HASH COLLISION EXPLOITATION

WITH FILES

A workshop by

**Ange Albertini**

WITH THE HELP OF

**Marc Stevens**

A.K.A.

**Let's Play**

**Colltris**
Welcome

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

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

The slides are public and will be improved whenever needed.

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

with remarks, one-liners, suggestions...

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

No gatekeeping, no dogma, no cult.
What’s a hash function? **MD5, SHA1...**

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

\[
\begin{align*}
\text{a} & \rightarrow \text{d41d8cd98f00b204e9800998ecf8427e} \\
\text{b} & \rightarrow \text{0cc175b9c0f1b6a831c399e269772661} \\
\text{A} & \rightarrow \text{92eb5ffe6ae2fec3ad71c777531578f} \\
\text{?} & \leftarrow \text{d41d8cd98f00b204e9800998ecf8427f} \\
\text{?} & \leftarrow \text{d41d8cd98f00b204e9800998ecf8427d}
\end{align*}
\]

Impossible to guess a content from its hash value.
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
  password → 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!>','%$", salt="br")
'brokenOz4KxMc'
>>> crypt.crypt("O!>','%$", "br") == crypt.crypt("5dUD&66", "br")
True
>>>
What’s the extent of a hash collision?

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

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

With some file types, we can **instantly** generate files that render the same way (via some tricks).
Results - 1/2

Instant MD5 collisions, with no recomputation

https://github.com/corkami/collisions

*SOME LIMITATIONS
 RESULTS - 2/2

*some limitations
Just new collisions?

Instant, reusable and generic collisions:

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

Extreme case: the colliding PDFs are 100% standard.

From a parser perspective,

the contents are unmodified: only the files’ structures are.
Taking collisions to extremes:
INSTANT & GENERIC PDF/PE/PNG/MP4 COLLISION
Don’t be fooled: shortcuts are necessary

Instant & generic collisions rely on attacks and file formats tricks. Some formats have no suitable tricks.

\[
\text{-} \rightarrow \text{no generic collisions for ELF, Mach-O, ZIP, TAR, Class.}
\]

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

the same JPEG trick was re-used with 3 hash collisions (MD5, MalSHA1, SHA1)
GOALS
Goals of this workshop

- **Understand hash collisions attacks and their exploits**
- **Create your own exploits**
Maybe you’ve heard of...

The Shattered Attack

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

What this slide deck is about
Uses of hashes

- Check if content has changed: ✓ Do nothing ✗ Refresh file [if newer...]
- Provide randomization: ✓ User ID ✗ Crypto key
- Match a file to a file/set (white/blacklisting, indexing)
  ✗ If the set is user-controlled
Use cases

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

Collide a normal* file with a malicious one. You can even do it on the fly!

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

*rendering-wise, not structure-wise.
Having troubles to convince?

Let files do the talking.

Threats? theory...

Exploits PoCs? reality!
MD5 COLLISIONS: A GOOD EXERCISE

Hacking a file format == reading + manipulating + abusing parsers
Crafting a re-usable collision requires all these skills, and leaves an undeniable proof.

A re-usable MD5 collision is a good & impactful exercise:
If the collision is instant, the files work and have the same MD5, it sets in stone your knowledge of that file format, and you have a proof (of concept).
Don't be scared...

\[ \Delta Q_j[i] = \hat{\Delta} Q_j[i] - \Delta Q_j[i] \text{ for } i \in I \text{ and } i = 0, \ldots, N - 1. \]

\[ \delta W_t = \hat{\delta} W_t \]

\[ \Delta F_t[i] = \hat{\Delta} F_t[i] \]

and \[ \hat{F}_t = j_{\text{bool}, t}(Q_{t-L+2}, \ldots, Q_t); \]

\[ \delta Y_{j,i} = RL(\hat{Q}_{t-L+i}^t, r_j, i) - 1, \ldots, V \text{ and } i = 1, \ldots, L; \]

\[ \delta Y_{j,L+1} = RL(\hat{F}_t, r_j, L+1) - 1, \ldots, V; \]

\[ \delta Y_{j,L+2} = RL(\hat{W}_t, r_j, L+2) - 1, \ldots, V; \]

\[ \delta Y_{j,L+3} = RL(\hat{T}_{t,j-1}^t, r_j, L+3) - 1, \ldots, V, \text{ where } \hat{T}_{t,i} \text{ and } \hat{T}'_t \text{ for } i = 0, \ldots, V \text{ are computed as in Section 5.3.4}; \]

TALES FROM THE CRYPTO
You don’t need to understand cryptography or maths...
(to be honest, I don’t either)

We’ll just use existing attacks:

FastColl, UniColl, HashClash, Shattered - yes, that’s all!
You don't have to be an expert in file formats.
You just need to know their overall structure.

(less complex than some lego models)
Even this is too much!
You only need to understand the high level structure (not the whole thing)
And we'll ignore most contents, so we'll just think in blocks.
To be honest

exploiting hash collisions feels a bit like...
...playing a puzzle game!

Colltris

Ange Albertini
You just need to know the rules of each block!
You know hexadecimal?
You know ASCII?

A standard encoding:
Characters $\leftrightarrow$ values

"A" $\leftrightarrow \text{0x41} = 65$
"Z" $\leftrightarrow \text{0x5A} = 90$
"a" $\leftrightarrow \text{0x61} = 97$
You know endianness?

> BIG DIGITS FIRST

01 23 → 0x0123 = 291

< LITTLE DIGITS FIRST

01 23 → 0x2301 = 8961

Formats

PNG, JPG, MP4, Class

TIFF

Exists in both endianness

PNG, JPG, MP4, Class

TIFF

ZIP, BMP, GZIP

Executables: ARM (default), x86, x64
You know hexadecimal viewers?

```
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
```

Start in the top-left corner +0x10 +1

Offsets

Contents

Kaitai
Okteta
Hex Fiend
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
1. **HashClash installed**
   
   [https://github.com/cr-marcstevens/hashclash](https://github.com/cr-marcstevens/hashclash)
   
   **CUDA is not required**

2. **A file format manipulation environment**
   
   Hex editor, assembly, scripting...
   
   Whatever rocks your boat and you’re familiar with.

3. **A copy of Corkami/collisions**
   
   *(Recommended)*

   [https://github.com/corkami/collisions](https://github.com/corkami/collisions)
OUR FIRST COLLISION
The first block in our game: An Identical Prefix Collision - FastColl
$ fastcoll -p empty
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'empty'
Using initial value: 0123456789abcdeffedefc9a876543210

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

$ _

From nothing, generate 2 files with the same MD5.
Our first hash collision

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

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'empty'
Using initial value: 0123456789abcdefedcba9876543210

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

$ _

Try again in the same conditions -> different computation time.
Our second colliding pair

- Completely different
- Still random-looking
→ Let’s ignore the ASCII!
- Differences at the same offsets
(That’s how it works)
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.

For more details, check [https://www.youtube.com/watch?v=iKE7DJd-PwU](https://www.youtube.com/watch?v=iKE7DJd-PwU)
Now let's add an input - our prefix.

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

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

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

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

Now let's add an input - our prefix.
- Padded to 64 bytes
- Collision blocks appended
- Differences at the same relative offsets

Similar blocks - added after padding to 64 bytes
MD5, SHA1 work by processing 64 bytes block, from start to end.
- Hash collisions always work with such alignment.
- Appending the same thing to two files with the same hash will give files with the same hash.
What can we do with this?

We can put whatever we want **before** and **after** the collision.

We need the following from the target file format:

- **Padding**, for alignments
- **Collision blocks' randomness** needs to be ignored
- **Differences** needs to be taken into account
- **Appended data** needs to be ignored
Certificate
Hash collision
Computed your first FastColl

_______   _______  

Ange Albertini
INSTRUCTOR
Hash collision blocks

- Start and end aligned to 64 bytes (via padding if needed)
- Totally random
- Tiny differences at fixed offsets

These properties are common to all the attacks on MD5 or SHA1.
An Identical Prefix hash Collision

- Takes a single input
- Prefix and suffix will be identical:
  - Files almost identical
  - Exploitation depends only on collision differences

These properties are common to FastColl, Unicoll and Shattered.
FastColl

→ HARD TO EXPLOIT!

The fastest but also the most limiting.

TWO BLOCKS
A FEW SECONDS
IN THE MIDDLE
(AWAY FROM START OR END)
What makes exploiting Fastcoll so difficult?

Every collision differences is surrounded by random data.

It’s hard to declare a structure **AND** its length in a single byte such as a variable-length comment that we need for IPC exploitation.

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

Adding some bruteforcing in the computation is an approach, but it’s a bit slow.
Bruteforcing is a solution

Extra constraint can be added manually inside FastColl source.

Cf PoCorGTFO 14:11

Thankfully there is UniColl.

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

```cpp
// change q17 until conditions are met on q18, q19 and q20
unsigned counter = 0;
while (counter < (1 << 7)) {
    const uint32 q16 = Q[Qoff + 16];
    uint32 q17 = ((xrng64() & 0x3ffd7ff7) | (q16&0xc0008008)) ^ 0x40000000;
    ++counter;
    uint32 q18 = GG(q17, q16, Q[Qoff + 15]) + tt18;
    q18 = RL(q18, 9); q18 += q17;
    if (0x00020000 != ((q18^q17)&0xa0020000)) continue;
    uint32 q19 = GG(q18, q17, q16) + tt19;
    q19 = RL(q19, 14); q19 += q18;
    if (0x80000000 != (q19 & 0x80020000)) continue;
    uint32 q20 = GG(q19, q18, q17) + tt20;
    q20 = RL(q20, 20); q20 += q19;
    if (0x00040000 != ((q20^q19) & 0x80040000)) continue;
    block[1] = q17-q16; block[1] = RR(block[1], 5); block[1] -= tt17;
    uint32 q2 = block[1] + tt1; q2 = RL(q2, 12); q2 += Q[Qoff + 1];
    block[5] = tt5 - q2;
    Q[Qoff + 2] = q2;
    Q[Qoff + 17] = q17;
    Q[Qoff + 18] = q18;
    Q[Qoff + 19] = q19;
    Q[Qoff + 20] = q20;
    MD5_REVERSE_STEP(2, 0x242070db, 17);
    counter = 0;
    break;
}
```
Instant computation doesn’t give any instant exploitation.

\[\Rightarrow\] Instant collision relies on pre-computed prefix.
The general structure of file formats

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

**Body**: comes after. Made of several sub-elements.

**Footer**: follows the body. Indicates that the file is complete. Any data is usually ignored after.
The "comment" block

Most formats 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**.
These files are equivalent (from a parser perspective). Same content, different structure.

Change from one to the other is usually (very) easy.
Take two files...
(of the same file type)
Create a super file combining both files’ data

Both files’ Body and Footer are kept original.
The header has to be a common ground.
Find a way to make the collision work with the file format.
NEXT LEVEL
Now let's look at something else.
Poetry...?
Now we hash md5,
no enemy cares!
Only we gave
the shards.

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

...
Our second block - another Identical Prefix Collision: UniColl.
$ ./poc_no.sh prefix
MD5 differential path toolbox
Copyright (C) 2009 Marc Stevens
http://homepages.cwi.nl/~stevens/

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

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

Mission

Run its script on a prefix

https://github.com/cr-marcstevens/hashclash/releases
Characteristics:
- Two blocks
- A few mins to compute

Important difference with FastColl:
- Prefix as part of the collision blocks!!
- Differences:
  10th char of prefix += 1
  10th char of 2nd block -= 1

Output of a UniColl computation
UniColl is an IPC with differences that you can easily predict

A true Unicorn of a collision
Rules of UniColl block prefix

- Length multiple of 4
- First block starting at last chunk of 64 bytes.
- The longer the prefix in the block, the longer the computation
- Maximum prefix in collision blocks:
  24 bytes - 9 minutes

$ cat prefix
Here is my long prefix!
$ time ./poc_no.sh prefix
MD5 differential path toolbox
[...]
Found collision!
6297bd824dc4a35ae83fe8e1bdceb9c2  collision1.bin
6297bd824dc4a35ae83fe8e1bdceb9c2  collision2.bin
[...]
user   9m12.321s
$
Sometimes, unicoll just... fail! Just retry then!

**Happy ending**

```
[...]  
262144 9  
370611 16  
524288 19  
Block 1: ./data/col1_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/col1_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  
Found collision!  
2b3663b299b72c6b40d13cc6c905a7d  collision1.bin  
2b3663b299b72c6b40d13cc6c905a7d  collision2.bin
```

**Bad endings**

```
[...]  
65536 4  
126153 8  
131072 8  
Block 1: ./data/col2_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/col1_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  
```

...
UniColl

Two blocks
A few minutes
In prefix

Slightly slower,
but easy to exploit.
Plan your exploit

Explore format specs for features you need.

Explore attacks: FastColl, UniColl [tree], HashClash

Plan your structure (pen & specs)

craft mockup files. Good to validate compatibility, CRCs…

Erase collision blocks to simulate colliding files.

Extract prefixes from mockups.

Run computations.
Plan your generic exploit

Explore format landscape, tools and options.

Some minor tool option might be a perfect fit.

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

Understand compatibility in depth.
What makes exploiting UniColl so easy?

The first difference is surrounded by chosen text:

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

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

I.E. one chunk will be exactly $0x100$ longer than the other,

which is bigger than the collision block

but doesn’t grow uncontrollably.
Exploiting our first hash collision attack
**Layout of a generic file format exploitation**

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

Case B (long comment)
Making it generic?

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

\(\rightarrow\) 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

Signature → Comment → Collision → ChunkA

Long collision comment

Signature → Comment → Collision → ChunkA
The **Portable Network Graphics format**

The most regular format:

A signature then a sequence of chunks.
The PNG format at chunk level

- Big endian length, on 4 bytes.
- A type, on 4 characters.

Critical chunks: **IHDR/PLTE/IDAT/IEND**

Lowercase starting types == ignored.

- CRCs are usually ignored
Overview of an actual PNG file

Signature

Header

Data

End

Fields

Values

signature

\x89 PNG
\r\n^Z \n
size

0x0000000D

IHDR

0x948283E3

id

CRC32

IDAT

0xE93261E5

size

0x00000015

CRC32

0xAE426082
import struct
import binascii

_Magic = r'\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'))

def make(chunks):
    s = [_MAGIC]
    for t, d in chunks:
        s += [
            struct.pack('>I', len(d)),
            t,
            d,
            struct.pack('>I', _crc32(t + d))
        ]
    return ''.join(s)

TRIVIAL TO PARSE OR MANIPULATE AT CHUNK LEVEL.
3 dummy chunks: alignment, collision and jump over (the first image) data
https://github.com/corkami/collisions/blob/master/scripts/png.py
Need to explore by yourself?

Open Kaitai IDE with the lightweight PoCs

Kaitai tricks

Only the high level structure is useful:

-> simpler grammar can be better.

Looser logic can be required:

Ex: IHDR chunk not in the first slot.

ICYDK You can directly edit the grammar in the IDE!

Simplified PNG grammar

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

All PNG viewers seem to ignore CRCs.
Most PNG viewers tolerate starting w/ a dummy chunk.
-\> generic collisions for any PNG pair
OS X (Safari, Preview) enforce an IHDR chunk first:
-\> dimensions and colorspace are in the common prefix

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

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

Collision exploit

Reusable PNG via UniColl

Ange Albertini
INSTRUCTOR
Back to exploiting FastColl...

It’s just a matter of getting a format to comply with the single byte differences.
The Graphics Interchange Format /dʒɪf/ JIF

- FastColl based exploit: instant computation
- Same dimensions and palettes, single frame.
- First image displayed for 10 minutes (each image is a different frame).
Overview of the Graphics Interchange Format

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

Header:
- Signature
- Version
- Width
- Height
- Flags
- Global Color Table

Local Screen Descriptor:
- Separator
- Block size
- Block data
- Block end

Image Descriptor:
- Trailer

GIF89a
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**

- **File-wide palette (optional)**
  - 3-768 bytes
  - Examples of extension:
    - Graphical Control
    - Defines delay between frames
    - Comment
    - Application
    - Defines looping

- **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.
Graphical Control Extension:
Max delay between images

Long comment

Short comment

10 minutes delay
GIF COMMENT MANIPULATION VIA FASTCOLL

Chunk length: 0x7B

Defines a comment

Chunk length: 0x33/0xB3
**GIF**

Special chunk structure -> single byte = “jump”

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

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

After 10 minutes, the images are identical.
Common headers?

Headers include palette, dimensions:

- Use 2 images of same dimensions
- Normalize palette

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

- Combine 2 frames in a single animation with a comment.
- Extend comment to align to 64 bytes with a jump to 0x7B (last difference in the collision blocks)
- Compute FastColl
- Append images suffix
- Adjust comments to:
  - Finish before first image: .! F9
  - Slide into first image data: 08 FE <high entropy>
00000300: [header palette ending..................] .! FE 2F
00000310: [comment for alignment............................
00000330: ............................................. 7B
00000340: [collision block with its last difference......
00000350: at relative offset of 7B.......................  
000003B0: .............................................
000003C0: [space to land to the shortest comment...........
000003D0: its length will vary, but........................
000003E0: the longest comment will always be 0x80 longer.

00000420: ................................80 [............................
000004A0: ................................14 00 .! F9 04 00 FF FF FF 00
000004B0: 2C 00 00 00 00 F4 01 F4 01 00 00 FE 00 59 09 BC

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

Collision exploit

Instant GIF collision via FastColl

Ange Albertini
INSTRUCTOR
The two Identical prefix collisions against MD5
IPCs limitations

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

Same prefix -> same file type

Same header -> same metadata

Enforced checksums prevent validity.

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

Chosen-Prefix Collisions
Our third block: a Chosen Prefix hash Collision
A WORD OF WARNING ON CPC

TAKES 72H.CORE HOURS TO COMPUTE - IF YOU’RE LUCKY.
REQUIRES SIMPLE MONITORING AND BACKTRACKING.
THE FEWER THE COLLISION BLOCKS,
THE LONGER TO COMPUTE.
LAUNCHING A HASHCLASH COMPUTATION

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

REQUIRES MONITORING: `ls -ltr | tail && date && tail screenlog.0`

IF A STEP TAKES MORE THAN 1H, KILL AND BACKTRACK AT PREVIOUS STEP:

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

Example of stalled HashClash computation:

```bash
[...]
-rwxr-xr-x 3 corkami corkami 4096 May 26 08:38 workdir4
-rw-r--r-- 1 corkami corkami 552859 May 26 08:45 screenlog.0
Sun May 26 15:22:41 UTC 2019
25: Q14Q3m14tunnel = 2
20: Q5m5tunnel = 6
20: Q4m5tunnel = 1
20: Q14m6Q3m5tunnel = 0
21: Q10m10tunnel = 2
21: Q9m10tunnel = 6
```
A 9-block CPC of YES and NO (differences are irrelevant)

Collision blocks

Random buffer (partial birthday attack bits)

Padding
So, we have two files. Any pair of files.
What a CPC does

Pad both files to the same length.

We compute a collision, that appends different blocks to both files.

Makes sense only if both formats tolerate appended data.
HashClash

Almighty, but slow
(requires some attention to compute)
**Input: Two arbitrary prefixes**

Their content and length don’t matter.
Shorter prefixes don’t make anything faster.

Both are padded to the same size.
The last 12 bytes before the collision blocks are used for the attack. Different on both sides.

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

CPC is made of 2 steps:

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

(Which may require backtracking)

Only the Birthday Search benefits from GPU.

The fewer the blocks, the more complex the BS: \(400\text{KH}\) for a single block CPC.

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

If two file formats tolerate appended data:
Compute collision. Done.

+ Straightforward
- Only works for a single pair.
Using CPC as a prefix like an IPC

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

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

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

A collision makes two different contents have the same hash.

→ They can be chained like a tree.

Top nodes can be an IDC, others CPCs.

→ Colliding more than 2 files

N collisions ⇒ N+1 colliding contents

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

https://github.com/corkami/collisions#pdf
MD5 is

cryptographic hash

to have fun!
All the known (implemented) collisions attacks on MD5:

- **FastColl**
  - Two blocks
  - A few seconds
  - In the middle
  - (Away from start or end)

- **UniColl**
  - Two blocks
  - A few minutes
  - In prefix

- **HashClash**
  - 7-9 blocks
  - A few hours
  - Irrelevant
A typical layout for an instant & generic collision via UniColl
Acknowledgments

They made this workshop possible:
Barbie Auglend, Christophe Brocas, Philippe Teuwen.

They made it better:
Jean-Philippe Aumasson, Nicolas Grégoire.
Thank you for making it this far! Any feedback is welcome!
<table>
<thead>
<tr>
<th>File format</th>
<th>Comment length</th>
<th>Generic collision</th>
<th>FastColl</th>
<th>UniColl</th>
<th>Shattered</th>
<th>HashClash</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF</td>
<td>32</td>
<td>✓</td>
<td>✓</td>
<td>✓ *</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JPG</td>
<td>16</td>
<td>✓ *</td>
<td>✓ *</td>
<td>✓ *</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PNG</td>
<td>32</td>
<td>✓ / X</td>
<td>✓ *</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MP4</td>
<td>32/64</td>
<td>✓ *</td>
<td>✓ *</td>
<td>✓ *</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PE</td>
<td>?</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GIF</td>
<td>8</td>
<td>✗</td>
<td>✓ *</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ZIP</td>
<td>16</td>
<td>✗</td>
<td>✓ *</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ELF/TAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mach-O/Class</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
UniColl attack patterns
PDF

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

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

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

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

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

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

Use **FF FE** comments for alignment, then a comment of length **0x77**

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

The other file will have a longer chunk of **0x177**.

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

CF https://github.com/corkami/collisions#jpg
MP4

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

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

The other file will have a longer chunk of 0x179.
0040: .. .. .. .. .. .. .. 00 00 01 79 .F .R .E .E ..

More details @ https://github.com/corkami/collisions#mp4-and-others
Comment segment in JPG: FF FE
Scans can be bigger than 64 KB -> split them via saving as progressive

JPEG file structure
Other UniColl Variants (for completeness)

- Less predictable difference
+ 16 fixed bytes after the first difference

Difference on the last byte
PE collisions
via a CPC used like an IPC
ANATOMY OF A PE FILE

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

Dos Header: declares Executable, points to PE Header.

Dos Stub
(old 16b code)

Rich Header
(MS linker information)

Points to

Dos Header:
declares Executable, points to PE Header.

Dos Stub
(16 bit code)

Rich Header
(MS Linker information)

PE Header
(critical)
Abusing PE files

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>VSize</th>
<th>Address</th>
<th>PSize</th>
<th>Offset</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.text</td>
<td>0003004A</td>
<td>00001000</td>
<td>00031000</td>
<td>00002000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.rdata</td>
<td>00007436</td>
<td>00032000</td>
<td>00000000</td>
<td>00035000</td>
<td>40000000</td>
</tr>
<tr>
<td>3</td>
<td>.data</td>
<td>000000B8</td>
<td>0003A000</td>
<td>00003000</td>
<td>0003B000</td>
<td>C0000040</td>
</tr>
<tr>
<td>4</td>
<td>.rsrc</td>
<td>00000320</td>
<td>0003F000</td>
<td>00001000</td>
<td>0003E000</td>
<td>40000000</td>
</tr>
</tbody>
</table>

Only Magic and pointers are important, can be discarded.
CPC-IPC exploitation for PE files

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

→ instant collision of any pair of PE files (with no code modification)
Windows PE collisions

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

→ Reusable and instant PE collision
SHATTERED: a SHA-1 IPC
Shattered

- an IPC for SHA1
- Computed only once (?)
- Differences at start and end
  -> "easy" to exploit

Official PoCs - JPGs in PDFs
PDFs embed JPGs natively
**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
Exploiting hash collisions
Ange Albertini
BlackAlps 2017
Switzerland

For more details about Shattered exploitation:
New in 2019

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

Gaëtan Leurent\textsuperscript{1} and Thomas Peyrin\textsuperscript{2,3}

SHA-1: faster practical CPC (never computed yet)

MD5: more efficient CPC in 2 blocks.
(little impact)
**Current hash collision complexity**

**SHA1**

<table>
<thead>
<tr>
<th>IPC</th>
<th>2^{65}</th>
<th>2017 Stevens (Shattered)</th>
<th>The first collision for full SHA-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPC</td>
<td>2^{77}</td>
<td>2013 Stevens</td>
<td>New collision attacks on SHA-1</td>
</tr>
<tr>
<td></td>
<td>2^{67}</td>
<td>2019 Leurent</td>
<td>From Collisions to Chosen-Prefix Collisions</td>
</tr>
</tbody>
</table>

**MD5**

<table>
<thead>
<tr>
<th>IPC</th>
<th>2^{16}</th>
<th>2009 Stevens (FastColl)</th>
<th>Short chosen-prefix collisions for MD5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPC</td>
<td>2^{39}</td>
<td>9 blocks</td>
<td>2009 Stevens (HashClash)</td>
</tr>
<tr>
<td></td>
<td>2^{53}</td>
<td>1 block</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>2^{46}</td>
<td>2 blocks</td>
<td>2019 Leurent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>From Collisions to Chosen-Prefix Collisions</td>
</tr>
</tbody>
</table>