2025

CTF INTERNACIONAL METARED ETAPA MEXICO

WRITEUP

CHALLENGE

Codes in the Silence

Description

Following an investigation into digital extortion, a message arrived as a form of threat. No keys, no encryption... just repeated text. But not every message screams. Some codes hide in the silence, between characters that leave no shadow or form. Look beyond the visible content. Sometimes, the most dangerous language is the one that cannot be read.

File: message.txt

Solution:

The description of the challenge hints at an invisible or silent form of communication. While the text appears to simply repeat “last warning…”, a closer look reveals that some characters might not be what they seem. The phrase “Some codes hide in the silence, between characters that leave no shadow or form” suggests the presence of zero-width characters — Unicode characters that don't display visibly but can encode information.

Upon opening **message.txt,** the content appears to be a repeated and redundant warning message. However, a raw hex or binary inspection (e.g., via a hex editor or cat -A) reveals an abnormal presence of characters like:

- \u200b (Zero Width Space)
- \u200c (Zero Width Non-Joiner)
- \u200d (Zero Width Joiner)

These characters are invisible in regular editors, but they do carry binary information, which is often used for steganographic purposes.

Decoding all the things reveals a .zip file encoded into the text, this file contains flag.txt.

This script reads the zero-width characters embedded in message.txt and decodes them into binary data. The binary is then converted into a byte array, which is saved directly into a ZIP archive that contains the flag.

```python
import zipfile

# Zero-width characters used in the encoding
ZW_SPACE = b"\xe2\x80\x8b"     # U+200B → bit 0
ZW_NONJOIN = b"\xe2\x80\x8c"   # U+200C → bit 1
BOM = b"\xef\xbb\xbf"          # Byte Order Mark

binary_data = b""

with open("message.txt", "rb") as input_file:
    for line in input_file:
        # Split line on BOM markers to isolate the hidden block
        parts = line.split(BOM)
        if len(parts) < 3:
            continue  # No valid encoded block in this line

        hidden_block = parts[1]  # The encoded data is between BOMs
        bits = ""

        # Read invisible characters in 3-byte chunks
        for i in range(0, len(hidden_block), 3):
            chunk = hidden_block[i:i+3]
            if chunk == ZW_SPACE:
                bits += "0"
            elif chunk == ZW_NONJOIN:
                bits += "1"

        # Convert full byte if 8 bits were found
        if len(bits) == 8:
            binary_data += bytes([int(bits, 2)])

# Write the recovered ZIP file
with open("message-recovered.zip", "wb") as output_file:
    output_file.write(binary_data)

print("[+] File 'message-recovered.zip' successfully written.")

# Optional: validate ZIP structure
try:
    with zipfile.ZipFile("message-recovered.zip", "r") as zip_check:
        test = zip_check.testzip()
        if test is None:
            print("[✓] ZIP file is valid and uncorrupted.")
            print("    Contents:", zip_check.namelist())
        else:
            print("[!] ZIP file has a problem with:", test)
except zipfile.BadZipFile:
    print("[✗] The recovered file is not a valid ZIP archive.")
```

Step-by-Step Explanation of the Code

1. Define the binary mapping:

- \u200b maps to binary '0'
- \u200c maps to binary '1'
- \u200d is ignored (possibly noise or filler)

2. Open the encoded message:

- message.txt is read in UTF-8 encoding to properly interpret Unicode characters.

3. Extract the zero-width characters:

- The script loops through the entire text.
- For each character found in the zwc dictionary, the corresponding binary value is added to a string called bits.

4. Convert the binary stream to bytes:

- Every 8 bits are grouped and converted into a decimal byte using int(byte, 2).
- Only complete 8-bit chunks are considered to avoid incomplete byte errors.

5. Reconstruct the ZIP file:
- The resulting bytes are written into a new file: message.zip.

6. Outcome:
- When unzipped, message.zip reveals the file flag.txt.

Final Flag

Once the ZIP file is extracted, the content of flag.txt is revealed to be:

flagmx{read_between_lines_and_bits}

Conclusion

This challenge cleverly hides binary data within text using zero-width characters. The solution involves understanding Unicode steganography, specifically how non-visible characters can encode binary data without altering visible text. The script extracts and decodes the hidden ZIP file byte by byte, leading to the successful retrieval of the flag.