WARGAMES MY CTF 2024

WRITE UP

Written By

cha

johnwayne

JobuTupak

WARGAMES.MY 2024

So, 3 of us joined this CTF game and somehow, ended up in 15th place. Here, we've shared our analysis on the challenges that we solved (and not) that came our way during the competition.

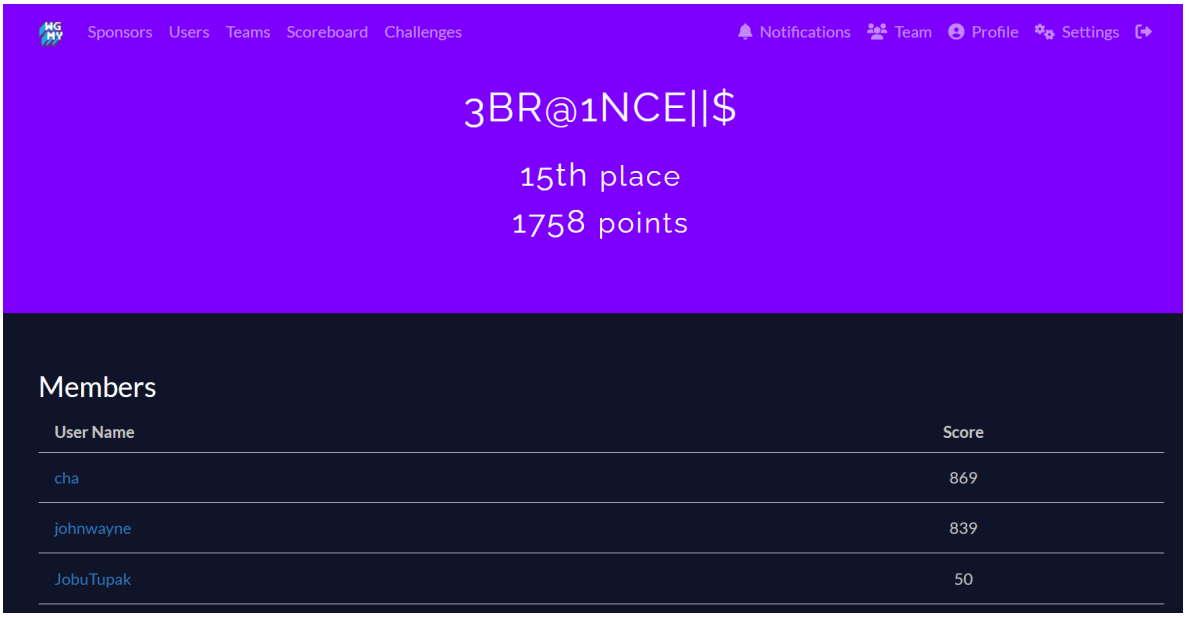
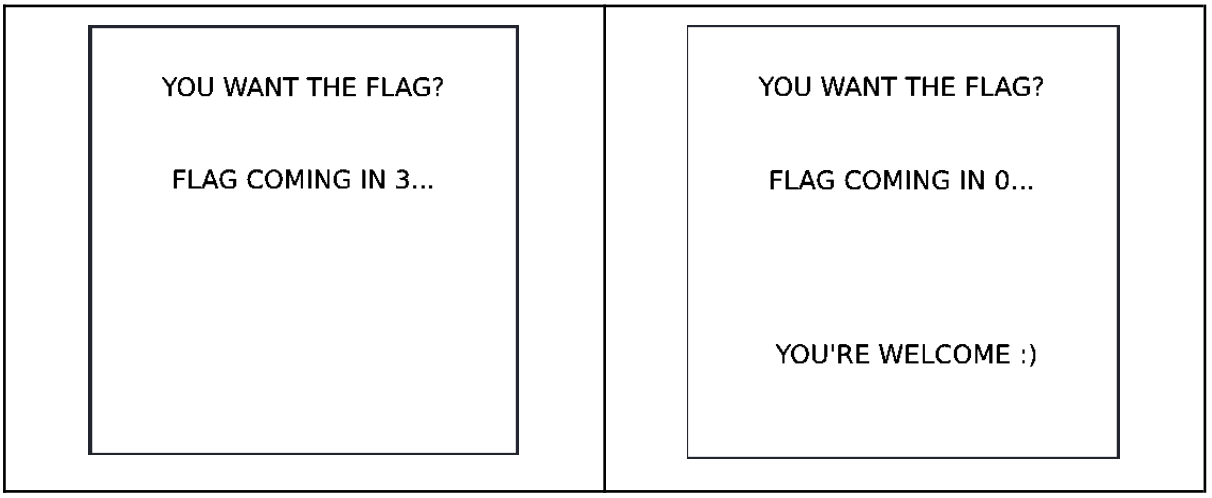


TABLE OF CONTENT

MISC

Invisible Ink

These are the first frame and last frame of the gif:



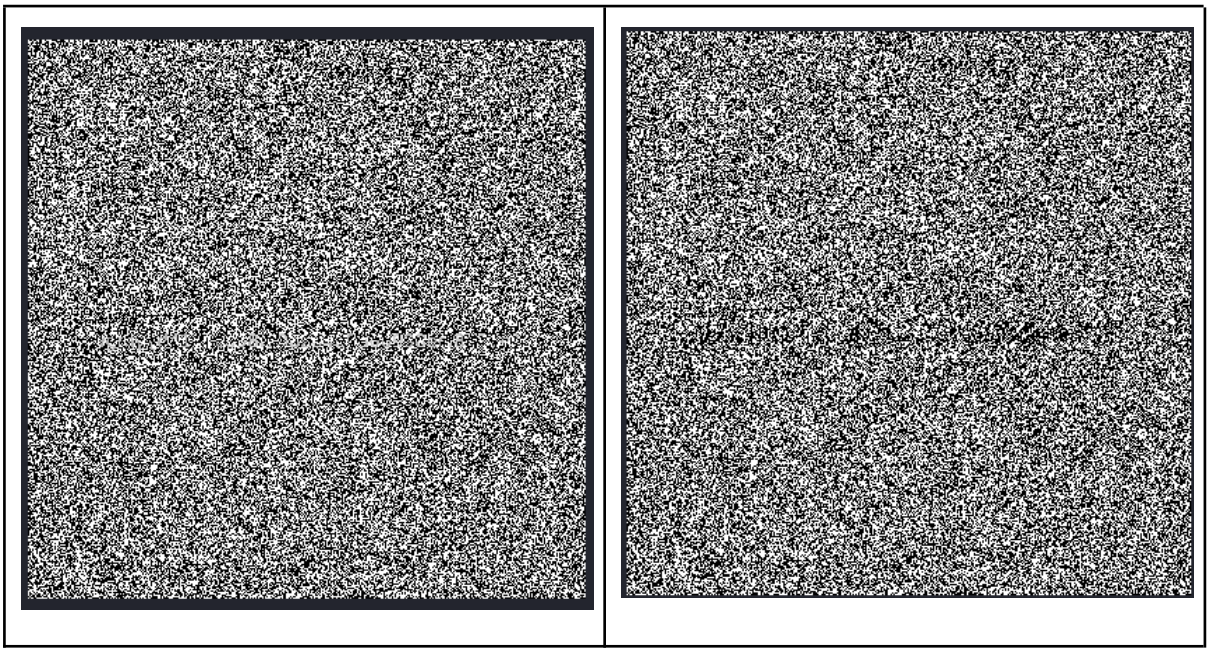
Analysis & Solution

Step 1: Analysing the .gif

- A. gif challenge that hides the flag in invisible ink, so it means I have to play with colors for the frames.
- So the first tool that came up to mind is StegSolve which is useful to check the frames of the gif.

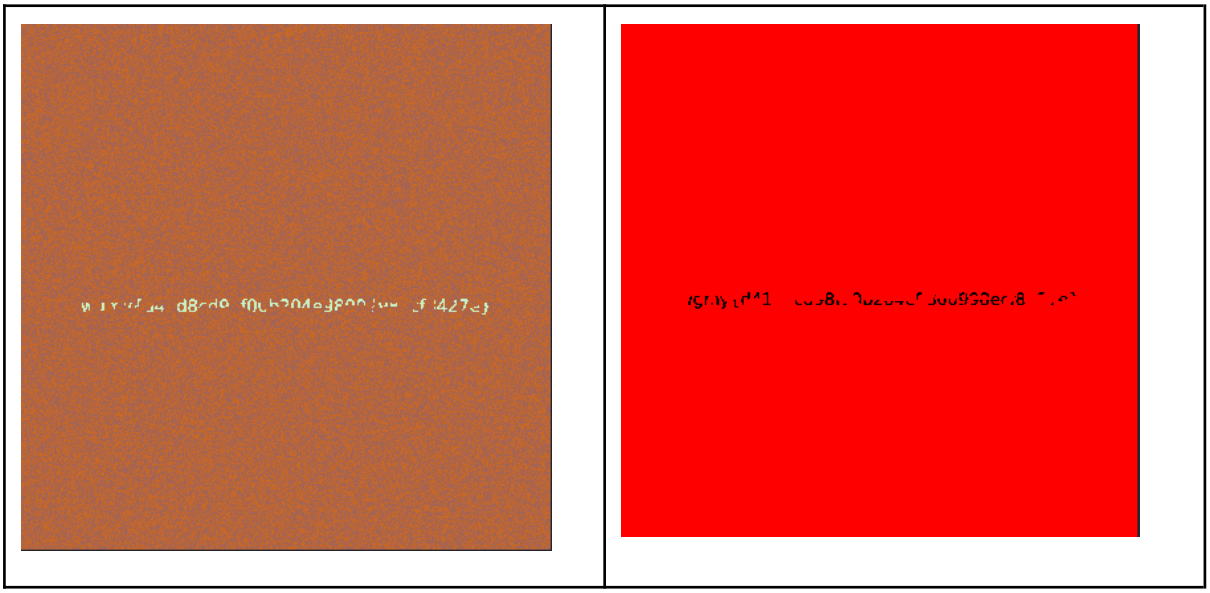
Step 2: Checking every frame of .gif using StegSolve.jar

- In the stegsolve, I used Analyse > Frame Browser to check every frame one by one.
- I noticed 2 frames have noise image and unclear text, that's when I confirmed it is a flag.



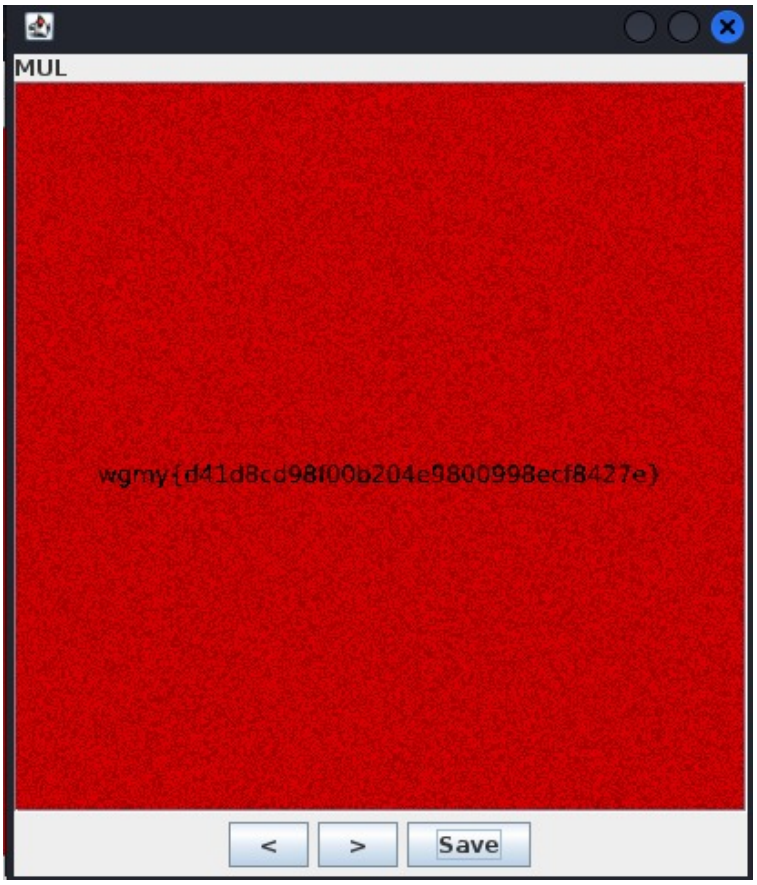
Step 3: Colouring the frame

- I saved the 2 frames (Frame 4 and 5) and opened them one by one with StegSolve again.
- I tried changing their colors by going through Random Color to make the frame as clear as possible.



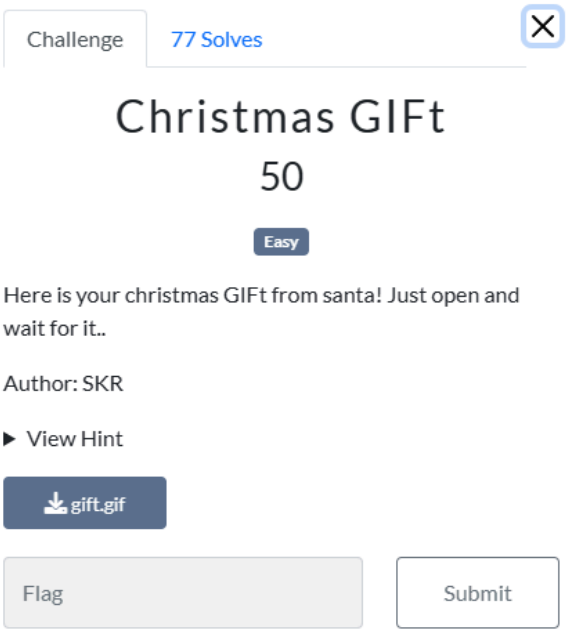
Step 4: Combining both frame and retrieve flag

- When both frames looks clear enough, I combined them through Analyse > Image Combiner in StegSolve.



Flag: wgmy{d41d8cd98100b204c9800998ecf8427e}

Christmas Gift



The prompt provided a .gif file of a present box being opened, but nothing came out of it.

Analysis & Solution

Step 1: Analyzing the File

- I began by examining the file format using various tools like file, exiftool, and binwalk to check for hidden data or embedded files.
- None of these tools revealed anything unusual about the .gif file.

Step 2: Visual Inspection in GIMP

- I opened the .gif file in GIMP to examine the individual frames.
- Upon scrolling through the frames, I discovered one frame that contained the flag.

Step 3: Submitting the Flag

- I noted down the text from the frame and submitted it as the flag.

Flag: wgmy{1eaa6da7b7f5df6f7c0381c8f23af4d3}

The DCM Meta

The challenge provided a challenge.dcm (and I realised that the hint they gave later on was sooo straightforward)

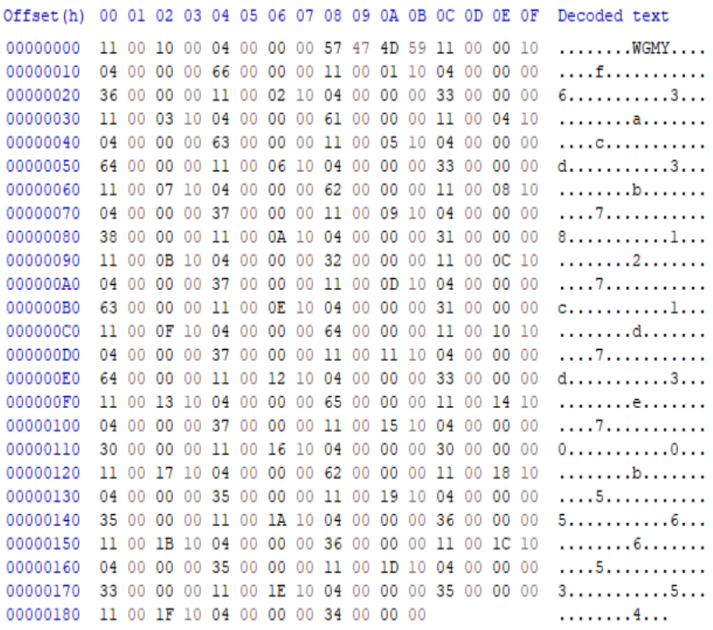
Analysis & Solution

Step 1: Inspecting the File

- Running the file command on challenge.dcm indicated that it was a generic data file, without identifying a specific format like DICOM or others.
- To investigate further, I opened the file in HxD, a hex editor.

Step 2: Analyzing Hex Dumps

- The hex dump revealed the presence of the string WGMY (the flag format) followed by a sequence of numbers.
- After hearing that my friend tried to sort it out, it gave me the idea that these are the flag strings but they were out of order.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	11	00	10	00	04	00	00	00	57	47	4D	59	11	00	00	10	........WGMY....
00000010	04	00	00	00	66	00	00	00	11	00	01	10	04	00	00	00	....f...........
00000020	36	00	00	00	11	00	02	10	04	00	00	00	33	00	00	00	6...........3...
00000030	11	00	03	10	04	00	00	00	61	00	00	00	11	00	04	10	........a.......
00000040	04	00	00	00	63	00	00	00	11	00	05	10	04	00	00	00	....c...........
00000050	64	00	00	00	11	00	06	10	04	00	00	00	33	00	00	00	d...........3...
00000060	11	00	07	10	04	00	00	00	62	00	00	00	11	00	08	10	........b.......
00000070	04	00	00	00	37	00	00	00	11	00	09	10	04	00	00	00	....7...........
00000080	38	00	00	00	11	00	0A	10	04	00	00	00	31	00	00	00	8...........1...
00000090	11	00	0B	10	04	00	00	00	32	00	00	00	11	00	0C	10	........2.......
000000A0	04	00	00	00	37	00	00	00	11	00	0D	10	04	00	00	00	....7...........
000000B0	63	00	00	00	11	00	0E	10	04	00	00	00	31	00	00	00	c...........1...
000000C0	11	00	0F	10	04	00	00	00	64	00	00	00	11	00	10	10	........d.......
000000D0	04	00	00	00	37	00	00	00	11	00	11	10	04	00	00	00	....7...........
000000E0	64	00	00	00	11	00	12	10	04	00	00	00	33	00	00	00	d...........3...
000000F0	11	00	13	10	04	00	00	00	65	00	00	00	11	00	14	10	........e.......
00000100	04	00	00	00	37	00	00	00	11	00	15	10	04	00	00	00	....7...........
00000110	30	00	00	00	11	00	16	10	04	00	00	00	30	00	00	00	0...........0...
00000120	11	00	17	10	04	00	00	00	62	00	00	00	11	00	18	10	........b.......
00000130	04	00	00	00	35	00	00	00	11	00	19	10	04	00	00	00	....5...........
00000140	35	00	00	00	11	00	1A	10	04	00	00	00	36	00	00	00	5...........6...
00000150	11	00	1B	10	04	00	00	00	36	00	00	00	11	00	1C	10	........6.......
00000160	04	00	00	00	35	00	00	00	11	00	1D	10	04	00	00	00	....5...........
00000170	33	00	00	00	11	00	1E	10	04	00	00	00	35	00	00	00	3...........5...
00000180	11	00	1F	10	04	00	00	00	34	00	00	00					........4...

Step 3: Sorting the Strings

- Using the array provided in the description, I wrote a really Python script to rearrange the strings based on the given sequence.

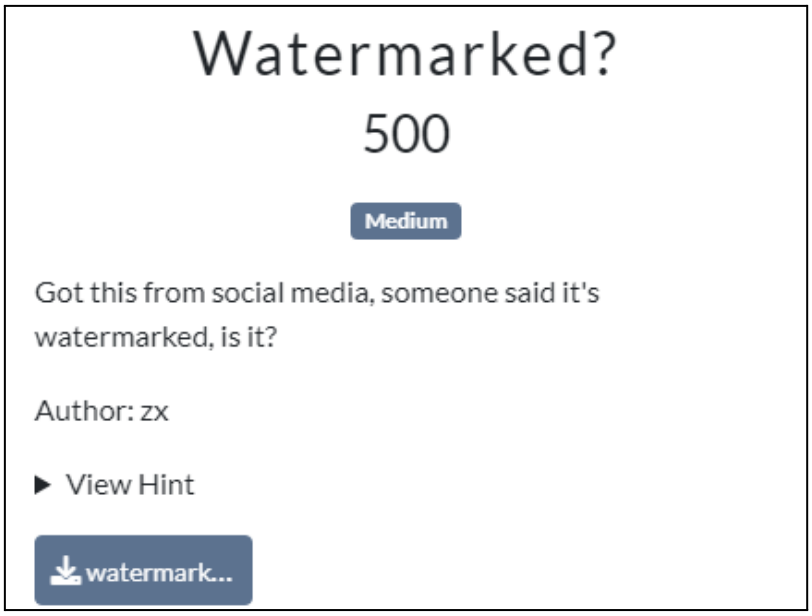
```
original_rank = "f63acd3b78127c1d7d3e700b55665354"
arrangement = [25, 10, 0, 3, 17, 19, 23, 27, 4, 13, 20, 8, 24, 21, 31, 15, 7, 29, 6, 1, 9, 30, 22, 5, 28, 18, 26, 11, 2, 14, 16, 12]

# Rearrange the original rank based on the given arrangement
reordered_rank = ''.join([original_rank[i] for i in arrangement])
print(reordered_rank)
```

Step 4: Run the Script

- Running the script produced the reordered string.
- The final flag was then wrapped in the required format: wgmy{reordered_string}.

Flag: wgmy{51fadeb6cc77504db336850d53623177}

Watermarked

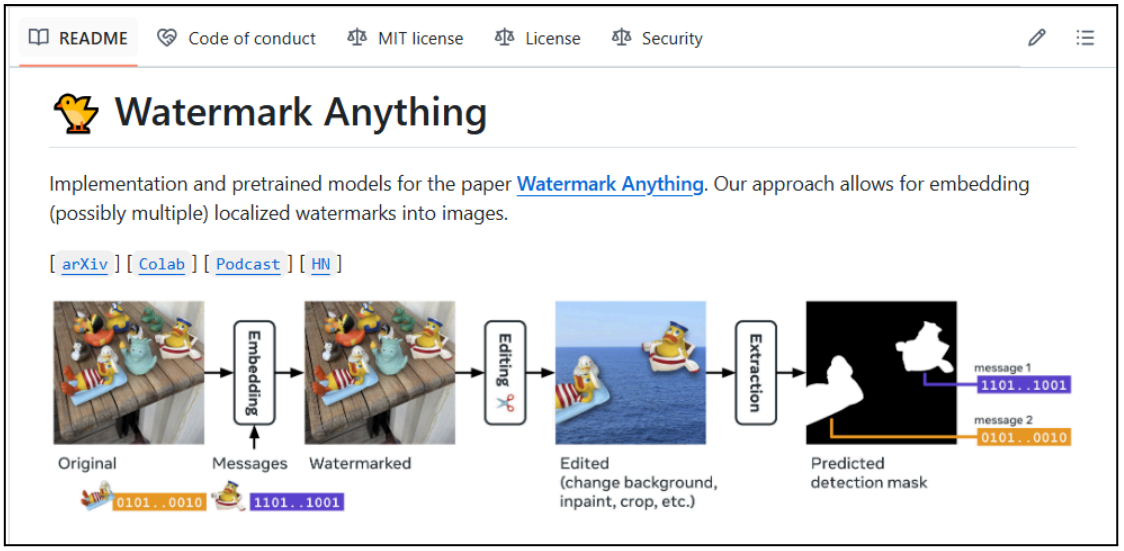


A gif again… but this time seriously we have no clue

Analysis

Step 1: Looking for tools to use

- I tried the basic tools to find anything about this .gif, but nothing really looks suspicious enough.
- Until the hint dropped, and found a github called Watermark-Anything.



Step 2: Using Watermark-Anything tool

- Well, I tried to use this but my kali linux terminal was full of errors during the process of this.

GAME

World 1

Challenge | 34 Solves

World 1

304

Easy

Game hacking is back!

Can you save the princess?

White screen? That is a part of the challenge, try to overcome it.

Author: Trailbl4z3r & Monaruku

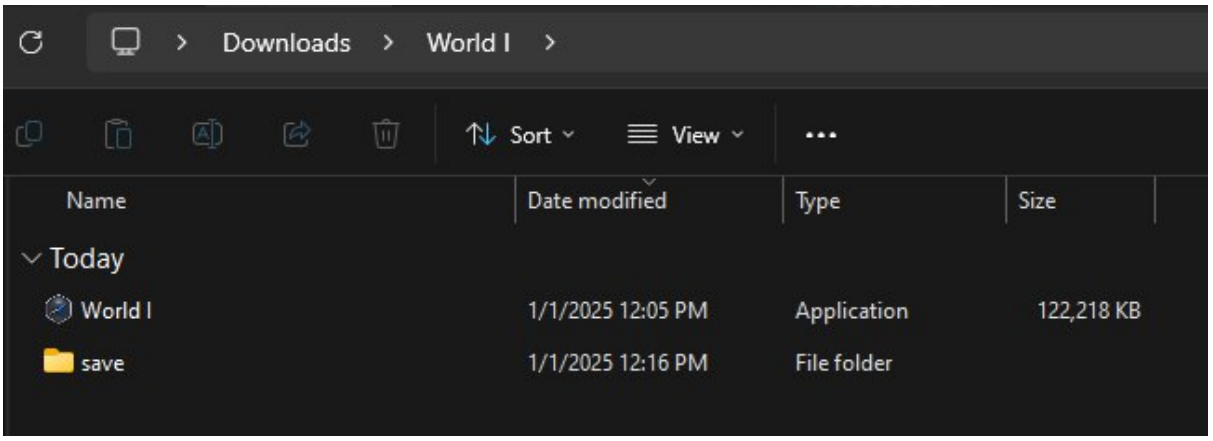
File:
https://drive.google.com/file/d/15Nfqmd2U0td57nmNt0ujy6Th__58och5/view

▶ View Hint

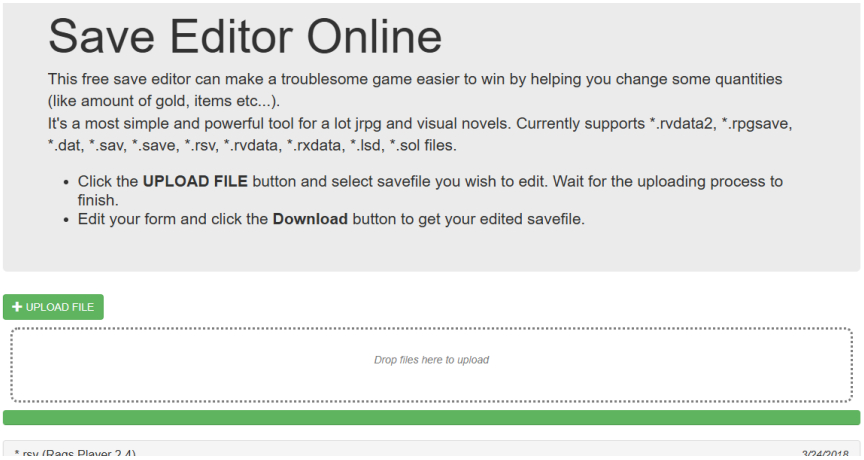
Flag | Submit

For the first time ever, we encountered this type of challenge. I opened the drive and downloaded the exe file. I opened the file and it is indeed a game.

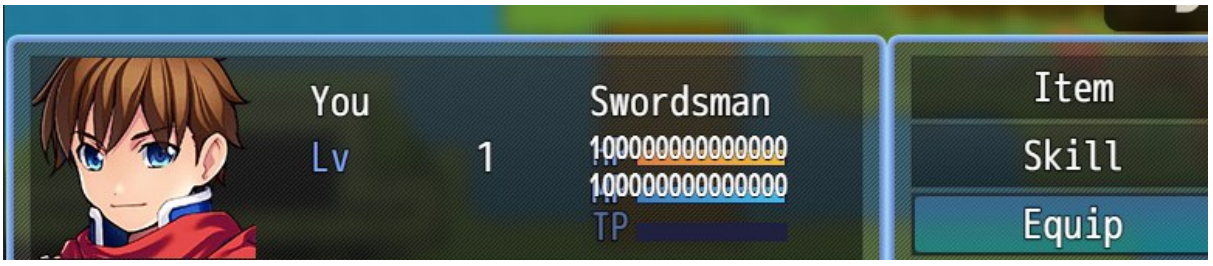
- I played the game like usual. Each victory gave me a chunk of flag. But once, I was at the end of the level. The boss was impossible to beat with the default level.



I save the process and look over the folder of the exe file. There was a save file located.



After that, I changed the value of the character status. I replaced the file with the original file.



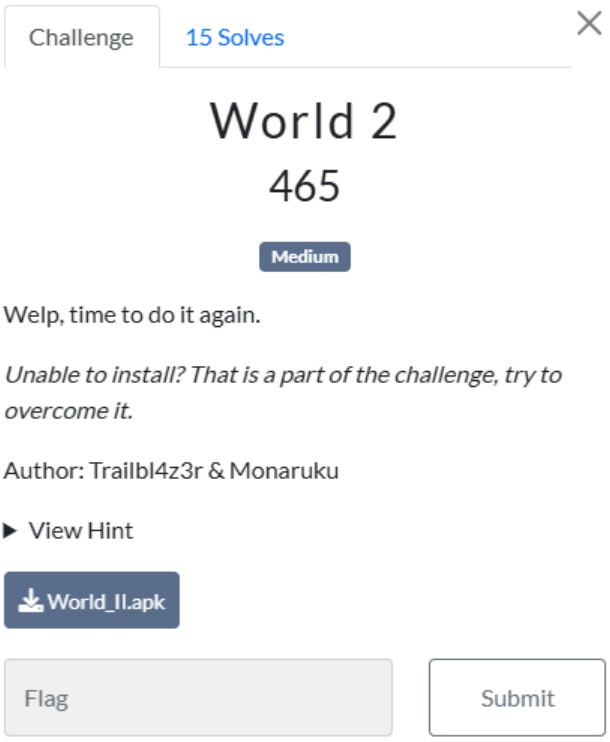
After defeating the boss level. We're required to enter a code with 4 lengths of words. I entered wgmy and it's correct. The 5th chunk of flag has been given in a qr code.

3fcaac2}

Above is the qr code output.

Flag: wgmy{5ce7d7a7140ebabf5cd43effd3fcaac2}

World 2



The prompt provided an APK file of a game similar to World 1.

Analysis & Solution

Step 1: Identifying the Challenge Context

This was my first time tackling a game hacking challenge, but I had seen my friend approach World 1. The description gave a clear hint with "time to do it again," suggesting a similar approach could work.

The key difference was that this game was on a different platform, which added a new layer of difficulty.

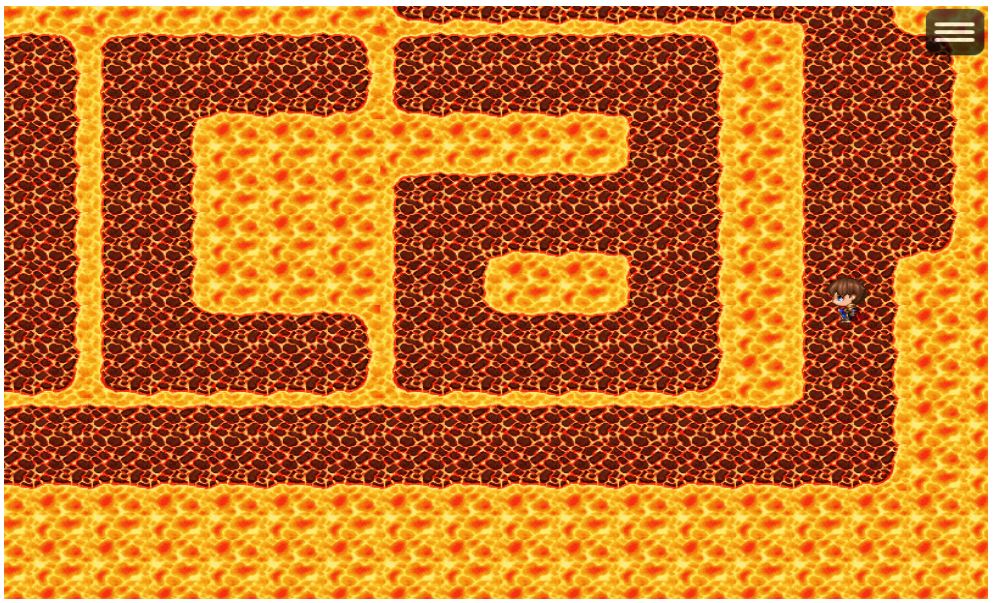
Step 2: Playing the Game

- I played the game for one round, saving the game after each session (a "round" involved fighting an opponent).
- Each round won provided me with a part of the flag.
- Each save generated an rmmsave file in the game's directory, which became crucial later.

Step 3: Overcoming Obstacles

Obstacle 1: Missing Part of the Flag

- The game provided four parts of the flag: Flag 1, Flag 2, Flag 3, and Flag 5. However, Flag 4 was missing.
- After some exploration and recalling my friend's experience in World 1, I discovered that Flag 4 was hidden in the lava section of the game. It was written directly on the lava, requiring careful navigation to piece it together. Credit to my friend for noticing this in World 1.



Obstacle 2: Defeating the Final Boss

- The final boss was unbeatable through regular gameplay—its attack dealt an absurd amount of HP damage. To overcome this, I modified the save file.
- I extracted the rmmsave file from the game folder and uploaded it to Save Editor Online. This tool allowed me to edit the file's attributes.
- I increased all critical stats (HP, attack, etc.) to atleast `1000000`, downloaded the modified file, and replaced the original in the game folder.

file1.rmmzsave

Gold	0

Param set #1 (You):

MHP	1000000	MMP	1000000	ATK	1000000	DEF	1000000
MAT	1000000	MDF	1000000	AGI	1000000	LUK	1000000
MyHP	1000000	MyMP	1000000				

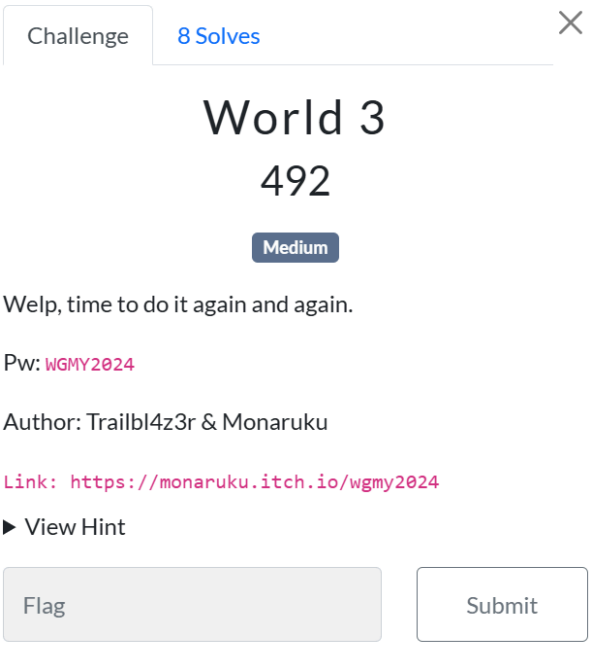
- After reloading the game, my character was now overpowered, capable of defeating the boss in one attack.

Step 4: Capturing the Flag

- Defeating the final boss rewarded me with Flag 5. Combining all five flag parts gave the full flag.
- While the process was straightforward in hindsight, navigating the lava section to read Flag 4 was particularly challenging.

Flag: wgmy{4068a87d81d8c901043885bac4f51785}

World 3



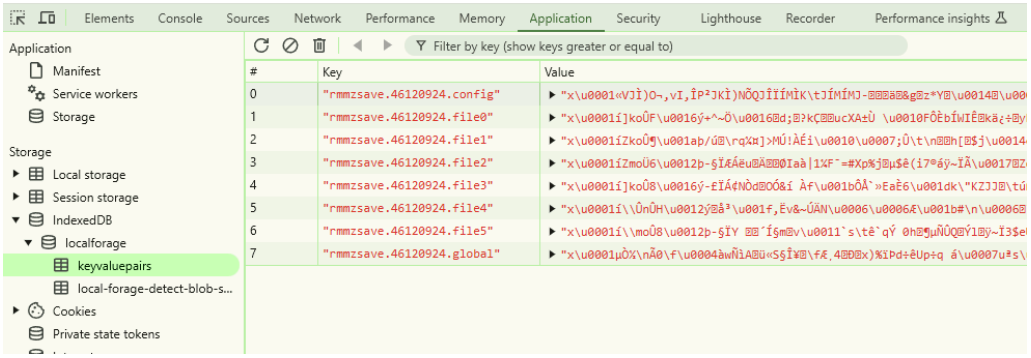
The challenge provided us with a link to a website containing a game similar to World 1 and 2.

Seeing that the description said “time to do it again and again”, I figure I have to do the challenge with the same approach for World 1 & 2.

Analysis

Step 1: Inspecting the Save File

- After accessing the game, I played it and saved my progress.
- I navigated to the Application tab in the browser's developer tools to locate the save file (rmmsave).
- Unfortunately, the file's content was unreadable and I could not find a way to modify it, so this approach did not lead to a solution.



Step 2: Exploring JavaScript Files

- I shifted my focus to the Sources tab to inspect the JavaScript files used by the game.
- While examining the scripts, I came across a potential method to manipulate the game using external plugins.
- I referred to this GitHub repository, which describes a cheat menu plugin for RPG Maker MV games.

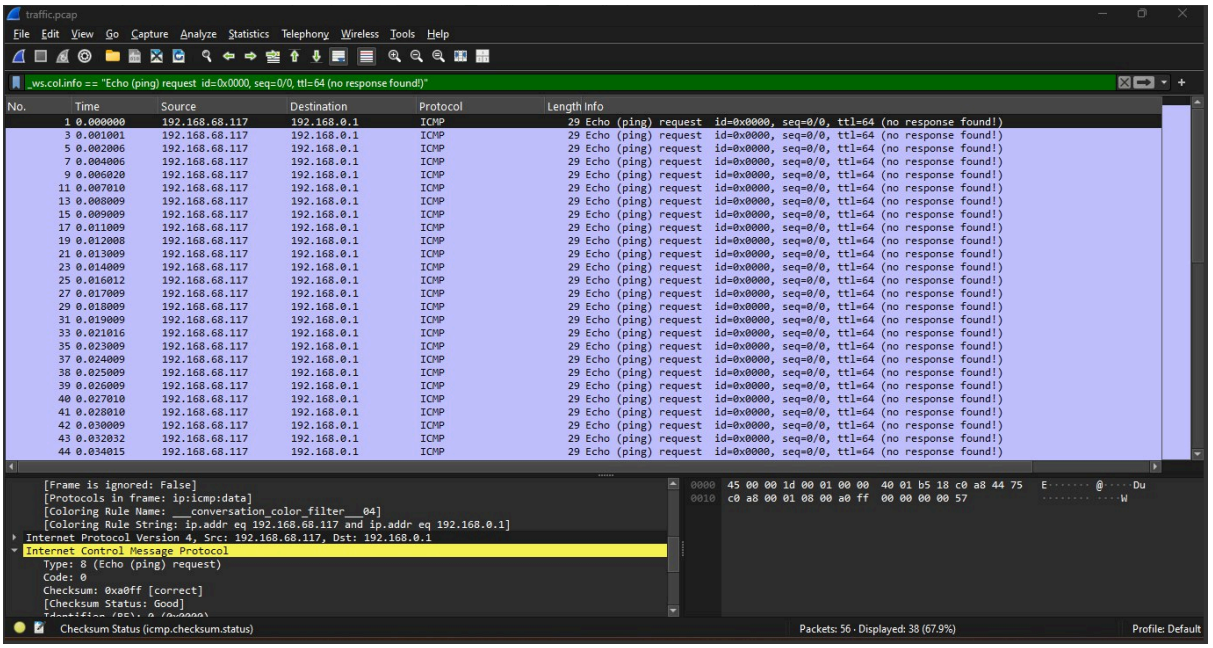
Step 3: Attempting to Use the Plugin

- Following the instructions from the GitHub repository, I attempted to integrate the cheat menu plugin to alter the game's behavior or retrieve the flag.
- Due to time constraints (only one hour remaining before the challenge ended), I could not successfully implement or test this approach.

FORENSIC

I Can’t Manipulate People

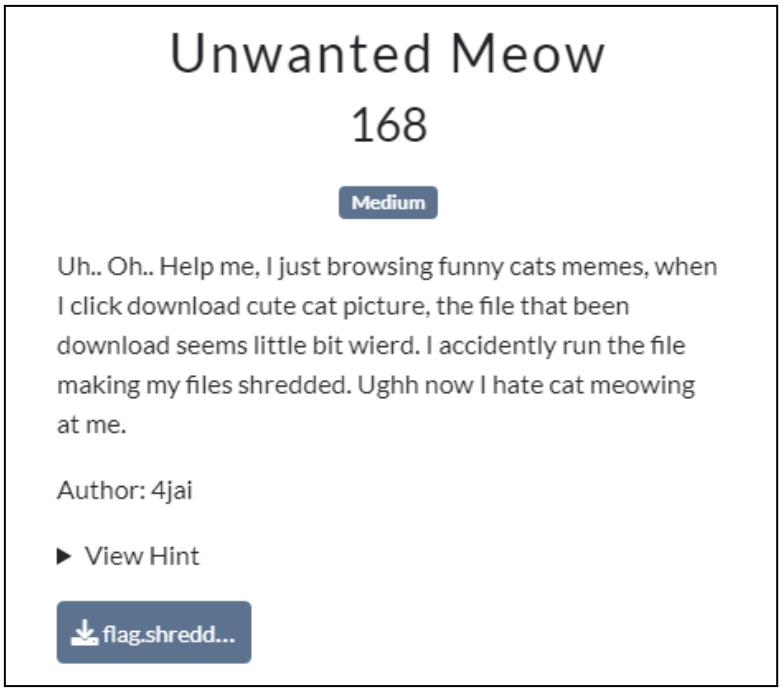
In this challenge, I tried to analyse the pcap packet that is given. I opened the ICMP packets and noticed the data of the packets is different in each one of them.



I then collected the data of all of the ICMP packets and realised that it’s a flag.

Flag: wgmy{1e3b71d57e466ab71b43c2641a4b34f4}

Unwanted Meow



Analysis & Solution

Step 1: Figuring out the issue

- When I download the attachment, it is corrupted and just a flag image.
- I checked the file type and exiftool. The image should've been a cat based on the exiftool information.
- So, I went to take a look at the strings and hex. And that's when I noticed that the hex is kind of weird, so I compared it to a sample for JFIF and yup now I know…

Step 2: Fixing the Hex Dump

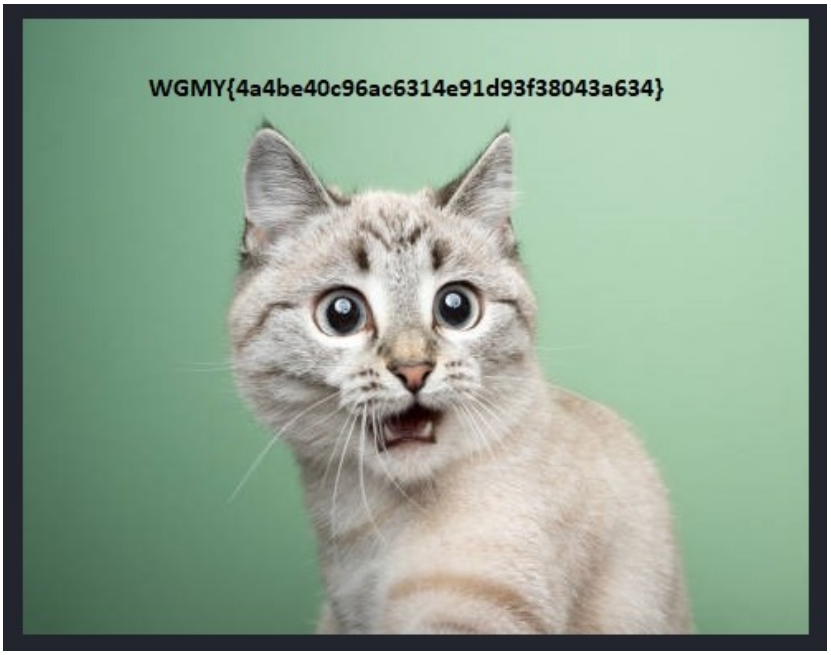
- I went to a website to edit the hex dump (https://hexed.it/) from the .jfif image.
- There is a repetitive 'meow' inside the hex, it is not supposed to be there.
- So I removed all the meow in the hex by replacing them. Because it is easier to just replace them all with a button than trying to scroll manually.

-Untitled- × flag.shredded ×

```
00000000  FF D8 FF E0 00 10 4A 46 49 46 00 01 01 6D 65 6F   ╪ α..JFIF...meo
00000010  77 01 01 2C 01 2C 00 00 FF E1 00 AC 45 78 69 66  w..,.,.. ß.¼Exif
00000020  00 00 4D 4D 00 2A 00 00 00 08 00 04 01 0E 00 02  ..MM.*..........
00000030  00 00 00 56 00 00 00 3E 01 12 00 03 00 00 00 01  ...V...>........
00000040  00 01 00 00 01 1A 00 05 00 00 00 01 00 00 00 94  ..............ö
00000050  01 1B 00 05 00 00 00 01 00 00 00 9C 00 00 00 00  ...........£....
00000060  66 75 6E 6E 79 20 63 61 74 20 6C 6F 6F 6B 69 6E  funny cat lookin
00000070  67 20 73 68 6F 63 6B 65 64 20 77 69 74 68 20 6D  g shocked with m
00000080  6F 75 74 68 20 6F 70 65 6E 20 70 6F 72 74 72 61  outh open portra
00000090  69 74 20 6F 6E 20 67 72 65 65 6E 20 62 61 63 6B  it on green back
000000A0  67 72 6F 75 6E 64 20 77 69 74 68 20 63 6F 70 79  ground with copy
```

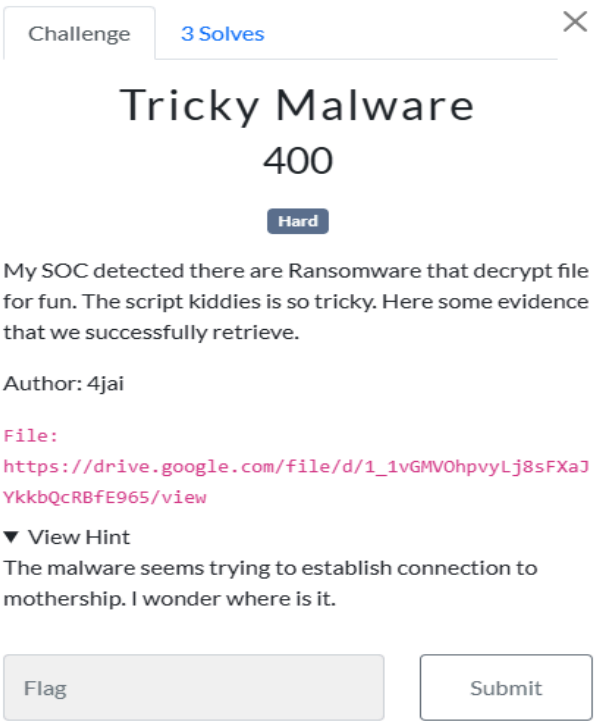
Step 3: Save the fixed file and retrieve the flag

- When the hex is cleared of ‘meow’. Save it and now, a really cute cat image with a flag came out :D



Flag: WGMY{4a4be40c96ac6314e91d93f38043a634}

Tricky Malware



The challenge provided us with a 'Evidence.rar' with 'network.pcap' file and 'memdump.mem' file inside.

Analysis

Step 1: Examining the memory dump file.

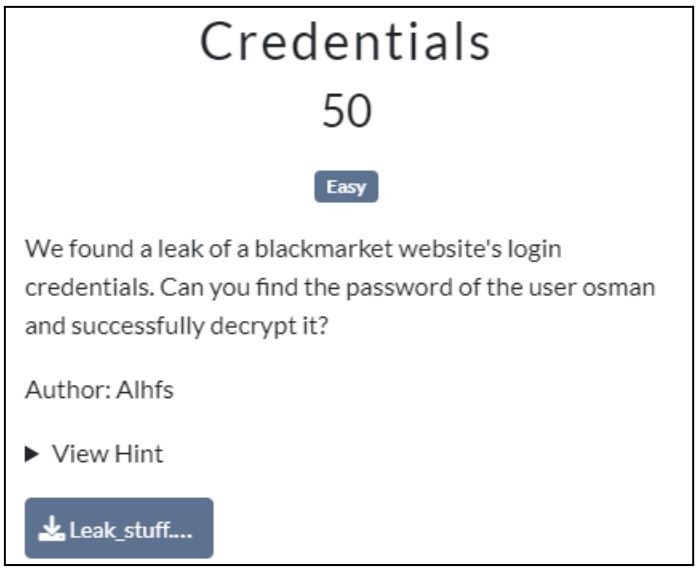
- I began by analyzing the provided memory dump file, 'memdump.mem'. My main objective was to identify any patterns or anomalies that might indicate the presence of a flag or hidden information
- I used Volatility3 to carve out useful information from the memdump.mem file, such as active processes, loaded modules, network connections, and potential artifacts.

Step 2: Discovering suspicious activity

- As I progressed, I identified a few suspicious areas, such as text that includes references to known ransomware families and detection names used by antivirus software. However, each of these leads either did not provide concrete results or required further exploration that I couldn't fully complete.
- I didn't get the time to explore the .pcap file before the competition ended.

CRYPTO

Credentials



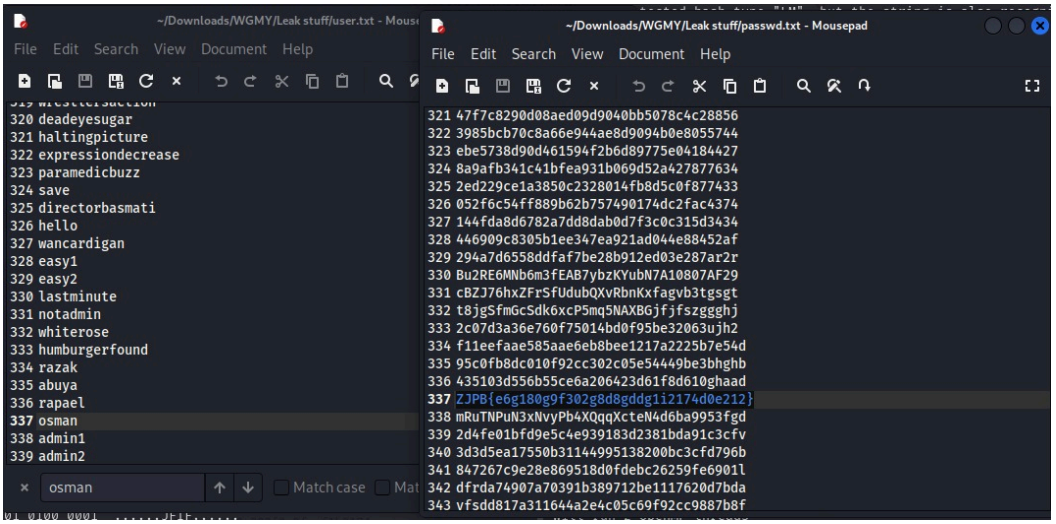
Analyse & Solution

Step 1: Analyse the files given

- I got a zipped file with 2 txt.file inside (passwd.txt and user.txt). Inside both txt files there are 505 user and password, so it's easy to look for the user's password.

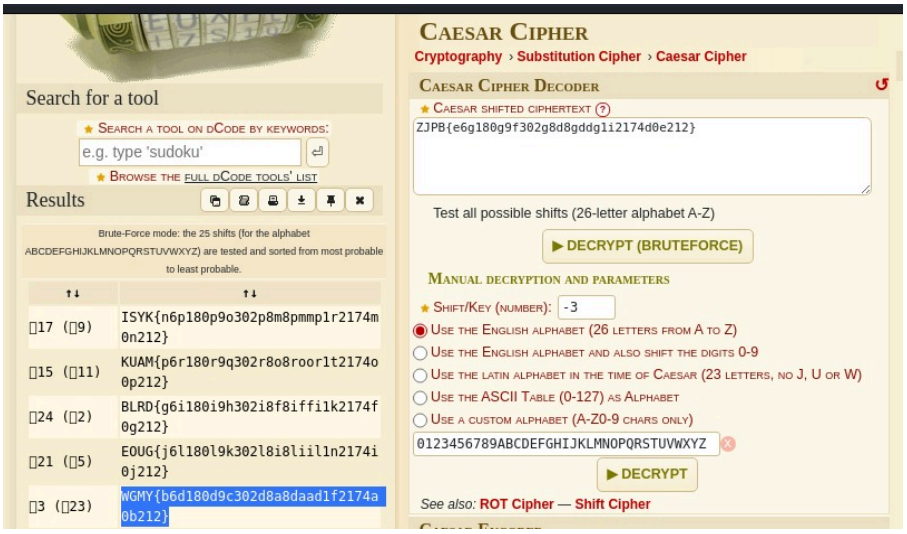
Step 2: Looking for the user mentioned in the description

- I went to search for user osman, and found the password at number 337, but yeah the password is encrypted but that definitely looks like a flag format.



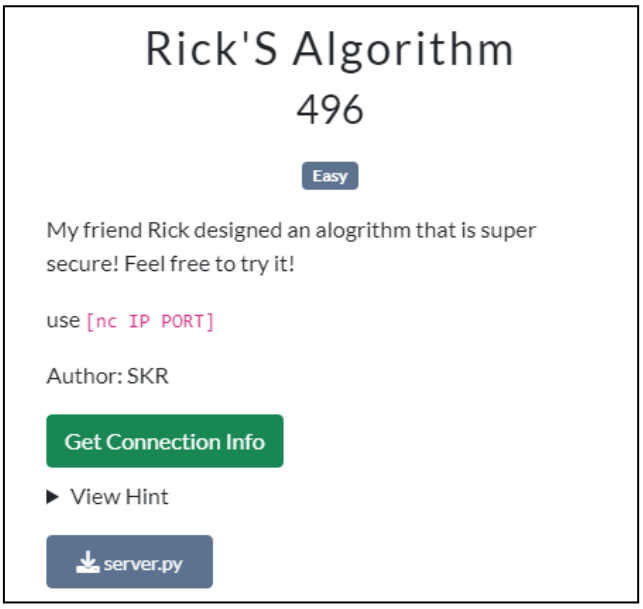
Step 3: Decrypt the password

- So, I went to *dcode.fr* to cipher it. I tried all the possible ciphers, and it turns out it is Caesar Cipher.



Flag: WGMY{b6d180d9c302d8a8daad1f2174a0b212}

Rick’S Algorithm

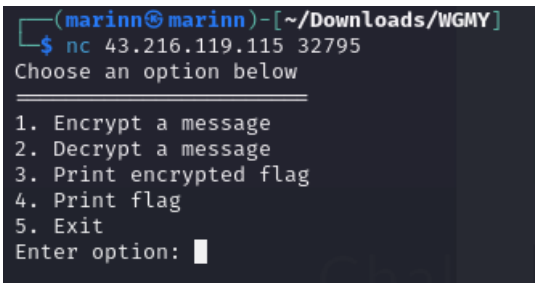


Well, the title looks fun so I tried… Unfortunately, I did not manage to find the flag because I do not understand the algorithm and modulus.

Analysis

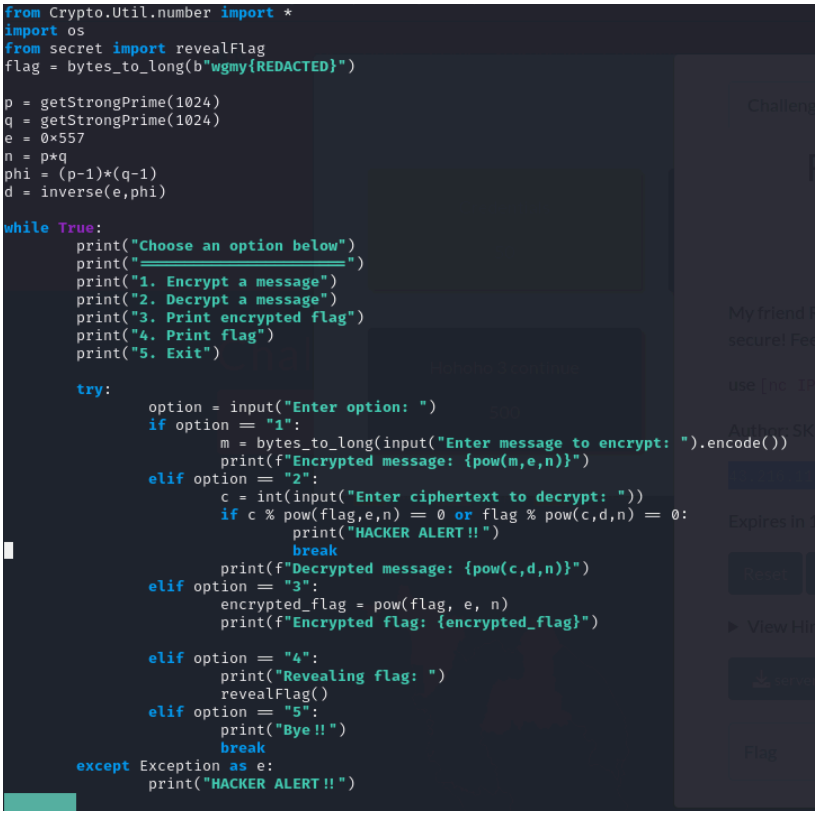
Step 1: Connecting to the server

- What I have tried is to enter the server using connection info and there are 5 options.
- I tried all options, got rickrolled… tried to decrypt the encrypted flag, and well.



Step 2: Analysing the coding

- I went to look inside the python code, I noticed the modular arithmetic, and from what I remember, I did not like math that much.
- I think I kind of understand how the server do encrypt/decrypt



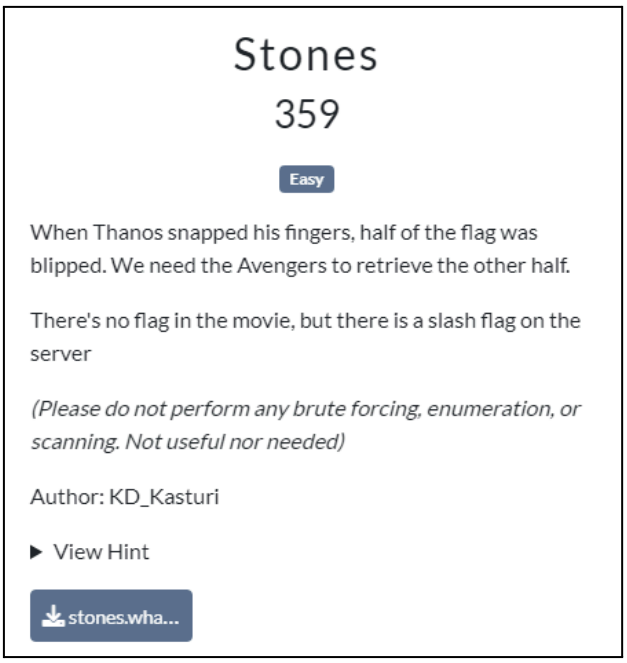
```python
from Crypto.Util.number import *
import os
from secret import revealFlag
flag = bytes_to_long(b"wgmy{REDACTED}")

p = getStrongPrime(1024)
q = getStrongPrime(1024)
e = 0x557
n = p*q
phi = (p-1)*(q-1)
d = inverse(e,phi)

while True:
        print("Choose an option below")
        print("=========================")
        print("1. Encrypt a message")
        print("2. Decrypt a message")
        print("3. Print encrypted flag")
        print("4. Print flag")
        print("5. Exit")

        try:
                option = input("Enter option: ")
                if option == "1":
                        m = bytes_to_long(input("Enter message to encrypt: ").encode())
                        print(f"Encrypted message: {pow(m,e,n)}")
                elif option == "2":
                        c = int(input("Enter ciphertext to decrypt: "))
                        if c % pow(flag,e,n) == 0 or flag % pow(c,d,n) == 0:
                                print("HACKER ALERT!!")
                                break
                        print(f"Decrypted message: {pow(c,d,n)}")
                elif option == "3":
                        encrypted_flag = pow(flag, e, n)
                        print(f"Encrypted flag: {encrypted_flag}")

                elif option == "4":
                        print("Revealing flag: ")
                        revealFlag()
                elif option == "5":
                        print("Bye!!")
                        break
        except Exception as e:
                print("HACKER ALERT!!")
```

Step 3: Changing the script coding

- I tried to use ChatGPT to give me possible coding to reverse this, but none of them work.
- I got stuck during the process of figuring out the variable n.

REVERSE

Stones



I like Avengers so I do this one, even though this is my first time doing the reverse. And I forgot the steps I actually did during the game, but I will try my best to explain here.

Analysis & Solution

Step 1: First attempt to check the vulnerabilities

- So when the hint dropped about travel back in time, I tried to change my kali's date one by one and by using script too, but the system just said I need a bigger brain, wrong date….

Step 2: Decompile the file

- So I really wanted to look at the coding, so I decompiled it and found the coding that provides server ip address and port.
- Also, the first flag chunk, so I tried to understand how it works based on the coding.

Step 3: Accessing the server

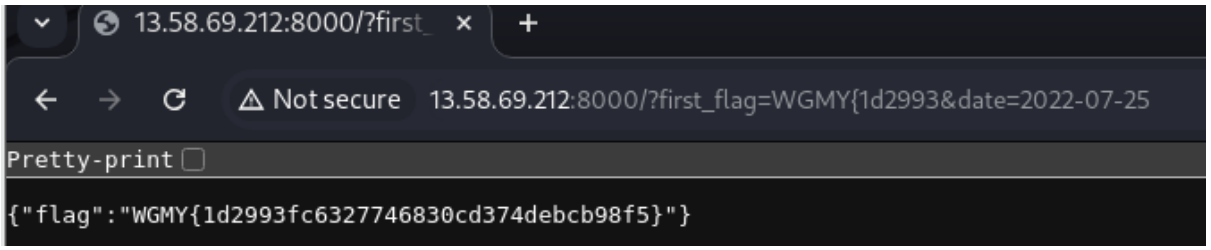
- I went to the server address/flag by using curl. There is an upload date and youtube link.
- But the youtube link provided is Avengers and the upload date is 25 July 2022. Once again, I tried to change the date but I was still wrong.

```
┌──(marinn㉿marinn)-[~/Downloads/WGMY]
└─$ curl "http://13.58.69.212:8000/flag"
{"Upload Date":"https://youtu.be/V0zJb2K4Yi8?si=xUTuXD3ppkJpU2Nw&t=75"}
```

Step 4: URL Manipulation based on the coding

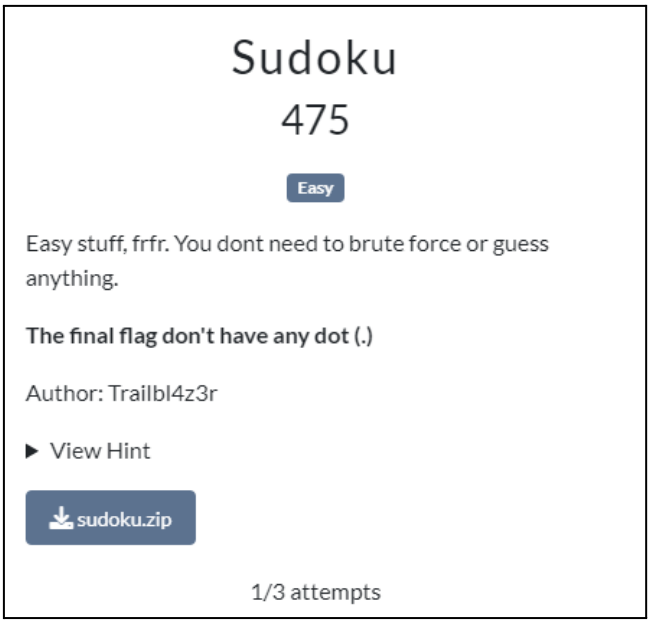
- I asked my best friend, ChatGPT and it advised me to time travel via URL Manipulation but this time straight to the browser since curl did not work.

```
┌──(marinn㉿marinn)-[~/Downloads/WGMY]
└─$ curl "http://13.58.69.212:8000/?first_flag=WGMY{1d2993&date=2022-07-25"
curl: (3) unmatched brace in URL position 42:
http://13.58.69.212:8000/?first_flag=WGMY{1d2993&date=2022-07-25
                                         ^
```



Flag: WGMY{1d2993fc6327746830cd374debcb98f5}

Sudoku



I like Sudoku too, so I tried it. But, no flag found……..

Analysis

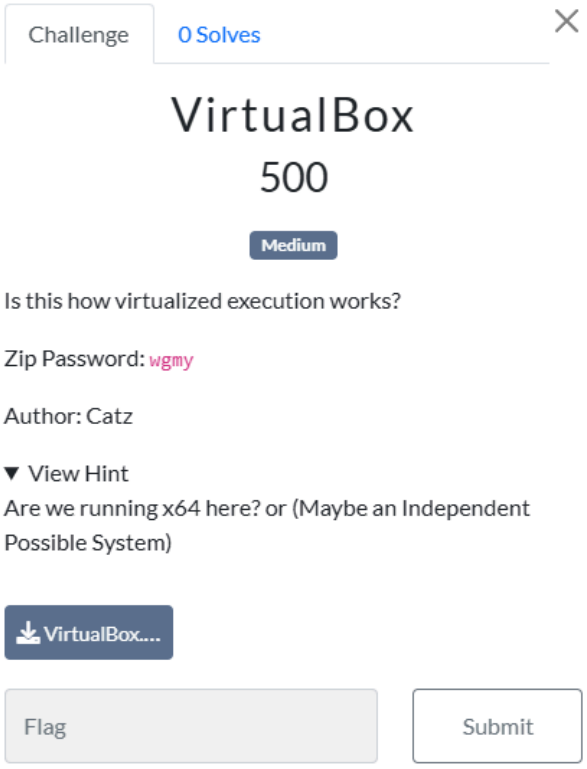
Step 1: Analysing the vulnerabilities

- Inside the sudoku.zip is out.enc and sudoku.exe.
- I tried to decrypt the out.enc content because it does look like a flag format, but I think it is maybe a key for the sudoku.exe

Step 2: Executing the Sudoku.exe

- To be very honest, I don't know what to do with these. I tried to run sudoku.exe using wine to get a clue, but it does not work. I also tried to decompile but I cannot find it.

Virtual Box



The challenge provided us with a 'ViritualBox.zip' folder with a 'check.exe' file contained inside.

Analysis

Step 1: Analysing the file

Using the file command, I am able to identify the file type as a Windows PE32+ binary.

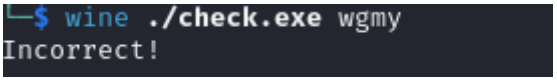
```
└─$ file check.exe
check.exe: PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows, 8 sections
```

Step 2: Observing file execution behaviour

- Attempted to execute the file on a Linux environment using wine:

```
└─$ wine ./check.exe
Usage: Z:\home\kali\Desktop\check.exe <flag>
```

- This indicates that the binary expects a single argument (<flag>), which will likely be validated internally.

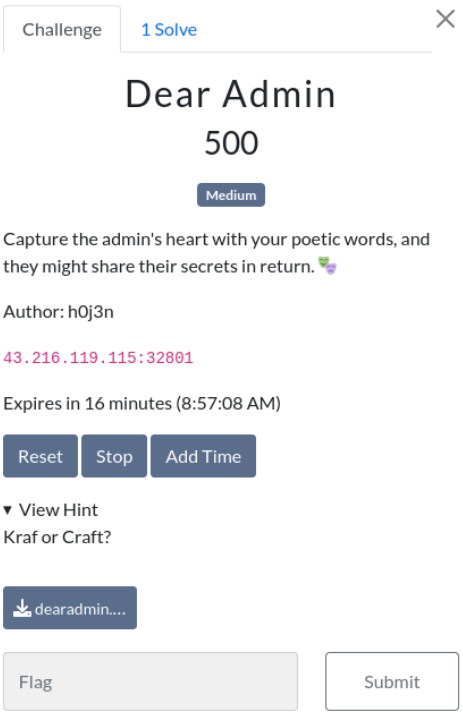


Step 3: Exploring MIPS (Microprocessor without Interlocked Pipeline Stages)

After checking the hint, I realised that it is running on MIPS architecture but due to my limited knowledge and short of time I was not able to fully analyse the binary and get the challenge done.

WEB

Dear Admin

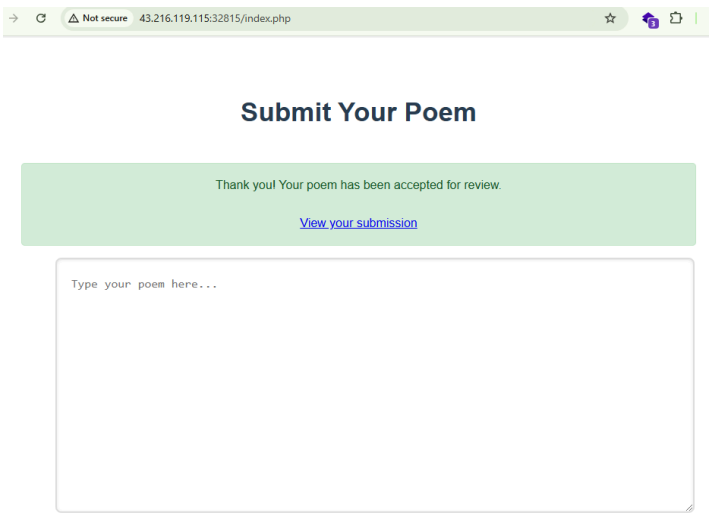


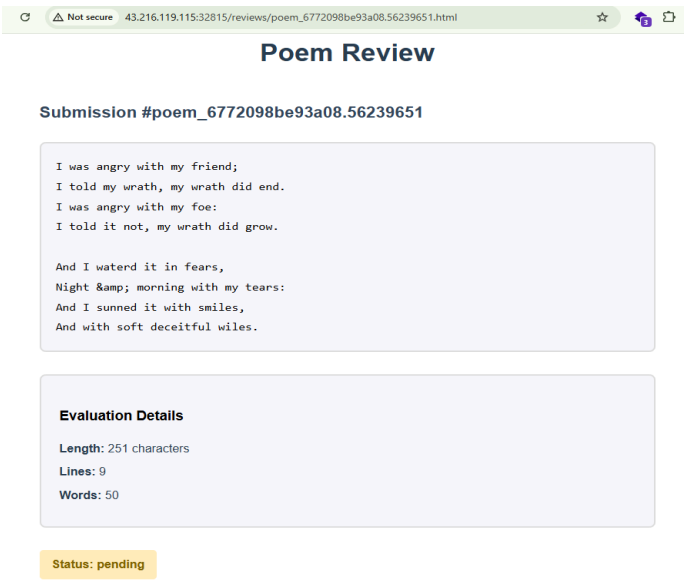
The challenge provides us with a link to the website and its source codes.

Analysis

Step 1: Exploring the Website

- The website contained a text box where we could insert a poem and see it on a .html file.





- As user input was allowed, I attempted various common input exploitation techniques such as:
    - **SQL Injection**
    - **Cross-Site Scripting (XSS)**
    - **Command Injection**
- Unfortunately, none of my approaches yielded any success.

Step 2: Analyzing the Hint

- After struggling with the initial attempts, I revisited the hint provided for the challenge. The hint mentioned **"kraf"** or **"craft".**
- This led me to research on anything regarding web that is related to those keywords.

Step 3: Discovering Craft CMS

- Through my research, I came across **Craft CMS**, a content management system I had not encountered before.

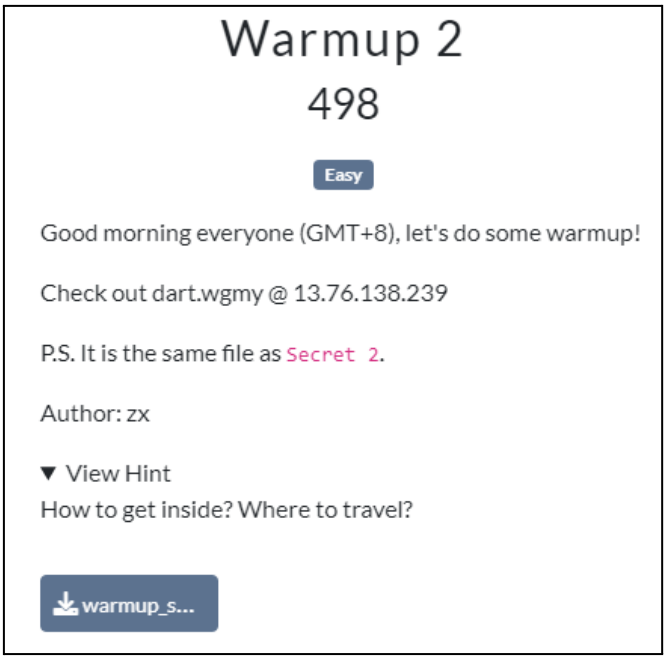
Craft is a self-hosted PHP application, built on Yii 2. It can connect to MySQL and PostgreSQL for content storage. Templating is powered by Twig.

- I explored potential vulnerabilities in Craft CMS and found an exploit on GitHub: CraftCMS_CVE-2023-41892.

Step 4: Attempting the Exploit

- I tried applying the exploit mentioned in the GitHub repository to the challenge website. Unfortunately, the exploit did not work for me. I hit a wall and moved on to another question.

Warm Up 2



Analysis

Step 1: Analyse the possible vulnerabilities

- Tried to do this challenge after the hint dropped, it sounds like Path Traversal.
- Analysing all the contents inside the chall.

Step 2: Trying to do Path Traversal

- I added /../../ direct to the URL in the browser, but still could not access.
- Even tried curl on the kali terminal, still cannot.

WordMarket

Tried this one, because it is WordPress. I used to be familiar with WordPress but still lack knowledge in exploitation.

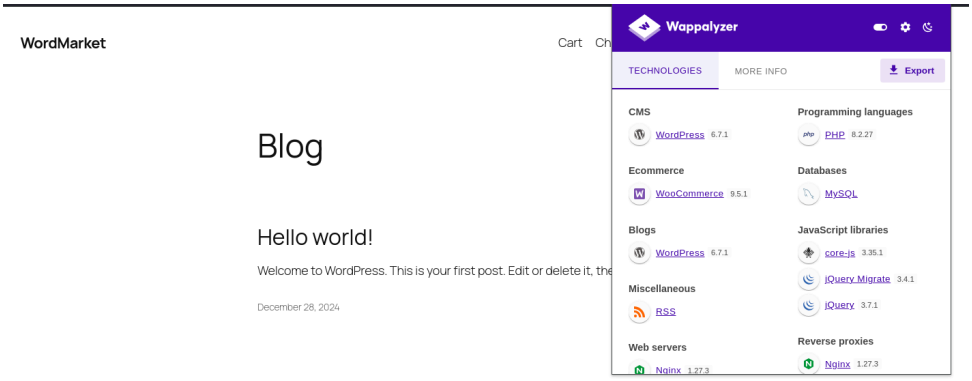
Analysis

Step 1: Analysing the website and vulnerabilities

- I unzipped the wordmarket.zip and there are many files.
- What I noticed is the Wordpress's plugin for this challenge which is…… WooCommerce and the city's shipping zone.
- So I tried to find CVE related questions and found this **CVE-2024-47309**.

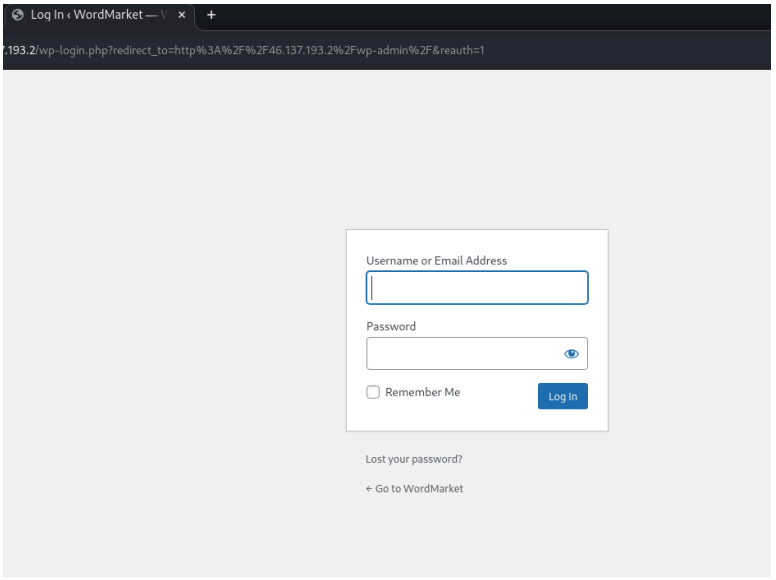
Step 2: Looking around in WordPress

- First of all, open the link given in the description, it is the client side of the WordPress website.



Step 3: Trying to access the WordPress's Admin Panel

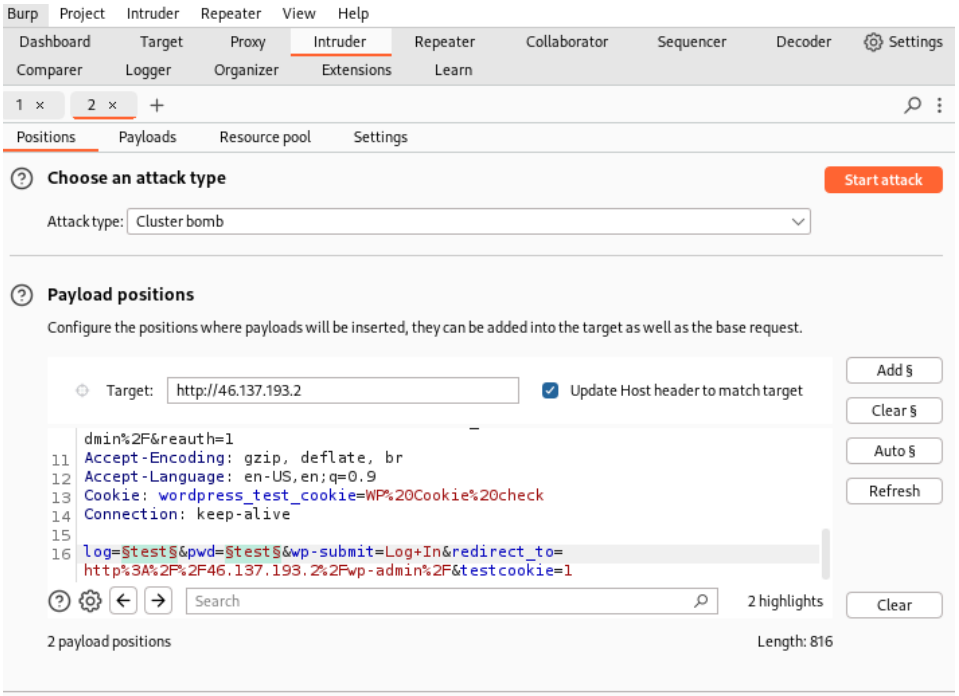
- I tried to access the wp-admin WordPress, so I added /wp-admin to the URL.

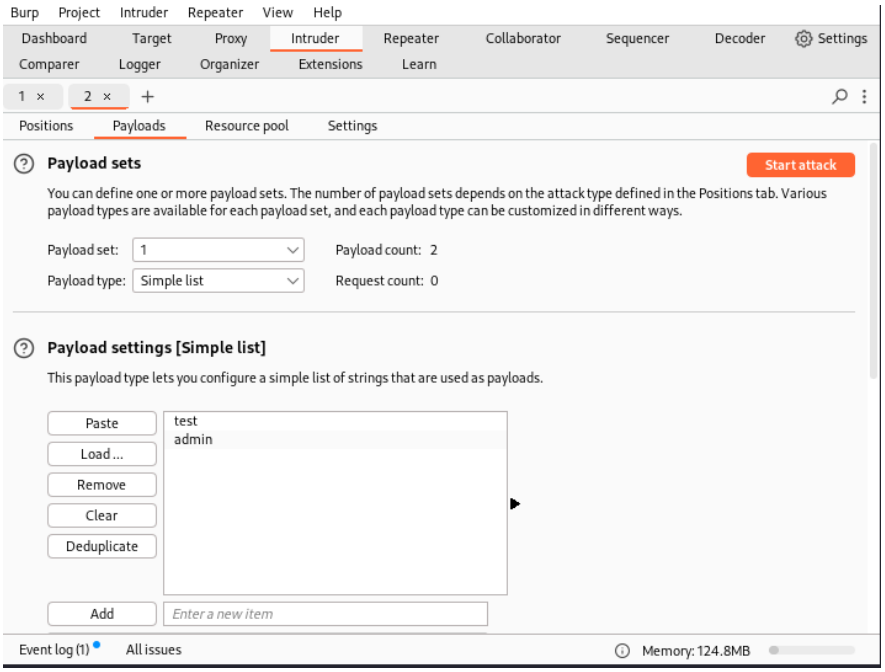


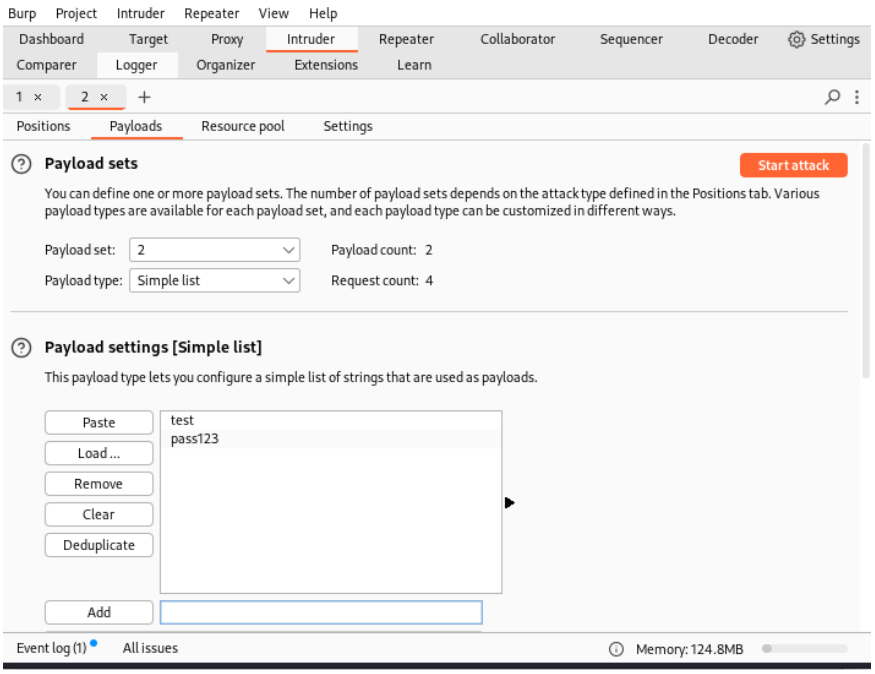
- Now, this is where I am stuck. I could not find the admin login and password to access the Admin Panel.
- Then, I tried BurpSuite to intercept and look around in Site Map and Proxy.

Step 4: Intercept the wp-admin page using BurpSuite

- I tried Bruteforce but it's still not working, maybe there is another way. But what I did was I tried to login with a random username and password on the browser, then send the request to the intruder.
- I went to read a write-up about accessing WordPress Admin. So, I tried to do a payload attack Cluster bomb. It does not give me the correct admin username and password though.


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Settings
Comparer Logger Organizer Extensions Learn
1 × 2 ×
Positions Payloads Resource pool Settings
Choose an attack type
Start attack
Attack type: Cluster bomb
Payload positions
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.
Target: http://46.137.193.2
Update Host header to match target
Add §
Clear §
Auto §
Refresh
dmin%2F&reauth=1
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: en-US,en;q=0.9
13 Cookie: wordpress_test_cookie=WP%20Cookie%20check
14 Connection: keep-alive
15
16 log=§test§&pwd=§test§&wp-submit=Log+In&redirect_to=
http%3A%2F%2F46.137.193.2%2Fwp-admin%2F&testcookie=1
Search
2 highlights
Clear
2 payload positions
Length: 816


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Settings
Comparer Logger Organizer Extensions Learn
1 × 2 ×
Positions Payloads Resource pool Settings
Payload sets
Start attack
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.
Payload set: 1
Payload count: 2
Payload type: Simple list
Request count: 0
Payload settings [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
Deduplicate
test
admin
Add
Enter a new item
Event log (1)
All issues
Memory: 124.8MB



So, just until here. Could not even enter the Wordpress Admin Panel, maybe it is actually easy or even provided but well, still learning…..

OSINT

迅帝

Analysis

Step 1: Analyse the title and description

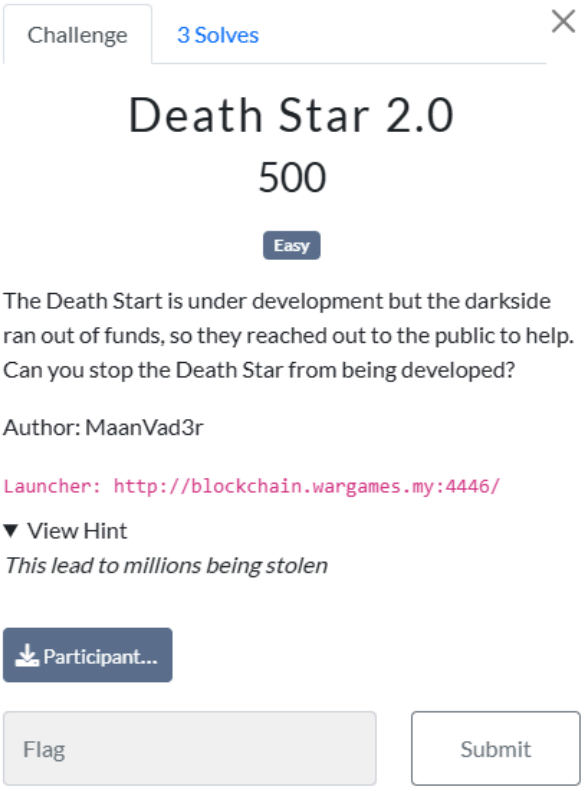
- I translated the title and looked around about Melancholy Angel.
- Eventually, I found out about a game ITC Xbox360 before the hint.
- But, I do not know what to do with the 2 files it give, which is BUILD.DAT and BUILD.TOC

Step 2: Researching the .DAT and .TOC file

- I tried to research the .DAT and .TOC file, but I am unsure what to do with these two files.
- Could not find the tools to do these, well that's why it is OSINT.

BLOCKCHAIN

Death Star 2.0



The challenge provided a link to an Ethernet Launcher where we could obtain the credentials needed to interact with a blockchain hosted on an RPC chain. Additionally, it included a file named 'Participants_2.0.zip' containing three Solidity (primary language for blockchain) source files: 'DarksidePool.sol, 'DeathStar.sol' and 'Setup.sol'.

Analysis

Step 1: Examining the Source Codes

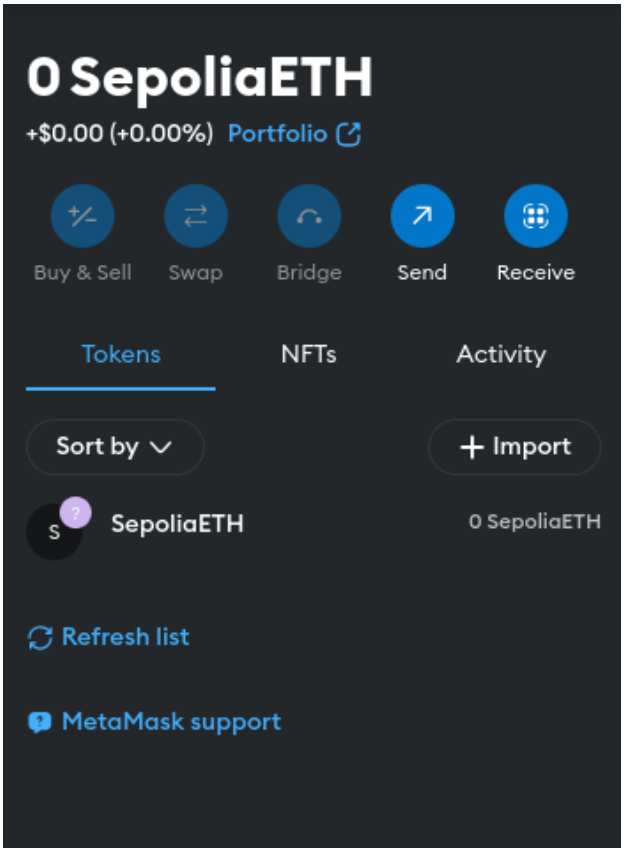
- I began by carefully examining the code provided to understand how the system functioned.
- While researching, I referred to a blog on Security Vulnerabilities in Web3 and Smart Contracts. This helped me focus on identifying potential flaws in the contract.
- To assist with my analysis, I used ChatGPT to gain deeper insights into the code. It suggested that the contract might be vulnerable to Reentrancy Attacks, a common issue in Ethereum-based smart contracts.

2. Reentrancy Attacks

Smart contracts on the Ethereum blockchain can have several types of vulnerabilities, and one common and severe vulnerability is the reentrancy attack. This type of attack occurs when a contract calls an external contract, and the external contract calls back into the original contract before the first call is finished, leading to unexpected behavior and potential exploitation.

Step 2: Interacting with the Blockchain

- To exploit the vulnerability, interacting with the blockchain required gas money in the form of ETH. Unfortunately, I didn't have any ETH in my wallet (I am broke) and couldn't find a reliable faucet to obtain free ETH within the 24-hour competition timeframe.



- Despite this, I was able to learn a great deal about blockchain vulnerabilities and smart contract exploitation. If anyone knows of a reliable source for free ETH, please share!