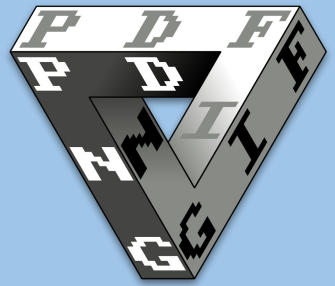
GENERATING WEIRD FILES

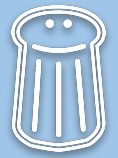
An introduction to Mitra



Ange Albertini

6/7/2021

Pass the SALT



About the author

My own views and opinions

- hacker since 1989, file format expert at Corkami, single dad of 3
- CPS-2 Shock, PoC or GTFO, Pwnie Award of Crypto 2017

Professionally

- 13 years of malware analysis
- 2 years of InfoSec Engineer at Google

My license plate is a CPU architecture
My phone case is a PDF doc
My resume is a Super NES/Megadrive rom

This talk

THE CURRENT SLIDE IS AN

HONEST TALK TRAILER

A CORKAMI ORIGINAL PRODUCTION

No new exploit, nothing to be patched

just file format tricks

Contents

Introduction to format abuses and Mitra

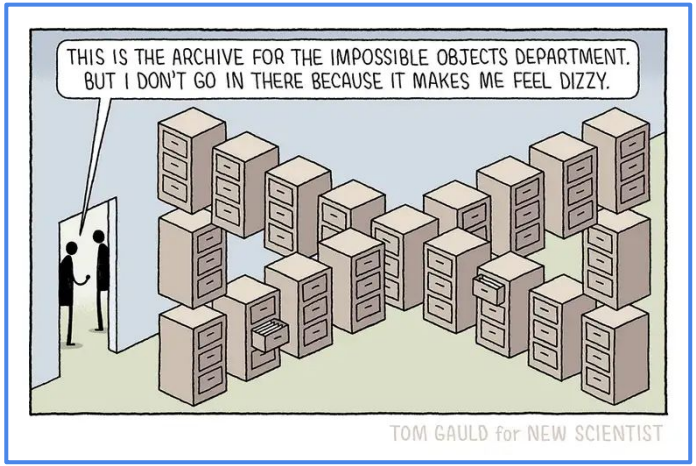
Strategies: concatenations, cavities, parasites, zippers

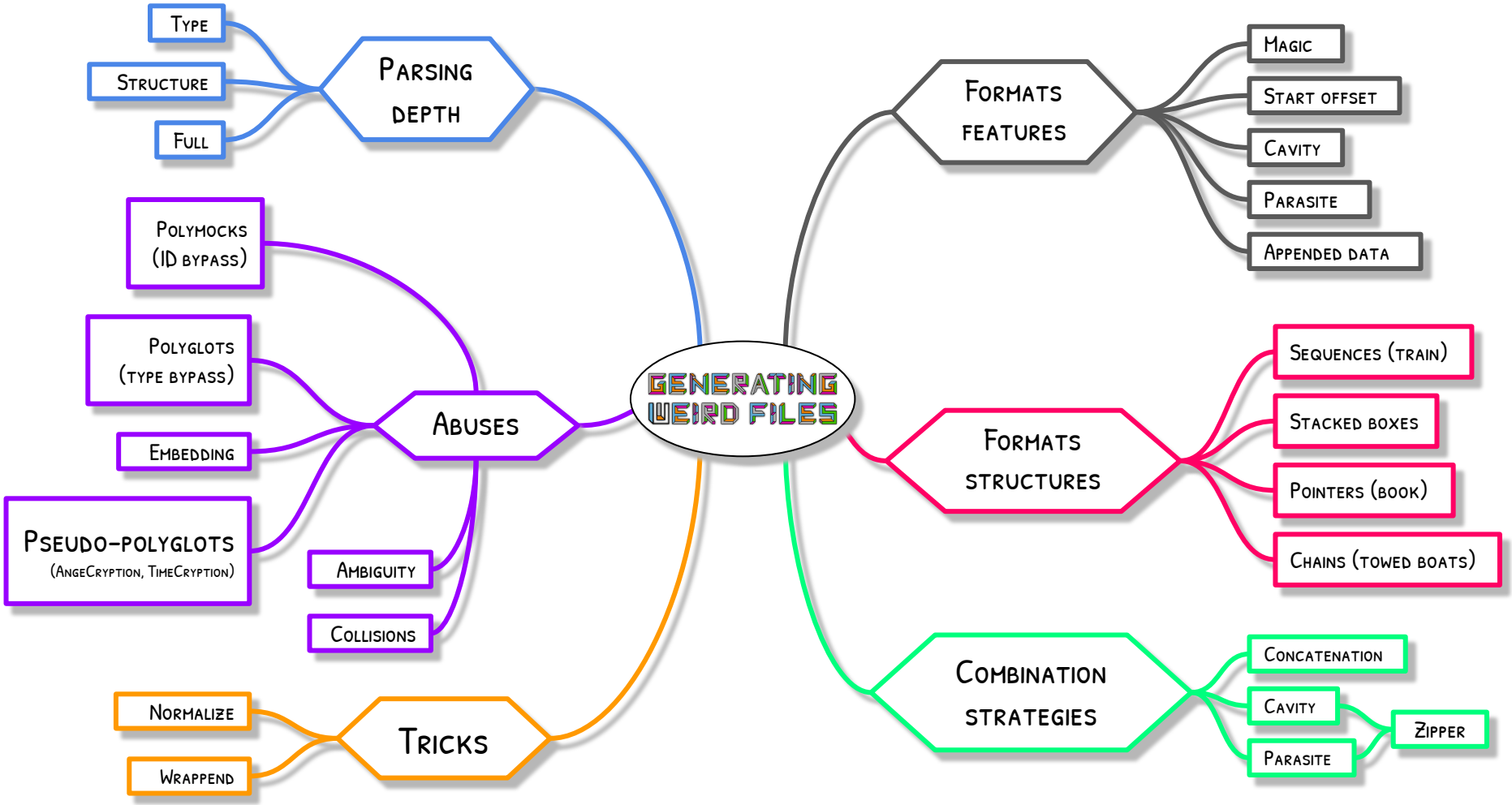
Categories: mocks, polymocks, polyglots

(how Mitra works, how to use it)

Near polyglots & cryptographic attacks

Conclusion, bonus

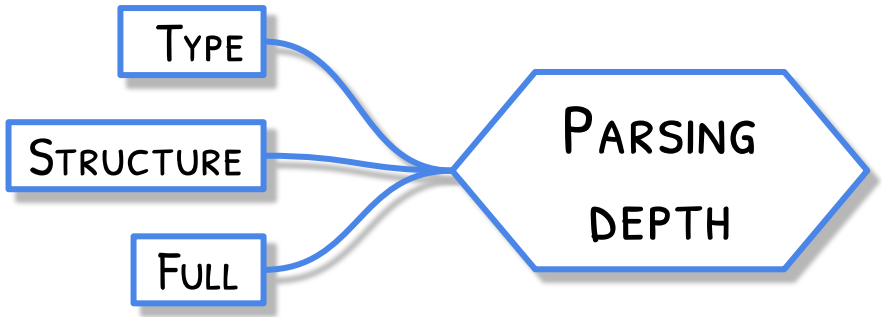




Covered topics

Different depths of file **parsing**

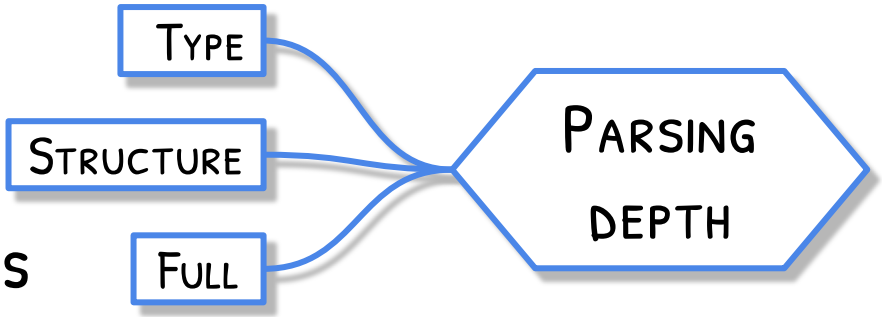
1. File type identification: just check the magic
2. Parse/validate the overall structure
3. Parse every element - e.g. to render it

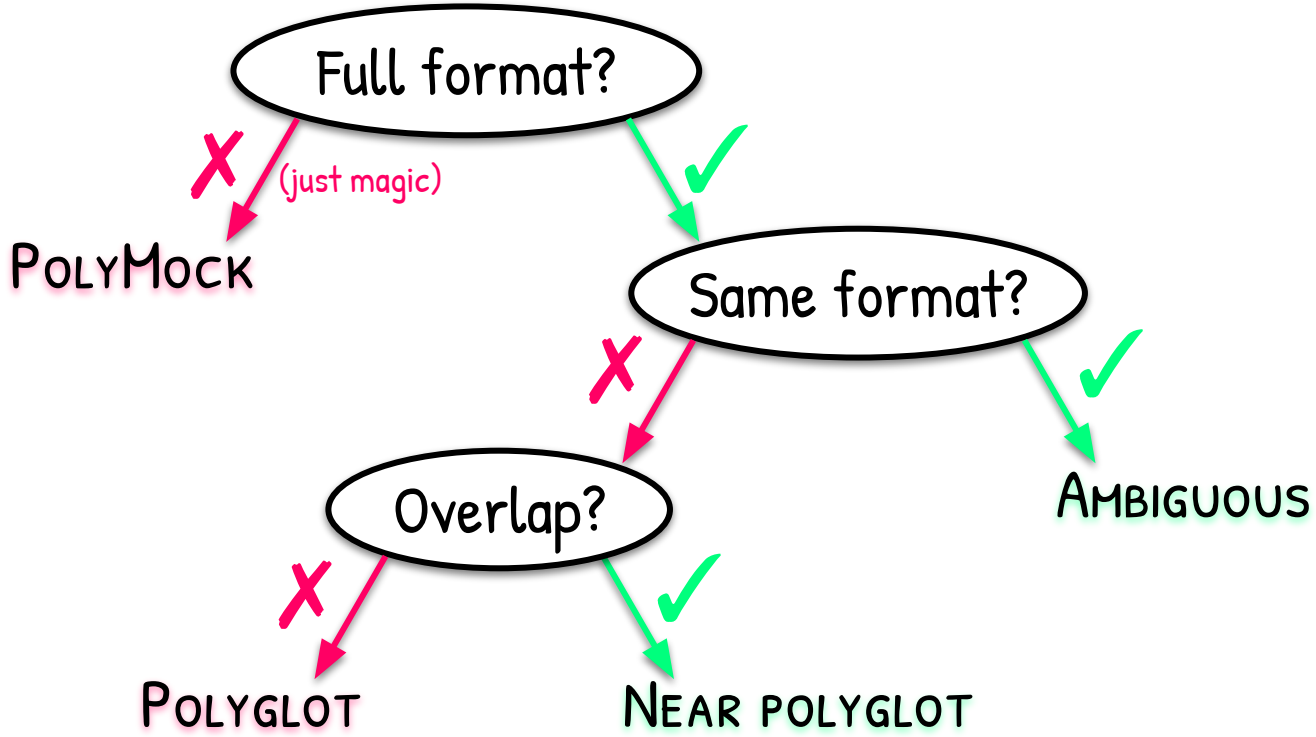


Different depths of file ABUSING

1. Add a fake magic to fool identification -> [poly]mocks
2. Store extra information:
   - Foreign payload
   - Extra file type -> polyglots
   - Hash collisions, near polyglots
3. Parser differences:

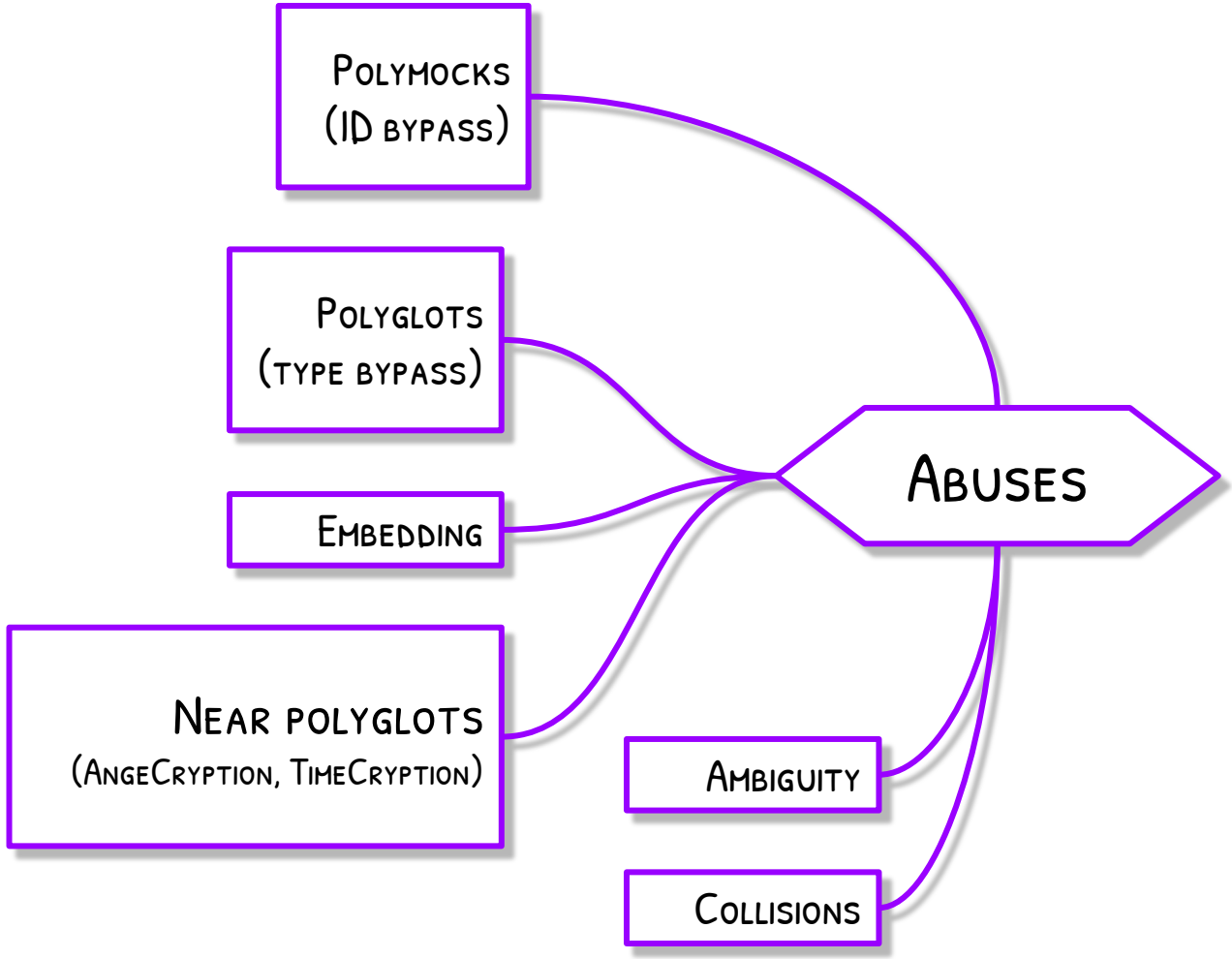
   -> ~~Schizophrenic~~ Ambiguous files

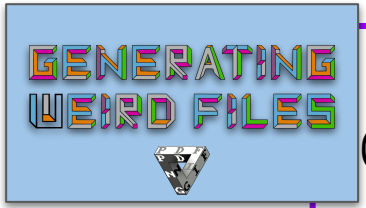




Clarifications

Abuses


Polymocks
(ID bypass)
Polyglots
(type bypass)
Embedding
Near polyglots
(AngeCryption, TimeCryption)
Abuses
Ambiguity
Collisions



Polymocks
(ID bypass)

Polyglots
(Type bypass)

Embedding

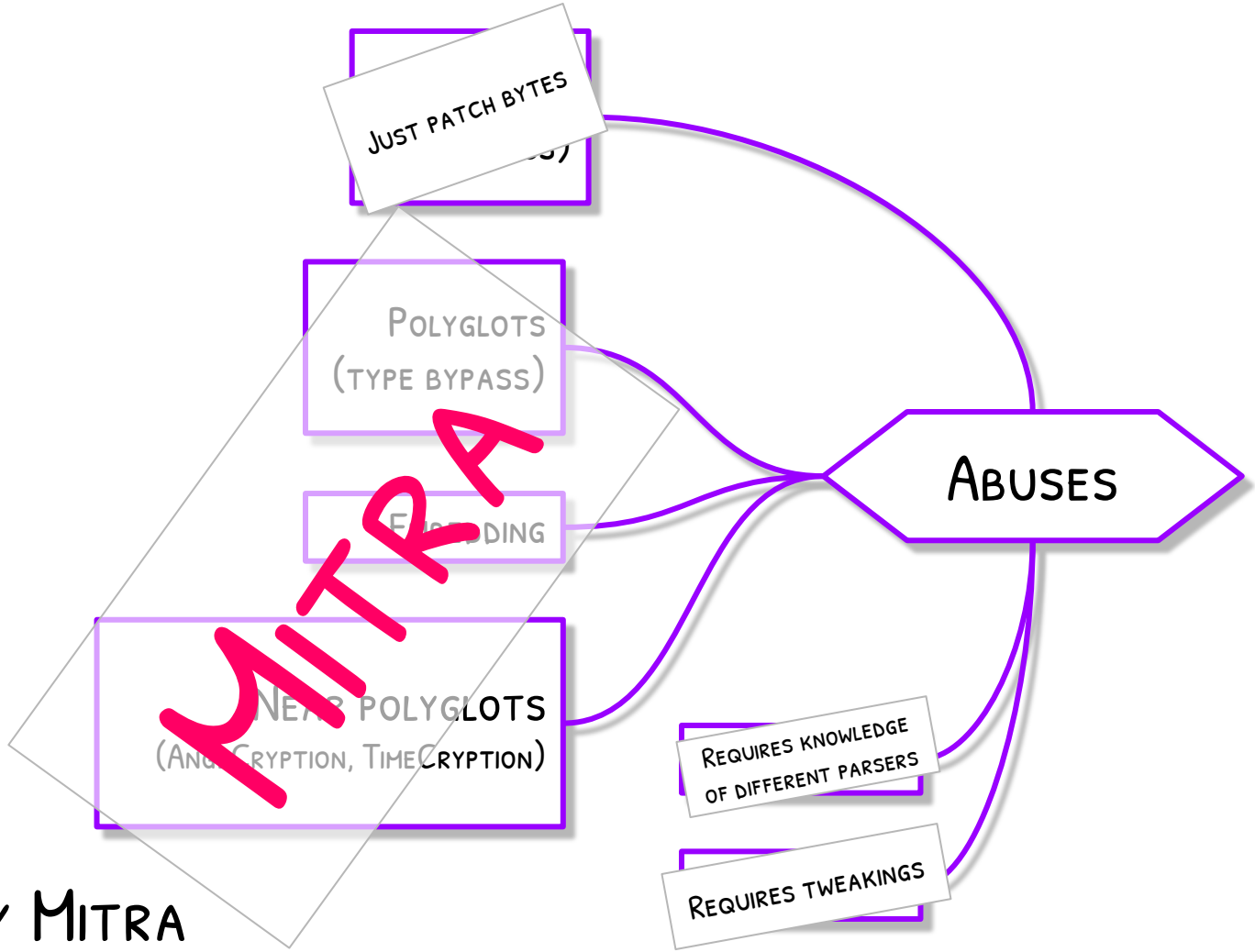
Abuses

Near polyglots
(AngeCryption, TimeCryption)

Ambiguity

Collisions

Talks on the topics



Covered by Mitra

Mitra https://github.com/corkami/mitra

Open-source software, MIT license

Takes 2 files as input, identifies file types

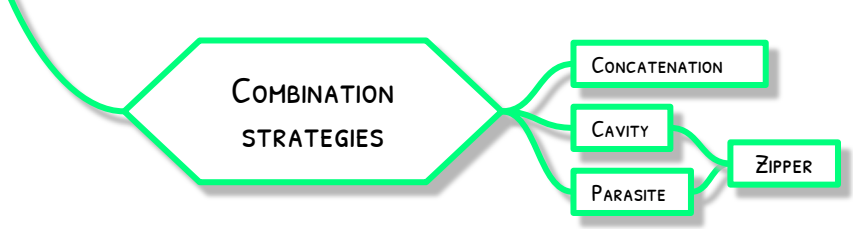
Generates possible polyglots

and optionally near polyglots

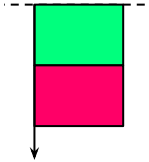
```
$ mitra.py dicom.dcm png.png
dicom.dcm
File 1: DICOM / Digital Imaging and Communications in Medicine
png.png
File 2: PNG / Portable Network Graphics

Zipper Success!
Zipper: interleaving of File1 (type DCM) and File2 (type PNG)
```

Named after Mithridates
(a famous polyglot)

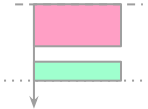
Combination strategies



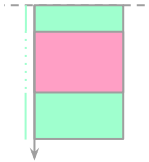
1. Concatenation (appended data)



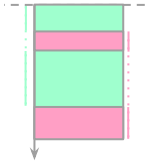
2. Cavities (filling empty space)

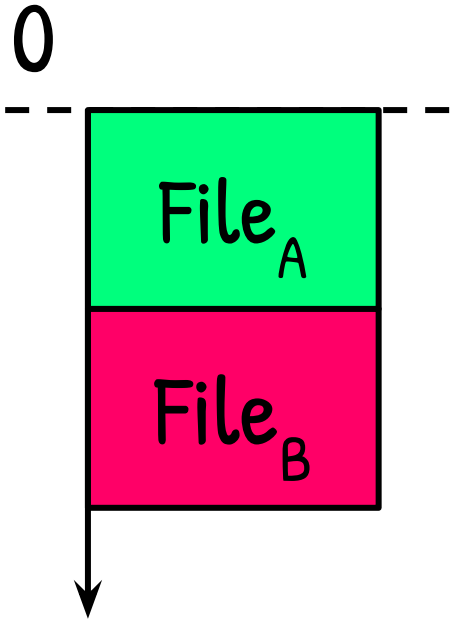


3. Parasite (comment)



4. Zipper (mutual comments)

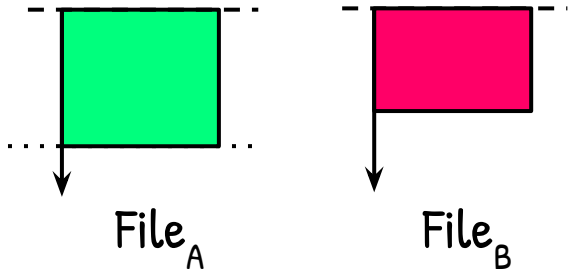




- $Type_A$ must tolerate **appended data**
- $Type_B$ must be allowed to start at $offset_B > size_A$

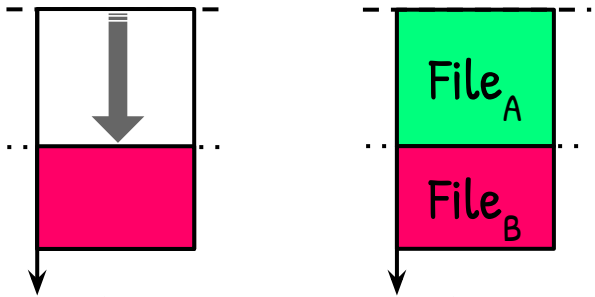
Polyglots by concatenation (appended data)

Start files



1. Relocating (changing offset)

2. Appending (concatenating)



most of the time, these don't require any data update

Making a polyglot by concatenation

Distribution of malicious JAR appended to MSI files signed by third parties

TUESDAY, JANUARY 15, 2019 | BERNARDO.QUINTERO | LEAVE A COMMENT

SHA-1	c6936693c44f1f8f6d0d667295aa6da836b17722
SHA-256	dd71284ac6be9758a5046740168164ae76f743579e24929e0a840afd6f2d0d8e
Vhash	8a55475fa971f5f610029d630ae49cdb
SSDEEP	12288:NhJ23hpbXTMnQX6LSxFEY2Oxlbo4wzYJhjbviA0:NW33T0s6LSx6Ys48AZbvi/
File type	Windows Installer
Magic	CDF V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Title: Inst Subject: Google Update Helper, Author: Google Inc., Keywords: Installer, Comments: (2010 Google Inc., Template: Intel
File size	411.98 KB (421863 bytes)

History

Creation Time	2018-05-08 22:47:50
Signature Date	2018-05-08 22:47:00
First Submission	2018-08-22 09:25:16
Last Submission	2018-08-22 09:25:16
Last Analysis	2020-08-17 01:26:48

Interesting tactic by Ratty & Adwind for distribution of JAR appended to signed MSI – CVE-2020-1464

June 28, 2020 | Adwind, binwalk, Bytecode Viewer, CVE-2020-1464, file, Glueball, IoCs, JAR, Java, MSI, RAT, Ratty, xxd, Yara, ZIP

MD5	63bed40e389b78379b47818ba912ee43
SHA-1	11c72a1239ffe8b6bcd2f5418c369b044f3bfc4a
SHA-256	90f613caa131c663e32aabc31b5fccc99edcfa874110d51cd627531d3a67b16d
Vhash	ae33802bf491401350747d9142922588
SSDEEP	6144:01kCxZjgS007NNMX/+DoklCAFNWCICA+jp02GmaZ/ZJSEPavLFjt+W8:
File type	Windows Installer
Magic	CDF V2 Document, Little Endian, Os: Windows, Version 6.0, Code page: 1252, Microsoft Software Update for Web Folders (English) 14, Author: Microsoft C Database, Release, Comments: This Installer database contains the logic and Software Update for Web Folders (English) 14., Template: Intel
File size	382.29 KB (391467 bytes)

History

Creation Time	2010-03-30 17:26:02
Signature Date	2010-03-30 18:28:00
First Submission	2020-06-03 12:00:45
Last Submission	2020-06-03 12:00:45
Last Analysis	2020-06-28 16:19:53

Names

29-05-2020.jar

An old trick that still works

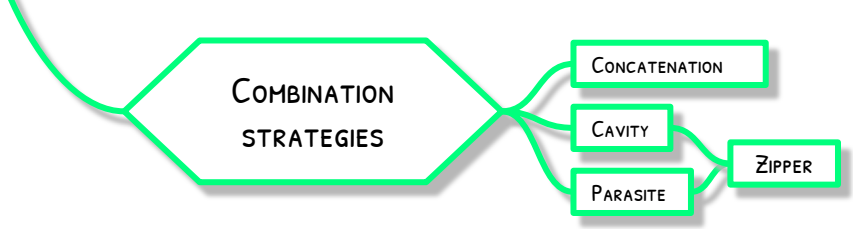
<rant>

MANY POLYGLOTS WOULD BE PREVENTED

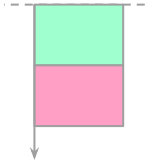
IF FORMATS WERE REQUIRED

TO START AT OFFSET ZERO

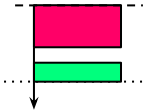
Combination strategies



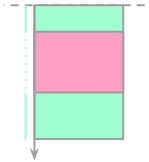
1. Concatenation (appended data)



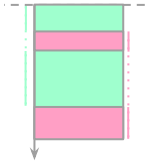
Cavities (filling empty space)



3. Parasite (comment)



4. Zipper (mutual comments)



Cavity

Some file formats start with ignored, empty space (cavity)

-> just copy a file small enough at that place

1. Host file must start with a big enough cavity
2. Parasite file must tolerate appended data

```
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: .D .I .C .M 02 00 00 00 55 4C 04 00 D4 00 00 00
```

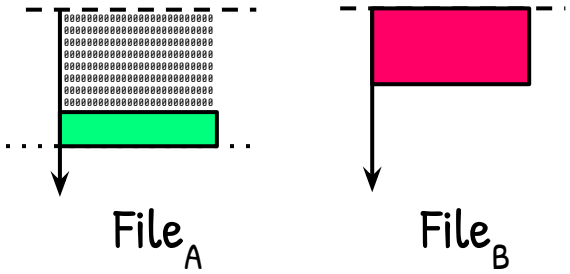
The 128 bytes preamble in a
Digital Imaging and Communications in Medicine file.

```
0000: 00 00 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
...
7FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8000: 01 .C .D .0 .0 .1 00 .L .I .N .U .X .  .  .  .
```

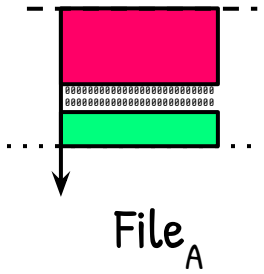
The first 16 sectors (32 KiB) of an ISO 9660 image.

Filling a cavity

Start files



1. Overwrite cavity



[POLY]MOCKS

Principles

File types are identified

1. by a magic
2. at a given offset [range]

`file` scans types by category in alphabetical order

acorn…console…images…filesystems…msdos…windows…zyxel

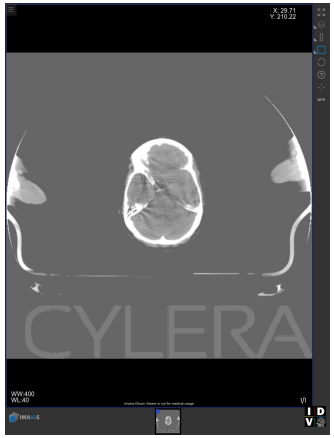
https://github.com/file/file/tree/master/magic/Magdir

acorn	coff	gnome	mcrypt	pgf	teapot
adi	commands	gnu	measure	pgp	terminfo
adventure	communications	gnumeric	mercurial	pgp-binary-keys	tex
aes	compress	gpt	metastore	pkgadd	tgif
algol68	**console**	gpu	meteorological	plan9	ti-8x
allegro	convex	grace	microfocus	plus5	timezone
alliant	coverage	graphviz	mime	pmem	tplink
alpha	cracklib	gringotts	mips	polyml	troff
amanda	crypto	guile	mirage	printer	tuxedo
amigaos	ctags	hardware	misctools	project	typeset
android	ctf	hitachi-sh	mkid	psdbms	uf2
animation	cubemap	hp	mlssa	psl	unicode
aout	cups	human68k	mmdf	pulsar	unisig
apache	dact	ibm370	modem	pwsafe	unknown
apl	database	ibm6000	modulefile	pyramid	usd
apple	dataone	icc	motorola	python	uterus
application	dbpf	iff	mozilla	qt	uuencode
applix	der	**images**	**msdos**	revision	vacuum-cleaner
apt	diamond	inform	msooxml	**riff**	varied.out
archive	dif	intel	msvc	rinex	varied.script
arm	diff	interleaf	msx	rpi	vax
asf	digital	island	mup	rpm	vicar
assembler	dolby	ispell	music	rpmsg	virtual
asterix	dump	isz	nasa	rst	virtutech
att3b	dyadic	java	natinst	rtf	visx
audio	ebml	javascript	ncr	ruby	vms
avm	edid	jpeg	neko	sc	vmware
basis	editors	karma	netbsd	sccs	vorbis
beetle	efi	kde	netscape	scientific	vxl
ber	elf	keepass	netware	securitycerts	warc
bflt	encore	kerberos	news	selinux	weak
bhl	epoc	kicad	nitpicker	sendmail	web
bioinformatics	erlang	kml	numpy	sequent	webassembly
biosig	espressif	lammps	oasis	sereal	**windows**
blackberry	esri	lecter	ocaml	sgi	wireless
blcr	etf	lex	octave	sgml	wordprocessors
blender	fcs	lif	ole2compounddocs	sharc	wsdl
blit	**filesystems**	linux	olf	sinclair	x68000
bm	finger	lisp	openfst	sisu	xdelta
bout	flash	llvm	opentimestamps	sketch	xenix
bsdi	flif	locoscript	os2	smalltalk	xilinx
bsi	fonts	lua	os400	smile	xo65
btsnoop	forth	luks	os9	sniffer	xwindows
c64	fortran	m4	osf1	softquad	yara
cad	frame	mach	palm	sosi	zfs
cafebabe	freebsd	macintosh	parix	spec	zilog
cbor	fsav	macos	parrot	spectrum	zip
cddb	fusecompress	magic	pascal	sql	zyxel
chord	games	mail.news	pbf	ssh	
cisco	gcc	make	pbm	ssl	
citrus	gconv	map	pc88	statistics	
c-lang	geo	maple	pc98	sun	
clarion	geos	marc21	pcjr	sylk	
claris	gimp	mathcad	pdf	symbos	
clipper	git	mathematica	pdp	sysex	
clojure	glibc	matroska	perl	tcl	

A Windows executable that starts with MZ (CVE-2019-11687)
is identified as DICOM medical image by `file`
because **`image`**`s` is scanned before **`m`**`sdos`
(even if the DOS magic is at 0, before the DICOM magic)

```
00  .M .Z 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
10  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
30  00 00 00 00 00 00 00 00 00 00 00 00 6C 01 00 00

40  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21  T  h
50   i  s     p  r  o  g  r  a  m     c  a  n  n  o
60   t     b  e     r  u  n     i  n     D  O  S
70   m  o  d  e  .  \r \r \n  $ 00 00 00 00 00 00 00

80  .D .I .C .M 02 00 00 00 55 4C 04 00 D0 00 00 00
90  02 00 01 00 4F 42 00 00 02 00 00 00 00 01 02 00
```

```
$ file justanotherwannacry.dcm
justanotherwannacry.dcm: DICOM medical imaging data
```



justanotherwannacry.dcm
63/71 on VirusTotal

Mock files

Just put a mock magic at the right offset

Trivial - and good enough to bypass security?

A POLYMOCK – A 190-IN-1 YET EMPTY FILE

The file is **mostly empty!** It only contains magics to fake file types.

https://github.com/corkami/pocs/tree/master/polymocks

```
multi: Windows Program Information File for \030(o\001
- MAR Area Detector Image,
- Linux kernel x86 boot executable RW-rootFS,
- ReiserFS V3.6
- Files-11 On-Disk Structure (ODS-52); volume label is '            '
- DOS/MBR boot sector
- Game Boy ROM image (Rev.00) [ROM ONLY], ROM: 256Kbit
- Plot84 plotting file
-  DOS/MBR boot sector
- DOSFONT2 encrypted font data
- Kodak Photo CD image pack file , landscape mode
- SymbOS executable v., name: HNRO0\334\247\304\375]\034\236\243
- ISO 9660 CD-ROM filesystem data (raw 2352 byte sectors)
- Nero CD image at 0x4B000 ISO 9660 CD-ROM filesystem data
- High Sierra CD-ROM filesystem data
- Old EZD Electron Density Map
- Apple File System (APFS), blocksize 24061976
- Zoo archive data, modify: v78.88+
- Symbian installation file
- 4-channel Fasttracker module sound data Title: "MZ`\352\210\360'\315!"
- Scream Tracker Sample adlib drum mono 8bit unpacked
- Poly Tracker PTM Module Title: "MZ`\352\210\360'\315!"
- SNDH Atari ST music
- SoundFX Module sound file
- D64 Image
- Nintendo Wii disc image: "NXSB\030(o\0
- Nintendo 3DS File Archive (CFA)
- Unix Fast File system [v1] (litt
- Unix Fast File system [v2] (litt
- Unix Fast File system [v2] (littl
- ISO 9660 CD-ROM filesystem data (
- F2FS filesystem, UUID=00000000-0000-0000-0000-000000000000, volume name ""
- DICOM medical imaging data
- Linux kernel ARM boot executable zImage (little-endian)
- CCP4 Electron Density Map
- Ultrix core file from 'X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVI...
- VirtualBox Disk Image (MZ`\352\210\360'\315!), 5715999566798081280 bytes
- MS Compress archive data
- AMUSIC Adlib Tracker MS-DOS executable, MZ for MS-DOS COM executable for DOS
- JPEG 2000 image
- ARJ archive data
- unicos (cray) executable
- IBM OS/400 save file data
- data
```

output from
`file --keep-going`

00	.M	.Z	60	EA	.j	.P	01	07	19	04	00	10	.S	.N	.D	.H
10	.N	.R	.O	.0	DC	A7	C4	FD	5D	1C	9E	A3	.R	.E	.~	.^
20	.N	.X	.S	.B	18	28	6F	01	.P	.K	03	04	.P	.T	.M	.F
30	.S	.y	.m	.E	.x	.e	.7	.z	BC	AF	27	1C	.S	.O	.N	.G
40	7F	10	DA	BE	00	00	CD	21	.P	.K	01	02	.S	.C	.R	.S
50	.R	.a	.r	.!	^Z	07	01	00	.L	.R	.Z	.I	.P	.L	.O	.T
60	.%	.%	.8	.4	.R	.a	.r	.!	^Z	07	00	00	00	.M	.A	.P
70	.	.(	FD	.7	.z	.X	.Z	00	04	22	4D	18	03	21	4C	18
80	.D	.I	.C	.M	.%	.P	.D	.F	.-	.1	..	.4	.	.o	.b	.j
…	…															

Many magics are at the start of the file.

This file is simultaneously detected as:

- DOS EXE, COM and MBR
- Zoo, ARJ, VirtualBox, MS Compress, 3DS
- ISO, RAW ISO, Nero, PhotoCD
- FastTracker, ScreamTracker, Adlib tracker, Polytracker, SoundFX
- Apple, IBM, HP, Linux, Ultrix, Raid, ODS, Nintendo, Kodak
- EZD, CCP4, Plot84, MAR, Dicom

…

```
0     0x0     Gameboy ROM,, [ROM ONLY], ROM: 256Kbit
80    0x50    RAR archive data, version 5.x
88    0x58    lrzip compressed data
89    0x59    rzip compressed data - versi
114   0x72    xz compressed data
120   0x78    LZ4 compressed data
...
```

output (150 sigs) from `Binwalk`

To make mock files

The polymock source references most `file` magic by offsets.

Just insert the right magic at the right offset.

```
00: 00 00 00 10  f  r  e  e 00 00 00 00 61 15 06 00
10: 00 00 00 1C  f  t  y  p  i  s  o  m 00 00 02 00
20:  i  s  o  m  i  s  o  2  m  p  4  1 00 00 00 08
```

An MP4 file being identified as a Berkeley DB

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File contents


_ats 0, {db "MZ"}       ; DOS executable
;ats 0, {db 0xff, 0xd8} ; JPEG
;ats 0, {db 0x1f, 0xb8} ; Gzip


_ats 2, {db 060h, 0EAh} ; ARJ
;ats 2, {db "-LH1-"},    ; LHA archive (c64)


_ats 4, {db "jP"}           ; JPEG 2000
;ats 4, {db 088h, 0F0h, 39} ; MS Compress ; requires extra stuff
;ats 4, {db "ftyp"}         ; ISO MEDIA
;ats 4, {db "RED1"}         ; REDCode video
;ats 4, {db "pipe"}         ; Clipper instruction trace
;ats 4, {db "prof"}         ; Clipper instruction profile
;ats 4, {db " A~"}          ; Mathematica .ml
;ats 4, {db "PK", 1, 2}     ; Mozilla archive
;ats 4, {dd 0xfb4a}         ; QDos executable


_ats 6, {dw 0x0701}              ; unicos (cray) executable
;ats 6, {db "%!FontType1"}       ; PostScript Type 1 font program data
;ats 6, {db "%!PS-AdobeFont-1."} ; PostScript Type 1 font program data
```

<rant++>

MANY EXPLOITS WOULD BE PREVENTED

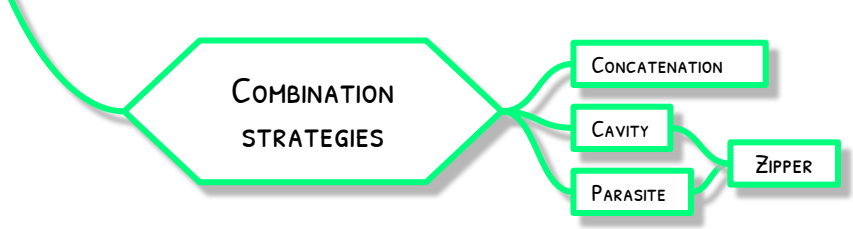
mockfiles

IF FORMATS WERE REQUIRED

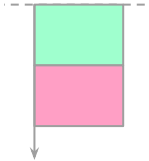
TO START AT OFFSET ZERO

Enforce magics
at offset zero !!1!

Combination strategies

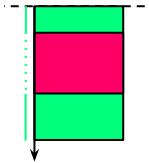


1. Concatenation (appended data)

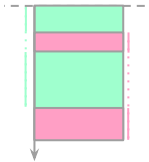


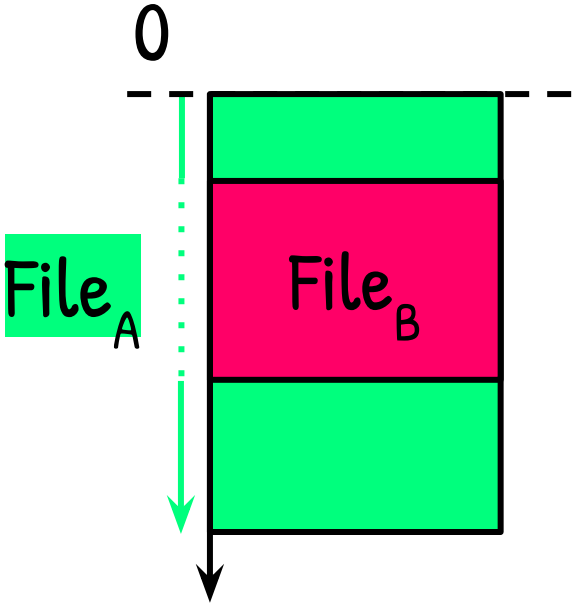
2. Cavities (filling empty space)

Parasite (comment)



4. Zipper (mutual comments)

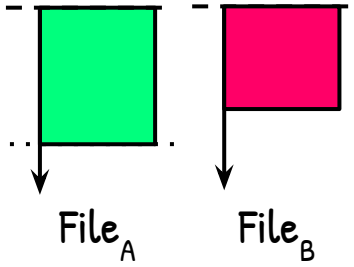




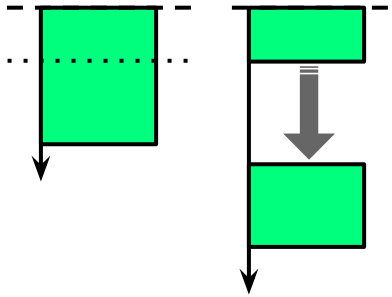
- $Type_A$ must tolerate **parasitizing data**
  typically a length restriction - sometimes contents too
- $Type_B$ must be allowed to start at $offset_B \geq ComStart_A$
  and tolerate appended data.

Abuse by parasite (comment)

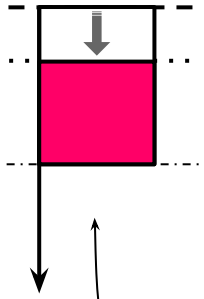
Start files



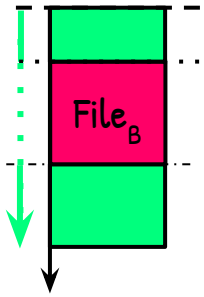
1. Make room
(declare a comment)



2. Relocating
(changing offset)



3. Combining



most of the time,
these don't require
any data update.

Making a polyglot by parasite

Single/small/text comment: 👌
Several/big/random comments: ⚠️

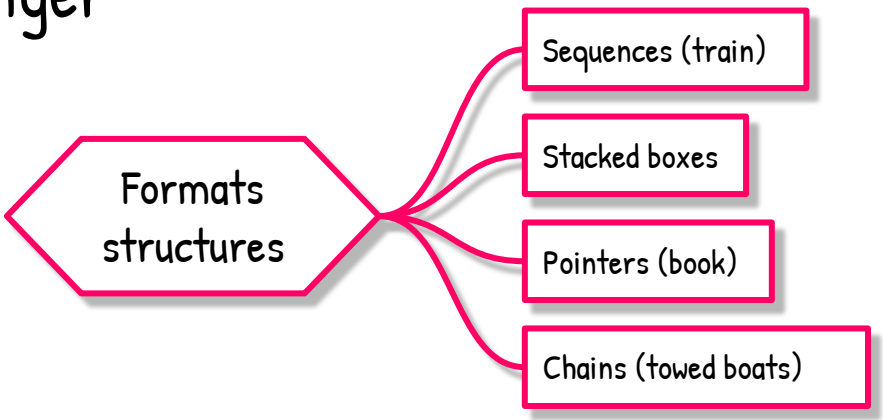
Comments are a normal feature

They're very useful!

However, they could be removed/merged/scanned

Parasitizing

- Train: add wagon, update wagons counter
- Stacked boxes: add a new box
- Book: add pages, update Table of Contents
- Towed boats: make towing rope longer



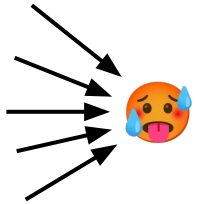
Normalize

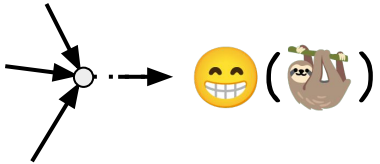
Some formats have **many** different forms (PDF, GIF...)

Some forms are awful to abuse color space, linearization, versions...

Find the right method to normalize to an abusable form

-> generic support of all files for that format





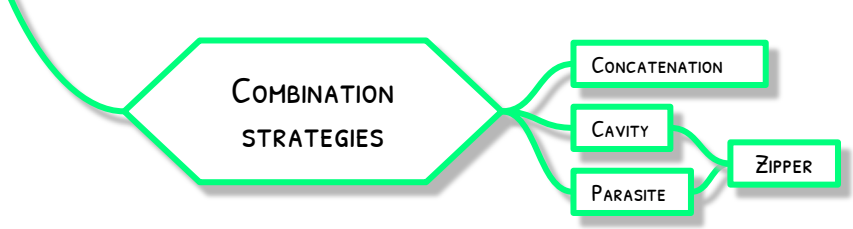
Wrappend

Some formats don't tolerate appended data:

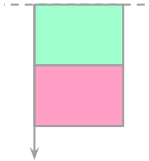
- pure sequences of chunk until EOF (PCAP, DICOM...)
- picky parsers (BPG, Java)
- formats w/ footers (ID3v1, XZ...)

- > Wrap appended data in a trailing chunk parasite
  -> "wrappending"

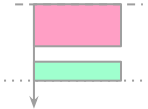
Combination strategies



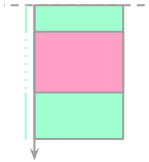
1. Concatenation (appended data)



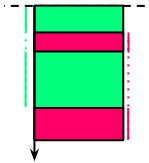
2. Cavities (filling empty space)



3. Parasite (comment)

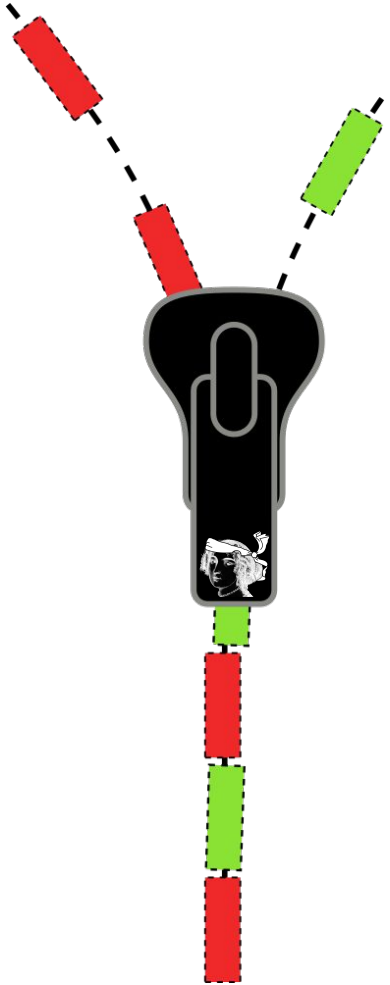


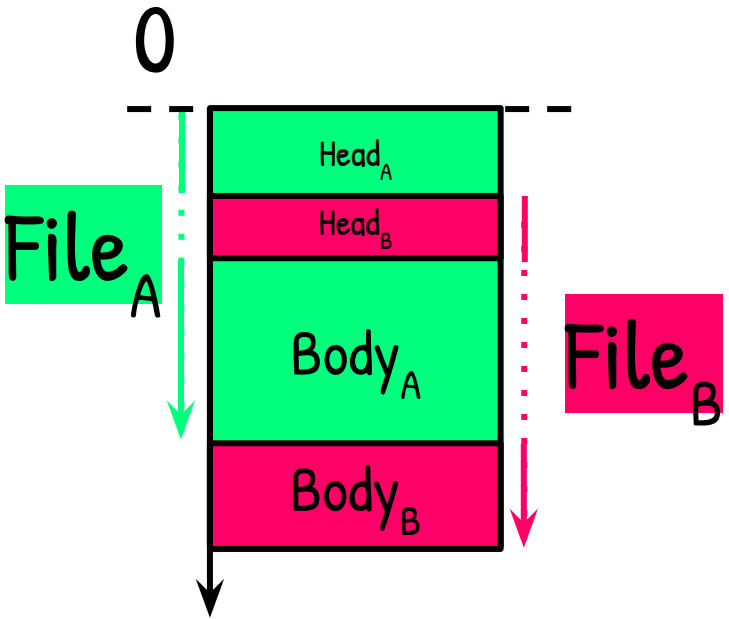
Zipper (mutual comments)



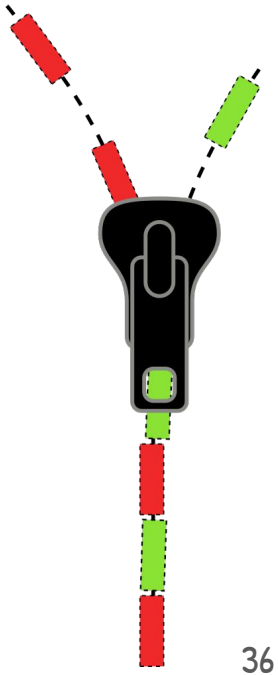
Zippers

I comment your elements out
You comment my elements out

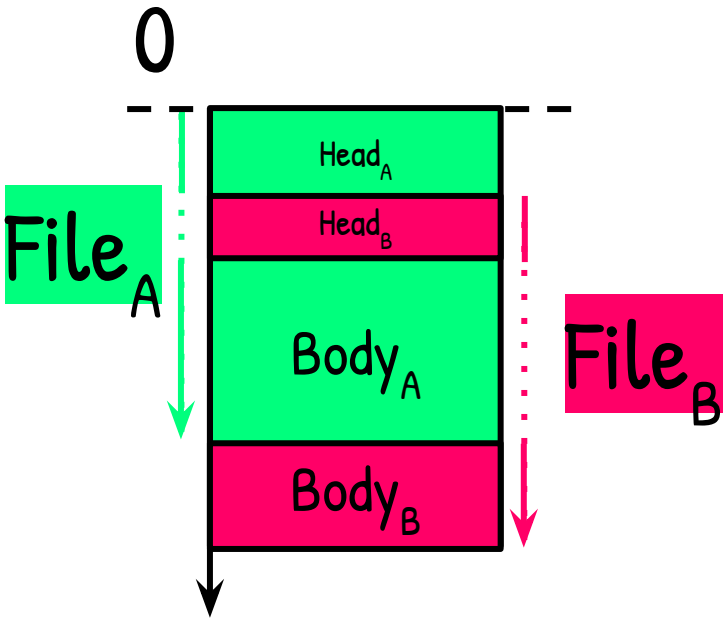




- Typically Head$_A$/Head$_B$/Body$_A$/Body$_B$
- Head$_B$ is a parasite for File$_A$
- Body$_A$ is a parasite for File$_B$
- Body$_B$ is a [wr]appended to File$_A$



Polyglots by zipper (mutual parasites)



File$_A$:

- parasite (even tiny)
- [wr]appended data

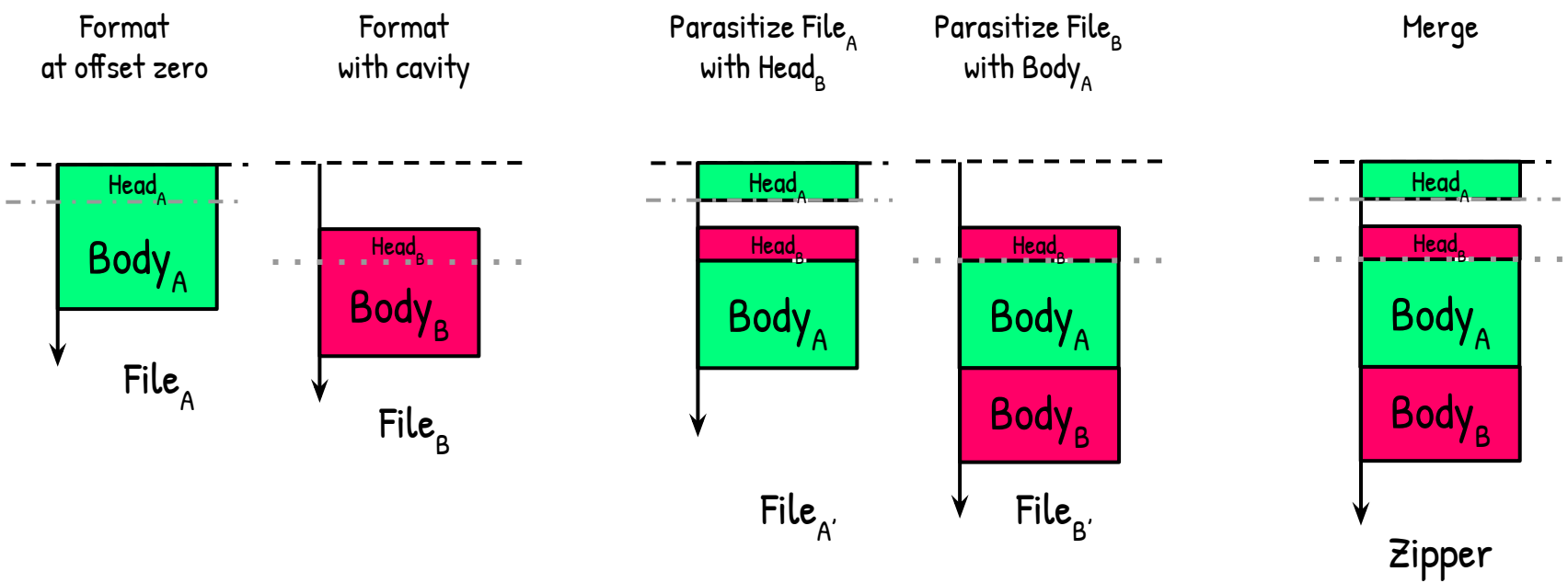
GIF: 255b
JPG, Java, PCAP: 64kb

File$_B$:

- cavity (PDF, DCM, ISO…)
- parasite

Required conditions

Start files



To make a zipper, parasitize then merge

What are zippers good for ?

Overcome constraints

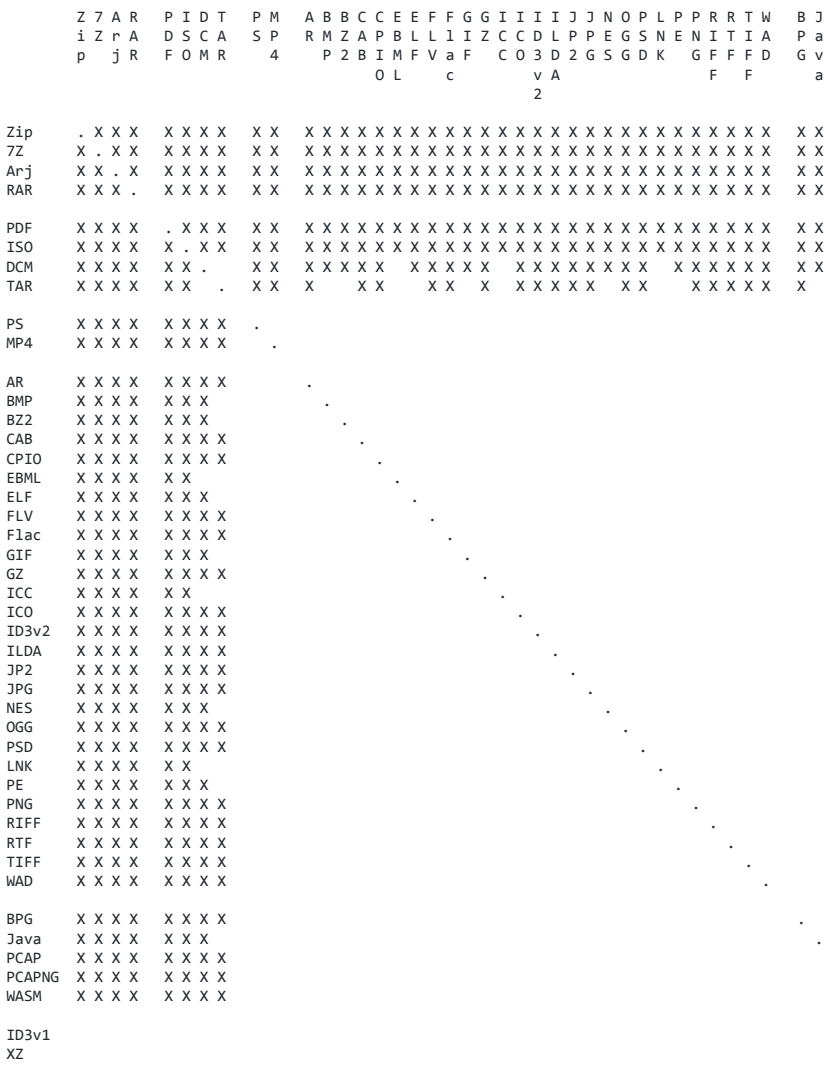
No matter the size of the cavity (Tar, Dicom…)

or the maximum length of a parasite (GIF, JPG, PCAP…)

Results

Many supported formats

Many combinations
via different strategies



Each format characteristic enables more possibilities

	Zip	7Z	Arj	RAR	PDF	ISO	DCM	TAR	PS	MP4	AR	BMP	BZ2	CAB	CPIO	EBML	ELF	FLV	Flac	GIF	GZ	ICC	ICO	ID3v2	ILDA	JP2	JPG	NES	OGG	PSD	LNK	PE	PNG	RIFF	RTF	TIFF	WAD	BPG	Java	PCAP	PCAPNG	WASM	ID3v1	XZ	
Valid at any offset																																													
Zip	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
7Z	X	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
Arj	X	X	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
RAR	X	X	X	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
Formats with cavities (->zippers)																																													
PDF	X	X	X	X	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
ISO	X	X	X	X	X	.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			41
DCM	X	X	X	X	X	X	.		X	X	X	X	X	X	X		X	X	X	X	X		X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X			37
TAR	X	X	X	X	X	X		.	X	X	X			X	X			X	X		X		X	X	X	X	X		X	X			X	X	X	X	X	X		X	X	X			30
PS	X	X	X	X	X	X	X	X	.																																				8
MP4	X	X	X	X	X	X	X	X		.																																			8
Formats enforcing magics at offset zero																																													
AR	X	X	X	X	X	X	X	X			.																																		8
BMP	X	X	X	X	X	X	X					.																																	7
BZ2	X	X	X	X	X	X	X						.																																7
CAB	X	X	X	X	X	X	X	X						.																															8
CPIO	X	X	X	X	X	X	X	X							.																														8
EBML	X	X	X	X	X	X										.																													6
ELF	X	X	X	X	X	X	X										.																												7
FLV	X	X	X	X	X	X	X	X										.																											8
Flac	X	X	X	X	X	X	X	X											.																										8
GIF	X	X	X	X	X	X	X													.																									7
GZ	X	X	X	X	X	X	X	X													.																								8
ICC	X	X	X	X	X	X																.																							6
ICO	X	X	X	X	X	X	X	X															.																						8
ID3v2	X	X	X	X	X	X	X	X																.																					8
ILDA	X	X	X	X	X	X	X	X																	.																				8
JP2	X	X	X	X	X	X	X	X																		.																			8
JPG	X	X	X	X	X	X	X	X																			.																		8
NES	X	X	X	X	X	X	X																					.																	7
OGG	X	X	X	X	X	X	X	X																					.																8
PSD	X	X	X	X	X	X	X	X																						.															8
LNK	X	X	X	X	X	X																									.														6
PE	X	X	X	X	X	X	X																									.													7
PNG	X	X	X	X	X	X	X	X																									.												8
RIFF	X	X	X	X	X	X	X	X																										.											8
RTF	X	X	X	X	X	X	X	X																											.										8
TIFF	X	X	X	X	X	X	X	X																												.									8
WAD	X	X	X	X	X	X	X	X																													.								8
BPG	X	X	X	X	X	X	X	X																														.							8
Java	X	X	X	X	X	X	X																																.						7
PCAP	X	X	X	X	X	X	X	X																																.					8
PCAPNG	X	X	X	X	X	X	X	X																																	.				8
WASM	X	X	X	X	X	X	X	X																																		.			8
Footers																																													
ID3v1																																											.		0
XZ																																												.	0

Magic signatures at offset zero

Under the hood

How Mitra works

Public Service Announcement

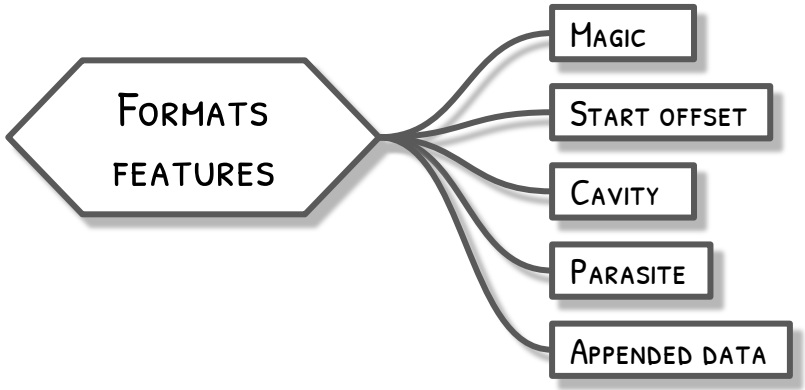
You don't have to fully understand a file format to abuse it

Identify the overall structure

Look for specific characteristics

Move blocks of data around

Adjust offsets and lengths



Mitra is a simple tool

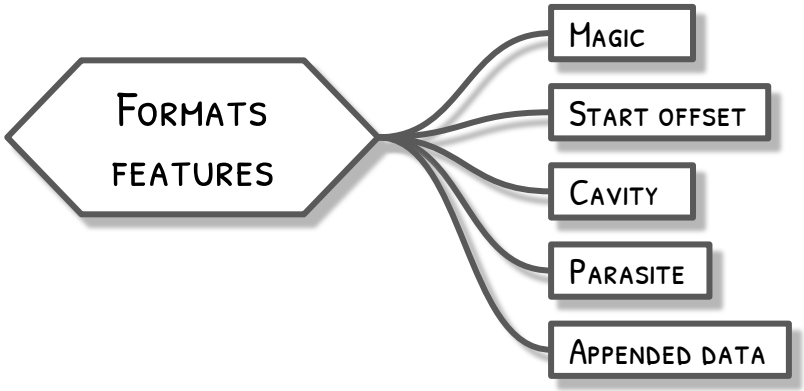
It only does basic identification and manipulations

It doesn't fully understand all formats, and expects **standard** files

It's not a full parser, nor an analysis tool

It does **not validate** output files

Use at your own risk!



Example

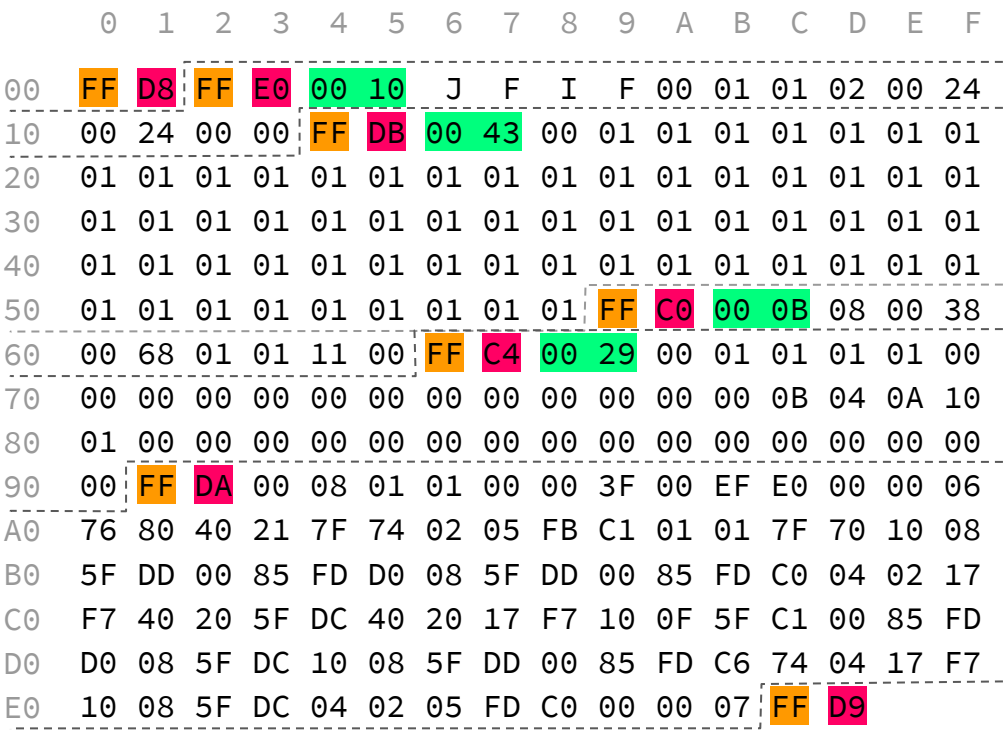
Abusing JPEGs like Mitra

(the laziest possible way)

JPEG is complex! And yet...

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	D8	FF	E0	00	10	J	F	I	F	00	01	01	02	00	24
10	00	24	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01
20	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
30	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
40	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
50	01	01	01	01	01	01	01	01	01	FF	C0	00	0B	08	00	38
60	00	68	01	01	11	00	FF	C4	00	29	00	01	01	01	01	00
70	00	00	00	00	00	00	00	00	00	00	00	00	0B	04	0A	10
80	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	FF	DA	00	08	01	01	00	00	3F	00	EF	E0	00	00	06
A0	76	80	40	21	7F	74	02	05	FB	C1	01	01	7F	70	10	08
B0	5F	DD	00	85	FD	D0	08	5F	DD	00	85	FD	C0	04	02	17
C0	F7	40	20	5F	DC	40	20	17	F7	10	0F	5F	C1	00	85	FD
D0	D0	08	5F	DC	10	08	5F	DD	00	85	FD	C6	74	04	17	F7
E0	10	08	5F	DC	04	02	05	FD	C0	00	00	07	FF	D9		

Let's look at a small JPEG file



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	D8	FF	E0	00	10	J	F	I	F	00	01	01	02	00	24
10	00	24	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01
20	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
30	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
40	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
50	01	01	01	01	01	01	01	01	01	FF	C0	00	0B	08	00	38
60	00	68	01	01	11	00	FF	C4	00	29	00	01	01	01	01	00
70	00	00	00	00	00	00	00	00	00	00	00	00	0B	04	0A	10
80	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	FF	DA	00	08	01	01	00	00	3F	00	EF	E0	00	00	06
A0	76	80	40	21	7F	74	02	05	FB	C1	01	01	7F	70	10	08
B0	5F	DD	00	85	FD	D0	08	5F	DD	00	85	FD	C0	04	02	17
C0	F7	40	20	5F	DC	40	20	17	F7	10	0F	5F	C1	00	85	FD
D0	D0	08	5F	DC	10	08	5F	DD	00	85	FD	C6	74	04	17	F7
E0	10	08	5F	DC	04	02	05	FD	C0	00	00	07	FF	D9		

00: FF D8 Start Of Image (size: n/a) Always first

02: FF E0 Application 0 (size: 10)

14: FF DB Define a Quantization Table (size: 43)

59: FF C0 Start Of Frame 0 (size: 0B)

66: FF C4 Define Huffman table (size: 29)

91: FF DA Start of Scan (size: n/a)

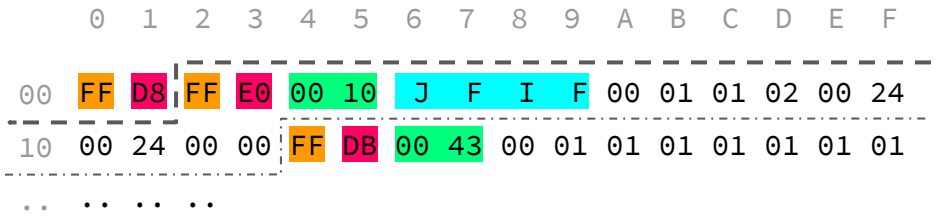
EC: FF D9 End Of Image (size: n/a) Always last

A JPEG FILE: A SEQUENCE OF FF MM LL LL SEGMENTS

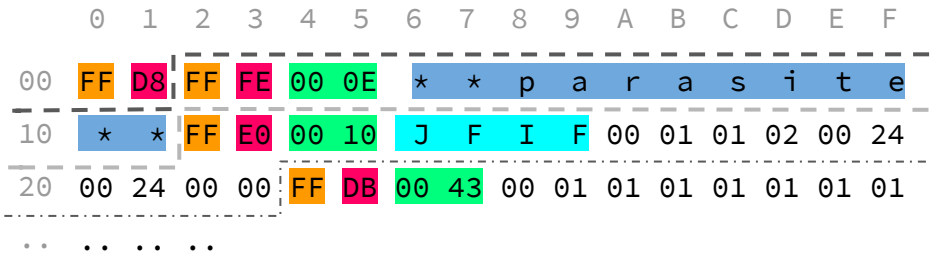
Fixed byte

Marker

Length



00: FF D8 Start Of Image (size: n/a)
02: FF E0 Application 0 (size: 10)
14: FF DB Define a Quantization Table (size: 43)
..: FF .. ...



00: FF D8 Start Of Image (size: n/a)
02: FF FE COMment (size: 0E)
12: FF E0 Application 0 (size: 10)
24: FF DB Define a Quantization Table (size: 43)
..: FF .. ...

len(FF D8)

Parasitizing: insert a COMment segment (FF FE) at offset 2

JPG support in Mitra

Mitra just knows:

- JPEG's magic signature
- Parasites are supported
  - Where to cut the file
  - How to wrap the parasite

(yes, that's the whole source file)

```
01  #!/usr/bin/env python3
02
03  from parsers import FType
04  from helpers import *
05
06
07  class parser(FType):
08        DESC = "JFIF / JPEG File Interchange Format"
09        TYPE = "JPG"
10        MAGIC = b"\xFF\xD8"
11
12        def __init__(self, data=""):
13              FType.__init__(self, data)
14              self.data = data
15
16              self.bParasite = True
17
18              self.parasite_o = 6
19              self.parasite_s = 0xFFFF - 2
20
21              self.cut = 2
22              self.prewrap = 1+1+2
23
24
25        def wrap(self, parasite, marker=b"\xFE"):
26              return b"".join([
27                    b"\xFF",
28                    marker,
29                    int2b(len(parasite)+2),
30                    parasite,
31                    ])
EOF
```

Want to know more ?

Check my PoCs, my docs...

(tiny PoCs here)

A walkthrough of Mitra

How to...

Embedding a payload in a file

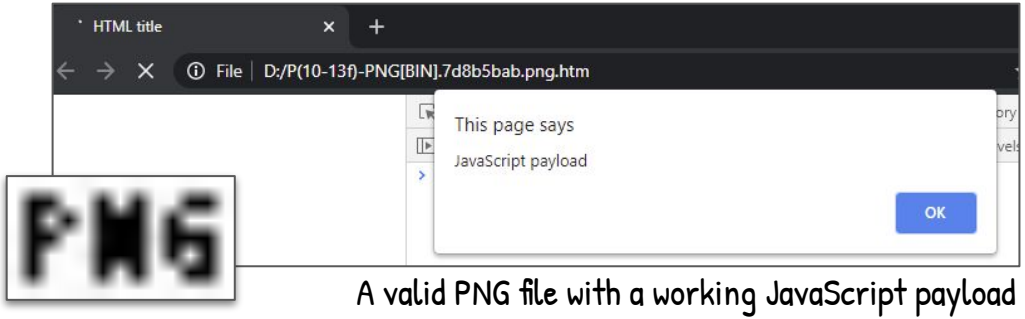
Just use any payload

Use `-f` to force it as a binary blob (with no type)

It's also useful to make room for some data.

Example

```
$ mitra.py png.png script.js -f
png.png
File 1: PNG / Portable Network Graphics
script.js
File 2: binary blob

Stack: concatenation of File1 (type PNG) and File2 (type BIN)
Parasite: hosting of File2 (type BIN) in File1 (type PNG)
```



A valid PNG file with a working JavaScript payload

```
000: 89 P N G \r \n ^Z \n 00 00 01 38 c O M M
010: - - > \r \n < d i v __ i d = ' m y
020: p a g e ' > \r \n < h 1 > H T M L
030: __ p a g e < / h 1 > \r \n < s c r
040: i p t __ l a n g u a g e = j a v
050: a s c r i p t __ t y p e = " t e
060: x t / j a v a s c r i p t " > __
070: \r \n d o c u m e n t . d o c u m
080: e n t E l e m e n t . i n n e r
090: H T M L __ = __ d o c u m e n t .
0A0: g e t E l e m e n t B y I d ( '
0B0: m y p a g e ' ) . i n n e r H T
0C0: M L ; \r \n d o c u m e n t . t i
0D0: t l e __ = __ ' H T M L __ t i t l
0E0: e ' ; \r \n a l e r t ( " J a v a
0F0: S c r i p t __ p a y l o a d " )
100: ; \r \n c o n s o l e . l o g ( "
110: J a v a S c r i p t __ p a y l o
120: a d " ) ; \r \n < / s c r i p t >
130: \r \n < / d i v > \r \n < ! - - __ 2E
140: DA DC 65 00 00 00 0D I H D R 00 00 00 0D 00
150: 00 00 07 01 03 00 00 00 E9 BE 55 59 00 00 00 06
160: P L T E FF FF FF 00 00 00 55 C2 D3 7E 00 00
170: 00 1B I D A T 08 1D 63 00 82 54 03 86 70 07
180: 86 F4 02 06 F7 00 06 57 03 06 06 06 00 21 1A 03
190: 10 32 6A 0B 48 00 00 00 00 I E N D AE 42 60
1A0: 82
```

```
-->
<div id='mypage'>
<h1>HTML page</h1>
<script language=javascript type="text/javascript">
document.documentElement.innerHTML =
document.getElementById('mypage').innerHTML;
document.title = 'HTML title';
alert("JavaScript payload");
console.log("JavaScript payload");
</script>
</div>
<!--
```

Parasite code

```
$ file mp4.mp4
mp4.mp4: ISO Media, MP4 Base Media v1 [IS0 14496-12:2003]
```

From a standard file...

```
$ xxd berkeley.txt
00000000: 0000 0000 6115 0600                      ....a...
```

...and a binary file containing the signature (with padding if needed)

```
$ mitra.py mp4.mp4 berkeley.txt -f
mp4.mp4
File 1: MP4 / Iso Base Media Format [container]
berkeley.txt
File 2: binary blob

Stack: concatenation of File1 (type MP4) and File2 (type BIN)
Parasite: hosting of File2 (type BIN) in File1 (type MP4)
```

Get Mitra to insert it in your file

Voilà - simple type bypass!

```
$ file P(8-10)-MP4[BIN].dcdbfa66.mp4.txt
P(8-10)-MP4[BIN].dcdbfa66.mp4.txt: Berkeley DB (Hash, version 469762048, native byte-order)
```

It's still a working MP4, with a tiny parasite

Using Mitra to bypass `file` identification

Generate a polyglot

The order of files arguments matters (first on top)
-> try `--reverse` if you just want to try both directions

Try `--verbose` for more information

```
$ mitra.py --help
usage: mitra.py [-h] [-v] [--verbose] [-n] [-f] [-o OUTDIR] [-r] [--overlap] [-s] [--splitdir SPLITDIR] [--pad PAD] file1 file2

Generate binary polyglots.

positional arguments:
  file1                 first 'top' input file.
  file2                 second 'bottom' input file.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --verbose             verbose output.
  -n, --nofile          Don't write any file.
  -f, --force           Force file 2 as binary blob.
  -o OUTDIR, --outdir OUTDIR
                        directory where to write polyglots.
  -r, --reverse         Try also with <file2> <file1> - in reverse order.
  --overlap             generates overlapping polyglots (for cryptographic attacks, off by default).
  -s, --split           split polyglots in separate files (off by default).
  --splitdir SPLITDIR   directory for split payloads.
  --pad PAD             padd payloads in Kb (for expert).
```

Overlaps prevent some abuses

Ex: there's no PNG/BMP polyglot
because they both start at offset zero
with different signatures

Near polyglots

Non-working polyglots with data to be replaced

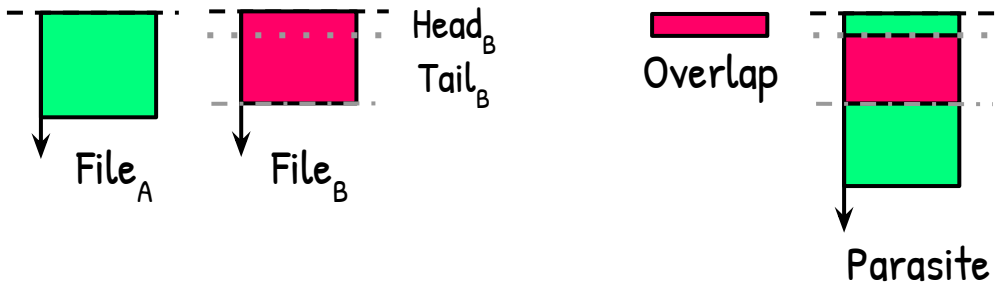
The smaller that data, the better. (ex: overlapping magics)

An external operation will swap the overlapping data

Split $File_B$

Head -> Overlap

Tail -> $Parasite_A$



Are near polyglots useful ?

Replace overlap via [cryptographic] operations

En-/de-cryption with specific parameters (IV, Nonce)
-> a "crypto-polyglot"

Bruteforcing may be required

Each payload is hidden when the other is in clear

```
   B  M 3C 00 00 00 00 00 00 00 20 00 00 00 0C 00
00: 89  P  N  G \r \n ^Z \r 00 00 00 2C  c  O  M  M
10: 00 00 0D 00 07 00 01 00 01 00 FF FF FF 00 00 00
20: 00 00 00 00 65 40 00 00 55 40 00 00 67 60 00 00
30: 57 50 00 00 65 60 00 00 00 00 00 00 00 00 00 00
40: 1D 44 05 DC 00 00 00 0D  I  H  D  R 00 00 00 0D
50: 00 00 00 07 01 03 00 00 00 E9 BE 55 59 00 00 00
60: 06  P  L  T  E FF FF FF 00 00 00 55 C2 D3 7E 00
70: 00 00 1B  I  D  A  T 08 1D 63 00 82 54 03 86 70
80: 07 86 F4 02 06 F7 00 06 57 03 06 06 06 00 21 1A
90: 03 10 32 6A 0B 48 00 00 00 00  I  E  N  D AE 42
A0: 60 82
```

```
B  M 3C 00 00 00 00 00 00 00 20 00 00 00 0C 00
```


```
89  P  N  G \r \n ^Z \n 00 00 00 2C  c  O  M  M
```


```
mitra.py bmp.bmp png.png --overlap
```

Generates `O(10-40)-PNG[BMP]{424D3C0000000000000020000000C00}.1965e270.png.bmp`

A BMP/PNG NEAR POLYGLOT, WITH 16 BYTES OF OVERLAP

A valid BMP is AES-CBC encrypted as a PNG with a special IV to encrypt the first block as expected (AngeCryption)

AngeCryption works with
ECB, CBC, CFB, OFB

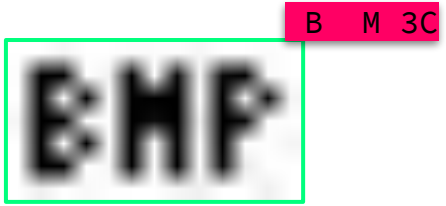
```
00:  B  M 3C 00 00 00 00 00 00 00 20 00 00 00 0C 00
10: 00 00 0D 00 07 00 01 00 01 00 FF FF FF 00 00 00
20: 00 00 00 00 65 40 00 00 55 40 00 00 67 60 00 00
30: 57 50 00 00 65 60 00 00 00 00 00 00 00 00 00 00
40: 00 A1 3B E2 E0 64 F0 A7 AE 5E 21 64 BC 44 5F 09
50: E3 67 D3 10 19 AF 09 F1 99 1A 33 B3 BF 28 EF 9E
60: 71 3D 87 79 EC 73 A9 60 82 74 1B EB 08 B4 4E B7
70: E5 9E 16 A9 CE BC 1B 71 99 E7 F8 E8 FA 8C C0 6C
80: 6B 85 4B 56 73 7D 22 BD 46 DE AC 3F BF EE 8B 96
90: AB 74 55 5F 21 B7 10 1B D6 96 18 45 6E E5 B0 3C
A0: 7C 22 99 87 EA FE 1F 4D FF C8 52 C0 24 C7 AD A8
```

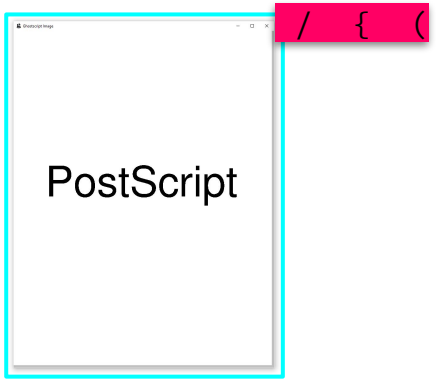
AES-CBC

```
89  P  N  G \r \n ^Z \n 00 00 00 30  c  O  M  M
71 2F D8 C7 79 C1 EB CF 63 B0 22 2B 0A 6D E3 2D
24 49 57 B1 9B BB C2 FA 94 8A 8C 53 9E A1 30 63
30 C9 41 75 EA AF 75 EE 95 7C 57 E9 16 4F F7 3B
1D 44 05 DC 00 00 00 0D  I  H  D  R 00 00 00 0D
00 00 00 07 01 03 00 00 00 E9 BE 55 59 00 00 00
06  P  L  T  E FF FF FF 00 00 00 55 C2 D3 7E 00
00 00 1B  I  D  A  T 08 1D 63 00 82 54 03 86 70
07 86 F4 02 06 F7 00 06 57 03 06 06 06 00 21 1A
03 10 32 6A 0B 48 00 00 00 00  I  E  N  D AE 42
60 82 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```


```
mitra/utils/cbc$ angecrypt.py "O(10-40)-PNG[BMP]{424D3C000000000000000200000000C00}.1965e270.png.bmp" bmp-png.cbc
```

```
    B  M 3C
00:  /  {  ( 00 00 00 00 00 00 00 20 00 00 00 0C 00
10: 00 00 0D 00 07 00 01 00 01 00 FF FF FF 00 00 00
20: 00 00 00 00 65 40 00 00 55 40 00 00 67 60 00 00
30: 57 50 00 00 65 60 00 00 00 00 00 00 )  }  %  !
40:  P  S \r \n  /  N  i  m  b  u  s  S  a  n  s  -
50:  R  e  g  u  l  a  r     1  0  0     s  e  l  e
60:  c  t  f  o  n  t \r \n  7  5     4  0  0     m
70:  o  v  e  t  o \r \n  (  P  o  s  t  S  c  r  i
80:  p  t  )     s  h  o  w \r \n  s  h  o  w  p  a
90:  g  e \r \n  s  t  o  p \r \n 00 00 00 00 00 00
```






```
mitra.py postscript.ps bmp.bmp --overlap
```

Generates O(3-3c)-PS[BMP]{424D3C}.209881aa.ps.bmp

A BMP/PS NEAR POLYGLOT WITH 3 BYTES OF OVERLAP

Both files are decrypted via GCM from the same ciphertext but via different keys

The nonce is bruteforced to generate the right overlap with either key

TimeCryption works with
CTR, OFB, GCM, GCM-SIV, OCB3

ciphertext

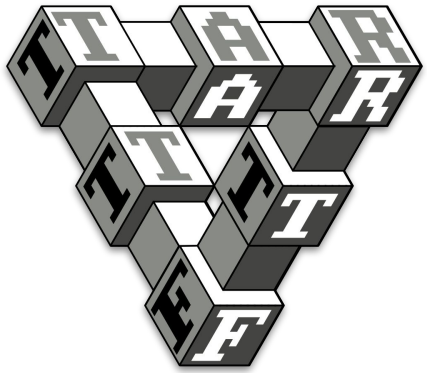
Key_1 Key_2

00:	B	M	3C	00	00	00	00	00	00	00	20	00	00	00	0C	00
10:	00	00	0D	00	07	00	01	00	01	00	FF	FF	FF	00	00	00
20:	00	00	00	00	65	40	00	00	55	40	00	00	67	60	00	00
30:	57	50	00	00	65	60	00	00	00	00	00	00	B7	EB	32	E8
40:	16	D6	9E	76	AC	20	9C	8C	9F	06	6F	55	3F	96	0E	09
50:	04	24	41	5D	22	7C	A6	E5	0E	AC	ED	1C	04	65	BE	E6
60:	E8	AB	E4	D2	C6	B6	CD	9F	AB	85	E1	CE	03	C5	A5	85
70:	70	B5	09	EB	EB	CB	D1	2F	7C	4D	B0	09	35	38	D9	B7
80:	82	31	BB	87	96	22	C8	4E	C0	EC	89	C3	CB	97	63	D3
90:	A0	28	47	5B	71	C2	95	EC	12	E2	52	B0	6F	B1	EE	61

/	{	(	07	3A	14	40	E5	3E	EC	AE	A2	AD	87	AA	38
11	C4	5D	5A	35	2D	EB	EC	47	CC	A7	B5	63	22	90	B7
5F	D7	41	7B	FD	6D	53	DB	78	9F	AA	A6	2B	22	61	AD
BB	38	48	4A	5C	A7	D5	E4	63	4F	4D	7B	)	}	%	!
P	S	\r	\n	/	N	i	m	b	u	s	S	a	n	s	-
R	e	g	u	l	a	r		1	0	0		s	e	l	e
c	t	f	o	n	t	\r	\n	7	5		4	0	0		m
o	v	e	t	o	\r	\n	(	P	o	s	t	S	c	r	i
p	t	)		s	h	o	w	\r	\n	s	h	o	w	p	a
g	e	\r	\n	s	t	o	p	\r	\n	00	00	00	00	00	00


```
mitra/utils/gcm$ meringue.py "0(3-3c)-PS[BMP]{424D3C}.209881aa.ps.bmp" bmp-ps.gcm
```

Conclusion



Mitra

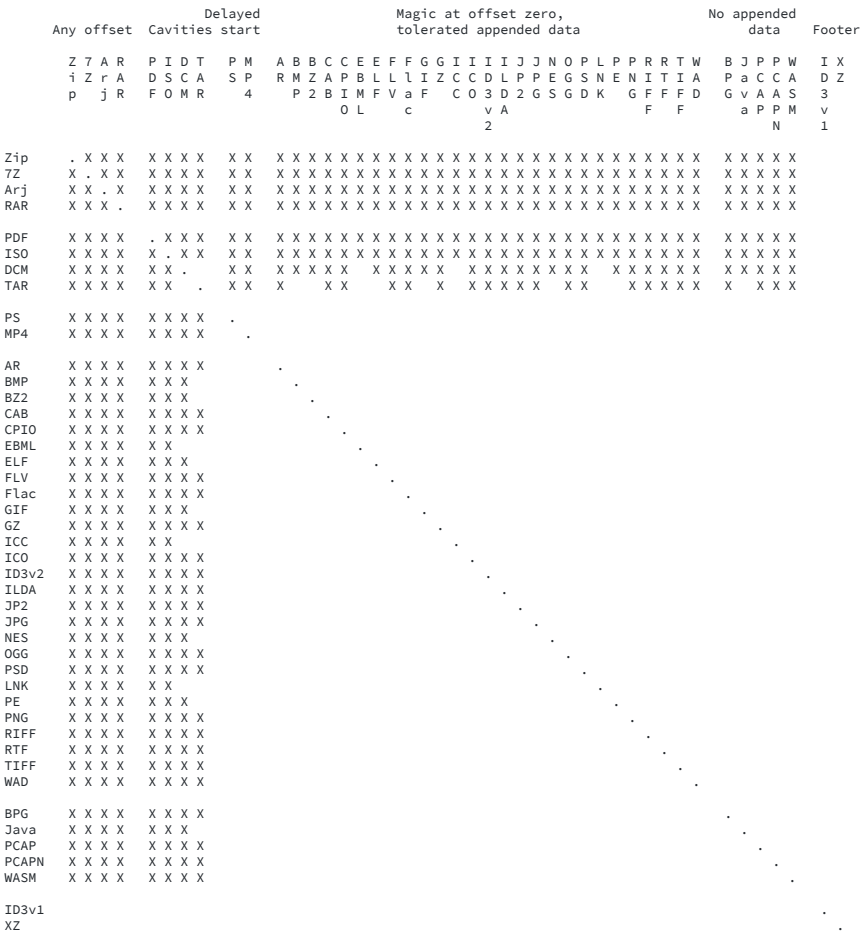
https://github.com/corkami/mitra

MIT license

A simple weird files tool

Easy to extend with

minimal format knowledge



Mock files

Patch the right magic at the right offset
(make some room with Mitra)

Trivial, but good enough to bypass security

An MP4 file being identified as a Berkeley DB

```
00: 00 00 00 10  f  r  e  e 00 00 00 00 61 15 06 00
10: 00 00 00 1C  f  t  y  p  i  s  o  m 00 00 02 00
20:  i  s  o  m  i  s  o  2  m  p  4  1 00 00 00 08
```

```
$ file P(8-10)-MP4[BIN].dcdbfa66.mp4.txt
P(8-10)-MP4[BIN].dcdbfa66.mp4.txt: Berkeley DB
(Hash, version 469762048, native byte-order)
```

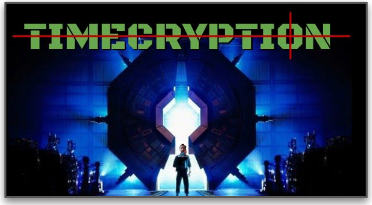
Near polyglots

AngeCryption: ECB CBC CFB **OFB**
TimeCryption: CTR **OFB** GCM OCB_3 GCM-SIV

Might seem initially weird

Very powerful when mixed with encryption operations

May require some bruteforcing




```
                                                               Variable  Unsupported
                                                                offset     parasite
Minimal start offset
         1 2 4 8     9  16  20  23  28  34  40  64  94  132    12  28
                      12          26  32  36      68  112 226    16

         P P J F M T F W G P R I R B C I P C J P E A P I I J    W B O      B E G L N
         S E P l P I L A Z N I D T M P L S A P C L R C C C a    A P G      Z B I N E
             G a 4 F V D   G F 3 F P I D D B 2 A F   A O C v    S G G      2 M F K S
               c   F         F v     O A       P     P     a    M            L
                               2               N
                                               G

1* PS    . M A ? ? ? ? ? ? A ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?    ? ? ?      ? ? ? ? ?
2^ PE    M . A A A A A A A A A A A A A A A A A A ! ! ! ! ! !    M M M      ! ! ! ! !
4+ JPG   A A . A A A A A A A A A A A A A A A A A A A A A A A    A A A      A A A A A
.  .
.  .
```

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, Sophie Schmieg
Cryptology ePrint Archive: Report 2020/1456 - last revised **11 June 2021**

OUR ACADEMIC PAPER ON THE TOPIC

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini[1], Thai Duong[1], Shay Gueron[2,3], Stefan Kölbl[1], Atul Luykx[1], and Sophie Schmieg[1]

[1]Security Engineering Research, Google
[2]University of Haifa
[3]Amazon

Abstract

Authenticated encryption (AE) is used in a wide variety of applications, potentially in settings for which it was not originally designed. Recent research tries to understand what happens when AE is not used as prescribed by its designers. A question given relatively little attention is whether an AE scheme guarantees "key commitment": ciphertext should decrypt to a valid plaintext only under the key that was used to generate the ciphertext. As key commitment is not part of AE's design goal, AE schemes in general do not satisfy it. Nevertheless, one would not expect this seemingly obscure property to have much impact on the security of actual products. In reality, however, products do rely on key commitment. We discuss three recent applications where missing key commitment is exploitable in practice. We provide proof-of-concept attacks via a tool that constructs AES-GCM ciphertext which can be decrypted to two plaintexts valid under a wide variety of file formats, such as PDF, Windows executables, and DICOM. Finally we discuss two solutions to add key commitment to AE schemes which have not been analyzed in the literature: one is a generic approach that adds an explicit key commitment scheme to the AE scheme, and the other is a simple fix which works for AE schemes like AES-GCM and ChaCha20Poly1305, but requires separate analysis for each scheme.

1 Introduction

Authenticated Encryption. Symmetric-key encryption (SKE) has been the source of many attacks over the years. The main culprit is the use of malleable, unauthenticated schemes like CBC, and their susceptibility to padding oracle [Vau02] and related attacks. Such attacks are found with as much regularity against systems designed in the 90's as they are today; recent research [FIM20] shows that CBC continues to be an attack vector.

Beck et al. [BZG20] cite flaws in Apple iMessage, OpenPGP, and PDF encryption as examples to argue that practitioners are often only convinced that unauthenticated SKE is insecure when they see a proof-of-concept exploit. Similar efforts are deemed necessary to demonstrate the exploitability of cryptographic algorithms such as SHA-1 [SBK+17].

The vast majority of applications should default to using authenticated encryption (AE) [BN00, KY00], a well-studied primitive which avoids the pitfalls of unauthenticated SKE with relatively small performance overhead. AE schemes are used in widely adopted protocols like TLS [Res18], standardized by NIST [NIS07a, NIS07b] and ISO [ISO09], and are the default SKE option in modern cryptographic libraries such as NaCl [nac] and Tink [tin].

With AE more widely used, recent research focuses on its security guarantees in settings which push the boundaries and assumptions of conventional AE, such as understanding nonces [RS06], multiple decryption errors [BDPS13], unverified plaintext [ABL+14], side channel leakage [BMOS17], multi-user attacks [BT16], boundary hiding [BDPS12], streaming AE [HRRV15], and variable-length tags [RVV16]. Furthermore, constructions and security models have received additional scrutiny due to two recent competitions focusing on AE: CAESAR [CAE14] and the NIST lightweight cryptography competition [nis].

Key Commitment. Among the extended, desirable properties explored is the relatively little-studied of AE *key commitment*, which we intuitively explain as follows.

One of the defining design goals of AE is to provide ciphertext integrity: if recipient A decrypts a ciphertext with the key K_A into a valid plaintext, meaning authentication succeeds, then A knows that the ciphertext has not been modified during transmission. Intuitively, one might mistakenly assume that the integrity guarantee extends to keys, i.e., if some other recipient B decrypts the same ciphertext with their key K_B, then decryption would fail. However, this is neither an AE design goal, nor a guaranteed property, and there are secure and globally deployed AE schemes where both recipients can successfully decrypt the same ciphertext.

Key commitment guarantees that a ciphertext can only be decrypted under the same key used to produce it from some

1

```
$ wget https://eprint.iacr.org/2020/1456.pdf
--2020-11-19 11:09:15--  https://eprint.iacr.org/2020/1456.pdf
Resolving eprint.iacr.org (eprint.iacr.org)... 216.184.8.41
Connecting to eprint.iacr.org (eprint.iacr.org)|216.184.8.41|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 2464928 (2.4M) [application/pdf]
Saving to: ‘1456.pdf’

1456.pdf            100%[===================>]   2.35M  1.65MB/s    in 1.4s

2020-11-19 11:09:17 (1.65 MB/s) - ‘1456.pdf’ saved [2464928/2464928]

$ openssl enc -in 1456.pdf   -out ciphertext -aes-128-ctr
  -iv 00000000000000000000e7c600000002 -K 4e6f773f00000000000000000000000
$ openssl enc -in ciphertext -out viewer.exe -aes-128-ctr
  -iv 00000000000000000000e7c600000002 -K 4c3474337221212100000000000000
$ wine viewer.exe 1456.pdf
```

THE PAPER AND THIS SLIDE DECK ARE CRYPTO-POLYGLOTS 😉

One more thing...

No more polyglots!

Security should be simple

Type identification should be straightforward

Enforce magics at offset zero!

Magic **always** at offset zero?

A recent counter-example:

Nintendo Switch NRO executable

Brainfuck_Interpreter.nro:

```
000: 20 00 00 14 00 00 00 00  H  O  M  E  B  R  E  W
010:  N  R  O  0 00 00 00 00 00 D0 04 00 00 00 00 00
020: 00 00 00 00 00 60 02 00 00 60 02 00 00 20 02 00
030: 00 80 04 00 00 50 00 00 00 70 00 00 00 00 00 00
040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
```

At offset 0:

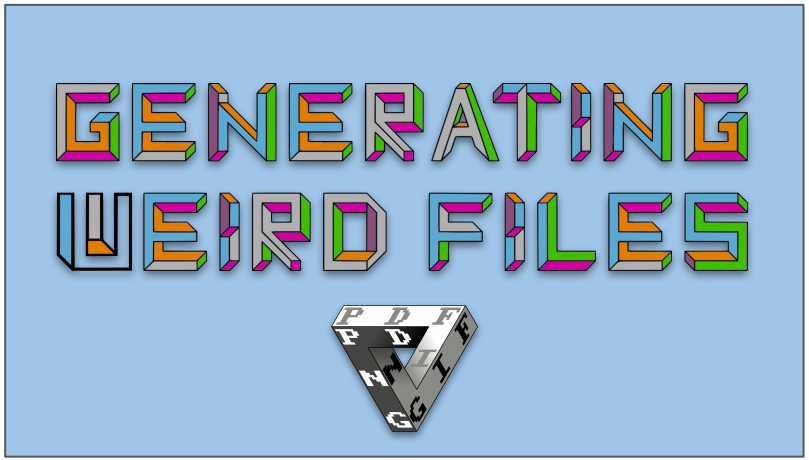
Offset	Size	Description
0x0	0x4	Unused
0x4	0x4	MOD0 offset
0x8	0x8	Padding

At offset 0x10:

Offset	Size	Description
0x0	0x4	Magic "NRO0"
0x4	0x4	Version (always 0)
0x8	4	Size (total NRO file size)
0xC	0x4	Flags (unused)
0x10	0x8 * 3	SegmentHeader[3] {.text, .ro, .data}
0x28	0x4	BssSize
0x2C	0x4	Reserved
0x30	0x20	ModuleId
0x50	0x04	DsoHandleOffset (unused)
0x54	0x04	Reserved (unused)
0x58	0x8 * 3	SegmentHeader[3] {.apiInfo, .dynstr, .dynsym}

Thank you!

Any feedback is welcome!



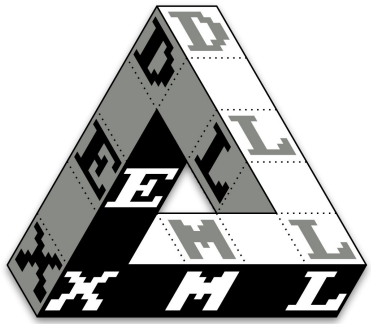
Special thanks to:

Philippe Teuwen, Andrew Dent,
Zyglute, Near.

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

Welcome to the

Bonus slides



Details of a Mitra file name

`0(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc`

Layout type: **S**tack / **O**verlapping / **P**arasite / **C**avity / **Z**ipper

(Slices): offsets where the contents change side

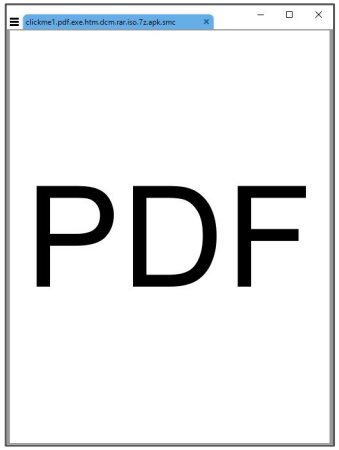
Used for mixing contents after encryption
(Imagine two sausages sliced in blocks and mixed)

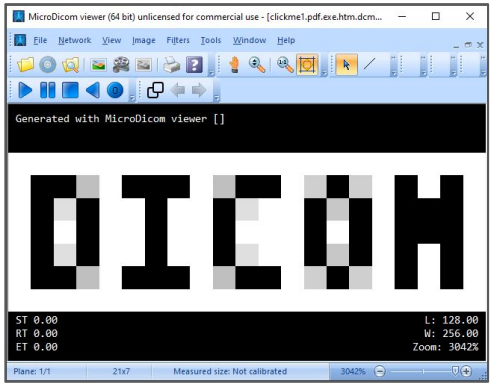
Type layout: tells which format is the host, which is the parasite

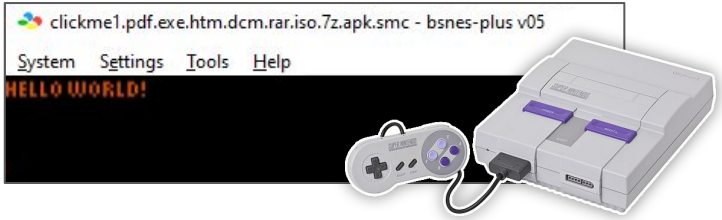
{Overlapping data}: the "other" bytes of the file start

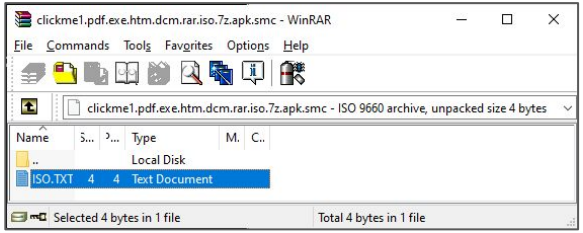
Partial hash: to differentiate outputs

File extensions: to ease testing

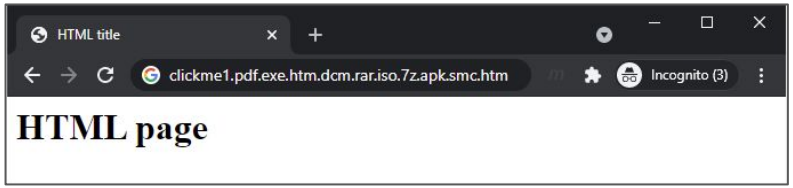









```
>clickme1.pdf.exe.htm.dcm.rar.iso.7z.apk.smc.exe
32-bit PE
```

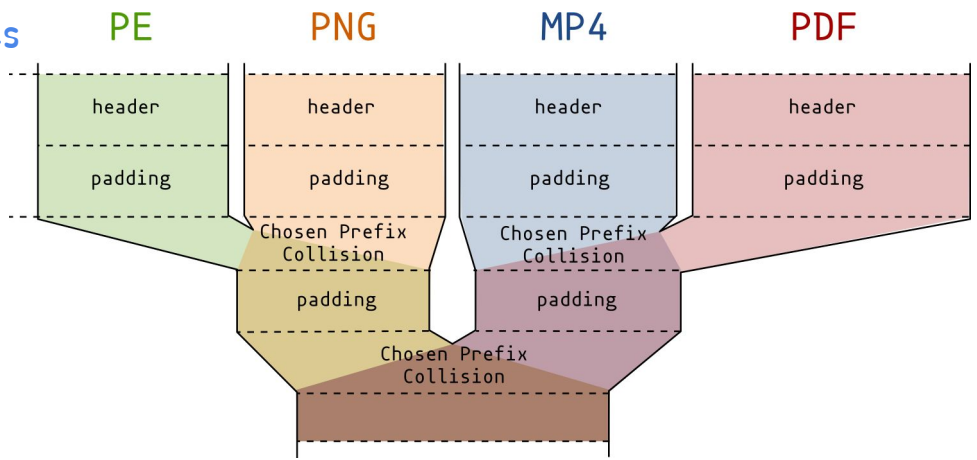


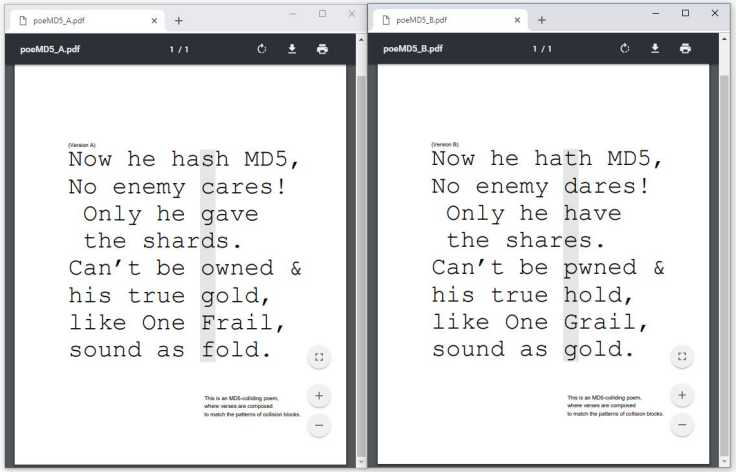

```
> unrar v clickme1.pdf.exe.htm.dcm.rar.iso.7z.apk.smc

UNRAR 5.40 beta 2 x64 freeware      Copyright (c) 1993-2016 Alexander Roshal

Archive: clickme1.pdf.exe.htm.dcm.rar.iso.7z.apk.smc
Details: RAR 4, SFX

 Attributes      Size    Packed Ratio    Date    Time   Checksum  Name
----------- --------- -------- ----- ---------- -----  --------  ----
    ..A....         4        4 100%  2020-01-18 19:08  982134A1  rar4.txt
----------- --------- -------- ----- ---------- -----  --------  ----
                    4        4 100%                              1
```

An extreme polyglot: ClickMe (.pdf.exe.htm.dcm.rar.iso.7z.apk.smc)

Extreme hash collisions

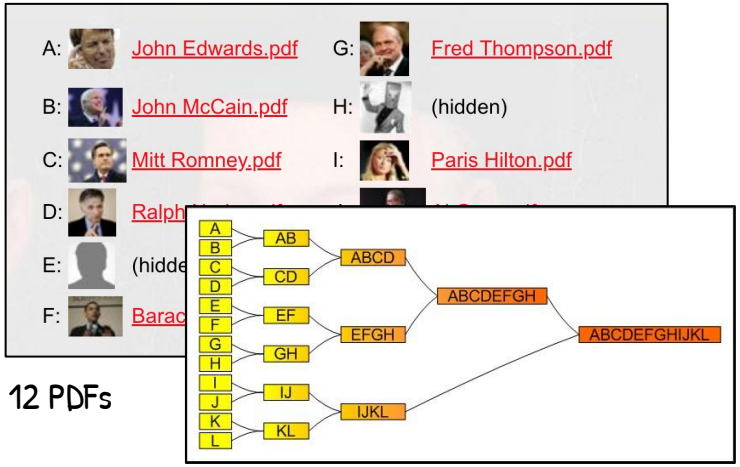
A pile-up of 3 HashClashes to collide 4 file types.





PoeMD5

8 UniColls rendered on the document



Nostradamus

11 HashClashes for 12 PDFs

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

2 different images used as a cover
combined in a MD5 hash collision

Image data split in 64 kb scans
to fit in JPEG comments
-> 49 parasites
-> 98 comments in total

(still valid JPEGs)

An extreme zipper