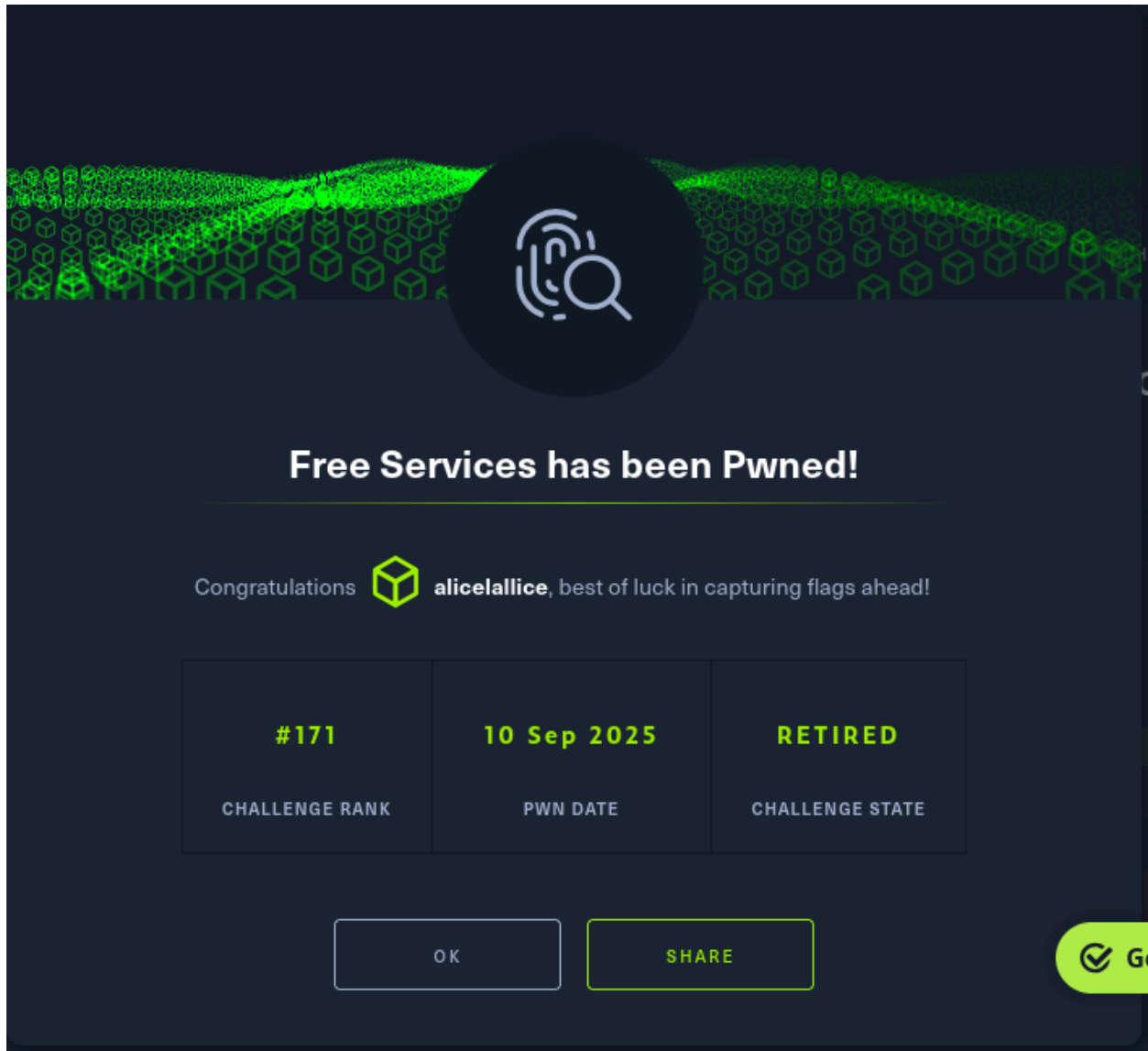


Free Service

Types	forensic
CTF	HTB



Challenge Overview

The challenge description indicated:

- The Intergalactic Federation intercepted a phishing campaign.
- A malicious `.xlsm` file was received (`free_decryption.xlsm`).
- The goal: Analyze the file to recover its “memory” (the hidden flag).

Observation: The `.xlsm` file might contain a **malicious macro**, but it was not a standard VBA macro.

```
(oletools-env)-(kali@kali)-[~/Desktop/htb]
$ olevba free_decryption.xlsm

olevba 0.60.2 on Python 3.13.7 - http://decalage.info/python/oletools
=====
FILE: free_decryption.xlsm
Type: OpenXML
No VBA or XLM macros found.
```

Initial Recon

```
olevba free_decryption.xlsm
```

Result:

```
| No VBA or XLM macros found
```

Unzipping the XLSM File

Since `.xlsm` is a ZIP archive:

```
unzip free_decryption.xlsm -d xlsx_extracted
cd xlsx_extracted
ls -R
```

```
(oletools-env)-(kali@kali)-[~/Desktop/htb]
$ unzip free_decryption.xlsm -d xlsx_extracted

Archive: free_decryption.xlsm
  inflating: xlsx_extracted/[Content_Types].xml
  inflating: xlsx_extracted/_rels/.rels
  inflating: xlsx_extracted/xl/workbook.xml
  inflating: xlsx_extracted/xl/_rels/workbook.xml.rels
  inflating: xlsx_extracted/xl/worksheets/sheet1.xml
  inflating: xlsx_extracted/xl/macrosheets/sheet1.xml
  inflating: xlsx_extracted/xl/theme/theme1.xml
  inflating: xlsx_extracted/xl/styles.xml
  inflating: xlsx_extracted/xl/sharedStrings.xml
  inflating: xlsx_extracted/xl/drawings/drawing1.xml
  extracting: xlsx_extracted/xl/media/image1.png
  inflating: xlsx_extracted/xl/worksheets/_rels/sheet1.xml.rels
  inflating: xlsx_extracted/xl/drawings/_rels/drawing1.xml.rels
  inflating: xlsx_extracted/xl/metadata.xml
  inflating: xlsx_extracted/xl/richData/rdrichvalue.xml
  inflating: xlsx_extracted/xl/richData/rdrichvaluestructure.xml
  inflating: xlsx_extracted/xl/richData/rdrichvaluetypes.xml
  inflating: xlsx_extracted/docProps/core.xml
  inflating: xlsx_extracted/docProps/app.xml

(koletools-env)-(kali@kali)-[~/Desktop/htb]
$ cd xlsx_extracted
```

```
(oletools-env)-(kali@kali)-[~/Desktop/htb/xlsx_extracted]
$ ls -R
.:
[Content_Types].xml  docProps  _rels  xl

./docProps:
app.xml  core.xml

./_rels:
trash  final_board...

./xl:
drawings  macrosheets  media  metadata.xml  _rels  richData  sharedStrings.xml  styles.xml  theme

./xl/drawings:
drawing1.xml  _rels

./xl/drawings/_rels:
drawing1.xml.rels

./xl/macrosheets:
sheet1.xml

./xl/media:
image1.png

./xl/_rels:
workbook.xml.rels

./xl/richData:
rdrichvaluestructure.xml  rdrichvaluetypes.xml  rdrichvalue.xml

./xl/theme:
theme1.xml

./xl/worksheets:
_rels  sheet1.xml

./xl/worksheets/_rels:
sheet1.xml.rels
```

Key location:

```
xl/macrosheets/sheet1.xml
```

Inspecting the Macrosheet

The macrosheet contained formulas like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="2" uniqueCount="2"><si><t>CALL("Kernel32","CreateThread","JJJJJJ",0,0,R2C6,0,0,0)</t></si><si><t>HALT()</t></si></sst>
~
~
```

Analysis:

- `VirtualAlloc` allocates memory.
- `BITXOR(...,24)` indicates each byte is **XORed with 24** for obfuscation.
- `WriteProcessMemory` writes the decoded payload.

Conclusion: The flag is hidden using **XLM macros + XOR obfuscation**.

Extracting the Payload

We wrote a Python script (`de.py`) to:

- Parse `sheet1.xml`.
- Reverse the `BITXOR` operations.
- Save the output as `decoded_payload.bin`.

```
import xml.etree.ElementTree as ET
```

```
# Load the Excel macro sheet XML
```

```
tree = ET.parse("sheet1.xml")
```

```
root = tree.getroot()
```

```

# Excel XML uses a namespace
ns = {"main": "http://schemas.openxmlformats.org/spreadsheetml/2006/main"}

decoded_bytes = []

# Iterate over all <c> (cell) elements
for c in root.findall(".//main:c", ns):
    # Get the cell reference like E10, F11, etc.
    cell_ref = c.attrib.get("r", "")
    if cell_ref.startswith(("E", "F", "G")):
        v = c.find("main:v", ns)
        if v is not None:
            try:
                num = int(v.text)
                decoded = num ^ 24 # XOR with 0x18
                decoded_bytes.append(decoded)
            except ValueError:
                pass

# Write raw bytes
with open("decoded_payload.bin", "wb") as f:
    f.write(bytearray(decoded_bytes))

print(f"[+] Extracted {len(decoded_bytes)} bytes → saved to decoded_payload.bin")

# Optional: also try printing ASCII (sometimes it's just the flag as text)
ascii_preview = "".join(chr(b) if 32 <= b < 127 else "." for b in decoded_bytes)
print("[+] ASCII preview:")
print(ascii_preview)

```

```

[oletools-env]-(kali@kali)-[~/htb/xlsm_extracted/xl/macrosheets]
$ python3 /home/kali/Desktop/htb/xlsm_extracted/de.py
[+] Extracted 772 bytes → saved to decoded_payload.bin
[+] ASCII preview:
.....1..d..BP.0~"RE.n..R0.C..r.(j...J.6.1....<a.|.7.{ .....U.."......R.W...R.....Jw<...L!..XB..H.....QA.7Y, W...S..I! ...q:I..G4..t.b..10...;....%...18..7u.....}N..j.)0$0u...uX..pX.
$. ....f...a..K...X.....e...V.B.f.P...D.$$.{.[a(Y02.Q...&j~ZW....R..)] .....0...PGHg1..P0..... ...X..V.h.....<...| .....vU7...G..6f o.j] ..S...J.ZRQf.G WA)dpD. Z"-H.K.LMM"\.530.
F.T.W.A7R.Ew\,M1XKc:r)0.s.o.f.thQWci.n.d o.w.s. N3T\,C0u5rIe.n.tCv.e.r.s.l.o.n.\,I'mmasg.eX, f.fL.e! NExx.eac.u.t.i.omm. 90.p.t.1$0.o.s\,u!t.i.l.e.a.n-{e.x.e."./:t{ ,RIE.G<_HS.Zx /vY .D.e.
b.u.g.g.ecra ./_d_ ".C:\3\w.i.n.d.o.wVs.\,siyQs.t.e.mV3.26\kTm.dl...e.xbe.". /wF.jKenc.h=0. ".H0T.BW{(1.s...t.h.1.s2_Sg.4.l.4fXcy...l+0.s.tQ.z1.nw_[tZ11m13.7.7!].}."6..

```

there we can see the pattern of the flag HTB..