How to better identify (weaponized) file formats

Philippe Lagadec – Hack.lu 2025

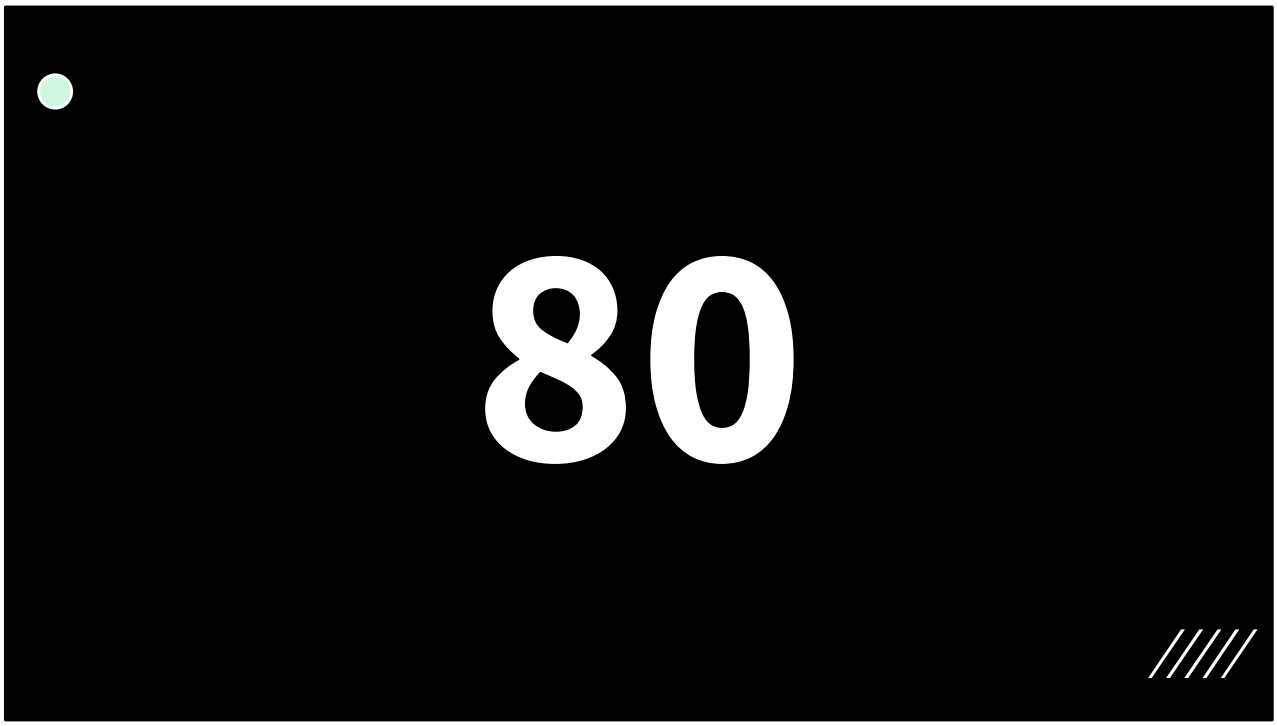
Why do we need to identify file formats accurately?

Why are the current solutions we use sometimes wrong?

And how can we do better?

I am Philippe Lagadec, and in the next 25 minutes I will cover those questions.
But for now, imagine you are in a game

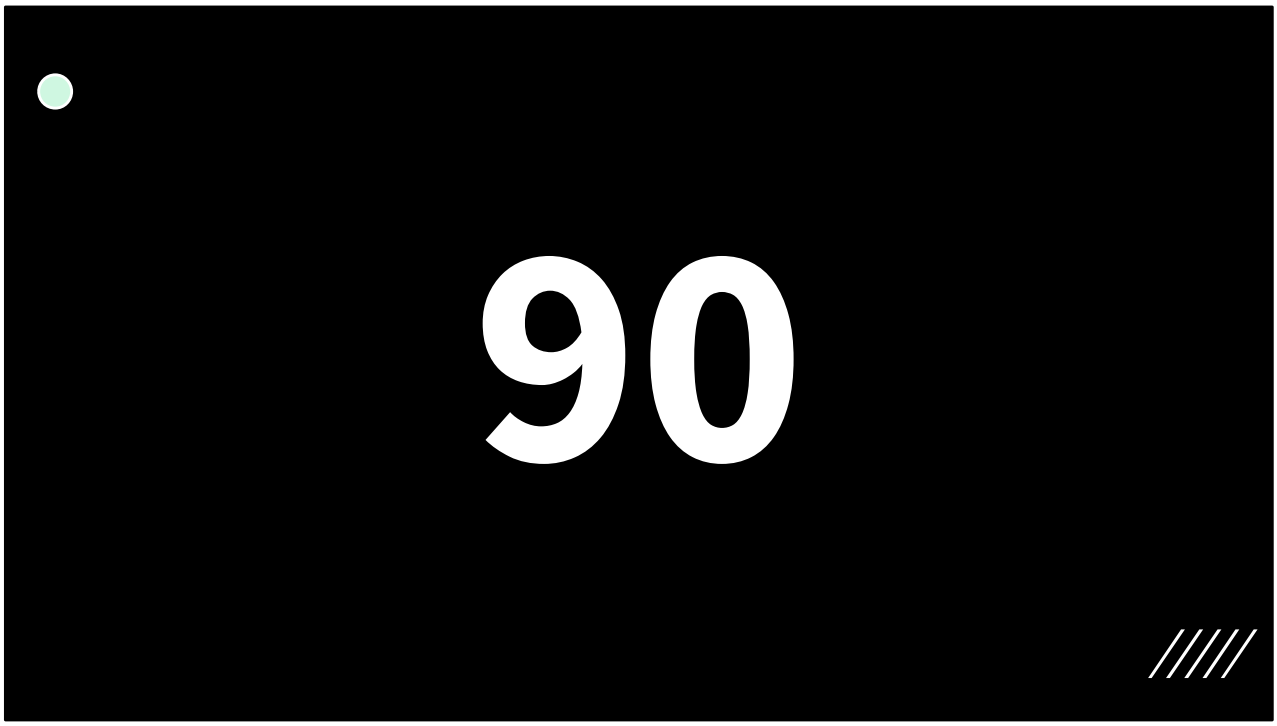
Imagine you are playing a game:
There is a person you don’t know in front of you, and your goal is to **guess their nationality**.
For this you can either **ask questions**, or check their **passport**.
Let’s pick someone randomly in the audience:
Alex?
Your accent sounds a bit like mine, so I guess you speak French.
You could probably be **French, Belgian or Swiss.**



How do you pronounce that number?

- **“quatre-vingt”**

I can tell you’re not Swiss otherwise you would have said **‘octante’**



How do you pronounce that one?

- “**Nonante**”

Ah! Then I can tell you’re probably Belgian, a French would have said “**quatre-vingt-dix”**.

And that one?

- **“Ouite”**

Then you’re definitely Belgian, a French would have said **“Uuite”**!
Thank you Alex!

Asking questions like these, I can guess the nationality of someone.
But some nationalities are harder to distinguish than others.
And if Alex had been a true polyglot (no pun intended ;-), it could have been much trickier.

But remember, in the game, we can also **check the passport**!
Then it would be much easier and more accurate, right?

Or even better, I can both check the passport AND ask questions to confirm the nationality.

OK, let's keep that in mind for later.

Quiz: What is the true format of this file?

```
goto BAT
<HTML><script>alert("I am HTML.");</script></HTML>
%PDF-1.
1 0 obj<</Pages 2 0 R>>endobj
2 0 obj<</Kids[3 0 R]/Count 1>>endobj
3 0 obj<</Parent 2 0 R /Contents 4 0 R>>endobj
4 0 obj<<>>
Stream
BT 9 Tf(I am PDF.)' ET endstream
endobj
trailer <</Root 1 0 R>>
:BAT
echo I am BAT.
```

Now a little quiz:
Who can tell me the true format of this file?
…
If you tell me it's a batch file, you're right.
If you tell me HTML, you're right too.
And PDF? You're also right!

In fact we are missing a key piece of information here to be able to answer for sure:
What is it?
The filename!
(and especially its **extension**)

How operating systems open a file

Do you know how all operating systems decide how to open a file?
Well, at least **Windows, macOS and even Linux use the file extension** to choose which application to launch when you double-click on a file.

Our small example is actually a **polyglot**.
A file which combines several formats in one.
Let's try to open that file when we change the extension.
DEMO:
- poly.bat opens in the text editor
- poly.html opens in the web browser (and runs javascript code)
- poly.pdf opens in the PDF viewer
- But all 3 are the same file, just renamed with a different extension.

Here it's a Linux system with XFCE, but it would be the same with GNOME or KDE.

This is the normal behaviour of xdg-open, from the freedesktop.org specifications:
https://www.freedesktop.org/wiki/Software/xdg-utils/
https://www.freedesktop.org/wiki/Specifications/shared-mime-info-spec/

Why do we need to identify file formats?

In cybersecurity and IT in general, there are many cases where we need to know the type of a file, in order to decide what to do with it.
This has a security impact especially in forensics, incident response, malware analysis and detection.
In malware detection and analysis (which is my main job currently), we need to identify the true file type, because our platform will then launch the scanning engines that are relevant for the file type. If it's a PDF, we launch the PDF scanner. If it's PowerShell or JavaScript, we launch our detection engine based on machine learning, etc.
But if we get it wrong, a malicious file might not be detected and go through.
Or the opposite, a legitimate file can be blocked by mistake.

This is also the same on many platforms, for example Gmail when scanning emails for malware.

How do we identify file formats?

In most cases, people rely on the **file tool or libmagic** to identify the format of a file.
File/libmagic have been around for decades.
There are some alternative tools, such as TrID or YARA rules.
And more recently, people have started to use **magika**, published by Google a couple years ago.

Who has already used file, libmagic or magika?

How does file/libmagic work?

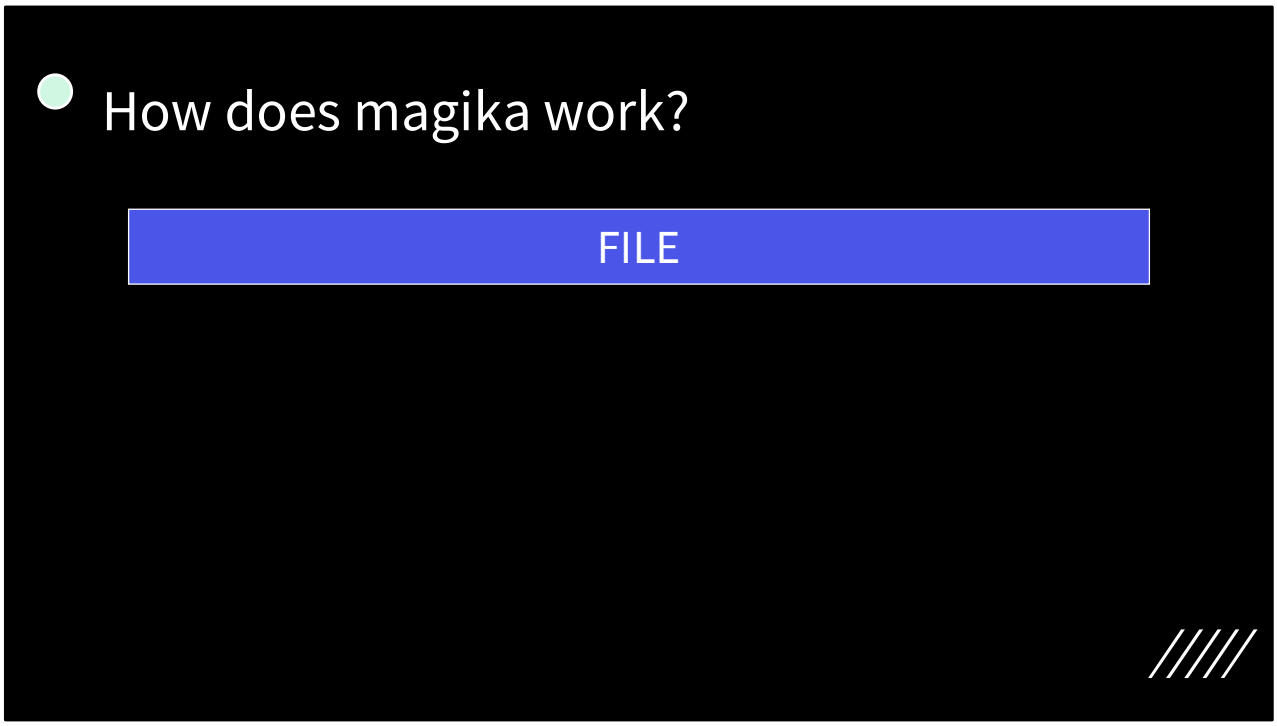
Libmagic, which is the engine used by the file tool, uses **thousands of rules** to scan file contents and identify file formats:

- Specific bytes at fixed locations
- Specific strings
- Regex

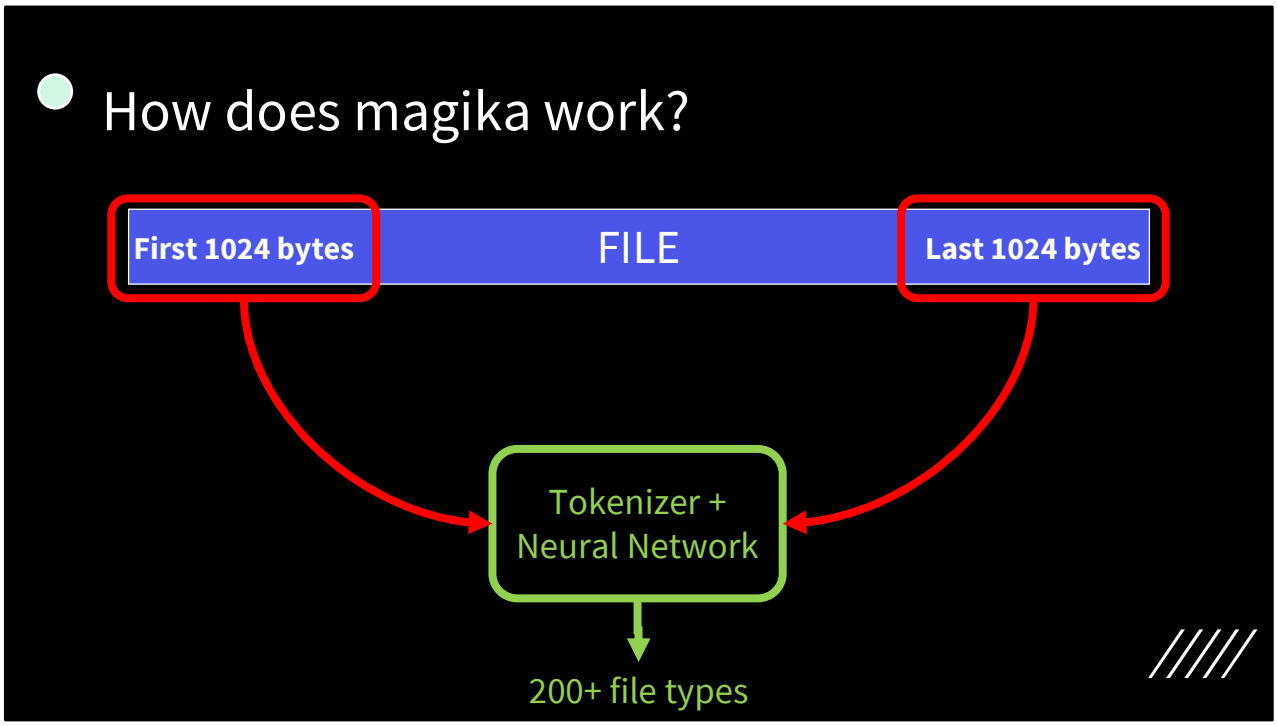
This works great for structured/binary formats.

Not so much for unstructured/text formats:
scripts, source, XML, HTML, …

How does magika work?



Magika is the new kid on the block for file format identification, published by Google a couple years ago.
Compared to libmagic it uses machine learning instead of predefined rules.



Magika takes the first and the last kilobyte of the file,
processes them with a tokenizer and a neural network,
and then identifies the most probable file format within 200 known file types.

Magika strengths

Better classification (F1 score) than file/libmagic, TrID and guesslang on the 200+ supported formats

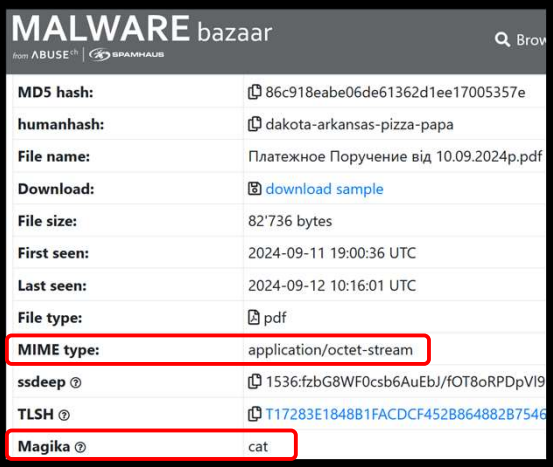
- 4% better on binary formats
- 22% better on text formats

Especially useful to **recognize unstructured text formats**:

- Libmagic is unable to identify scripts
- Magika is way better

Downside: only 200 supported formats – thousands for libmagic!
=> best approach: combination of libmagic + magika

Example: **PDF file** on MalwareBazaar



$ file sample.pdf
sample.pdf: data

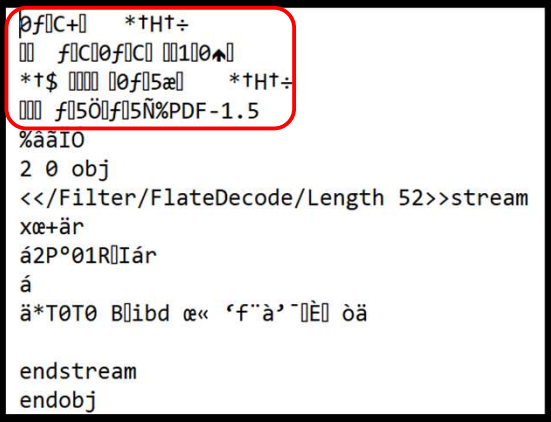
$ file -i sample.pdf
sample.pdf: application/octet-stream; charset=binary

$ magika sample.pdf
sample.pdf: Certificates (binary format) (text)

Most of the time, libmagic and magika give really good results.
But sometimes, it does not work as expected…

For example, here is a malicious PDF file found on MalwareBazaar:
It is really a PDF file, but libmagic says it's unknown binary data, and magika says it's a certificate!

Example: **PDF file** on MalwareBazaar

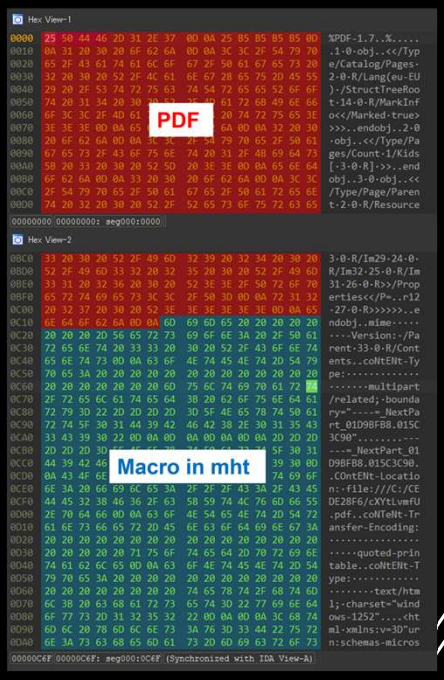


- A few random bytes before the %PDF header

Why on Earth is it not recognized?
In fact this file is exploiting something weird in the PDF format:
It’s not in the specifications, but **almost all PDF viewers do accept up to 1 KB of random data before the actual PDF header.**
This has been the case since the birth of PDF in the nineties, when web browsers did not know how to open a PDF file.
So it was possible to insert a small HTML web page before the PDF header to explain how to download a viewer.
Today it is still an effective technique to bypass many PDF filters.

Other examples

- PDF/MHT reported by JPCERT in 2023
- File/libmagic => PDF
- But the file has the .doc extension, opens in MS Word
  - Malicious VBA Macro
- https://blogs.jpcert.or.jp/en/2023/08/maldocinpdf.html

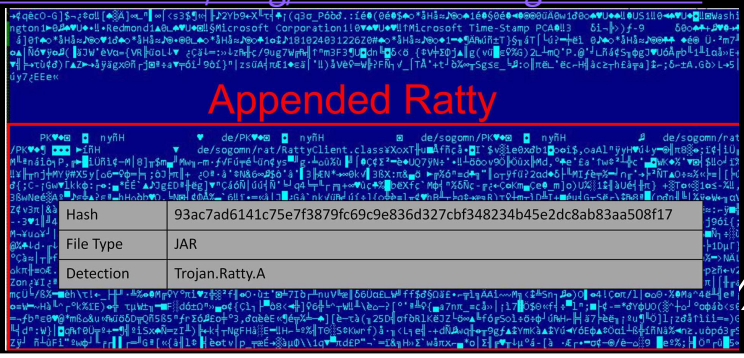


Here is another case of malware reported by the Japanese CERT in 2023:
The file is identified as PDF by libmagic because it starts by a PDF header.

But the file has the .doc extension, so it opens in Word.
And this is actually a Word document in MTHML format, which runs a malicious VBA macro!

Other examples

- Signed MSI + malicious JAR in 2020
  - File/libmagic => MSI
  - Magika => sometimes JAR, sometimes MSI
- https://www.gdatasoftware.com/blog/how-malware-gets-a-free-pass



A third example from 2020:
This time it's a legitimate MSI file (Windows Installer package), followed by a JAR file.
Libmagic and magika see it as MSI,
But it has the .JAR extension so it runs the malicious Java code.

Hard case: identifying script formats

- PowerShell, JavaScript, BAT, shell scripts , …
  - very hard to identify by content only

- **PowerSheLLM** : PowerShell malware detection model
  - https://github.com/glimps-re/PowersheLLM

PowerShell, JavaScript, BAT, shell scripts , ...
very hard to identify by content only

Example: PowerSheLLM, a malware detector published by GLIMPS

Works great on real PowerShell, but in practice lots of files are incorrectly tagged as PS1 by Magika

Much more common problem than actual polyglots!

A story of BASE64

- `$ cat ALERT.MSG`
- `IAAhAEIAQQBTAEUANgA0AEUARAAhACAAQQBwAGUAeAAgAE8AbgBlACAAYQAgAGQA6QB0AGUAYwB0AOkAIAB1AG4AIABvAHUAIABwAGwAdQBzAGkAZQB1AHIAcwAgAHYAaQByAHUAcwAvAHAA[…]BuAHMALgAAAA==`

- `$ magika ALERT.MSG`
- `ALERT.MSG:` **`Powershell source (code)`**

One day a colleague sent me a false positive, a legitimate file that was blocked by our PowerShell detection engine

Here is the content of the file: it's actually a simple text message encoded in BASE64!

It turns out that Magika must have learned that PowerShell scripts use a lot of BASE64 encoding.
Then some BASE64 files are mistakenly identified as PowerShell.

Magika weaknesses

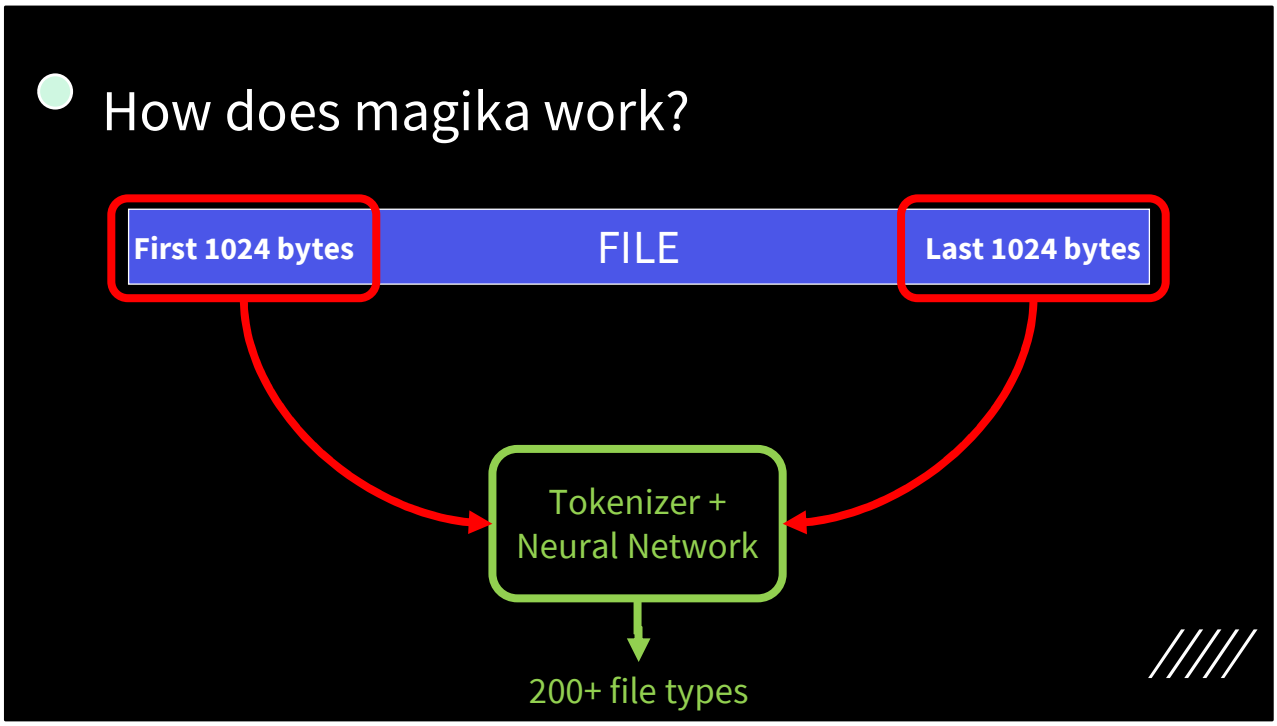
- Recent Google study (Oct 2025): https://arxiv.org/abs/2510.01676

*"by **changing just 13 bytes** of a malware sample, we can successfully **evade Magika in 90% of cases** and thereby allow us to send malware files over Gmail."*

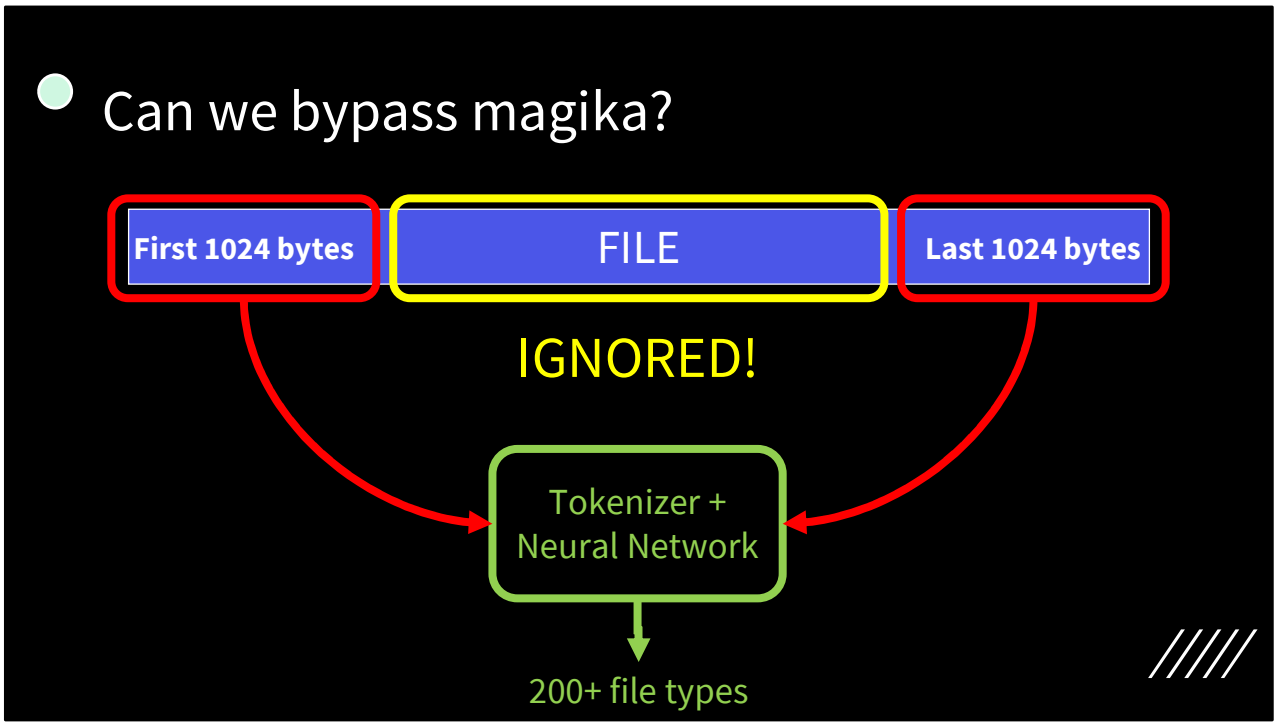
Recent Google study (Oct 2025): https://arxiv.org/abs/2510.01676
Researchers have used adversarial machine learning techniques to attempt to fool magika. (and succeeded)

*"by **changing just 13 bytes** of a malware sample, we can successfully **evade Magika in 90% of cases** and thereby allow us to send malware files over Gmail."*

But wait, there's more...



Remember I told you magika takes the first and the last kilobytes of data?



So what happens to the rest of the file? It is simply IGNORED! By design.
A neural network has a fixed input size, so it is not possible to feed it with the whole file.

The other day I was thinking:
What happens if I take a PowerShell script or JavaScript, and if I add a few KB of whitespaces before and after the script?
What do you think?

To test it I took more than 1000 powershell scripts and javascript files from MalwareBazaar.
In every case, magika told me those were text files!
(DEMO)

So this is a very simple way to bypass magika for all kinds of script files.
This is not a vulnerability, it is magika's core design.

NOTE: I will soon publish an add-on for magika to mitigate this kind of bypass.

Recap: Why are the current solutions we use sometimes wrong?

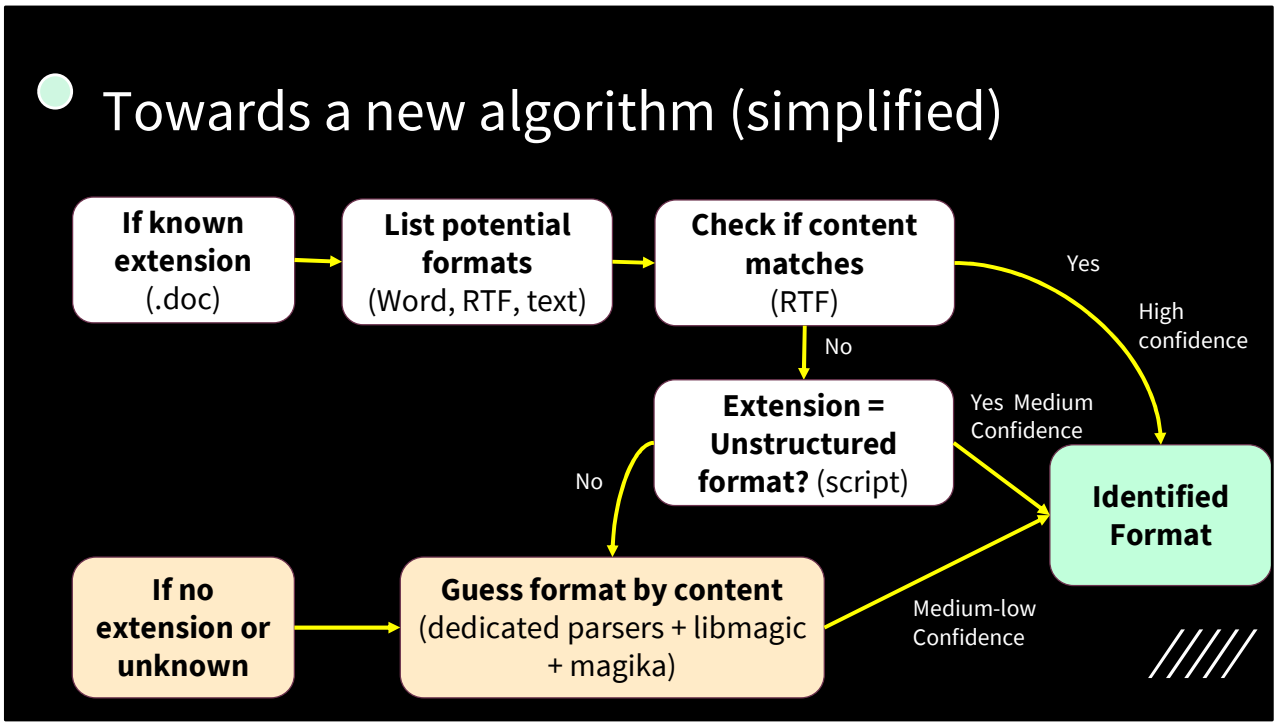
- 1. Slightly malformed files exploiting **parsers flexibility**
- 2. **Polyglot files**, combining several formats in one
- 3. Unstructured, **hard to identify formats** such as scripts
- 4. Magika bypass tricks
- *Fundamental flaw: guessing by content only*

All those issues are due to the same fundamental flaw:

Libmagic and magika are only trying to guess by looking at the content of a file.
(and this is a hard problem)

How can we do better?

Remember the game “guess the nationality” we played at the beginning?
Well we are in a quite similar situation:
In the game we could ask questions, or check the passport. If you can check the passport it’s much better, right?
Tools like libmagic and magika only look at file contents.
They ignore file extensions.
File extension is like the passport of a file.
Extensions don’t lie, because that’s what operating systems check to open a file.
So how can we design a better algorithm, checking both the extension and the contents of a file?



Here is an experimental algorithm that I started to implement in my tool ftguess:

1. Starting with the extension, we identify the possible formats.
   For example, a .DOC file could be a Word document, an RTF file, or a text file.
2. For each of these formats, we use a small dedicated parser (or simple magic) to check whether the content matches what is expected.
   If so, then we have found the right format, with high confidence.
3. If not, then we check if the extension is for an unstructured format, such as a script.
   If so, we select that format only based on the extension, with medium confidence.
4. If we haven't found it or if the file has no extension, then we switch to classic detection based on content alone. We can use a combination of libmagic and magika.

This algorithm is more accurate than just looking at file contents, even if it does not address all issues.

Ftguess

- Disclaimer: this is experimental, work in progress!

- Origin: a tool in oletools to better identify MS Office formats with precise parsing
- Now implementing this new algorithm, supporting many formats

- To test ftguess :
- pipx install git+https://github.com/decalage2/oletools.git@feat/ftguess-strict-mode
- ftguess -s <file>

Ftguess is originally a tool in my project oletools that was meant to identify MS Office documents more precisely.
I have now extended it to support many more formats, and to implement the new algorithm that I just presented.

Give it a try with the commands shown on the slides, and send me your feedback.

Ftguess on malformed PDF

```
$ ftguess –s sample.pdf
ftguess 0.61.dev1 on Python 3.9.0 - http://decalage.info/python/oletools

File      : sample.pdf
File extension: pdf
File Type  : PDF
Description: PDF - Portable Document Format (.pdf)
Application: Any PDF viewer (Adobe Reader, Acrobat, Firefox, Chrome, etc)
Container  : PDF
Content-type(s) : application/pdf
PUID      : fmt/276
File type identification method: strict mode (file extension and content)
```

Here is ftguess analysing the slightly malformed PDF file that I showed earlier, with random data inserted before the PDF header.

Ftguess – future work

- Implement the full algorithm
- Test on large datasets, with all tricky cases
- Go version for better performance

- Proposal for malware handling: file.ext.infected

Very often malware analysts have the habit of renaming suspicious files to remove their extension, so that malware cannot be run by accident.
Since we do need the original file extension to better identify file formats, I propose an alternate solution:
If you **renamed a file by just appending ".infected" at the end**, tools such as ftguess could extract the original extension and give more accurate results.

Key takeaways

- **We need to identify file formats more accurately**
- **File extensions don’t lie** – Do not rename suspicious samples!
- Current tools (libmagic, magika) **only look at file contents**
  - Sometimes it’s just wrong
  - **Easy bypass tricks** for script formats
- We can do better by **checking both file extension and contents**

Contact

- **Philippe Lagadec**
- https://linktr.ee/decalage => GitHub, X, Mastodon, Bsky
- https://github.com/decalage2/oletools

Contact me:
- https://linktr.ee/decalage
- https://x.com/decalage2
- https://mastodon.social/@decalage
- https://bsky.app/profile/decalage.bsky.social
- https://github.com/decalage2
- https://decalage.info/