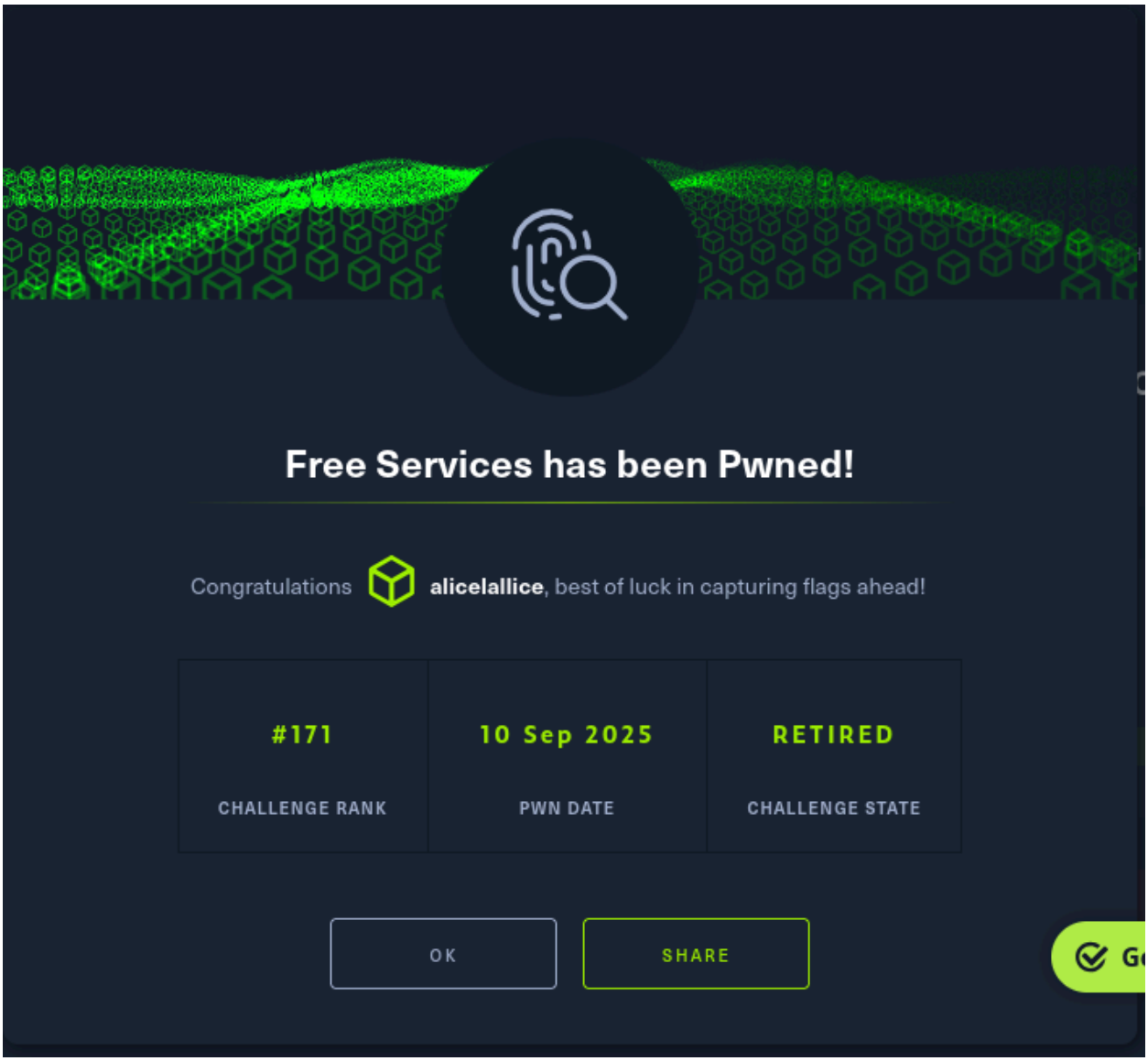
Free Service

Types	forensic
CTF	HTB

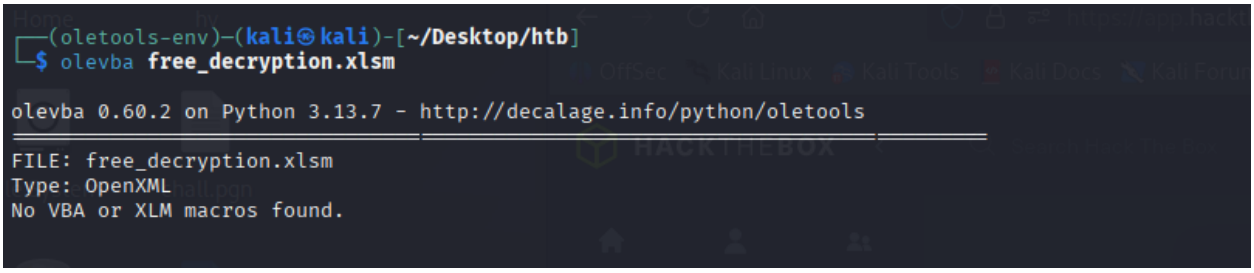


Challenge Overview

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.



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 xlsm_extracted
cd xlsm_extracted
ls -R
```

```
┌──(oletools-env)─(kali㉿kali)-[~/Desktop/htb]
└─$ unzip free_decryption.xlsm -d xlsm_extracted

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

┌──(oletools-env)─(kali㉿kali)-[~/Desktop/htb]
└─$ cd xlsm_extracted
```

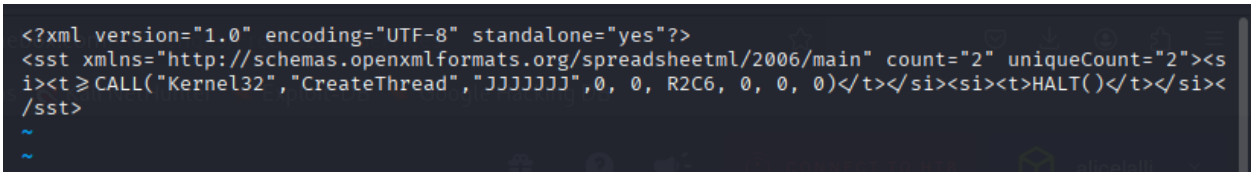
```
┌──(oletools-env)─(kali㉿kali)-[~/Desktop/htb/xlsm_extracted]
└─$ ls -R
.:
'[Content_Types].xml'   docProps   _rels   xl

./docProps:
app.xml  core.xml

./_rels:
./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:



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**.

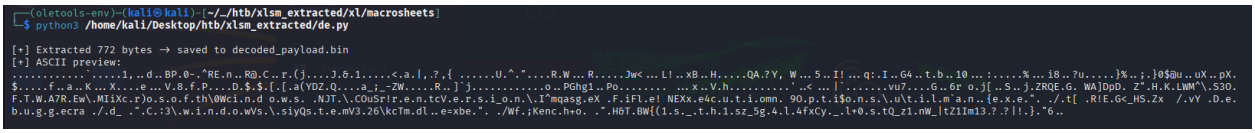
Extracting the Payload

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/mai
n"}

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_payloa
d.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)
```



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