EVERYTHING STAYS LOCAL

Document malware, and sanitising it

August 21, 2026

Matthew J. Harmon

THE SHAPE OF THE PROBLEM

A document is not a picture of information. It is a **container format** with an execution surface bolted on over thirty years.

What users think they receive

- text and pictures
- a thing to read
- inert

What actually arrives

- a ZIP archive, or an OLE2 (Object Linking and Embedding) filesystem
- scripting engines, event hooks
- outbound network references
- other files, whole and intact

WHY THIS IS STILL THE DELIVERY VEHICLE OF CHOICE

A document is the one executable payload a person is expected to open as part of their job, from a stranger, without suspicion. The CV. The invoice. The court filing. Refusing the attachment is refusing the work.

METHODS

LEGACY CONTAINERS AND THINGS CARRIED INSIDE

OLE2 / CFB (Compound File Binary) — `.doc`, `.xls`, `.ppt`

A small filesystem in a file: storages, streams, a FAT (File Allocation Table).

- `_VBA_PROJECT`, `Macros`, `PROJECT`
- `ObjectPool` — embedded objects
- `\{}x01Ole10Native` — the Packager stream: an arbitrary file, with its original name, opened on double-click

Magic bytes worth finding anywhere

- `4D 5A` — PE (Portable Executable) **HIGH**
- `D0 CF 11 E0` — OLE2 inside something else **HIGH**
- `50 4B 03 04` — nested ZIP / OOXML **HIGH**
- `25 50 44 46` — PDF inside a document **HIGH**
- `FWS` / `CWS` — Flash **HIGH**

DEMONSTRATION: IT IS ALREADY A ZIP

A document from the machine in front of you, not a fixture

```
$ cp ~/Documents/report.docx /tmp/report.zip
$ unzip -l /tmp/report.zip | head -7
  Length      Date    Time    Name
---------  ---------- -----   ----
     1417  1980-01-01 00:00   [Content_Types].xml
      590  1980-01-01 00:00   _rels/.rels
    24911  1980-01-01 00:00   word/document.xml
     8365  1980-01-01 00:00   word/theme/theme1.xml
```

FIVE FAMILIES, ONE PAGE

Family	Mechanism	Artifact it leaves
Macro code	VBA (Visual Basic for Applications) runs on open or on event	`vbaProject.bin`, `Basic/`
Embedded object	A second file carried inside the first	`embeddings/`, magic bytes
Remote fetch	Document reaches out when opened	external relationship targets
Reader features	The viewer's own scripting and actions	`/OpenAction`, `/JS`, `/Launch`
Legacy container	Pre-XML formats with richer object models	OLE2 streams, `Ole10Native`

Everything that follows is one of these five wearing a different hat. **The artifact column is the whole basis of detection** — a sanitiser never sees intent, only structure.

DEMONSTRATION: ONE FIXTURE PER FAMILY

The whole demo set, and the single command that exposes each

Fixture	Family	The one command
`sample.docm`	macro code	‘unzip -l sample.docm
`carrier.docx`	embedded object	‘unzip -p carrier.docx word/embeddings/o1.bin
`invite.docx`	remote fetch	`unzip -p invite.docx word/_rels/settings.xml.rels`
`actions.pdf`	reader features	`pdfid.py actions.pdf`
`invoice.doc`	legacy container	`oledump.py invoice.doc`
`clean.docx`	—	the control. Every demo set needs one.

THESE ARE FIXTURES, NOT SAMPLES

Nothing here is live malware, and it does not need to be. Every macro body is a `MsgBox`, every remote target is loopback, and the carried “executable” is a valid PE header followed by nothing. **A sanitiser never sees intent** — so a fixture that only has the structure exercises every path the real thing would.

OOXML: IT IS A ZIP FILE

Office Open XML — `.docx`, `.xlsx`, `.pptx`

These are ZIP archives of XML. Rename one to `.zip` and it opens. So does everything inside it.

Executable content

- `word/vbaProject.bin` **HIGH**
  compiled VBA; the classic
- `word/activeX/` **HIGH**
  COM (Component Object Model) controls, instantiated on load
- `xl/queryTables/`, `xl/connections/` **REVIEW**
  data connections that fetch on refresh

Carried content

- `word/embeddings/` **HIGH**
  arbitrary files, often OLE2 droppers
- `customXml/`, `docProps/custom.xml` **REVIEW**
  a place to hide data that survives edits
- `*/media/*` with non-image magic **HIGH**
  a ”picture” that is not one

The macro is the famous one. The other rows are the ones that still work when macros are disabled by policy.

DEMONSTRATION: THE PICTURE THAT IS NOT ONE

`*/media/*` with non-image magic headers — the row nobody expects

```
$ unzip -l sample.docm | grep -E 'vbaProject|media'
    12288  1980-01-01 00:00   word/vbaProject.bin
     4096  1980-01-01 00:00   word/media/image1.png
$ unzip -p sample.docm word/media/image1.png | head -c 8 | xxd
00000000: 4d5a 9000 0300 0000                      MZ......
```

The extension says PNG. The first two bytes say `MZ`, and Word never looks — it is not asked to render this part, only to carry it. **HIGH**

The same command against `clean.docx` returns `89 50 4e 47`. Two bytes, one comparison — and nothing else in the pipeline was making it.

OOXML: THE RELATIONSHIP GRAPH

Every part of an OOXML document is wired to others through `.rels` files. Those relationships can point **outside the document**.

- `TargetMode="External"` — fetch on open
- `attachedTemplate` — **remote template injection**: the document is empty and harmless; the *template* it loads is not
- `oleObject`, `frame`, `hyperlink` — each a documented path to somewhere else
- `docProps/app.xml` `<Template>` pointing at a UNC (Universal Naming Convention) path — `\{}\{}host\{}share` leaks credentials on open, before any content is rendered

THE LESSON THAT SHAPED THE TOOL

A file can be malicious while containing no malicious content. Scanning the bytes of the document tells you nothing; the payload is at the other end of a URL, fetched at open time, and it can differ per recipient.

DEMONSTRATION: A DOCUMENT WITH NO PAYLOAD IN IT

Remote template injection, against loopback

```
$ unzip -p invite.docx word/_rels/settings.xml.rels
<Relationship Id="rId1" TargetMode="External"
  Type=".../officeDocument/2006/relationships/attachedTemplate"
  Target="http://127.0.0.1:8000/template.dotm"/>
$ grep -Eaic 'shell|autoopen|powershell' invite.docx
0
```

Now serve the loopback address and open the document.

```
$ python3 -m http.server 8000
127.0.0.1 - - [17/Aug/2026 09:14:02] "GET /template.dotm" 404 -
```

The 404 *is* the demonstration. No template is served and none is needed — the request proves the document reached out on open, and the grep above proves there was nothing in the file to find.

ODF: THE SAME SHAPE, DIFFERENT NAMES

OpenDocument Format is also a ZIP archive. The families map across almost exactly.

- `Scripts/`, `Basic/` **HIGH**
  StarBasic; same power as VBA
- `META-INF/manifest.xml` **HIGH**
  script entries declare what runs
- `Configurations2/` **REVIEW**
  toolbar and menu customisations

Inside the XML itself

- `<script`, `onload=`, `onclick=` in `content.xml` / `styles.xml`
- `<style>` blocks carrying `animation` — a timer that fires without a click

The distinction that matters: `Scripts/` is a *part you can delete*. A script element inside `content.xml` is *inside the document body* — delete the part and you have deleted the document.

PDF: THE READER IS THE ATTACK SURFACE

Several PDF object types mean “do something” rather than “draw something”.

Name	What it means	Severity
`/OpenAction`	run this the moment the file opens	HIGH
`/AA`	additional actions: on page open, on close, on focus	HIGH
`/JavaScript`, `/JS`	the reader's own JS engine	HIGH
`/Launch`	execute an external program	HIGH
`/EmbeddedFile`	a whole file carried inside	HIGH
`/RichMedia`	Flash and friends, still parsed by some readers	HIGH
`/XFA`, `/AcroForm`	form engines (XML Forms Architecture) with their own scripting	REVIEW
`/GoToE`, `/GoToR`, `/SubmitForm`	navigate to embedded / remote / post data	REVIEW
`/URI`, `/Annots`	links and comments — mostly benign	INFO

Note the last row. A scanner that flags `/Annots` on every PDF with a hyperlink teaches its users to ignore it.

DEMONSTRATION: THE FILE INSIDE THE FILE

`Ole10Native` still carries the original filename

```
$ oledump.py invoice.doc
    1:       114 '\x01CompObj'
    2:      4096 '\x01Ole10Native'
    3:       417 'Macros/VBA/Module1'
$ oledump.py -s 2 -d invoice.doc | xxd | head -2
00000000: 3c10 0000 0200 7570 6461 7465 2e65 7865  <.....update.exe
00000010: 0000 0000 0000 4d5a 9000 0300 0000 0400  ......MZ........
```

Two facts in one hexdump: the Packager stream stores the name the file will be written as, and the payload behind it is a PE. **HIGH**

The stream is 4096 bytes, which puts it exactly at the container's mini-stream cutoff — the boundary MS-OVBA decompression turns on later.

OBFUSCATION, AND WHY NAIVE SCANNING FAILS

PDF names take hex escapes

```
/JavaScript
/J#61vaScript
/#4A#61#76#61#53#63#72#69#70#74
```

All three are the same name to the reader. A substring search finds one of them.

Normalise before matching, never after.

Incremental updates

A PDF may contain several revisions, appended. The first `%%EOF` is not the end of the file.

Scan the whole byte range, not the first revision — otherwise the trailing revision, which is the one the reader actually resolves, goes unread.

Compression hides structure

Object streams and deflated content mean the keyword you are looking for may not appear as bytes at all.

DEMONSTRATION: GREP LOSES TWICE ON ONE FILE

Hex escapes, then the revision that is not the first one

```
$ grep -c '/JavaScript' obf.pdf
0
$ grep -c '/J#61vaScript' obf.pdf
1
$ grep -abo '%%EOF' obf.pdf
1180:%%EOF
2743:%%EOF
```

The first search is the one a reasonable person writes, and it returns zero on a file that has JavaScript in it. **Normalise before matching, never after.**

Two `%%EOF` markers means two revisions. A scanner that stops at the first one read a version of this document that the reader will never resolve.

THE AWKWARD PART

WHY THE USUAL ANSWERS DO NOT FIT

- **“Antivirus will catch it.”**
  Retrospective by construction — and remote template injection ships a document with no payload in it at all.
- **“Just don’t enable macros.”**
  Correct, and insufficient: ActiveX, external relationships, embedded objects and every PDF action above are untouched by it.
- **“Upload it to a scanning service.”**

UPLOADING A SUSPICIOUS DOCUMENT IS A DISCLOSURE DECISION

The file you are unsure about is a legal filing, a medical record, a source document from a confidential contact. Sending it to a third party to ask whether it is dangerous *publishes it* — and multi-scanner services have historically made samples available to subscribers.
You cannot ask that question about a document you are not allowed to share.

DEMONSTRATION: THE QUESTION YOU CANNOT ASK

Three files, one question, and nowhere to ask it

Three attachments arrive this morning

1. a CV from a candidate **INFO**
2. a sealed exhibit from opposing counsel **HIGH**
3. a discharge summary from a clinic **HIGH**

Same question about all three: *is this dangerous?* Show of hands — which may be uploaded to a multi-scanner service to find out?

The second and third are indefensible. The first is comfortable only until you notice the CV carries a home address, a phone number and a date of birth.

WHY THE CONSTRAINT IS NOT OPTIONAL

The constraint is not paranoia and not policy theater. It is that **the question and the disclosure are the same act** — and for most of the documents anyone actually worries about, asking is the thing you are not allowed to do.

WHAT YOU WRITE IN THE TICKET

The descriptive layer

Everything so far answers *is this dangerous*. None of it answers the question an analyst asks second: **what do I write down, and what do I pivot on?**

Per item, not just per file

- size, method, CRC-32, timestamp
- Shannon entropy
- MD5 and SHA-256
- what its magic bytes say it really is

And across the file

- author, last-modified-by, application
- every timestamp, oldest first
- URLs, IPs, UNC paths, registry keys

WHY MD5 IS IN A TOOL WRITTEN IN 2026

Collision-broken since 2004, and it decides nothing here. It is printed because intel sharing still runs on it. A triage tool that cannot produce an MD5 sends its user to fetch another tool — which is how a local-only tool stops being local.

DEMONSTRATION: ENTROPY POINTS, MAGIC DECIDES

Per entry, because per file tells you nothing

```
$ tools/entries.py suspect.docx
 name                       size   method   entropy  magic
 word/document.xml         24911   deflate     4.41  XML
 word/media/image1.png      4096   store       7.98  PE32
 word/media/image2.png     88214   store       7.94  PNG
 word/embeddings/o1.bin    12288   deflate     7.99  OLE2
```

Rows two and three have nearly the same entropy and nothing else in common. **Entropy tells you where to look; the magic column is what you write down.**

A tool that flagged on entropy alone would flag every PNG in every document ever sent — the `/Annots` mistake again, wearing a number.

VERIFICATION

TOOLS USED IN THE DEMONSTRATIONS

None of them mine, and all of them worth having

- `pdfid.py`, `oledump.py`
  Didier Stevens, public domain. `pdfid` is also where the length-preserving disarm comes from — the technique is his.
- `olevba` — oletools
  Philippe Lagadec. `pip install oletools`
- `mutool` — MuPDF, and LibreOffice
  The independent second opinion, and the convert-and-re-scan advice.

And your distribution's

`unzip` `xxd` `file`
`strings` `grep` `sha256sum`

```
apt install unzip xxd file
    binutils mupdf-tools
```

THE LIST, WITH LINKS

pres.toiletduck.cleaning — linked rather than bundled. Take the current version from the author, not a copy frozen inside a talk.

Everything stays local. One file. One set of tools.