



## WARGAMES MY 2024 mikimaus' writeup

Writeup made by:  
4ngs4  
hazo  
R4N4WB0I

mikimaus  
Malaysia  
24th place  
1293 points

Members

User Name	Score
4ngs4	834
R4N4WB0I	50
hazo <small>Captain</small>	409

*We placed 24th on wgmy24!*

# Table of Content

<b>CHALLENGES</b>		<b>PAGE</b>
<b>GAMES</b>		
1. World 1.....		3
<b>REVERSE</b>		
2. Stones.....		9
<b>CRYPTO</b>		
3. Credentials.....		12
<b>MISC</b>		
4. Christmas GIFT.....		14
5. Invisible ink.....		16
6. The DCM Meta.....		19
<b>FORENSIC</b>		
7. Unwanted Meow.....		21
8. I Cant Manipulate People.....		23

## GAME SOLVED (1/3)

### 1. World 1



First we were given a World1.exe file and by looking at the interface and template, we could conclude that it is a game made with rpg maker mz.



Just like the classic rpg games, we have to save princesses and beat these bosses so I just walk through all the bosses and gain flags.

wgmy{5ce



Flag 1

7d7a7140



Flag 2

Flag Three:

ebabf5cd



Flag 3



Everything was a smooth sail until I met this absolutely-not-overpowered-boss named Antares (his skill literally dealt 84,200,353). Thus I tried to fight him over and over again and hopefully I would land a critical strike.

file10.rmmzsave

Gold	0						
Param set #1 (You):							
MHP	0	MMP	0	ATK	1000000000000000	DEF	0
MAT	0	MDF	0	AGI	0	LUK	0
MyHP	838	MyMP	76				
Inventory:							
Item#33 :(1)		Item#34 :(1)					
The item names is stored here \www\data\items.json, you can upload this file to the save editor.							

On my 20th attempt (yes, I counted), I felt frustrated and realized that I could temper the save file with this sweet save editor I found online.



Then I tried fighting the boss again with my pure skill and technique (hacking technique). I managed to drop an absolutely reasonable skill. Trust me it is super satisfying dealing this much damage against someone who one-shotted you for 20 times.



Then this NPC with an office suit in a castle appeared out of nowhere and told me that the princess was in another castle. Thankfully he gave me a “code”. Since i realized w is the 23th letter in alphabet i knew the password is “wgmy”



I was given a QR code. God forbid these authors to ease our life (jk i love the authors). The flag obtained from the QR is “3fcaac2}”.



Then I realized there are only 4 flags which means I missed the 4th flag.

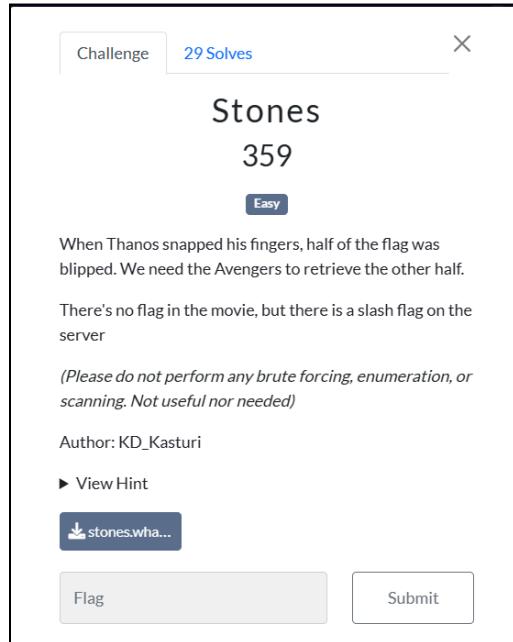


After 15 minutes of exploring the map from the start, I finally found the 4th flag on the volcano map.

Flag: wgmy{5ce7d7a7140ebabf5cd43effd3fcaac2}

# Reverse Solved(1/4)

## 1. Stones



We were provided with a “stones.whatdis” i wonder what file is it

```
└$ file stones.whatdis
stones.whatdis: PE32+ executable (console) x86-64, for MS Windows, 6 sections
```

By using file command, we can see that it is an exe file so I just rename it to .exe.

```
└$ ./stones.exe
We're gonna need a really big brain; bigger than his?
Traceback (most recent call last):
  File "urllib3\connection.py", line 198, in _new_conn
    File "urllib3\util\connection.py", line 85, in create_connection
      File "urllib3\util\connection.py", line 73, in create_connection
TimeoutError: [WinError 10060] A connection attempt failed because the connected party did not properly respond after a
period of time, or established connection failed because connected host has failed to respond

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "urllib3\connectionpool.py", line 787, in urlopen
    File "urllib3\connectionpool.py", line 493, in _make_request
      File "urllib3\connection.py", line 445, in request
        File "http\client.py", line 1278, in endheaders
          File "http\client.py", line 1038, in _send_output
            File "http\client.py", line 976, in send
              File "urllib3\connection.py", line 276, in connect
                File "urllib3\connection.py", line 207, in _new_conn
urllib3.exceptions.ConnectTimeoutError: (<urllib3.connection.HTTPConnection object at 0x000001E486AE49D0>, 'Connection t
o 3.142.133.106 timed out. (connect timeout=None)')
```

I tried to run it but seems to be there is an error with the file and it is from a .py file so now we knew that this exe is created using python

```

└$ ls
api-ms-win-core-console-l1-1-0.dll      api-ms-win-core-sysinfo-l1-1-0.dll      _hashlib.pyd
api-ms-win-core-datetime-l1-1-0.dll       api-ms-win-core-timezone-l1-1-0.dll    libcrypto-1_1.dll
api-ms-win-core-debug-l1-1-0.dll          api-ms-win-core-util-l1-1-0.dll       libssl-1_1.dll
api-ms-win-core-errorhandling-l1-1-0.dll   api-ms-win-crt-conio-l1-1-0.dll       _lzma.pyd
api-ms-win-core-file-l1-1-0.dll           api-ms-win-crt-convert-l1-1-0.dll     pyiboot01_bootstrap.pyc
api-ms-win-core-file-l1-2-0.dll           api-ms-win-crt-environment-l1-1-0.dll  pyimod01_archive.pyc
api-ms-win-core-file-l2-1-0.dll           api-ms-win-crt-fsystem-l1-1-0.dll     pyimod02_importers.pyc
api-ms-win-core-handle-l1-1-0.dll         api-ms-win-crt-heap-l1-1-0.dll       pyimod03_ctypes.pyc
api-ms-win-core-heap-l1-1-0.dll          api-ms-win-crt-locale-l1-1-0.dll     pyimod04_pywin32.pyc
api-ms-win-core-interlocked-l1-1-0.dll    api-ms-win-crt-math-l1-1-0.dll      pyi_rth_inspect.pyc
api-ms-win-core-libraryloader-l1-1-0.dll  api-ms-win-crt-process-l1-1-0.dll    python310.dll
api-ms-win-core-localization-l1-2-0.dll   api-ms-win-crt-runtime-l1-1-0.dll    PYZ-00.pyz
api-ms-win-core-memory-l1-1-0.dll        api-ms-win-crt-stdio-l1-1-0.dll     PYZ-00.pyz_extracted
api-ms-win-core-namedpipe-l1-1-0.dll     api-ms-win-crt-string-l1-1-0.dll    _queue.pyd
api-ms-win-core-processenvironment-l1-1-0.dll  api-ms-win-crt-time-l1-1-0.dll     select.pyd
api-ms-win-core-processthreads-l1-1-0.dll api-ms-win-crt-util-l1-1-0.dll     _socket.pyd
api-ms-win-core-processthreads-l1-1-1.dll base_library.zip      _ssl.pyd
api-ms-win-core-profile-l1-1-0.dll       _bz2.pyd                         struct.pyc
api-ms-win-core-rtlsupport-l1-1-0.dll    certifi                          ucrtbase.dll
api-ms-win-core-string-l1-1-0.dll        CHAL-stones.pyc                  unicodedata.pyd
api-ms-win-core-synch-l1-1-0.dll         charset_normalizer             VCRUNTIME140.dll
api-ms-win-core-synch-l1-2-0.dll         _decimal.pyd

```

Extract the python file from exe using pyinstxtractor and this is the result. We can see there are a lot of files but there is an eye-catching file named "CHAL-stones.pyc"

```

└$ cat beep.py
# Source Generated with Decompyle++
# File: CHAL-stones.pyc (Python 3.10)

import requests
from datetime import datetime
from urllib.request import urlopen
from datetime import datetime
server_url = 'http://3.142.133.106:8000/'
current_time = urlopen('http://just-the-time.appspot.com/')
current_time = current_time.read().strip()
current_time = current_time.decode('utf-8')
current_date = current_time.split(' ')[0]
local_date = datetime.now().strftime('%Y-%m-%d')
if current_date == local_date:
    print("We're gonna need a really big brain; bigger than his?")
first_flag = 'WGMY{1d2993'
user_date = current_date
params = {
    'first_flag': first_flag,
    'date': user_date }
response = requests.get(server_url, params, **('params',))
if response.status_code == 200:
    print(response.json()['flag'])
    return None
None(response.json()['error'])

```

By using pycdc, we can decompile the "CHAL-stones.pyc" file and read the content. I threw the code into chatgpt to explain it for me

The program fetches the current date from an external service.  
It compares the external date with the local system date.  
If they match, it sends a GET request to a server at  
<http://3.142.133.106:8000/>, including the date and a partial flag  
(WGMY{1d2993}) as parameters.  
The server is expected to respond with a full flag, which is printed out.

So I tried to send a request to the url by using curl and received a youtube video which was uploaded on 25th July 2022!

```
import requests

# Define the server URL and the parameters
server_url = 'http://3.142.133.106:8000/'
params = {
    'first_flag': 'WGMY{1d2993',
    'date': '2022-07-25'
}

# Send the GET request to the server
response = requests.get(server_url, params=params)

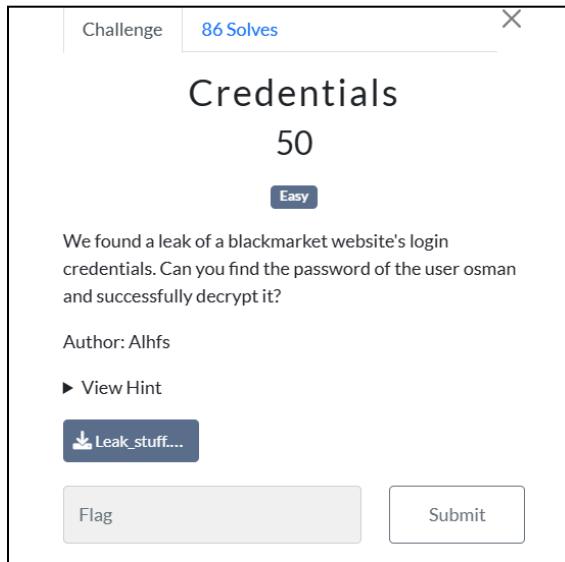
# Check if the response status is OK (200)
if response.status_code == 200:
    # If the response contains the flag, print it
    print("Flag received:", response.json()['flag'])
else:
    # If there was an error, print the error message
    print("Error:", response.json().get('error', 'Unknown error'))
```

As usual, I asked chatgpt politely to modify the program and put the right date to get the flag.

Flag: wgmy{1d2993fc6327746830cd374debc98f5}

# Crypto Solved (1/5)

## 1. Credentials



At first, we were provided with user.txt and passwd.txt. We were asked to find Osman's Password.

```
abuya
rapael
osman
admin1
admin2
user1
user2
```

Ln 337, Col 6 | 6.809 characters

We need to find Osman's password so first we found out that Osman's name is on line 337 thus the password must be on line 337 too.

```
2c07d3a36e760f75014bd0f95be32063ujh2
f11eefaae585aae6eb8bee1217a2225b7e54d
95c0fb8dc010f92cc302c05e54449be3bhghb
435103d556b55ce6a206423d61f8d610ghaad
ZJPB{e6g180g9f302g8d8gddg1i2174d0e212}
mRuTNPuN3xNvypb4XQqqXteN4d6ba9953fgd
2d4fe01bfd9e5c4e939183d2381bda91c3cfv
3d3d5ea17550b31144995138200bc3cf796b
847267c9e28e869518d0fdebc26259fe69011
df6da274097a79201b289742ba1117629d7bda
```

On line 337 in passwd.txt, we could find the ZJPB{e6g180g9f302g8d8gddg1i2174d0e212}. Looks like a flag format!

→3 (←23)

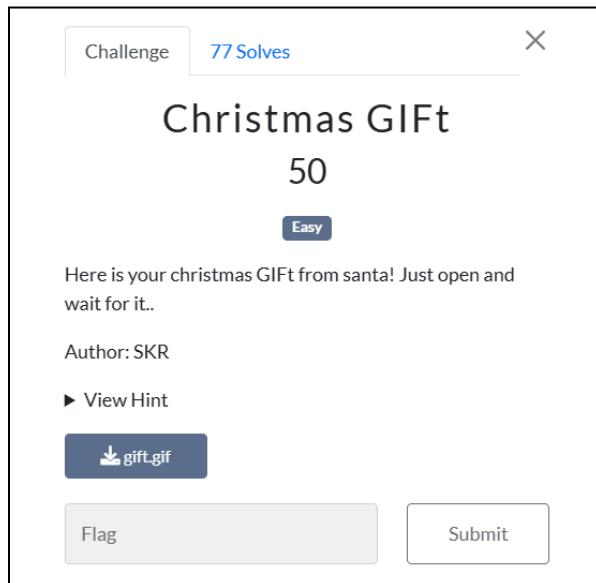
WGMY{b6d180d9c302d8a8daad1f2174a  
0b212}

Since ZJPB = wgmy, we can simply conclude that it is a caesar cipher because it uses the substitution method (z=w and j=g is both 3 steps back). Just put it on decode and bruteforce the output.

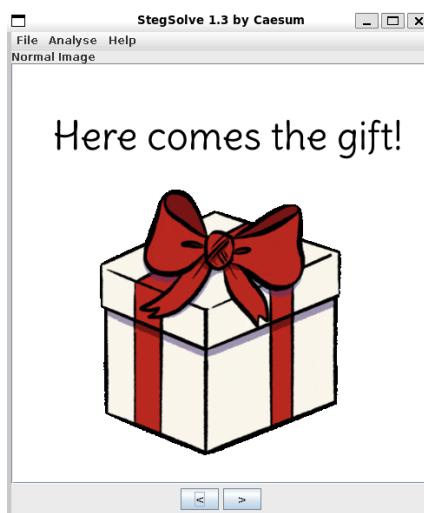
Flag: wgmy{b6d180d9c302d8a8daad1f2174a0b212}

## MISC Solved(3/4)

### 1. Christmas GIft



A Christmas gift! An interesting challenge indeed!



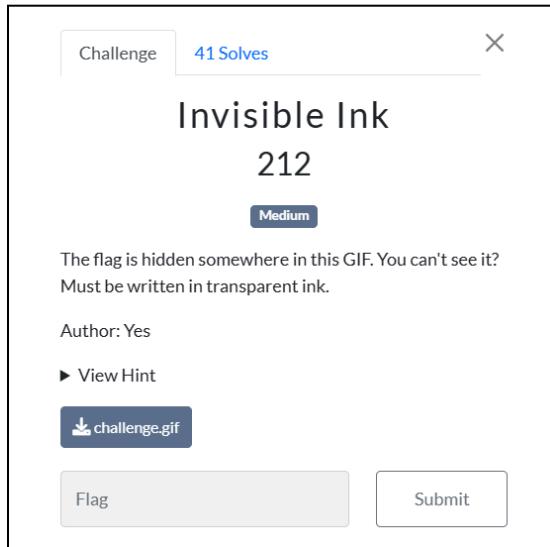
Inspecting the gif, we could guess that this gif exposed something at the end but it is too fast for my weak eyes.. So i decided to use stegsolve to inspect frame by frame



So there are 1402 frames and the flag is in the last frame!

Flag: wgmy{1eaa6da7b7f5df6f7c0381c8f23af4d3}

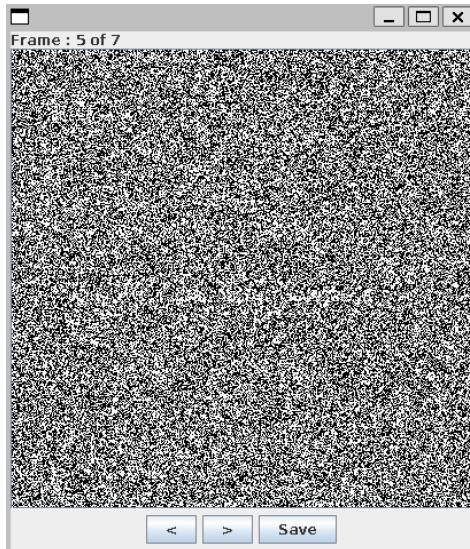
## 2. Invisible ink



We were provided with another interesting gif!



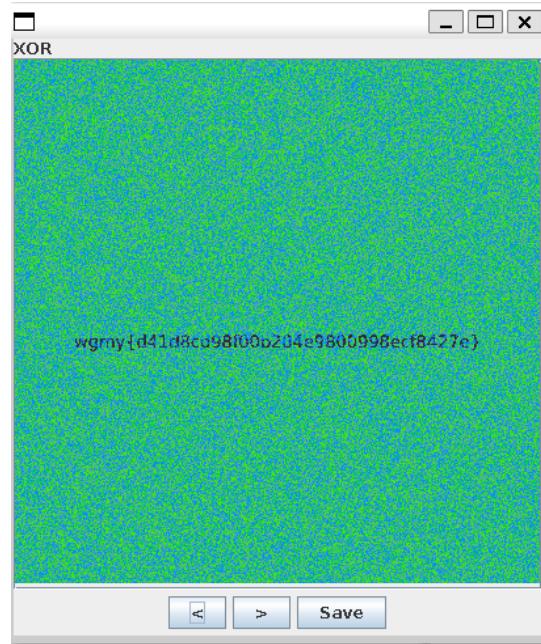
I decided to use stegsolve again to inspect the gif since aperisolve and ezigif didnt help much



By using the frame browser, we could inspect that there are weird images on frame 5 and 6! Save the frames as .bmp (bitmap file)



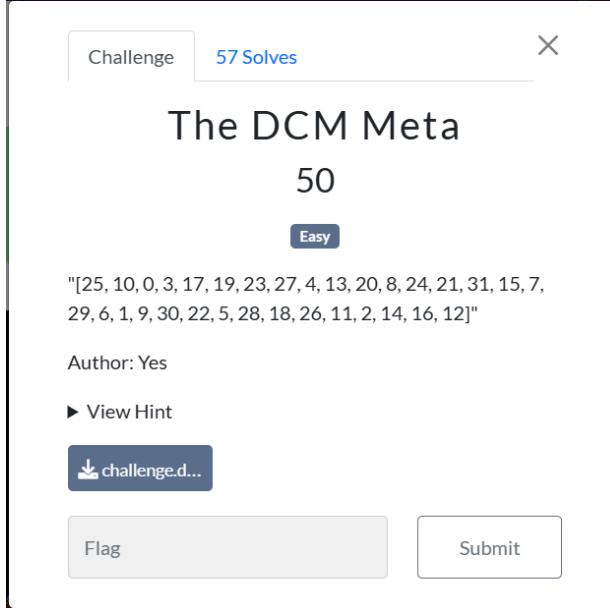
After several minutes of inspecting the bmp in a different colour map, we finally found an incomplete flag. Save this bmp and repeat the same step on another frame.



Then we can use an image combiner to combine both images to form a flag.

Flag: wgmy{d41d8cd98f00b204e9800998ecf8427e}

### 3. The DCM Meta



We were provided with a challenge.dcm file and a sequence of numbers that should help in getting the flag.

```
# Dicom-File-Format
# Dicom-Meta-Information-Header
# Used TransferSyntax: Unknown Transfer Syntax

# Dicom-Data-Set
# Used TransferSyntax: Little Endian Implicit
(0011,0010) LO [WMJY] # 4, 1 PrivateCreator
(0011,1000) ?? 66\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1001) ?? 36\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1002) ?? 33\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1003) ?? 61\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1004) ?? 63\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1005) ?? 64\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1006) ?? 33\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1007) ?? 62\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1008) ?? 37\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1009) ?? 38\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100a) ?? 31\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100b) ?? 32\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100c) ?? 37\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100d) ?? 63\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100e) ?? 31\00\00\00 # 4, 1 Unknown Tag & Data
(0011,100f) ?? 64\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1010) ?? 37\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1011) ?? 64\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1012) ?? 33\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1013) ?? 65\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1014) ?? 37\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1015) ?? 30\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1016) ?? 30\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1017) ?? 62\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1018) ?? 35\00\00\00 # 4, 1 Unknown Tag & Data
(0011,1019) ?? 35\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101a) ?? 36\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101b) ?? 36\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101c) ?? 35\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101d) ?? 33\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101e) ?? 35\00\00\00 # 4, 1 Unknown Tag & Data
(0011,101f) ?? 34\00\00\00 # 4, 1 Unknown Tag & Data
```

I'm using the dcmdump tool to read the content of the .dcm file.

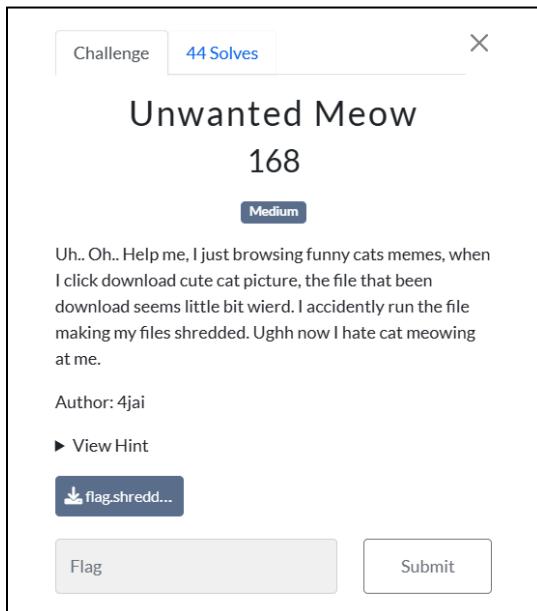
```
$ python
Python 3.12.7 (main, Nov  8 2024, 17:55:36) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> hex_values = [
...     0x66, 0x36, 0x33, 0x61, 0x63, 0x64, 0x33, 0x62,
...     0x37, 0x38, 0x31, 0x32, 0x37, 0x63, 0x31, 0x64,
...     0x37, 0x64, 0x33, 0x65, 0x37, 0x30, 0x30, 0x62,
...     0x35, 0x35, 0x36, 0x36, 0x35, 0x33, 0x35, 0x34
... ]
>>> decoded_chars = ''.join(chr(val) for val in hex_values)
>>> print(decoded_chars)
f63acd3b78127c1d7d3e700b55665354
>>>
>>> sequence = [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]
>>> ordered_chars = ''.join(decoded_chars[i] for i in sequence)
>>> print(f"Flag: WGMY{{{{ordered_chars}}}}")
Flag: WGMY{51faddeb6cc77504db336850d53623177}
>>> █
```

I got the help of chatgpt to decode the hex value in dcmdump and then use the sequence to reorder the decoded characters and then we got the flag!!!

Flag : wgmy{51faddeb6cc77504db336850d53623177}

## Forensic solved(2/4)

### 1. Unwanted Meow



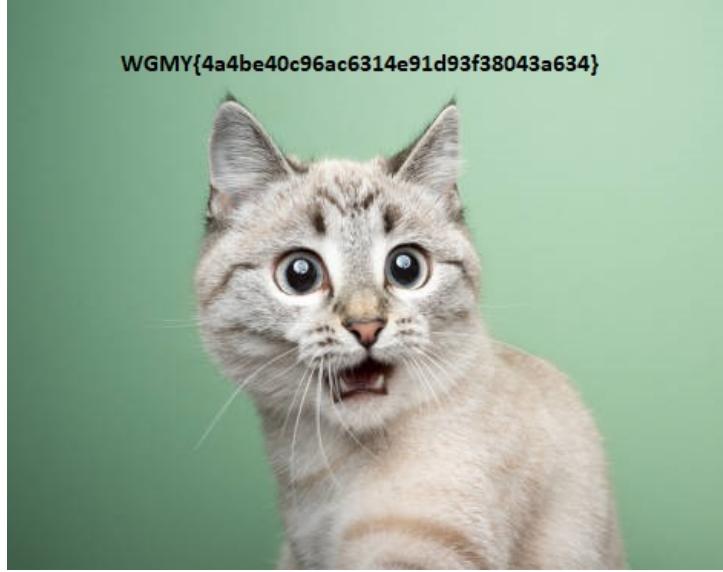
In this challenge, we were provided with a shredded file. Let's inspect the file.

```
4xmpExif/2003-02-29T xmpRights= "meow@cp://ns.adobe.com/xap/1.0" GettyImagesGIFT:AssetID="1361306507" xmpRights:WebStatement="http://medium=organic&utm_source=google&utm_campaign=iptcurl" plus:IBITED-EXCEPTSEARCHENGINEINDEXING">
<dc:creator><rdf:Seq><rdf:mememeowli>Nils Ja
<rdf:Alt><rdf:li xml:lang="x-default">funny cat looking shockemeowd
py space</rdf:li></rdf:Alt></dc:description>
<plus:Licensor><rdf:Seq><rdf:li rdf:type="http://kphoto.com/photo/license-gm1361306507?utm_medium=organic&utm_so
RL"></rdf:li></rdf:Seq></plus:Licensor>
<meow</rdf:Description>
</rdf:RDF>
</x:xmpmeta>
<?xpacket end='w'?>
Photoshop 3.0
8BIM
Nils Jacobi
Vfunny cat looking shocked with mouth open portrait on green background
Getty Images/iStockphoto
meow
$3br
%&'()*)456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
meow
meow
meow
meow
```

By inspecting the file with the command 'strings' we can see that there are a lot of "meow" in the file. Seems very suspicious to me especially when the challenge name is "unwanted meow".

Untitled-1	flag shredded	Search
00000000	FF D8 FF E0 00 10 1A 46 49 46 00 01 01 6D 65 6F	<input checked="" type="checkbox"/> α..JFIF...mo
00000010	77 01 01 2C 00 00 FF E1 00 AC 45 78 69 66	<input checked="" type="checkbox"/> w...,,. B.. <b>Exif</b>
00000020	00 00 4D 4D 00 2A 00 00 08 00 04 01 0E 00 02	<input checked="" type="checkbox"/> ..MM.*.....
00000030	00 00 00 56 00 00 03 E0 01 12 00 03 00 00 00 01	<input checked="" type="checkbox"/> ..V...>.....
00000040	00 01 00 00 01 1A 00 05 00 00 00 01 00 00 00 94	<input checked="" type="checkbox"/> .....δ
00000050	01 1B 00 05 00 00 00 01 00 00 09 0C 00 00 00 00	<input checked="" type="checkbox"/> .....E....
00000060	66 75 6E 79 20 63 61 74 20 6C 6F 6F 6B 69 6E	<input checked="" type="checkbox"/> funny cat lookin
00000070	67 20 73 68 6F 63 6B 65 64 20 77 69 74 68 20 6D	<input checked="" type="checkbox"/> g shocked with m
00000080	6F 75 74 68 20 6F 70 65 6E 20 70 6F 72 74 72 61	<input checked="" type="checkbox"/> outh open portra
00000090	69 74 20 6F 6E 20 67 62 75 65 6E 20 62 61 63 6B	<input checked="" type="checkbox"/> it on green back
000000A0	67 72 6F 75 6E 64 20 77 69 74 68 20 63 6F 70 79	<input checked="" type="checkbox"/> ground with copy
000000B0	20 73 70 61 63 65 00 00 01 2C 00 00 00 00 01 00 00	<input checked="" type="checkbox"/> space...,.....
000000C0	01 2C 00 00 00 01 FE 01 05 DF 68 74 74 70 3A 2F	<input checked="" type="checkbox"/> ..,... B.. <b>http://</b>
000000D0	2F FE 73 2E 61 64 6F 62 65 2E 63 6F 6D 2F 78 61	<input checked="" type="checkbox"/> /ns.adobe.com/xfa
000000E0	70 2F 31 2E 30 2F 00 03 3C 78 70 61 63 6B 65 74	<input checked="" type="checkbox"/> p/1.0/.<?xpacket
000000F0	20 62 65 67 69 6E 3D 22 EF BB BF 22 20 69 64 3D	<input checked="" type="checkbox"/> begin="n <sub>1</sub> " id=
00000100	22 57 35 40 3D 70 43 65 68 69 48 7A 72 65 53	<input checked="" type="checkbox"/> "WSMOpMcphiltzreS
00000110	7A 4E 54 63 7A 6B 33 69 4A 22 3F 3E 0D 0A 3C 78	<input checked="" type="checkbox"/> zNTczkc9d?>...<x
00000120	3A 78 6D 70 6D 65 74 61 20 78 6D 6C 6E 73 3A 78	<input checked="" type="checkbox"/> :xmpmeta xmlns:x
00000130	3D 22 61 60 65 6F 77 64 6F 62 65 3A 6E 73 3A 6D	<input checked="" type="checkbox"/> ="ameowdobe:ns:m
00000140	65 74 61 60 65 6F 77 2F 22 3E 0D 0A 09 03 3C 72 64	<input checked="" type="checkbox"/> etameowb/...<.rd
00000150	66 3A 52 44 46 20 78 6D 6E 73 3A 72 64 66 3D	<input checked="" type="checkbox"/> f:RDF xmlns:rdf=
00000160	22 68 74 74 70 3A 2F 2F 77 77 72 77 33 2E 6F	<input checked="" type="checkbox"/> "http://www.w3.o
00000170	72 67 2F 31 39 39 39 2F 30 32 2F 32 32 2D 72 64	<input checked="" type="checkbox"/> rg/1999/02/22-rd
00000180	66 2D 73 79 6E 74 61 78 2D 6E 73 23 22 3E 0D 0A	<input checked="" type="checkbox"/> f-syntax-ns?>..
00000190	09 09 3C 72 64 66 3A 44 65 73 63 6D 65 6F 77 72	<input checked="" type="checkbox"/> ..<rdf:Descremowr
000001A0	69 70 74 69 6F 6E 20 72 64 66 3A 61 62 6F 75 74	<input checked="" type="checkbox"/> iption rdf:about
000001B0	3D 22 22 20 78 6D 6C 73 73 7A 78 68 6F 74 6F 73	<input checked="" type="checkbox"/> ="" xmlns:photos
000001C0	68 6F 70 3D 22 68 74 74 70 3A 2F 2F 6E 73 2E 61	<input checked="" type="checkbox"/> hop="http://ns.a
000001D0	64 6F 62 65 2E 63 6F 6D 2F 70 68 6F 74 6F 73 68	<input checked="" type="checkbox"/> dobe.com/photosh
Search for		<input type="text"/> meow
Data Type		<input type="checkbox"/> 8-bit Integer <input type="checkbox"/> 16-bit Integer <input type="checkbox"/> 24-bit Integer <input type="checkbox"/> 32-bit Integer <input type="checkbox"/> 64-bit Integer <input type="checkbox"/> 16-bit Floating Point <input type="checkbox"/> 32-bit Floating Point <input type="checkbox"/> LEB128 <input type="checkbox"/> VLQ <input type="checkbox"/> Rational <input type="checkbox"/> Hexadecimal Values <input checked="" type="checkbox"/> Text
Text Encoding		<input type="checkbox"/> All <input type="checkbox"/> Transform backslashes
Case Sensitivity		<input type="checkbox"/> Match Case (faster)
Byte Order		<input checked="" type="checkbox"/> Little-endian <input type="checkbox"/> Big-endian
Search Type		<input type="checkbox"/> List all occurrences <input checked="" type="checkbox"/> Enable replace
Replace by		
Find previous		
Find next		

We can simply remove the meows by using online tools such as hexed.it to modify the strings. Find all meow and replace with blank.



Save the modified file and there you go! We finally got the fully recovered file with the flag.

Flag: wgmy{4a4be40c96ac6314e91d93f38043a634}

## 2. I Cant Manipulate People

Challenge    93 Solves    X

### I Cant Manipulate People

50

Easy

Partial traffic packet captured from hacked machine, can you analyze the provided pcap file to extract the message from the packet perhaps by reading the packet data?

Author: Ap0k4L1p5

► View Hint

traffic.pcap

Flag    Submit

A challenge with pcap, straight to wireshark it is.

No.	Time	Source	Destination	Protocol	Length Info
1	0.000000	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
2	0.057003	192.168.68.117	192.168.0.11	DNS	64 Standard query 0x0000 A dummy1.example.com
3	0.001001	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
4	0.061003	192.168.68.117	192.168.0.12	TCP	55 28098 → 48105 [ACK] Seq=1 Ack=1 Win=8192 Len=15
5	0.002006	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
6	0.053028	192.168.68.117	192.168.0.10	HTTP	88 GET /dummy1.html HTTP/1.1
7	0.004006	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
8	0.061003	192.168.68.117	192.168.0.13	UDP	43 1097 → 8790 Len=15
9	0.006020	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
10	0.055012	192.168.68.117	192.168.0.10	HTTP	88 GET /dummy3.html HTTP/1.1
11	0.007010	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
12	0.055012	192.168.68.117	192.168.0.10	HTTP	88 GET /dummy4.html HTTP/1.1
13	0.008009	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
14	0.064004	192.168.68.117	192.168.0.12	TCP	55 59705 → 40604 [ACK] Seq=1 Ack=1 Win=8192 Len=15
15	0.009009	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
16	0.059004	192.168.68.117	192.168.0.12	TCP	55 58256 → 29512 [ACK] Seq=1 Ack=1 Win=8192 Len=15
17	0.011009	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
18	0.058004	192.168.68.117	192.168.0.11	DNS	64 Standard query 0x0000 A dummy2.example.com
19	0.012008	192.168.68.117	192.168.0.1	ICMP	29 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)

Inspecting the network packet, we can see that there are a lot of ICMP protocols.

```
0000  45 00 00 1d 00 01 00 00  40 01 b5 18 c0 a8 44 75  E.....@.....Du
0010  c0 a8 00 01 08 00 7c ff  00 00 00 00 7b  .....|..{
```

After spending a good amount of time, I realized that each ICMP packet contains each letter of the flag!

```

import pyshark

def extract_last_byte_from_icmp(pcap_file):
    """Extracts the last byte of each ICMP packet from a pcap file."""
    last_bytes = []
    try:
        # Load the pcap file with a display filter for ICMP
        cap = pyshark.FileCapture(pcap_file, display_filter='icmp')

        for packet in cap:
            # Check if the ICMP layer and data field exist
            if 'icmp' in packet and hasattr(packet.icmp, 'data'):
                data_layer = packet.icmp.data
                if data_layer:
                    # Extract the last byte (2 characters of hex)
                    last_byte = data_layer[-2:]
                    last_bytes.append(last_byte)

        cap.close()
    except Exception as e:
        print(f"Error reading the pcap file: {e}")

    return last_bytes

def hex_to_ascii(hex_list):
    """Converts a list of hex values to an ASCII string."""
    try:
        return ''.join(chr(int(h, 16)) for h in hex_list)
    except ValueError as e:
        print(f"Error converting hex to ASCII: {e}")
        return ""

if __name__ == "__main__":
    # Path to your pcap file
    pcap_file = "traffic.pcap"

    # Step 1: Extract last bytes from ICMP packets
    last_bytes = extract_last_byte_from_icmp(pcap_file)

    # Step 2: Convert the hex values to ASCII
    decoded_text = hex_to_ascii(last_bytes)

    # Step 3: Print the result
    if decoded_text:
        print(f"Decoded Text: {decoded_text}")
    else:
        print("No valid ICMP data extracted.")

```

Since i am too lazy to extract each letter from the packets, I asked my best friend(chatgpt) to create a script for me

```

└$ python3 flag.py
Decoded Text: WGMY{1e3b71d57e466ab71b43c2641a4b34f4}

```

As expected, we obtained the flag.

Flag: wgmy{1e3b71d57e466ab71b43c2641a4b34f4}