Shadows Within the Layers - Writeup

Challenge Name: Shadows Within the Layers

Category: Forensics, Cryptography

CTF Event: SeaTF 2025

1. Challenge Description:

A cryptic message has been buried beneath layers of deception. What seems ordinary is anything but—the truth is locked away, hidden in silence, and distorted beyond recognition. To uncover it, you must unravel the steps taken to conceal it.

Mission:

Somewhere in the provided files, a truth long forgotten awaits discovery. But be warned—nothing is as it seems.

Hints:

- A locked vault guards the first secret—its key is known, yet unseen.
- The secrets are hidden in plain sight, but locked away with a key. First, uncover the key from the encrypted vault, then use it to reveal the hidden message within the image. A trail of hex awaits your discovery!
- Even after revealing the message, its true form remains disguised. Look beyond its current state—transform it, and another layer of the secret will unfold.
- The message is still cloaked, but a simple shift can set it straight. Rotate the letters, and what was once scrambled will start to make sense.
- The message is locked with a familiar twist—shift it back to uncover the truth. But the key to the shift? That's for you to discover.
- The journey through ciphers and transformations nears its end. Strip away this final layer, and the true message will finally come to light.

2. Files Provided:

- challenge_1.jpg
- password.enc

3. **Approach & Solution:**

Step 1: Decrypting password.enc

The file password.enc suggests that it is encrypted. A common encryption scheme for such files is AES, often requiring a password-based decryption. The hint mentions a 'locked vault' and 'key is known, yet unseen,' implying that the key may be something obvious or provided in the challenge itself.

We use OpenSSL to decrypt the file with a known passphrase "securekey":

```
openssl enc -aes-256-cbc -pbkdf2 -d -in password.enc -out
password.txt -pass pass:"securekey"
```

Checking the contents:

```
cat password.txt
```

```
Output: Grabbed
```

This reveals the password: "**Grabbed**", which will be useful for the next step.

Step 2: Extracting Data from challenge_1.jpg

Image files often contain hidden messages embedded via steganography. The challenge hints at a 'hidden message within the image' and 'locked away with a key.' Given the format of the image, Steghide is a well-known tool for extracting hidden data using a password.

```
steghide extract -sf challenge_1.jpg -xf extracted.txt -p
"Grabbed"
```

```
Output: wrote extracted data to "extracted.txt".
```

Checking the contents:

```
cat extracted.txt
```

```
Output: hex_encoded_string
```

Step 3: Decoding the Extracted Hex String

The extracted text appears as a hex-encoded string, which is a common way to store encoded data. Converting hex back into readable text may reveal another layer of encoding. The output reveals a base64 encoded string with ROT13 transformation.

```
echo -n "hex_encoded_string" | xxd -r -p
```

```
Output: ROT13_encoded_string
```

Step 4: Applying ROT13

The hint mentions 'rotate the letters,' which strongly suggests ROT13, a simple letter substitution cipher that shifts characters by 13 places.

```
echo "ROT13_encoded_string" | tr 'A-Za-z' 'N-ZA-Mn-za-m'
```

```
Output: base_64_encoded_string_encrypted_using_caesar_cipher
```

This reveals a base64 encoded string encrypted using caesar cipher

Step 5: Applying the Caesar Cipher (Shift -3)

The challenge hints at a 'familiar twist' and 'shift it back.' A common cipher used in CTFs is the Caesar cipher, where characters are shifted by a fixed amount. The participants must determine the correct shift value by trial or educated guessing (e.g., shift -3 is common).

Python Code to decrypt the string

```
def caesar_cipher(text, shift):
      result = ""
```

```
    for char in text:
    if char.isalpha():
        shift_amount = shift % 26
        new_char = chr(((ord(char) - ord('A' if char.isupper()
else 'a') + shift_amount) % 26) + ord('A' if char.isupper() else
'a'))
        result += new_char
    else:
        result += char
    return result

decrypted =
caesar_cipher("base_64_encoded_string_encrypted_using_caesar_ciph
er", -3)
print(decrypted)
```

```
python3 decrypt_script.py
```

```
Output: base64_encoded_flag
```

This output reveals the base64 encoded flag.

Step 6: Decoding the Base64 String

The final step involves decoding a Base64-encoded string. The last hint states, "The journey through ciphers and transformations nears its end. Strip away this final layer, and the true message will finally come to light." This suggests that Base64 is the last encoding method used.

```
echo -n "base64_encoded_flag" | base64 -d
```

```
Output: SeaTF{D1g1tal_W4v3s_Cr0ss1ng_B0rd3rs}
```

This reveals the final flag.

4. Flag Extraction

Final Flag: SeaTF{D1g1tal_W4v3s_Cr0ss1ng_B0rd3rs}

5. Learning Outcomes:

- **AES Encryption & Decryption** – Using OpenSSL to decrypt files with known passwords.
- **Steganography with Steghide** – Extracting hidden data from image files.
- **Hex Encoding & Decoding** – Converting hexadecimal representations into readable formats.
- **Caesar Cipher Decryption** – Understanding character shifts and applying Python scripting to decrypt text.
- **Base64 Encoding & Decoding** – Recognizing and decoding Base64-encoded strings.
- **Logical Problem-Solving** – Interpreting cryptographic hints and applying common CTF techniques systematically.

6. References & Tools Used:

- **OpenSSL**: Used for AES decryption.
- **Steghide**: Extracts hidden data from image files.
- **xxd**: Converts hex-encoded strings back to readable text.
- **Python**: Implementing the Caesar cipher shift.
- **Base64**: Encoding and decoding textual data.
- **Linux Command Line**: Various CLI tools for file handling and text processing.

*****