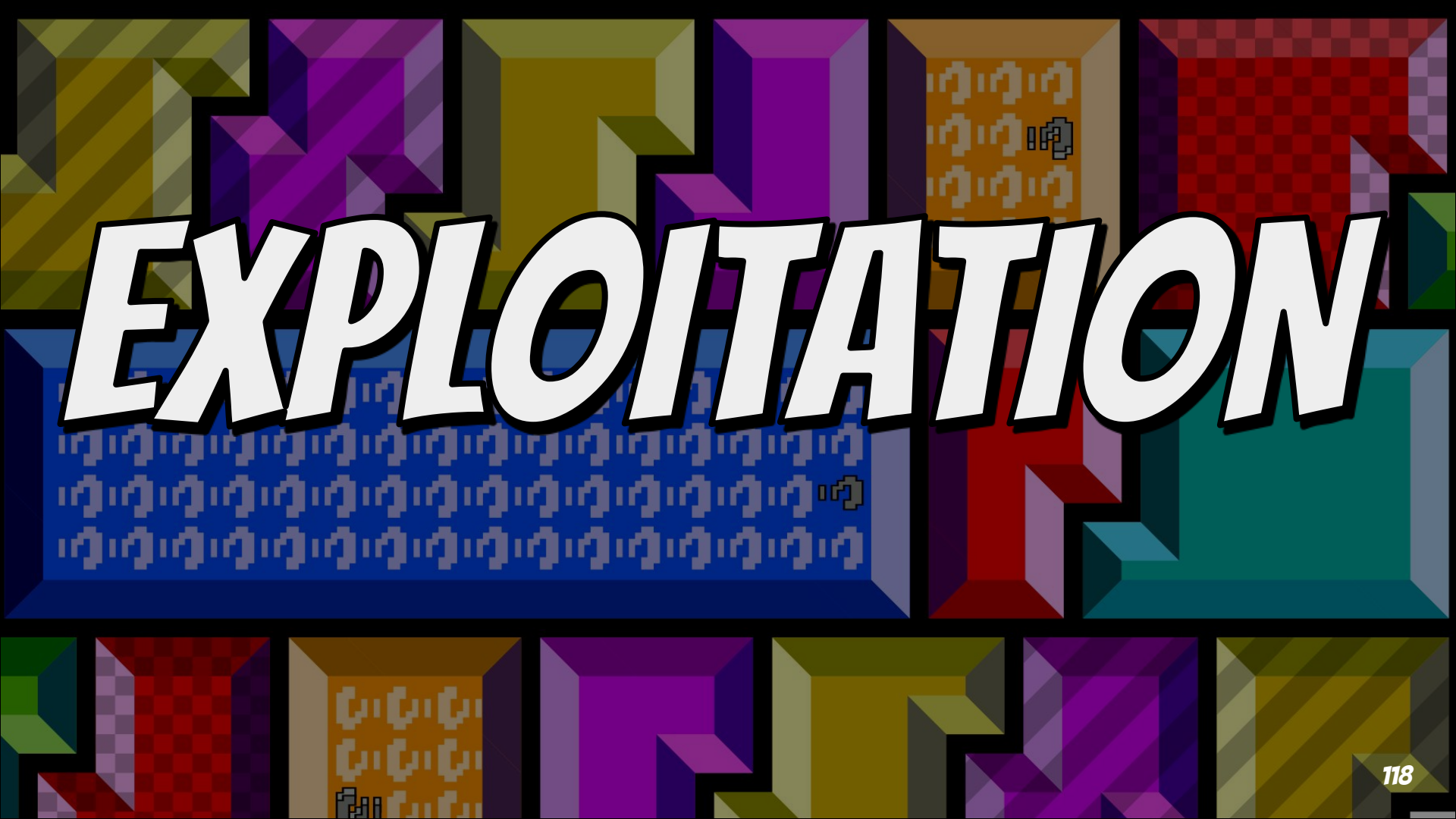
UniColl

two blocks
a few minutes
In prefix

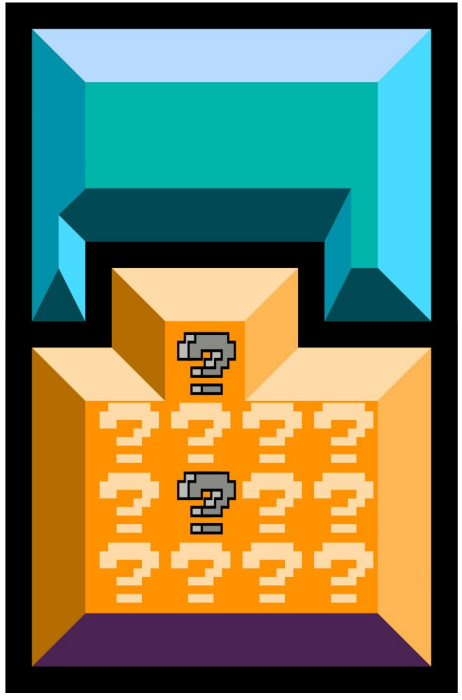
Slightly slower,
but easy to exploit.

```
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
```


EXPLOITATION

Your first hash collision exploit



Plan your exploit

Prepare

1. Study format specs, look for features you need.
2. Choose attack: FastColl, UniColl [tree]...
3. Plan your file structure (pen & specs).

Craft

4. Craft mockup files: check compatibility, CRCs...
5. Ignore collision ranges to simulate colliding files.

Compute

6. Extract prefixes from mockups.
7. Run computation(s).

⇤ ⇥	Padding, for alignments
# & % ! @	collision blocks' randomness needs to be ignored
... ? ...	Differences need to be taken into account
... 🛑 ?	Two contents need to co-exist.

A mockup file before computation

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,

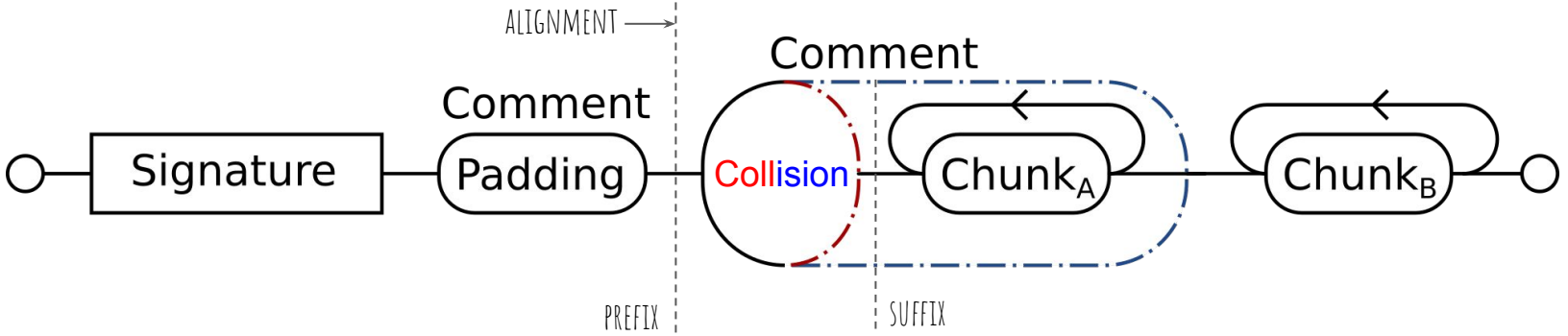
which is bigger than the collision block

but doesn't grow uncontrollably.

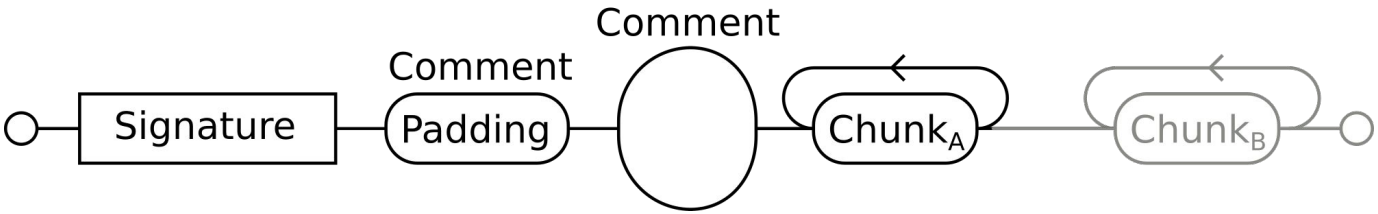
```
FastColl: 61 52 3E ⇔ 61 D2 3E
UniColl:  00 71 .c .O .L .L ⇔
          01 71 .c .O .L .L
```

Layout of a classic collision+format exploitation

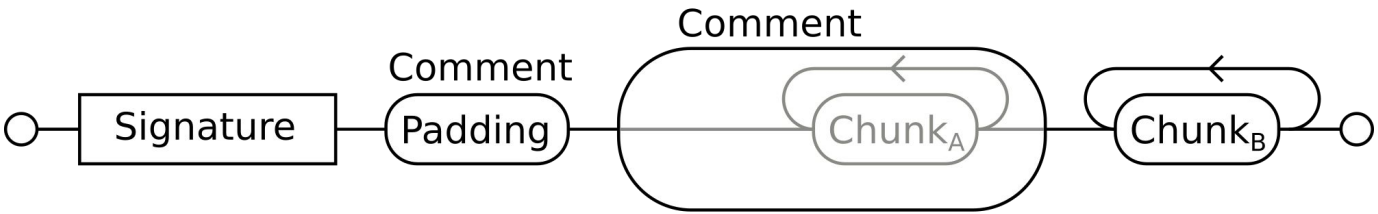
1. A fixed-length comment for padding.
2. A variable length comment at the start of collision blocks.
3. Using collision blocks to grow this comment over a first file's data, followed by a second's file data.



Case A (short comment)



Case B (long comment)



Plan your generic exploit

Getting an exploit PoC (pair) is great to convince/test!
Making a script to instantly generate any PoC is even better!

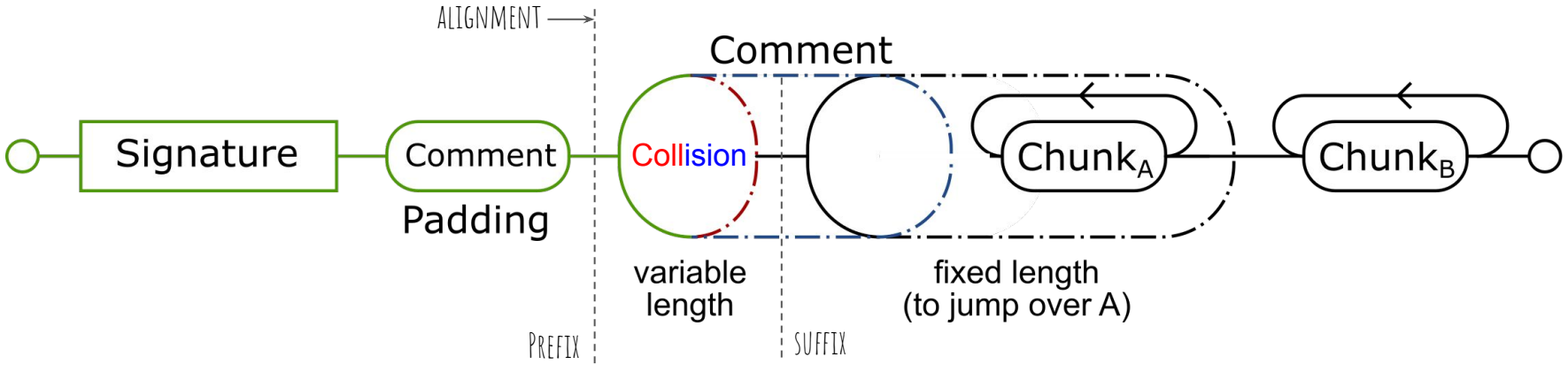
Explore the format landscape, standard implementations.
Understand compatibility in depth.

Making it generic

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

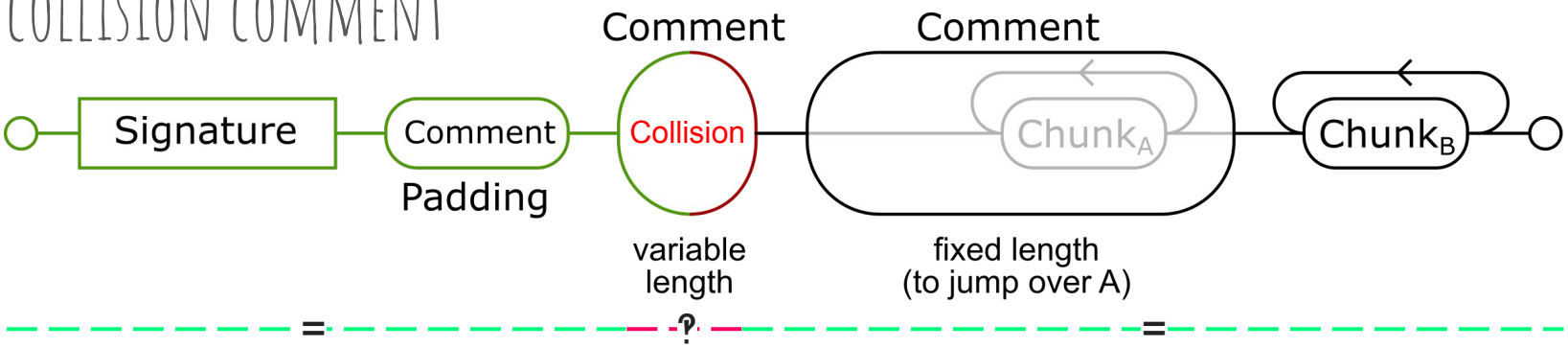
-> one extra comment to jump over these chunks

with its **declaration** switched on/off by the variable comment

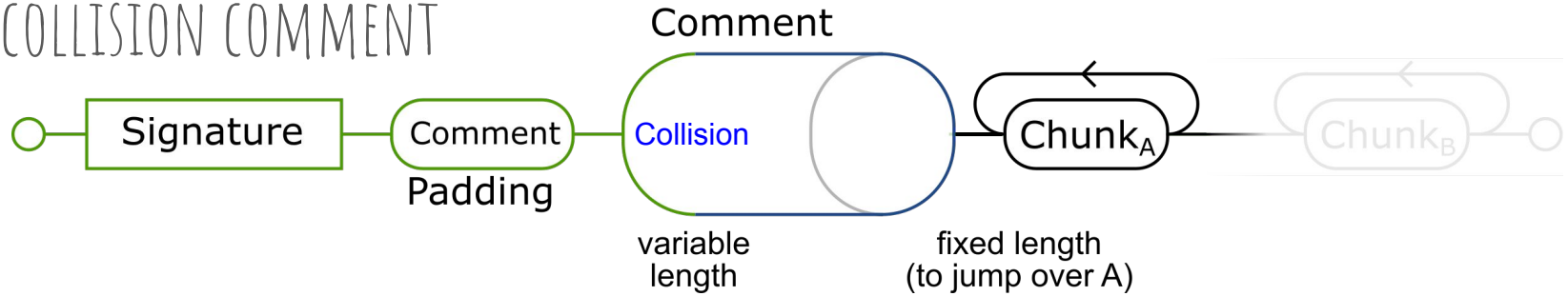


A chain of three comments

Short collision comment



Long collision comment



It's not always easy.

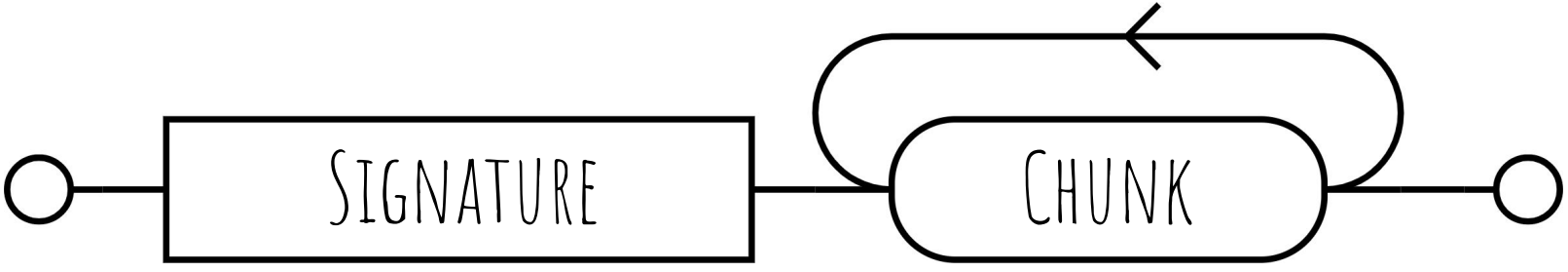
Identify required structures of the format.
Check structure sizes: constant? if not, what are the margins?

Explore tools and options:
Merging (PDF pages, GIF frames) then selectively hiding
 is a quick way to normalize 2 contents.
Some minor tools' output might be optimal for manipulation.

Exploiting PNG with UniColl

The Portable Network Graphics format

/ˌpiːɛnˈdʒiː/ PEE-EN-JEE
/pɪŋ/ PING



The most regular of the common formats:
a signature then a sequence of chunks.

The PNG signature

- enforced at offet **0**
- fixed, always the same **8** bytes:

```
89 50 4E 47 0D 0A 1A 0A
```

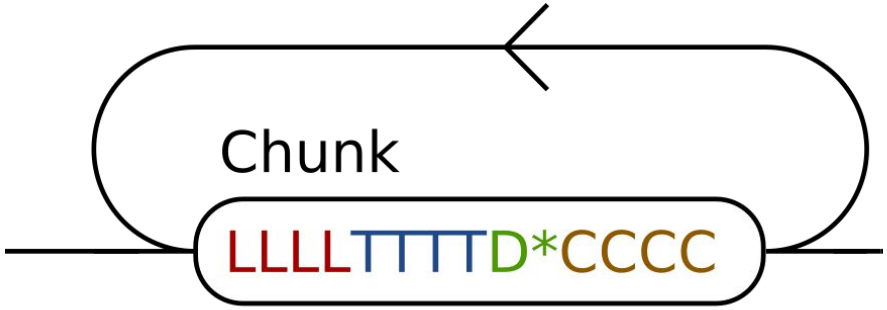
(Trivia) it's made of special characters to detect various errors:

```
\x89 P N G \r \n ^Z \n
```

non ascii · carriage return · line feed · end of file · line feed

The PNG format at chunk level

- the **Length**, big endian on **4** bytes.
- the **Type**, on **4** letters.
- the **Data**, of the given **Length**.
- the **CRC** of **Type** and **Data**.
  - they are usually ignored



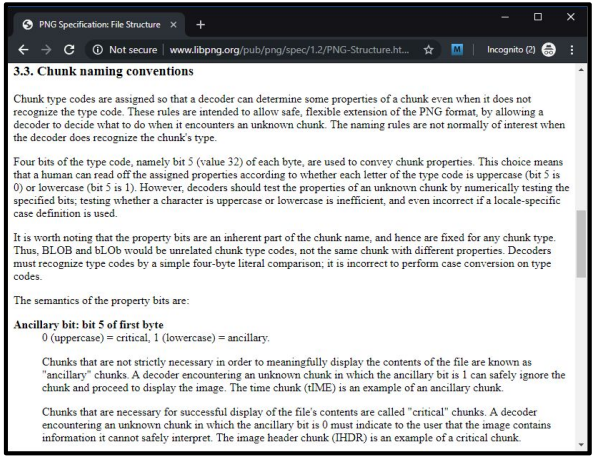
Length	>u	4
Type	c	4
Data (type+data)	b	Length
CRC32	>u	4

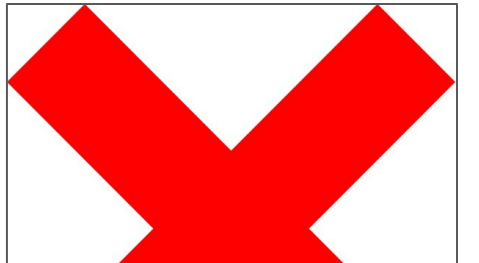
Lower case-typed chunks are ignored

first letter:

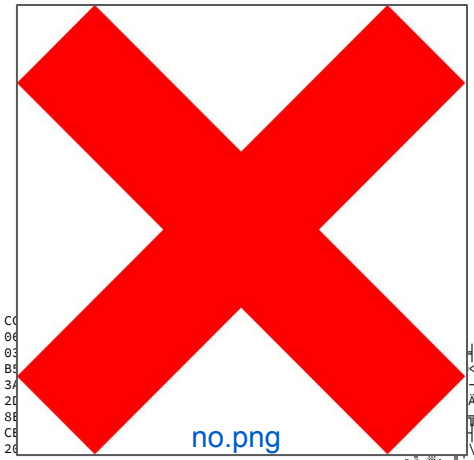
- uppercase == critical:
  - Standard: `IHDR` Header / `PLTE` Palette / `IDAT` Data / `IEND` End
  - Non-standard: `CgBI` Apple custom PNG
- lowercase == secondary:
  - In the specs: `bKGD cHRM gAMA hIST iCCP pHYs tIME tRNS sBIT sPLT sRGB tEXt iTXt zTXt`
    - ignored if not in the specs: aLIG / cOLL / sKIP

http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html#Chunk-naming-conventions





LET'S WALK THROUGH A SIMPLE PNG FILE...



Reminder

A PNG file is a **signature**,

Then a sequence of Chunks:

starts with a **length** then a **type**, and ends with a **CRC**.

To walk through a PNG

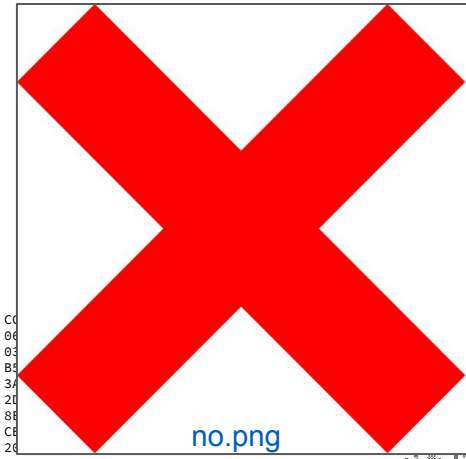
check the **signature**, followed by a chunk start (typically **IHDR**)

then look for sequences like:

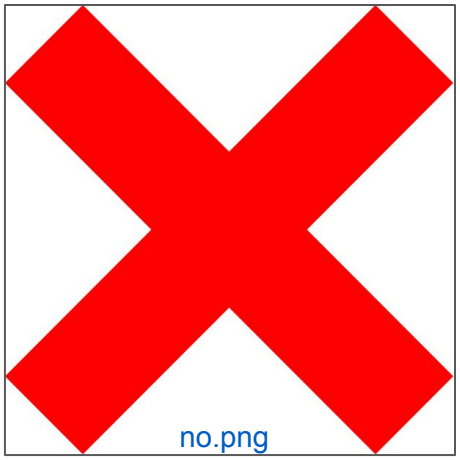
XX XX XX XX **LL LL LL LL** **AA AA AA AA**

- 4 random-looking bytes — Ex: 88 CF CD 8E
- a big-endian length — Ex: 00 00 00 24
- a 4-bytes alpha string — Ex: PLTE

- Note the chunk type.
- Skip L bytes
- (Repeat until chunk IEND).



```
0000:  89 .P .N .G \r \n ^Z \n 00 00 00 0D .I .H .D .R
0010:  00 00 02 64 00 00 02 64 08 03 00 00 00 88 CF CD
0020:  8E 00 00 00 24 .P .L .T .E FF FF FF FF 7D 7D FF
0030:  6F 6F FF FE FE FF 7E 7E FF 00 00 FF 80 80 FF 6D
0040:  6D FF 6E 6E FF 30 30 FF FD FD FF 2F 2F A6 F0 78
0050:  4E 00 00 0A D9 .I .D .A .T 78 DA EC D2 D1 09 5A
0060:  01 14 86 31 B5 DE EE 3F 72 1F CF 00 3F 94 20 5F
....
0B20:  A8 65 FB 31 78 59 C7 A8 65 5F 3A 96 FC 01 2A EE
0B30:  B0 6C 0B 6E B3 BA 00 00 00 00 .I .E .N .D AE 42
0B40:  60 82
```



The same file, displayed with HexII and skipping data ranges

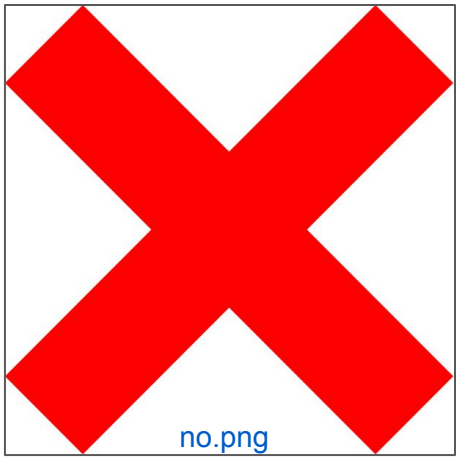
```
0000:  89 .P .N .G \r \n ^Z \n 00 00 00 0D .I .H .D .R
0010:  00 00 02 64 00 00 02 64 08 06 00 00 00 88 CF CD
0020:  8E 00 00 00 24 .P .L .T .E FF FF FF FF 7D 7D FF
0030:  6F 6F FF FE FE 00 00 FF 80 80 FF 6D
0040:  6D FF 6E 6E FF D FF 2F 2F A6 F0 78
0050:  4E 00 00 0A D9 .I .D .A .T 78 DA EC D2 D1 09 5A
0060:  01 14 B5 DE EE 3F 72 1F CF 00 3F 94 20 5F

0B20:  A8 65 FB 31 78 59 C7 A8 65 5F 3A 96 FC 01 2A EE
0B30:  B0 6C 0B 6E B3 BA 00 00 00 00 .I .E .N .D AE 42
0B40:  60 82
```

HEADER

PALETTE

DATA

END



0000-0007:	Signature	89 .P .N .G \r \n ^Z \n
0008-0020:	Image Header	00 00 00 0D .I .H .D .R…88 CF CD 8E
0021-0050:	Palette	00 00 00 24 .P .L .T .E…A6 F0 78 4E
0051-0B34:	Data	00 00 0A D9 .I .D .A .T…0B 6E B3 BA
0B35-0B41:	End	00 00 00 00 .I .E .N .D AE 42 60 82

The file's chunk map

```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)
```

The PNG format is really

easy to parse or manipulate at chunk level.

Cf minipng.py

Recap: exploit strategy

- lower-case chunk are ignored. aLIG/cOLL/sKIP/wHAT/eVER/…
- 3 chunks to add:
  1. alignment
  2. collision:aligned with UniColl's 10th character
     to jump over collision blocks with variable length.
  3. skip: one to land successfully, and jump over the first image.
- we can copy the whole images' contents after their signature:
  they're made of sequence of chunks, no need to parse them.

```
0000:  89 .P .N .G \r \n ^Z \n 00 00 00 33 .a .L .I .G
0010:  00 00 00 00 00 00 00 00 0                 00 00
0030:  00 00 00 00 00 00 00 00 0                 00 00
0040:  00 00 00 57 93 27 27 00 00 00 71 .c .O .L .L 00
0050:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00B0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0:  32 B8 1F CE 00 00 00 0D .I .H .D .R 00 00 02 64
00D0:  00 00 02 64 08 03 00 00 00 88 CF CD 8E 00 00 00
00E0:  24 .P .L .T .E FF FF FF FF 7D 7D FF 6F 6F FF FE

0BE0:  A8 65 FB 31 78 59 C7 A8 65 5F 3A 96 FC 01 2A EE
0BF0:  B0 6C 0B 6E B3 BA 00 00 00 00 .I .E .N .D AE 42
0C00:  60 82
```

ALIGNMENT

UNICOLL

```
0000-0007: Signature
0008-0046: Alignment      00 00 00 33 .a .L .I .G…57 93 27 27
0047-00C3: UniColl        00 00 00 71 .c .O .L .L…32 B8 1F CE
00C4-00DC: Image Header   00 00 00 0D .I .H .D .R…88 CF CD 8E
00DD-010C: Palette        00 00 00 24 .P .L .T .E…A6 F0 78 4E
010D-0BF1: Data           00 00 0A D9 .I .D .A .T…0B 6E B3 BA
0BF2-0BFD: End            00 00 00 00 .I .E .N .D AE 42 60 82
```

Source example:

```
chunks = [
  ["aLIG", 0x33*"\0"],
  ["cOLL", 0x71*"\0"],
  ] + read(fNo)
```

Inserting chunks still keeps a PNG file valid.

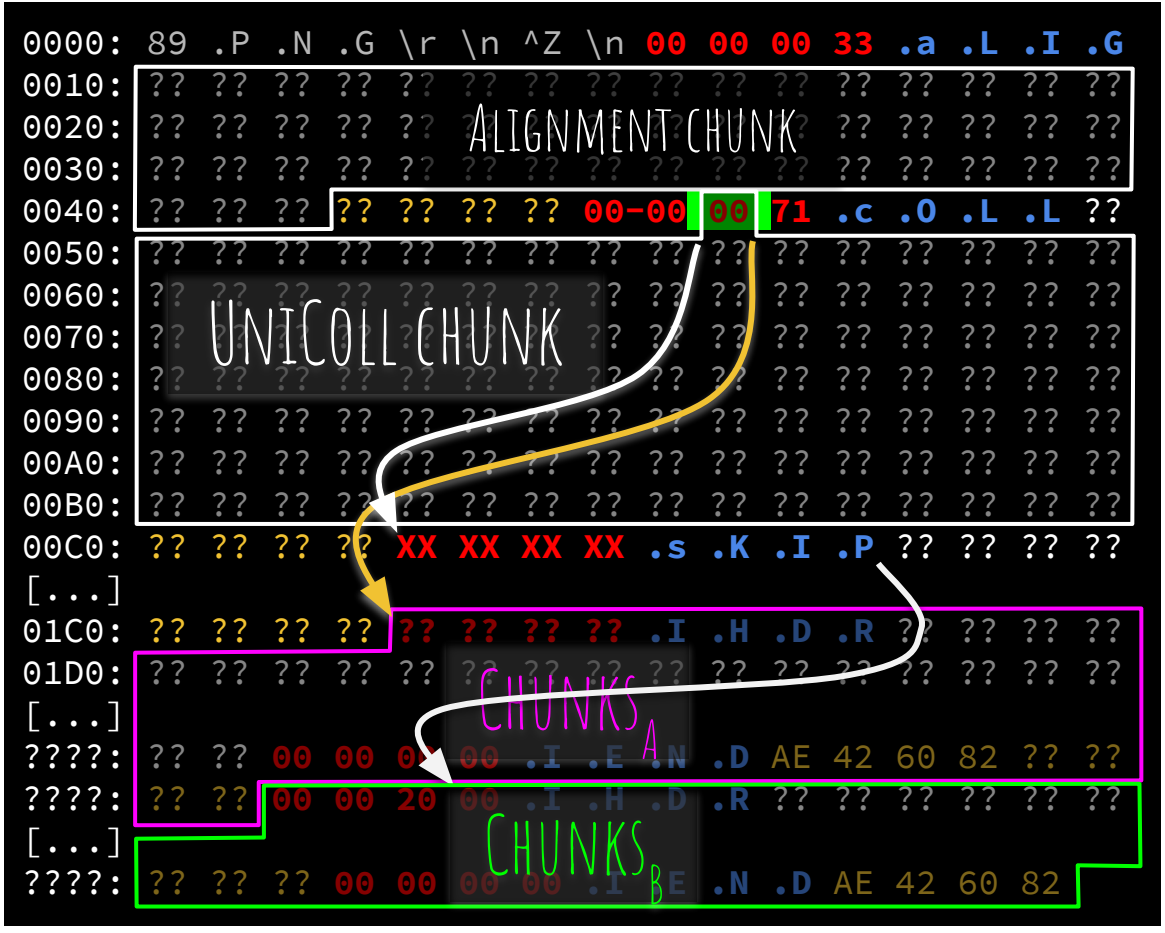
```
0000: 89 .P .N .G \r \n ^Z \n 00 00 00 33 .a .L .I .G
0010: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0020: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0030: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0040: ?? ?? ?? ?? ?? ?? ?? 00-00 00 71 .c .O .L .L ??
0050: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0060: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0070: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0080: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
0090: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00A0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00B0: ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
00C0: ?? ?? ?? ?? XX XX XX XX .s .K .I .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

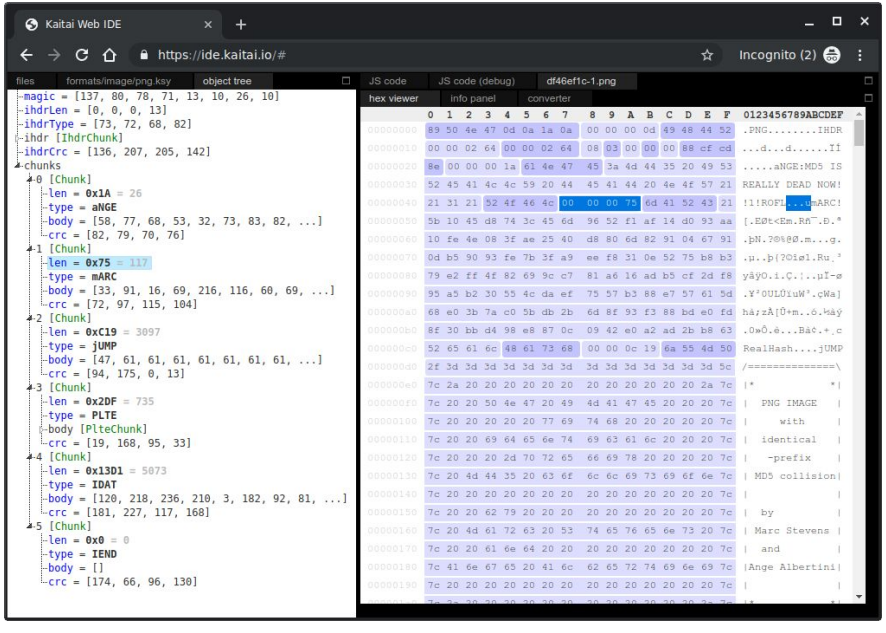
CRC32

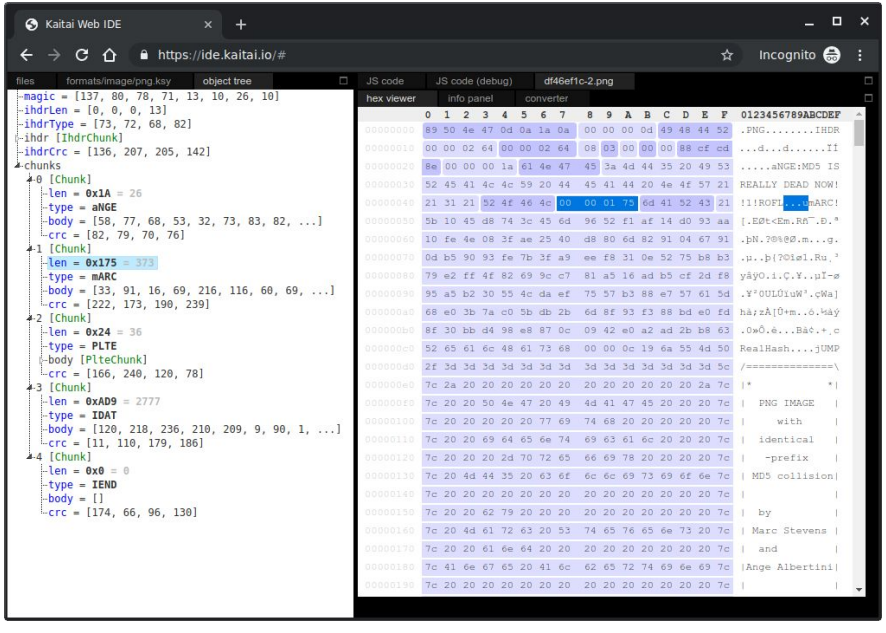
3 DUMMY CHUNKS: ALIGNMENT, COLLISION AND JUMP OVER (THE FIRST IMAGE) DATA



HERE'S A WORKING script FOR REFERENCE.

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!

(the modified copy will be automatically saved in your local storage)

```
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
```

Simplified PNG grammar

png_simple.ksy

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:

and dimensions and colorspace are in the common prefix

-> Study the landscape to understand the scope of your exploit.

They support the undocumented `CgBI` chunk,
but they enforce the specs like no one else!

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

Certificate (medium)

Collision exploit

Reusable PNG via UniColl

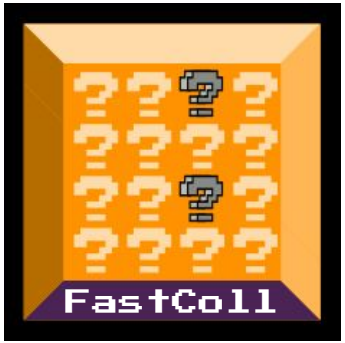
Ange Albertini

INSTRUCTOR

Correct CRCs in UniColliding-PNG ?

- CRCs are ignored
- the collision blocks have different contents
+ the collision blocks also change the length of the chunk
-> 2 different CRCs for different lengths of the collision chunk
  -> correct CRCs are possible


CHOSEN
PREFIX



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

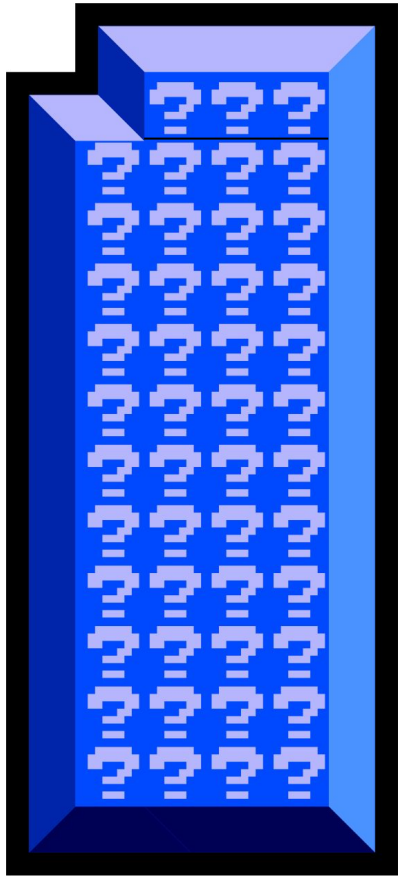
Enforced checksums prevent validity.

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

The ultimate attack

Chosen-Prefix Collisions

HashClashCPC



Our third block:
a Chosen Prefix hash Collision

A word of warning on CPC

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

Often requires backtracking, but it's now automated..

The fewer the collision blocks,

the longer to compute.

example of a computation log:

Corkami collisions:examples/cpc.txt

Launching a hashclash computation

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

No monitoring required (unlike previous versions)

```
...
[*] Timeout reached. Killing process with pid 5760
Killed
[*] Step 5 failed. Backtracking to step 4
[*] Number of backtracks until now: 1
[*] Time before backtrack: 2400 s
[*] Starting step 4
...
```

A 9-block CPC of YES and NO (differences are irrelevant)

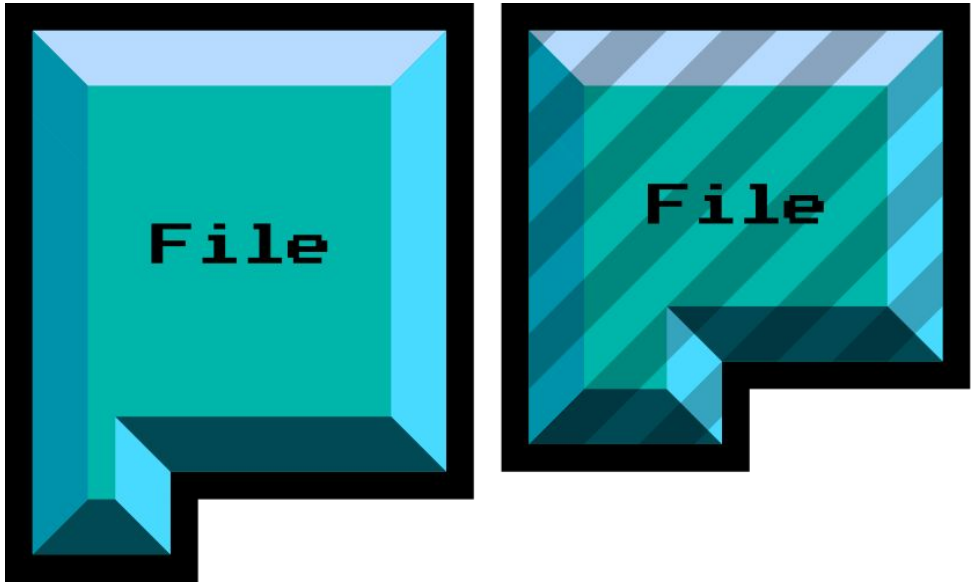
```
0000:  .y .e .s 00-00 00 00 00-00 00 00 00-00 00 00 00
0010:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0020:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0030:  00 00 00 00-00 00 00 00-B7 46 38 09-8A 46 F1 7B
0040:  F3 45 26 13-66 60 C8 01-B9 2A 75 25-5A 67 23 A6
0050:  92 3D EB 8D-B0 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
```



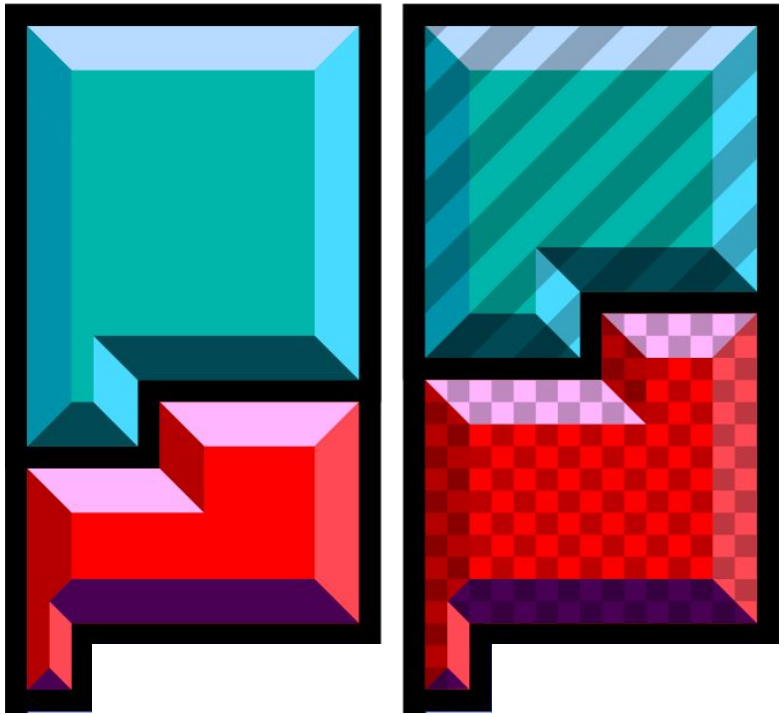
So, we have **two** files. Any pair of files. We can see them as prefixes.

Padding, as usual

Pad both files to the same length,

to a block boundary minus 12(0xC) bytes.

Padding content doesn't matter.



Computation

We compute a collision,

that appends different blocks to both files.

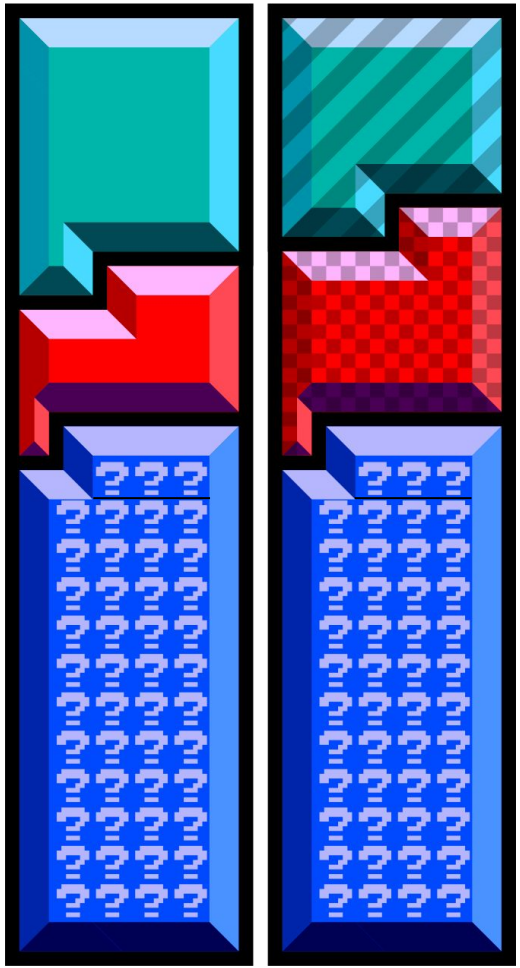
It makes sense only if

both formats tolerate appended data.

(or cover it by a comment).

Block differences are irrelevant in this case.

(we **entirely** control both prefixes).



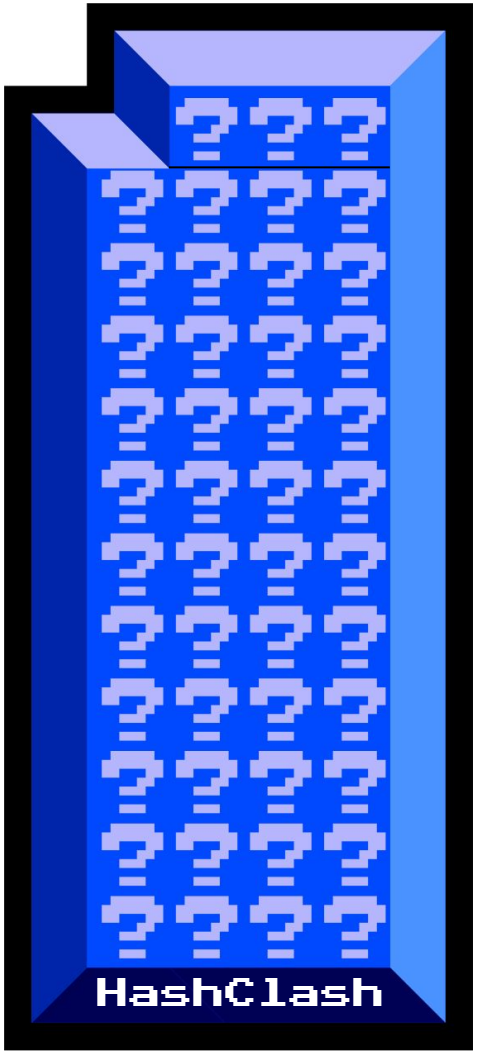
HashClash

7-9 blocks

A few hours

Irrelevant

Almighty, but slower



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.

They're always different.

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

`--maxblocks 9`

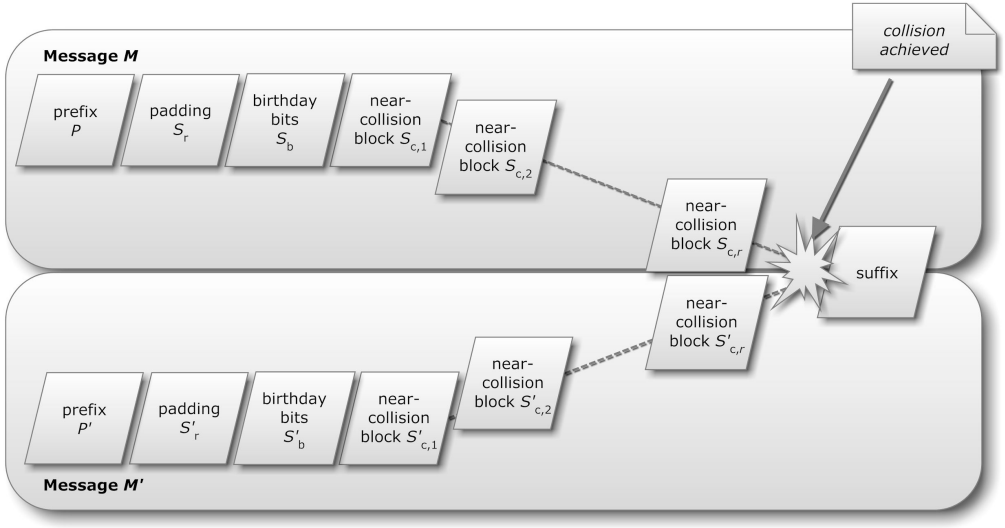
The 2 steps of a CPC

First, an single **Birthday Search**,
then near-collision computation for **each** block.
(which may require **backtracking**)

Only the Birthday Search benefits from GPU.
The B.S. doesn't require any backtracking.

The **fewer** the blocks, the more complex the B.S.

- 400kH for a single block CPC.
- 7-9 blocks is a good trade-off for desktop computation.



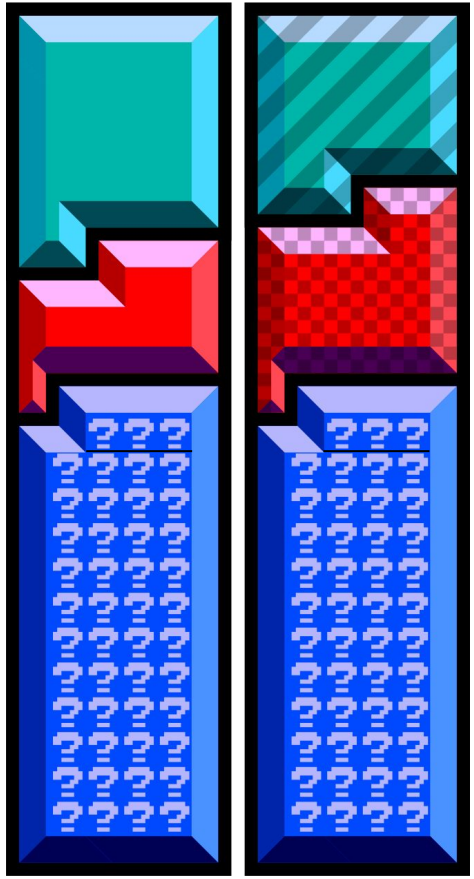
# Blocks	Complexity	Attack publication
9	2^{39}	2009
3	2^{49}	2009
2	**2^{46}**	**2019**
1	2^{53}	2009

Impact of a CPC

If two files formats tolerate appended data:

Compute collision. Done.

+ Straightforward
- Only works for a single pair

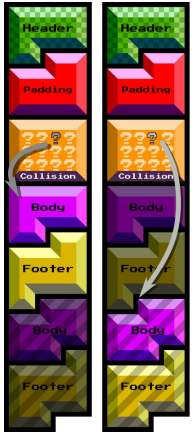


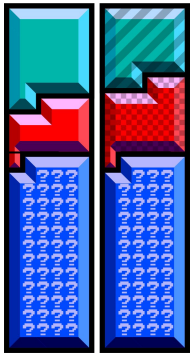
RECAP

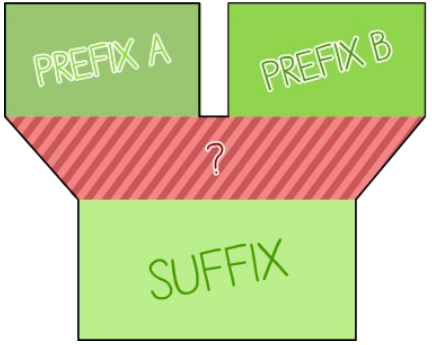
Identical prefix

vs

Chosen Prefix

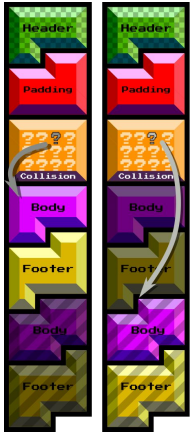






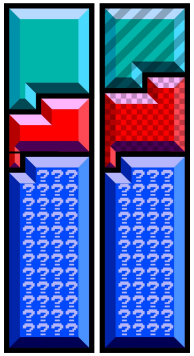
IPC: both contents in each file.

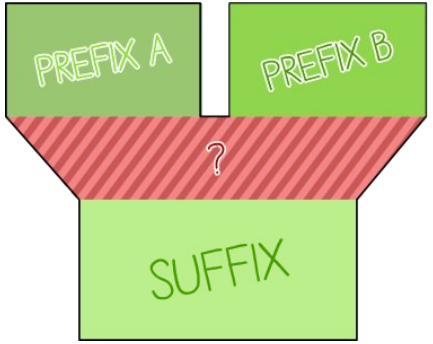
(You might leak your payload)



CPC: Only one content per file.

(evil payload not in the good file)

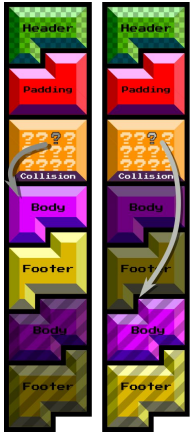




From a content perspective

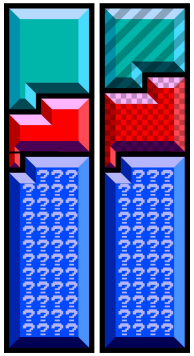
IPC: Deeply modified file structures.

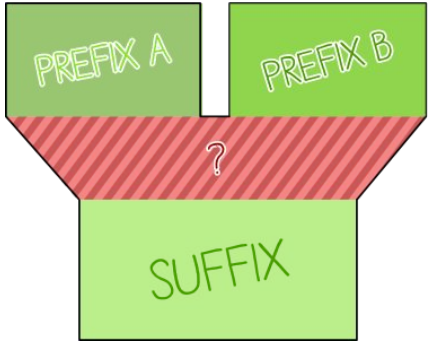
Weird header, 2 payloads, possibly split.



CPC: Each file starts exactly like before.

It's just appended data.





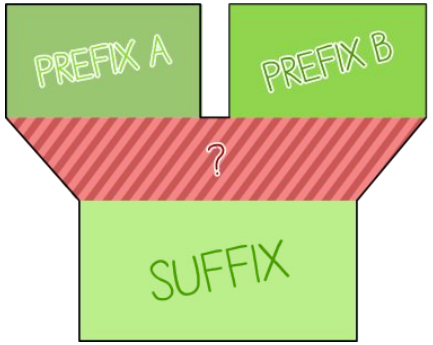
From a file perspective

IPC: Can reuse precomputed collision

so instant generation can be possible.

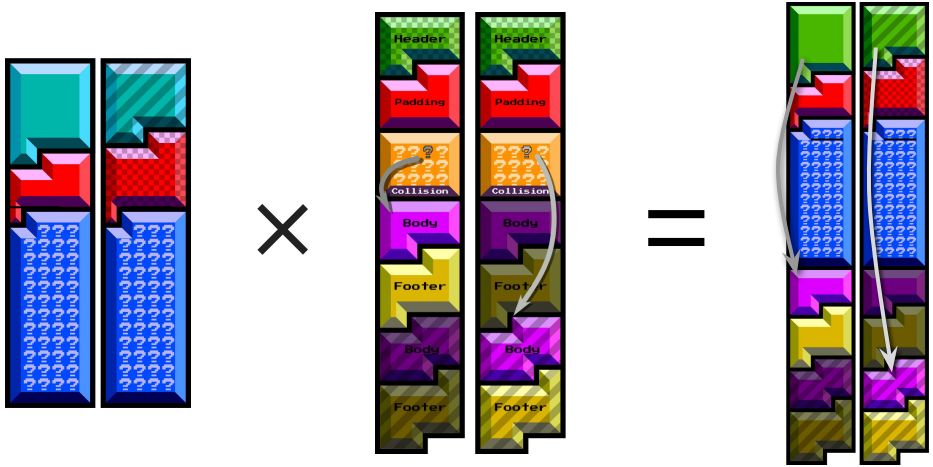
CPC: No re-usability: same computation every time.

-> no optimization, no shortcut.



From a computing perspective

Combining CPC flexibility with IPC re-usability

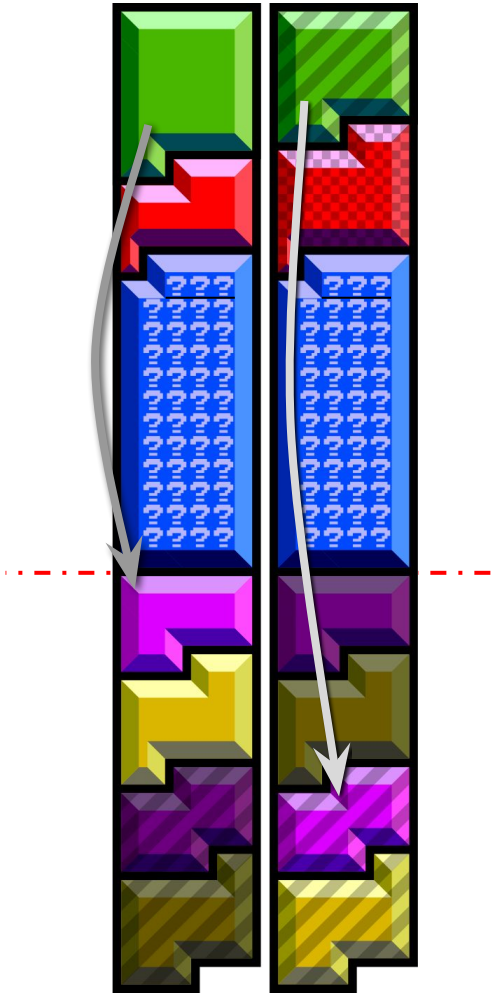


Using CPC as a prefix like an IPC

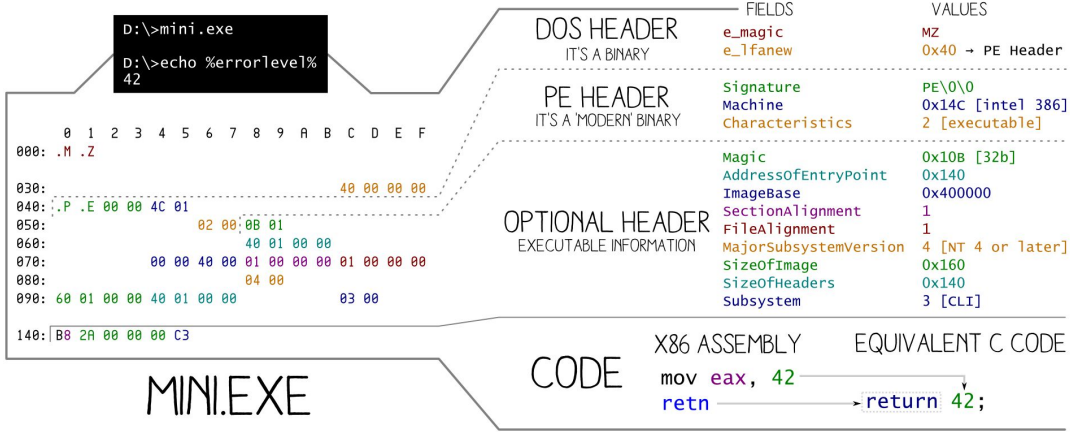
More computing than IPC, but less restrictive.

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

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

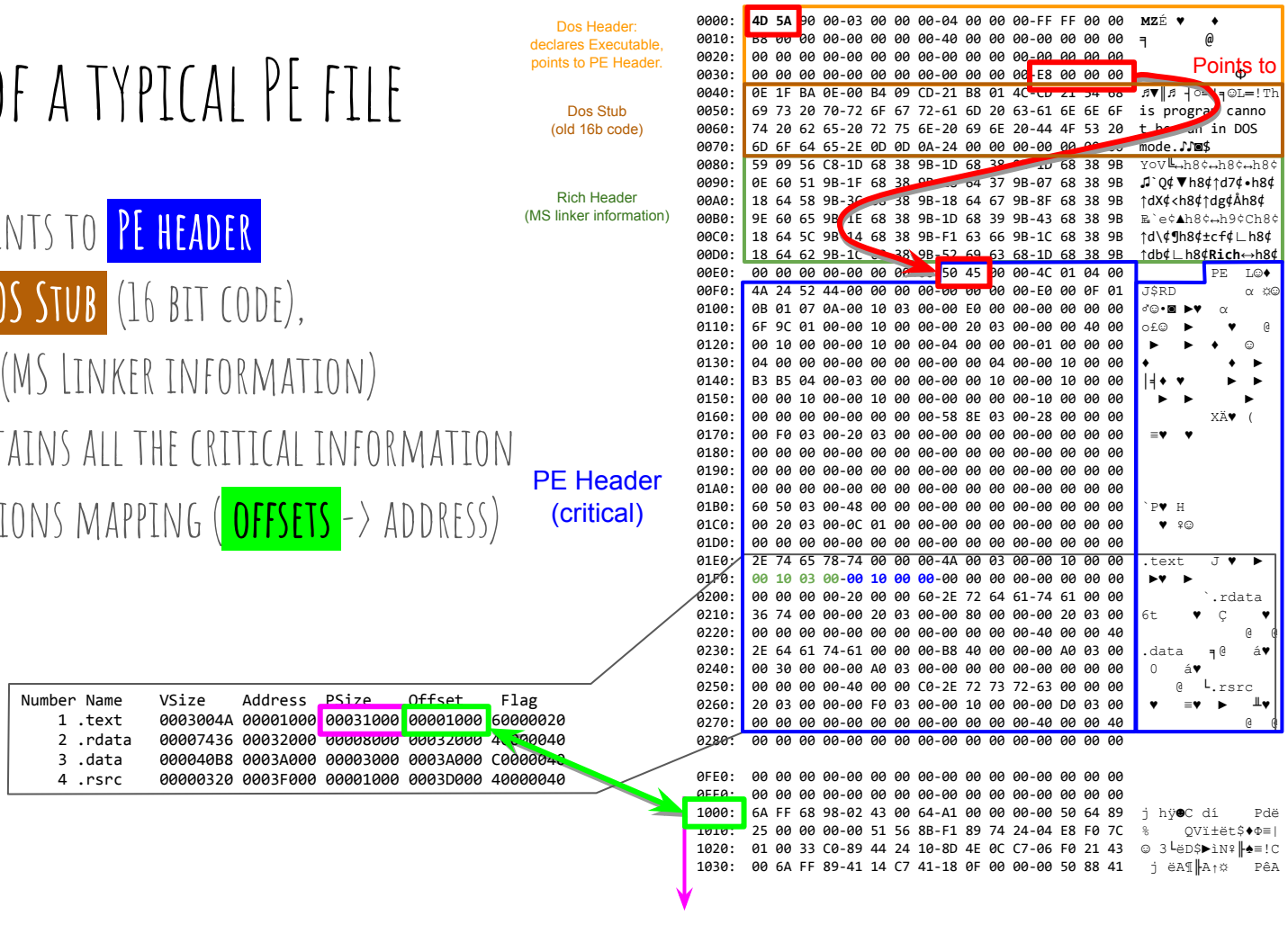


PE COLLISIONS VIA A CPC USED LIKE AN IPC



Anatomy of a typical 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 including sections mapping (offsets -> address)



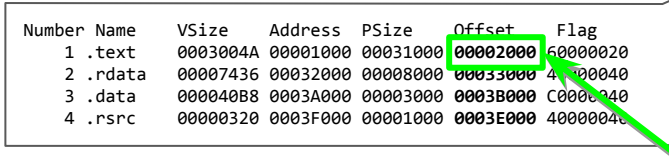
Abusing PE files

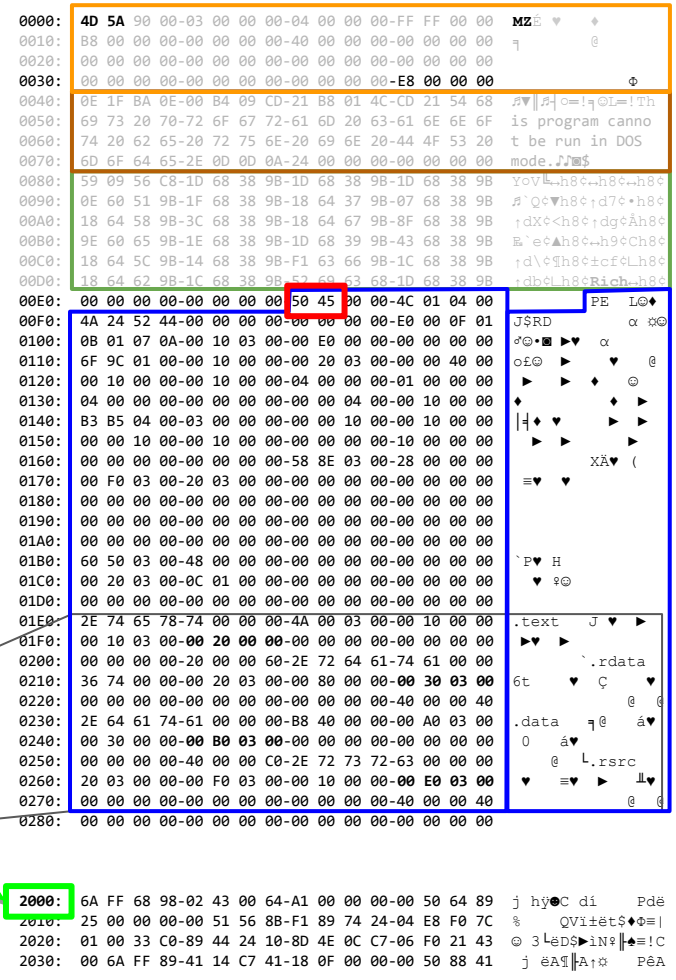
- **DOS header** only contains 2 important fields, the rest is irrelevant.
- **Dos Stub** and **Rich header** can be removed.
- **PE header** can be moved further: just update its **pointer**.
- Sections can be moved further: just adjust **offsets**.

only Magic and pointers are important

can be removed

can be removed

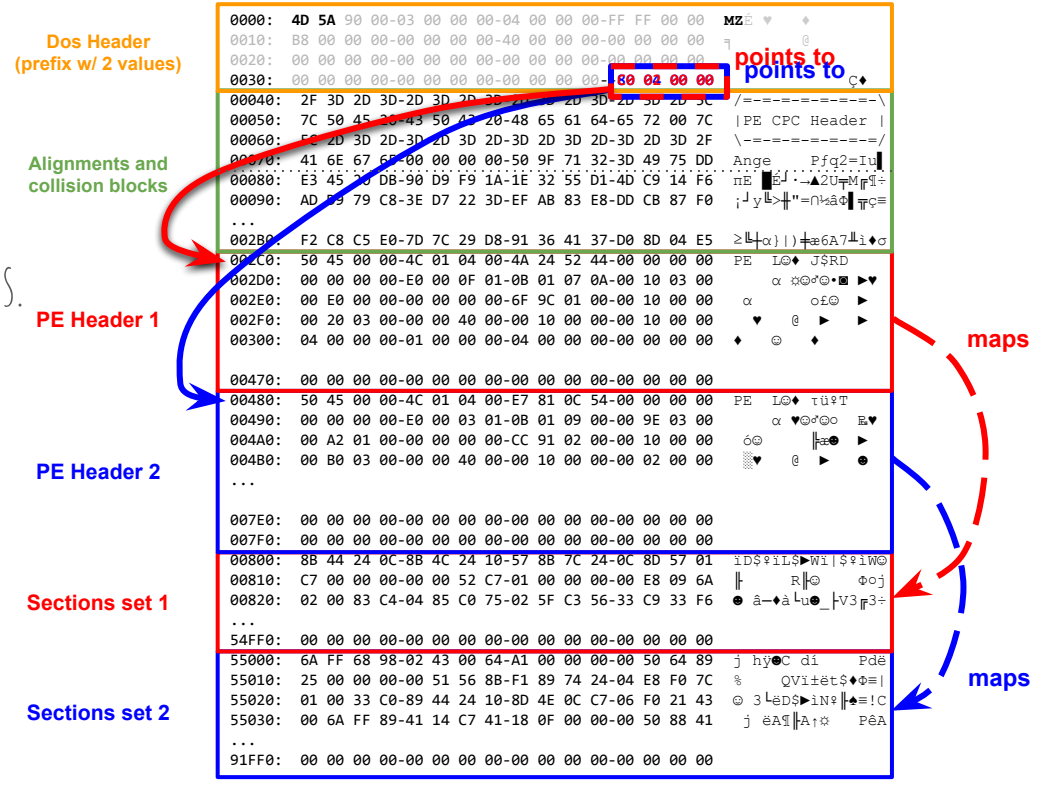




Windows PE collisions

- 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.

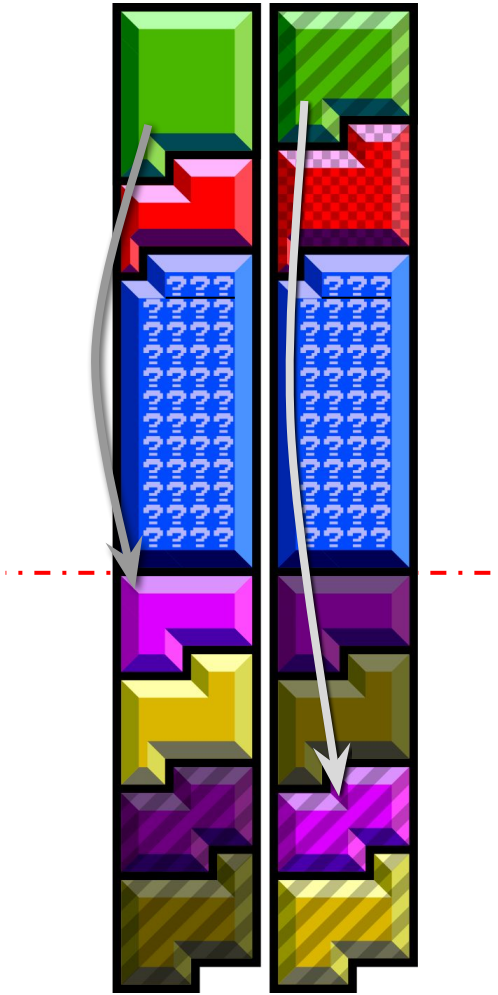
-> Reusable and instant PE collision



Recap: CPC-IPC exploitation for PE files

1. Craft 2 DOS headers with enough difference:
   - 1 block of alignment, 9 blocks of collision + 1 PE header
2. Compute CPC (a few hours)
3. Copy PE headers. Append sections.
   Adjust Sections offsets.

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



Chaining collisions

Chaining collisions

A collision makes two different contents have the same hash.

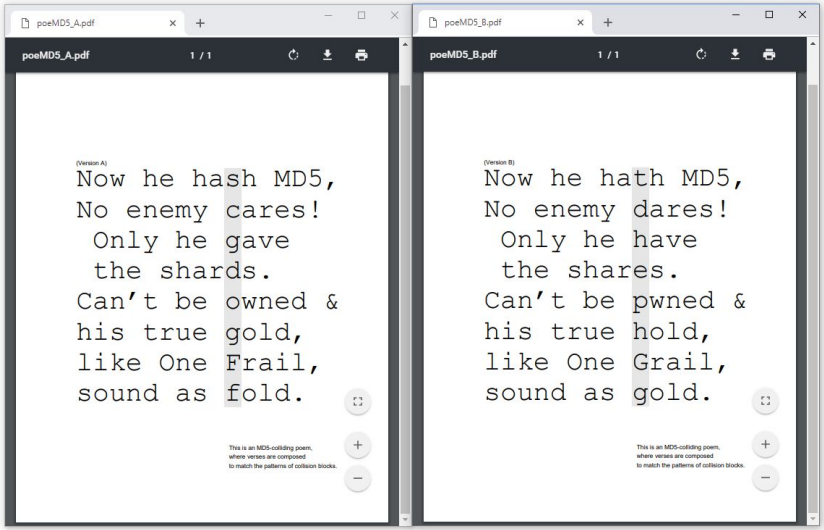
-> they can be chained like a tree.

Top nodes can be an IPC, others CPCs or UniColl

-> colliding more than 2 files

N collisions makes **N+1** contents collide

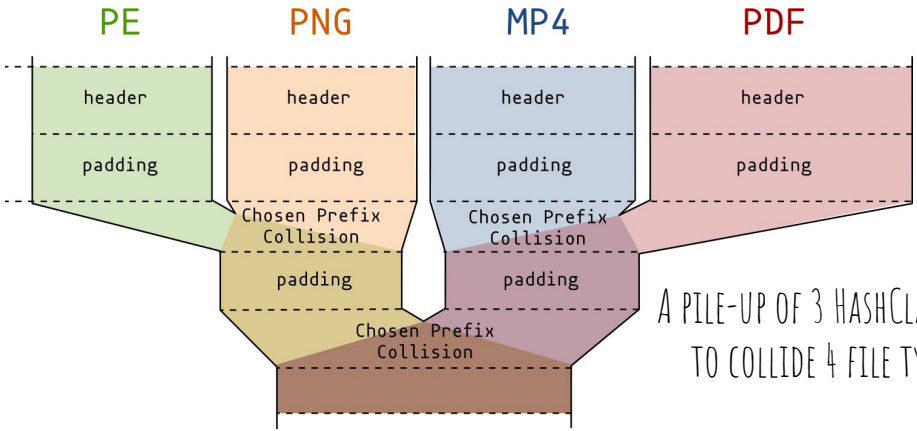
Collisions trees



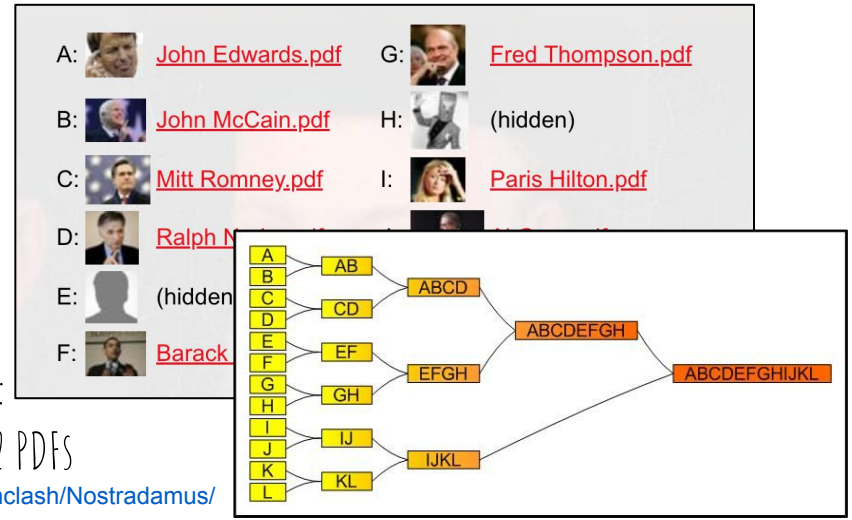
PoeMD5:

8 UniColls displayed on the page.

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



A pile-up of 3 HashClashes to collide 4 file types.



Nostradamus (2007):

11 HashClashes for 12 PDFs

https://www.win.tue.nl/hashclash/Nostradamus/

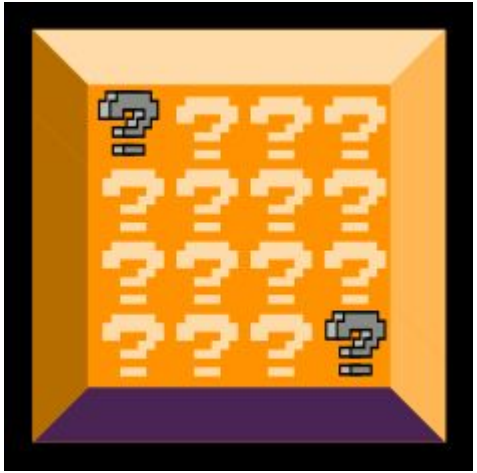
Shattered

a SHA-1 IPC

Shattered

- an IPC for Sha1
- Computed only once (?)
- Differences at start and end
-> "easy" to exploit

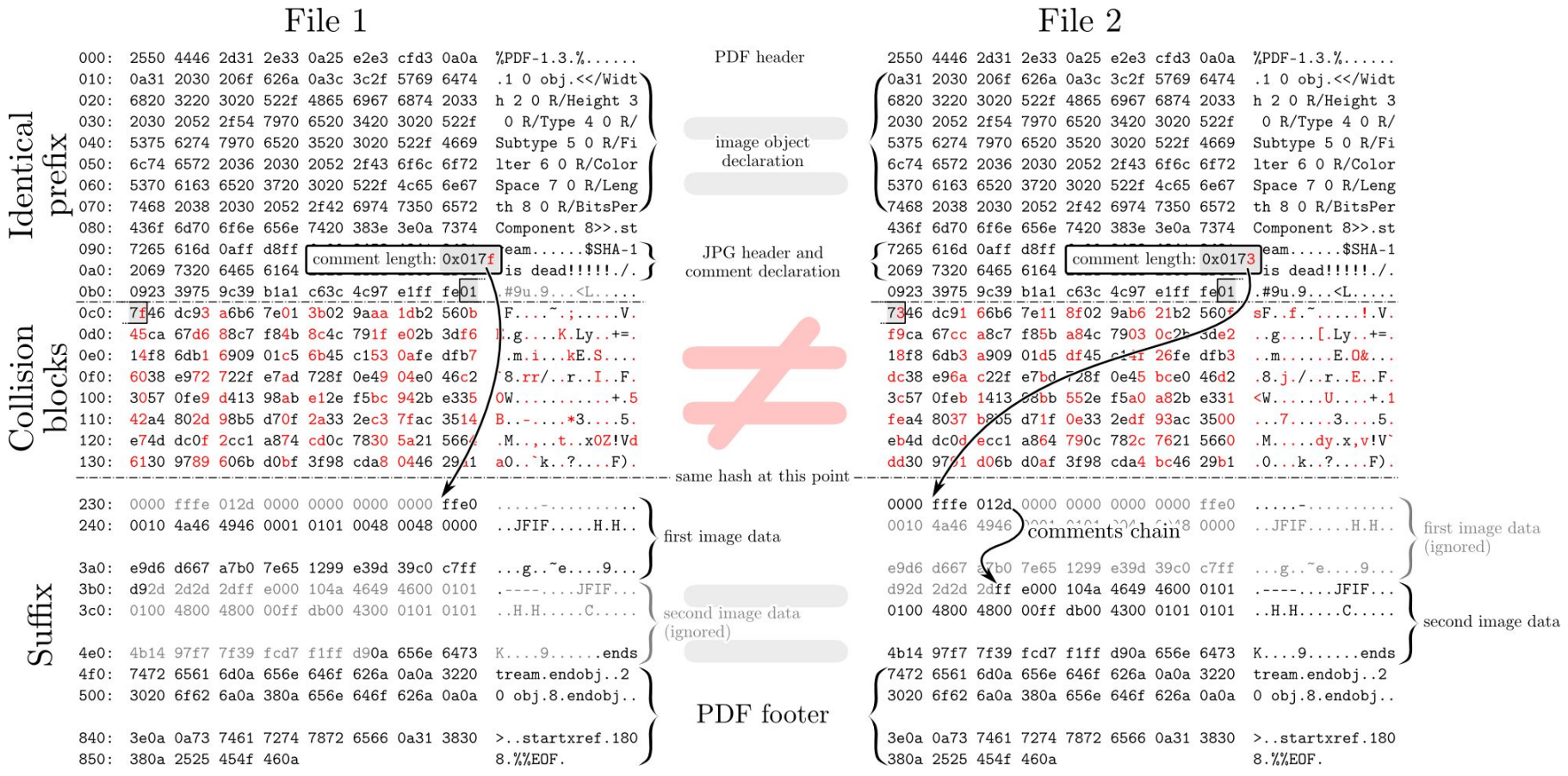
Official PoCs == JPGs in PDFs
(PDFs embed JPGs natively)



2 blocks

6K years

Start & end



Shattered files layout

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)

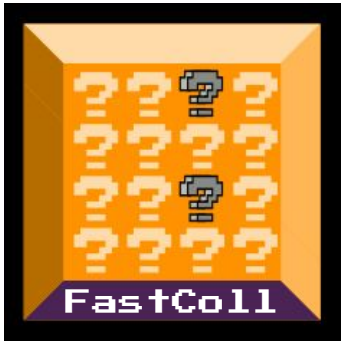
then abuse length with collision difference.

*with 64b lengths

For more details about Shattered exploitation:

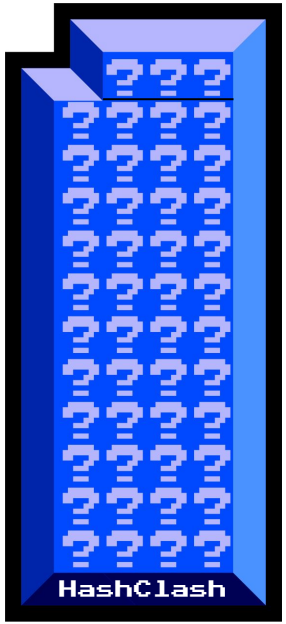
https://speakerdeck.com/ange/exploiting-hash-collisions (2017)


WRAP UP



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

Common points of all these attacks

They all appends random-looking blocks with tiny differences.

No, there's no other kind of attacks!

nothing like:

- ASCII-only
- incomplete blocks
- modify in the middle

Only 3 attacks?

In 2008, a CPC Attack used to create a rogue SSL certificate.

(200 PS3, signing at an exact second, 4 attempts, 2 days of computation each

-> MD5 was considered dead for good.

-> no incentive for any further research.

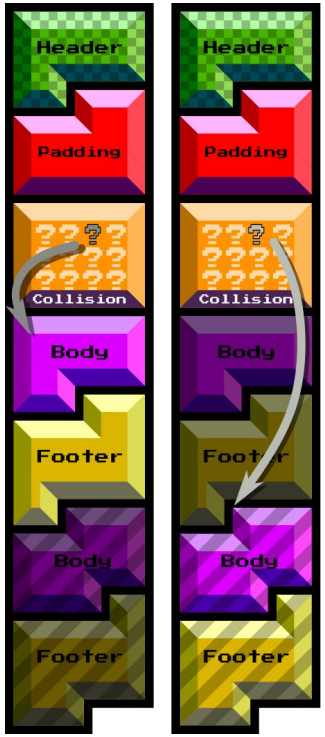
https://medium.com/@sleevi_/a-history-of-hard-choices-c1e1cc9bb089

Exploitations patterns

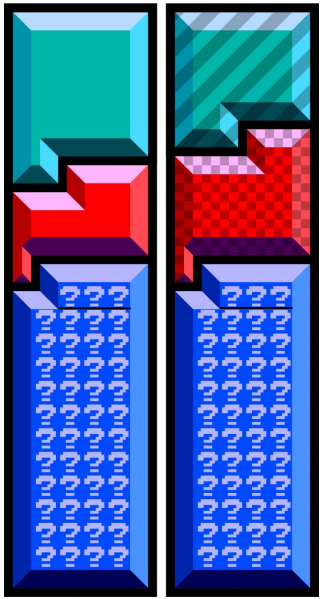
File
(prefix)
Comment
(padding)

Header
Body
(chunks)
Footer

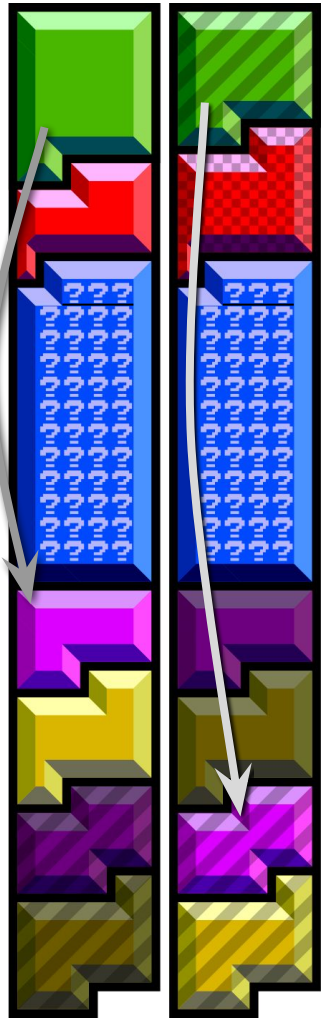
Identical Prefix
Chosen Prefix



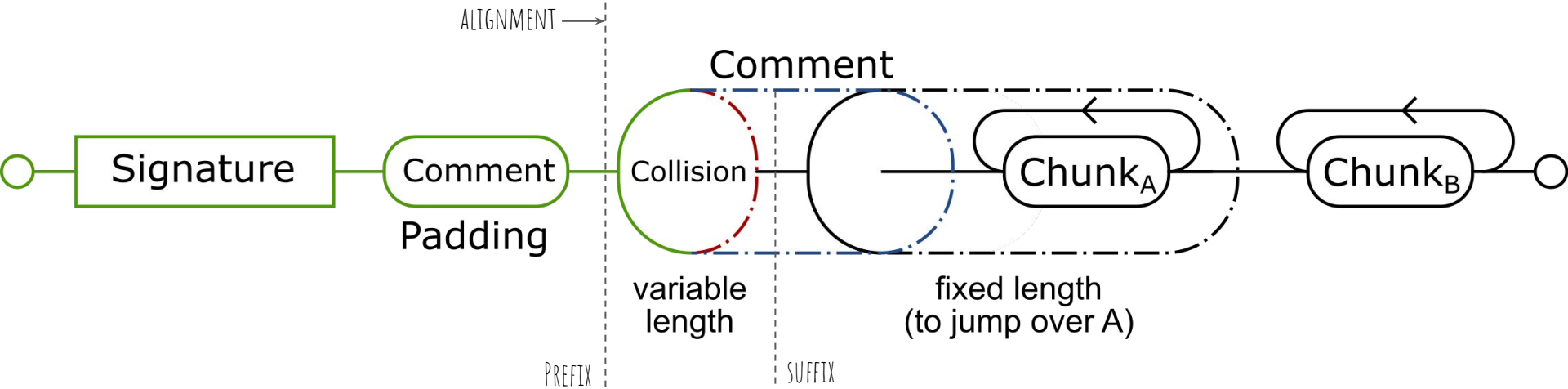
Reusable IPC



CPC



Reusable CPC



LAYOUT OF A RE-USABLE COLLISION EXPLOIT

MD5 IS

~~a cryptographic hash~~

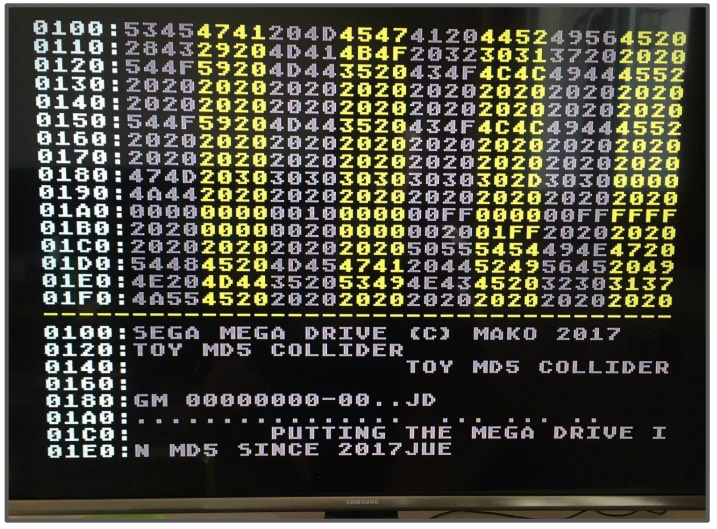
a toy function

...have fun!

Mako's "Toy MD5 Collider" for the Mega Drive dd49d7eb...

It takes 2 hours

1988: Sega Mega Drive/Genesis - 1992: MD5



EXTRAS

Other UniColl-based exploits for other formats

For more, see https://github.com/corkami/collisions

PDF

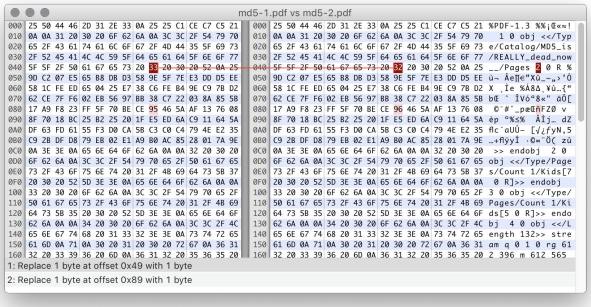
Merge both documents, split **/Kids** in 2 part showing pages sets separately.

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

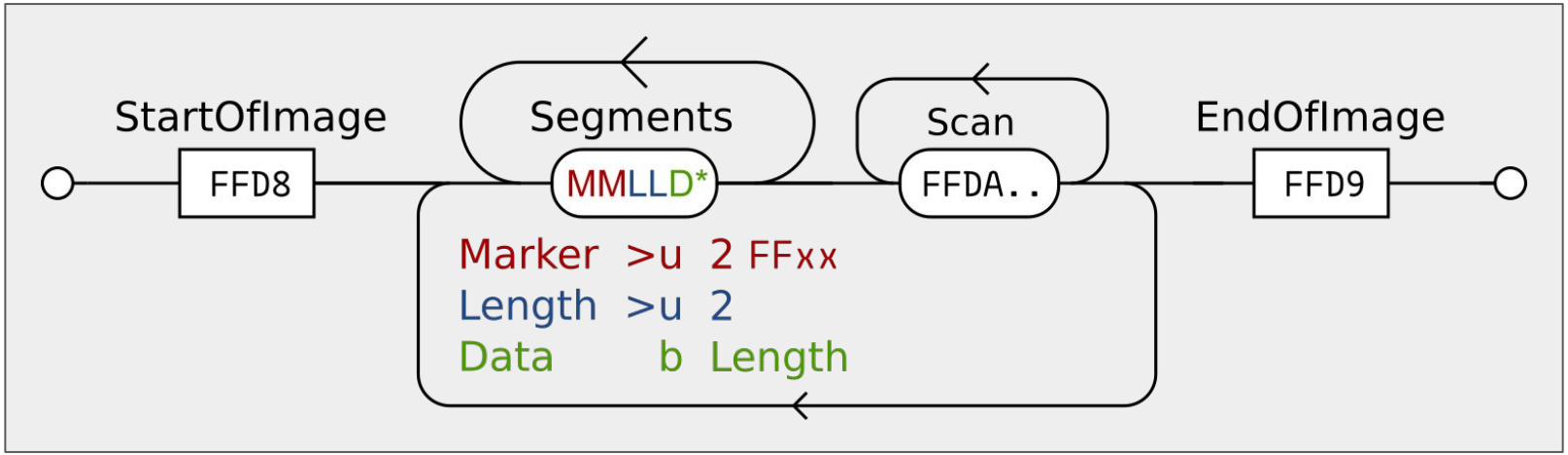
```
0040: .. .. ./ .P .a .g .e .s .  .2 .  .0 . .R \n .%
```

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



Comment segment in JPG: `FF FE`

Scans can be bigger than 64 kB -> split them via saving as progressive

JPEG file structure

JPG

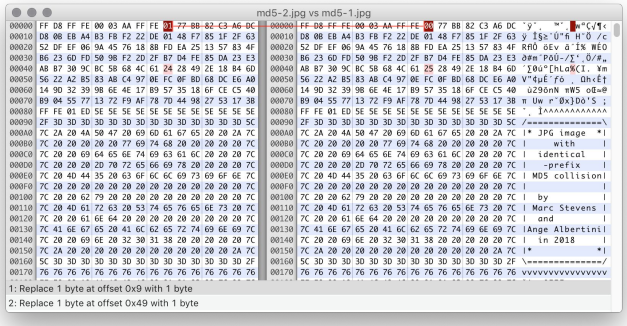
In JPG specifications, chunks are called segments

Use a **FF FE** COMment segments 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 segment of **0x177**.

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



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

MP4 (+JP2, HEIF...)

In MP4 specifications, chunks are called atom/boxes

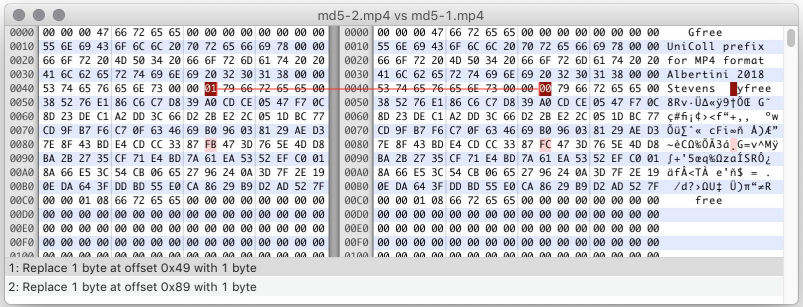
Use **FREE** atoms, for ALIGNEMENT then of length **0x79**

```
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 ..
```

Adjusting absolute (!) offsets in sample tables is required.



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

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		✗				✓

EXTRA

Exploiting FastColl

It **seems** hard, but not necessarily impossible.

It's a matter of finding the right file format.
(AKA aligning planets)

Recap on FastColl

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

⇤ ⇥	Padding, for alignments
# & % ! @	collision blocks' randomness needs to be ignored
… ? …	Differences need to be taken into account
…  ?	Two contents need to coexist

```
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

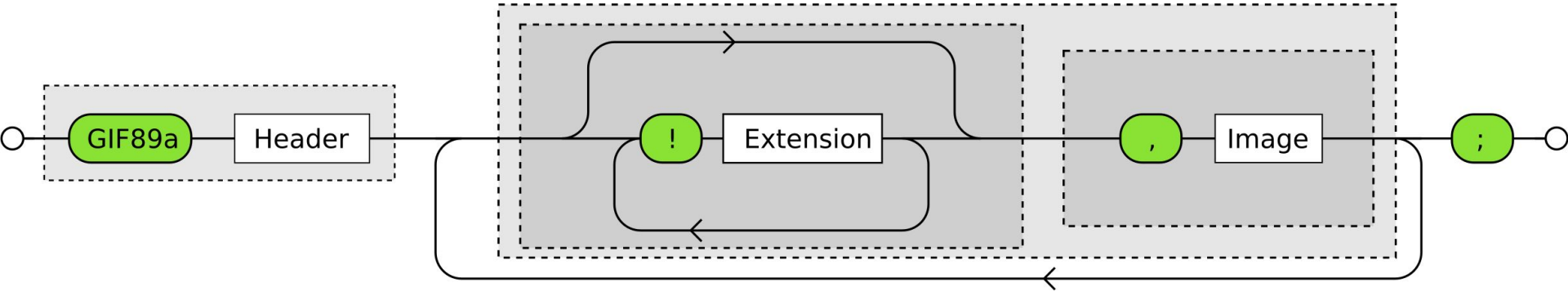
/dʒɪf/ JIF
/gɪf/ GHIF

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

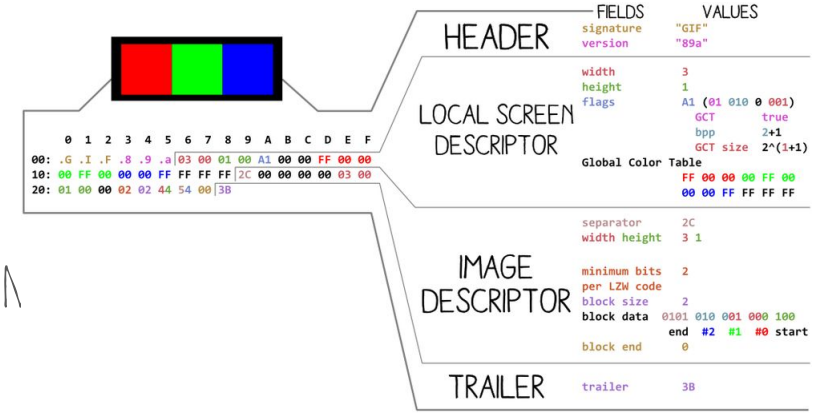

```
5c827c0eba9cfaa647c1a489bea77c60 *collision1.gif
5c827c0eba9cfaa647c1a489bea77c60 *collision2.gif
```

Specifications from 1989

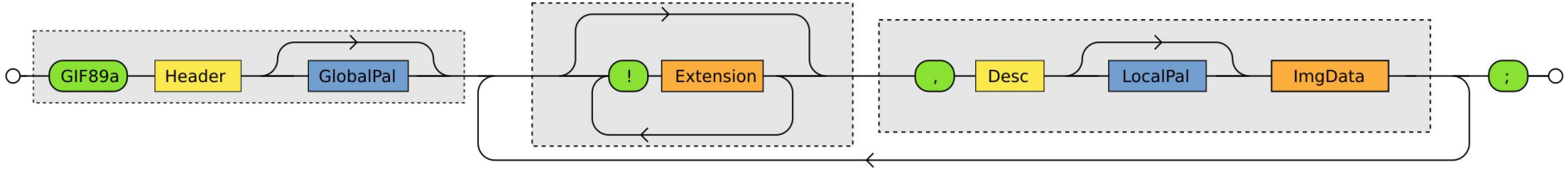
Overview of the Graphics Interchange Format



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

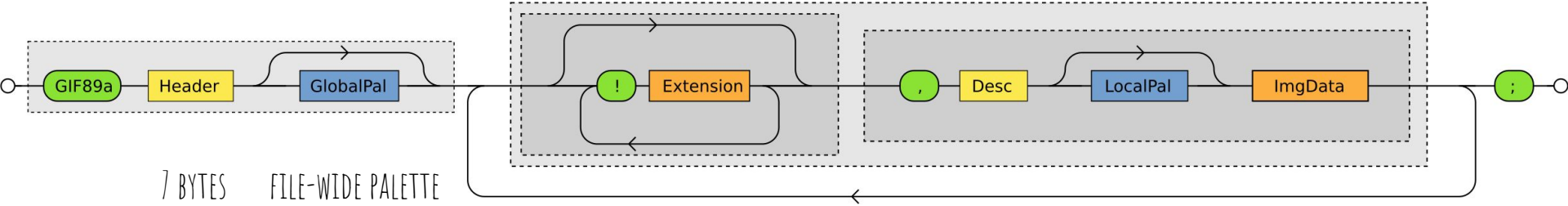


More details



- 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

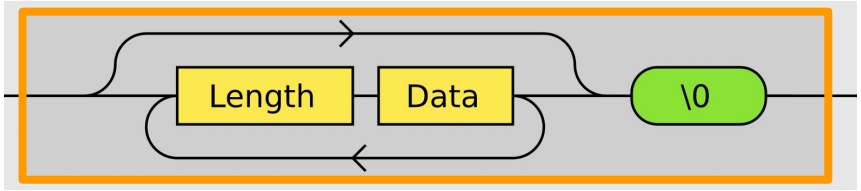
Image-wide palette
(optional)
3-768 bytes

Fixed size

SubBlocks-based

Variable-sized
(3-768 bytes)

GIF SubBlocks structures



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

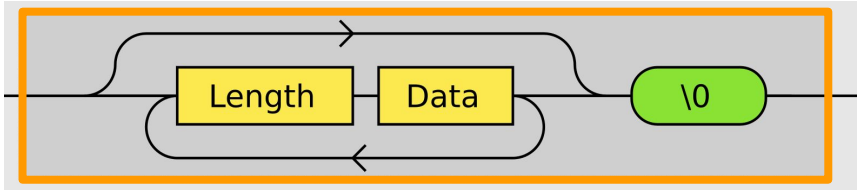
Cut in chunks of 255 bytes max, starting with their length, until **00**:

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



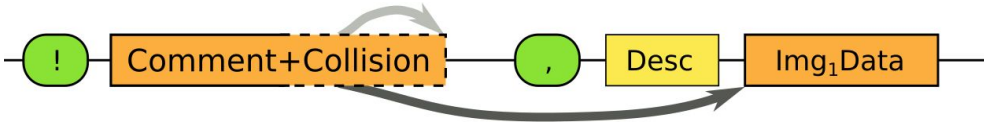
- can't jump over anything longer than 255 bytes.
  -> **very** restrictive
+ turns any non-null byte into a forward jump:
  good for FastColl

GIF data sled

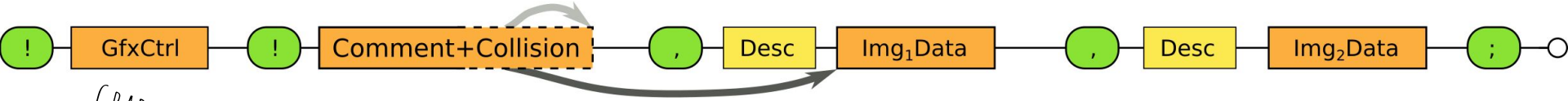
Subblocks are common to image data and extensions (like comments):

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

Reliable w/ minor overhead.



Common schema



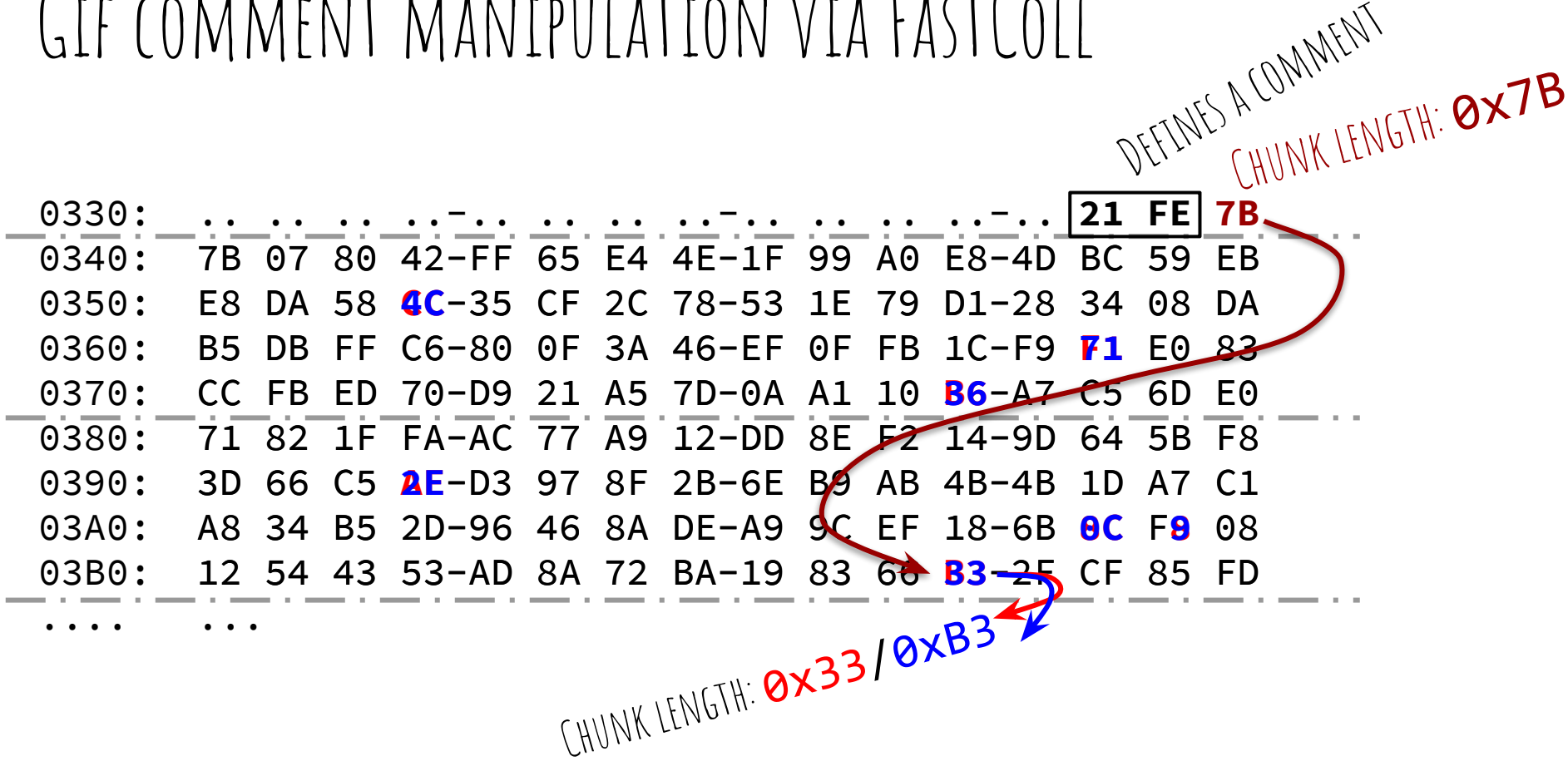
Graphical control extension:
Max delay between images

Long comment

Short comment

10 minutes delay

GIF comment manipulation via FastColl

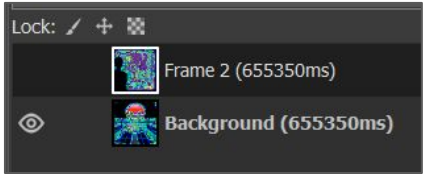


GIF

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

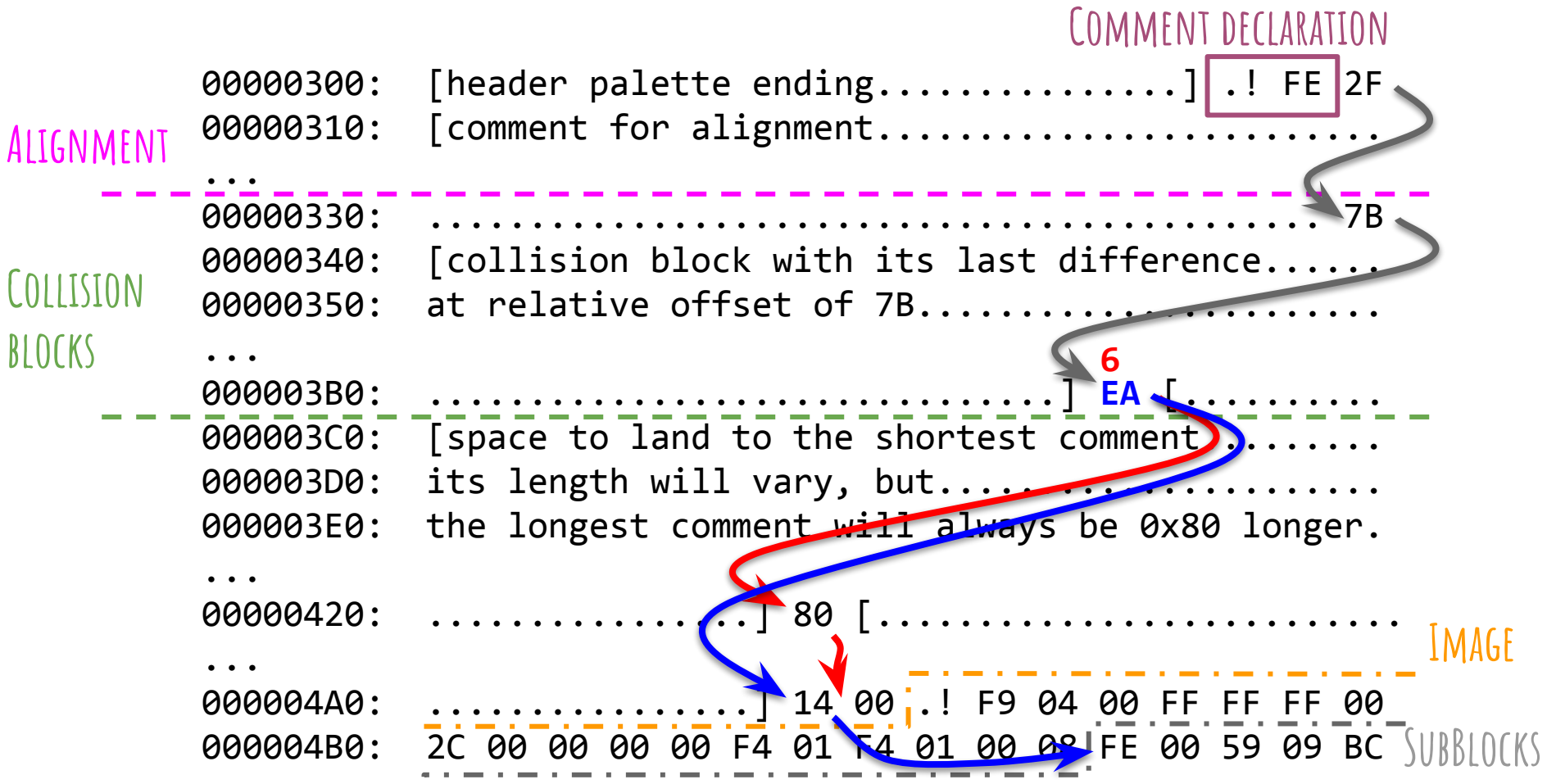
-> normalize palette

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 of `0x7B` (points to the 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>`

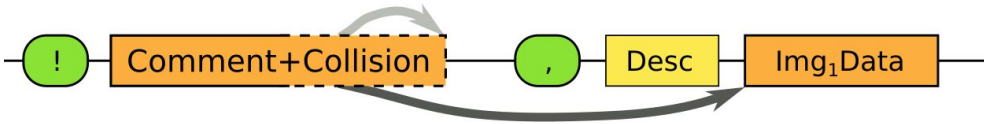
Actual example



CF working script for reference

Recap on GIF exploitation via FastColl

+ Comments are made of SubBlocks: declaration are separated from lengths.
  -> compatible with FastColl (!)
- lengths are stored on a single byte -> can't skip more than 255 bytes
+ image data is stored in SubBlocks too:
  -> extend comment to image data (!)



+ use max delay (10 min) to the 1st frame to hide the 2nd one.

Certificate (hard)

Collision exploit

Instant GIF collision via FastColl

Ange Albertini

INSTRUCTOR

FAQ

(FGA)

What did SHA-1 break?

- Git doesn't use SHA-1 on files, but on objects.

  https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

- SubVersion servers were unexpectedly broken.
- Bittorrent indexes blocks with SHA-1.

-> Marc created a SHA-1 collision detection library

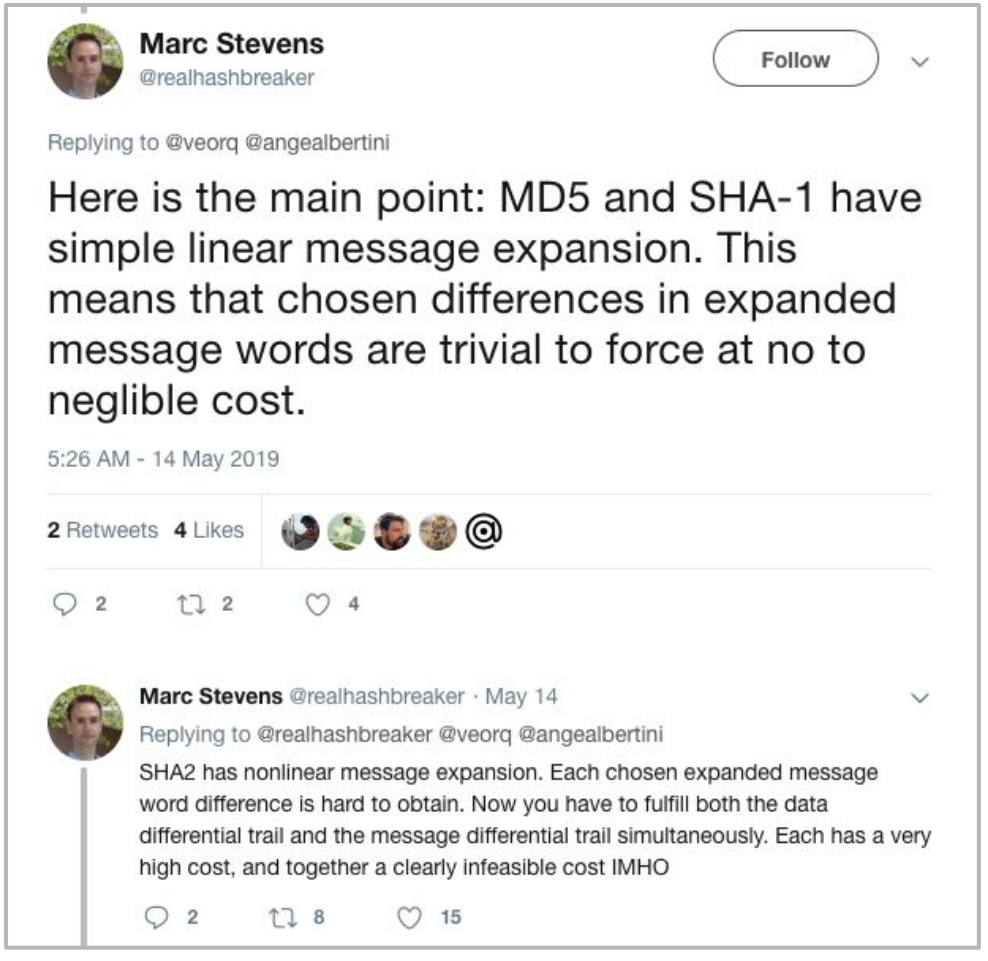
https://github.com/cr-marcstevens/sha1collisiondetection

What about SHA-2?

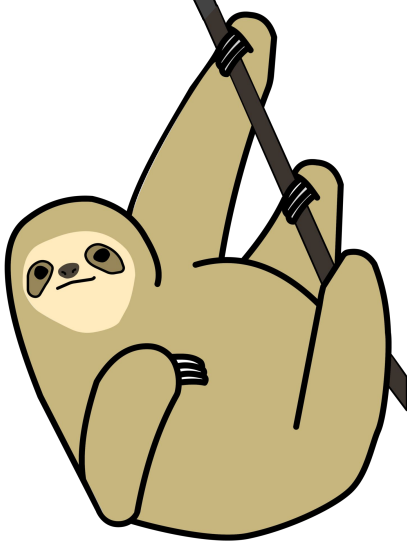
May 2019:

- MD5/SHA-1: "trivial".
- SHA-2: "clearly infeasible".

https://twitter.com/realhashbreaker/status/1128275424574832640



Only files?



Protocols can also be attacked:

SLOT - CVE-2015-7575

Security Losses from Obsolete and Truncated Transcript Hashes

https://www.mitls.org/pages/attacks/SLOTH

Paper: Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH

References

IPC EXPLOITS PAPERS

2004: Dan Kaminsky: MD5 To Be Considered Harmful Someday
https://eprint.iacr.org/2004/357.pdf
https://dankaminsky.com/2004/12/06/46/

2004: Ondrej Mikle: Practical Attacks on Digital Signatures Using MD5 Message Digest
https://eprint.iacr.org/2004/356.pdf

- 2005
  Max Gebhardt, Georg Illies, Werner Schindler
  A Note on the Practical Value of Single Hash Collisions for Special File Formats

  Slides a6cb4934...
  Paper ac7a05b4...

- 2014 MalSHA1
  Malicious Hashing: Eve's Variant of SHA-1
  Ange Albertini, Jean-Philippe Aumasson, **Maria Eichlseder**, Florian Mendel, Martin Schläffer

  Jean-Philippe's Slides aba7833e...
  Paper 5c763e29...

- 2017 Shattered
  The first collision for full SHA-1
  Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov

  Paper a3396362...
  Marc's Crypto17 video
  Elie's BlackHat slides video 1a17c315...
  Pierre's RWC video slides 08e83064...

file format collaborator instigator

New in 2019

(Not implemented yet AFAIK)

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)

https://eprint.iacr.org/2019/459

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

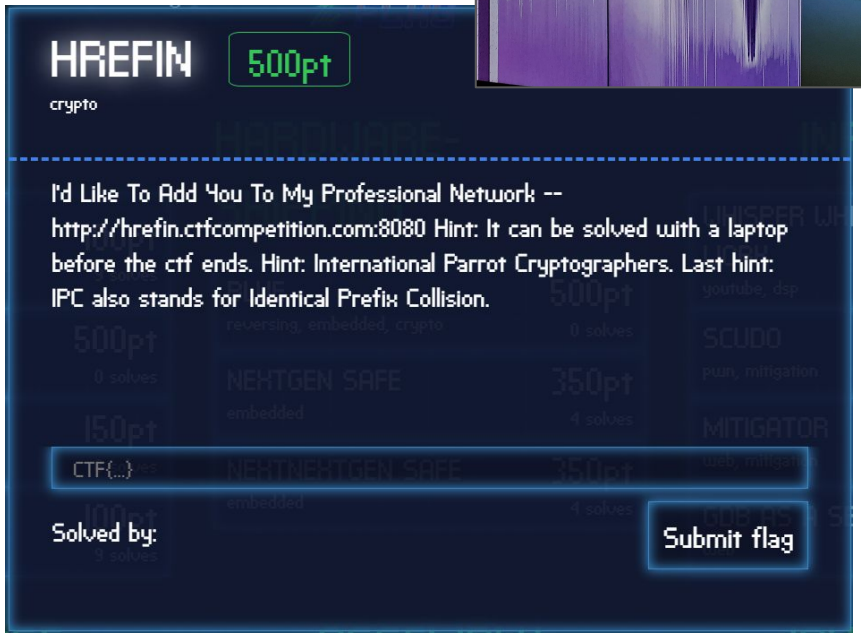
- 2^{39}: 9 blocks 2009 Stevens (HashClash) Short chosen-prefix collisions for MD5
- 2^{53}: 1 block … …
- 2^{46}: 2 blocks **2019 Leurent** **From Collisions to Chosen-Prefix Collisions**

Hash collisions in CTF

HrefIn

MD5 collision on certificates by Enzo.

500 pts, 0 solves :(



Looking Glass

MD5 collision on ProtoBuf by mlen.

330 pts, 11 solves.

Looking glass

Web / Cryptography, 330

Difficulty: medium (18 solvers)

We found some shady Looking Glass service. Pretty sure there is a way to get that delicious /flag.

http://lg.hackable.software:8080/

- src.tgz

Acknowledgments

They made this workshop possible:

Barbie Auglend, Christophe Brocas, Philippe Teuwen.

They made it better:

Jean-Philippe Aumasson, Nicolas Grégoire

Enzo Puig, Mateusz Lenik and Enrico Bacis,

and my audiences.

Thank you for making it this far!

Any feedback is welcome!

@angealbertini or ✉ ange@corkami.com

ANGE ALBERTINI
reverse engineering
VISUAL DOCUMENTATIONS
@angealbertini
ange@corkami.com
http://www.corkami.com