Raise the Flag and Capture it

Lorenzo Cappetti Mat: 7165929
Chiara Peppicelli Mat: 7146573
Jay Senoner Mat: 7152874

Group name: Charizard-EX
Group code 888

February 26, 2025

1 CTF Building: Raising the flag

In this section, we provide a detailed explanation of how our Capture the Flag (CTF) challenge was constructed. To deploy the challenge, we used Docker containers to ensure easy access and cross-platform compatibility, allowing other students to participate in solving the challenge and capturing the flag. Our group worked efficiently and collaboratively throughout the project, ensuring smooth teamwork and a balanced contribution from every member.

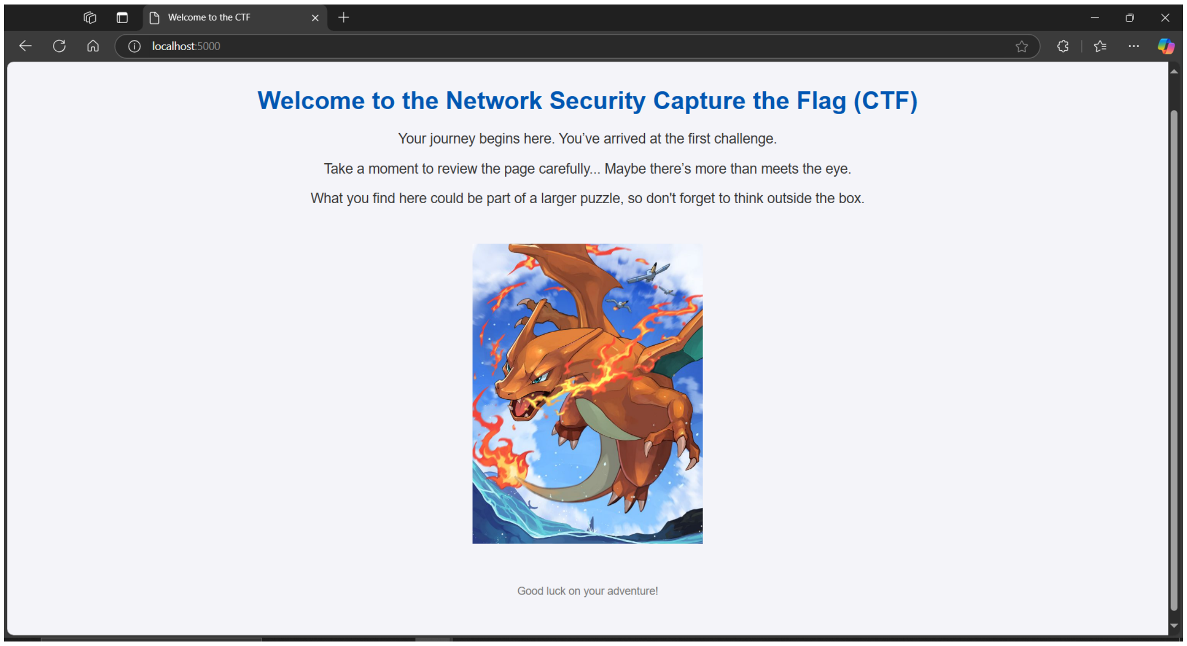
The **first part** of the challenge focuses on the vulnerability of transmitting unencrypted data via the HTTP protocol. This scenario highlights the importance of understanding the risks associated with the lack of encryption in communications—a persistent issue in many legacy systems and outdated applications. The objective is to teach participants how to use network sniffing tools, such as Wireshark, to identify sensitive information transmitted in plain text while emphasizing the critical need to secure such communications with encrypted protocols.

The **second part** of the challenge addresses vulnerabilities related to weak access credential configurations. This section aims to raise awareness about the risks of using easily guessable or poorly secured credentials—a common issue that undermines the security of many systems.

1.1 The First Challenge

The first part of the challenge involves configuring an HTTP server that transmits unencrypted messages. Once launched, this server will be accessible via a web browser at the address `http://localhost:5000`. Upon opening this page, participants will encounter the screen shown in Figure 1.

At first glance, the page does not display any useful information. There are no visible messages or immediate hints, and it might seem as though there is nothing to do. However, the true objective of this phase is to analyze the network traffic generated by the HTTP server to uncover hidden information. This step is crucial, as the credentials needed to access the second part cannot be obtained otherwise.



Welcome to the Network Security Capture the Flag (CTF)

Your journey begins here. You've arrived at the first challenge.

Take a moment to review the page carefully... Maybe there's more than meets the eye.

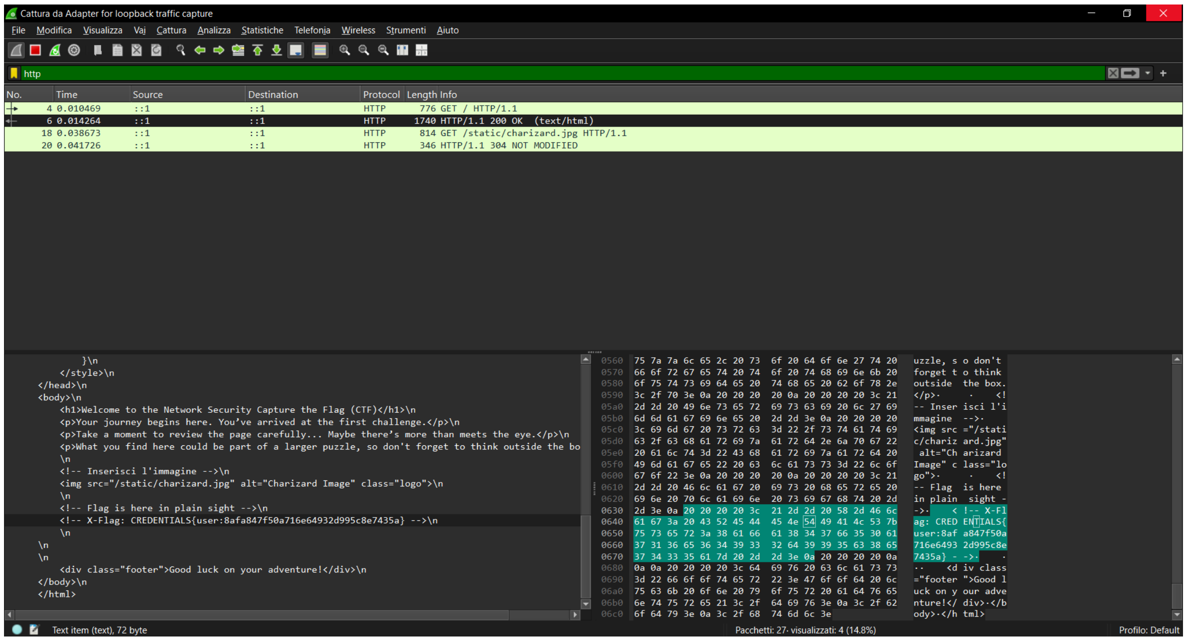
What you find here could be part of a larger puzzle, so don't forget to think outside the box.

Good luck on your adventure!

Figure 1: Access at `http://localhost:5000`

To proceed, the participant must use a network traffic sniffing tool, such as Wireshark. By configuring it to monitor the port used by the HTTP protocol (in our case, port 5000), the participant can view all traffic generated by the server.

When traffic capture is initiated in Wireshark and HTTP packets are filtered, a screen similar to the one shown in Figure 2 will appear.


Cattura da Adapter for loopback traffic capture
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto
http

```
No.   Time      Source   Destination   Protocol Length Info
    4 0.010469  ::1      ::1           HTTP      776 GET / HTTP/1.1
    6 0.014264  ::1      ::1           HTTP     1740 HTTP/1.1 200 OK  (text/html)
   18 0.038673  ::1      ::1           HTTP      814 GET /static/charizard.jpg HTTP/1.1
   20 0.041726  ::1      ::1           HTTP      346 HTTP/1.1 304 NOT MODIFIED
```

```
            }\n
        </style>\n
    </head>\n
    <body>\n
        <h1>Welcome to the Network Security Capture the Flag (CTF)</h1>\n
        <p>Your journey begins here. You've arrived at the first challenge.</p>\n
        <p>Take a moment to review the page carefully... Maybe there's more than meets the eye.</p>\n
        <p>What you find here could be part of a larger puzzle, so don't forget to think outside the bo
        \n
        <!-- Inserisci l'immagine -->\n
        <img src="/static/charizard.jpg" alt="Charizard Image" class="logo">\n
        \n
        <!-- Flag is here in plain sight -->\n
        <!-- X-Flag: CREDENTIALS{user:8afa847f50a716e64932d995c8e7435a} -->\n
        \n
    \n
    \n
        <div class="footer">Good luck on your adventure!</div>\n
    </body>\n
    </html>
```

```
0560  75 7a 7a 6c 65 2c 20 73  6f 20 64 6f 6e 27 74 20   uzzle, s o don't 
0570  66 6f 72 67 65 74 20 74  6f 20 74 68 69 6e 6b 20   forget t o think 
0580  6f 75 74 73 69 64 65 20  74 68 65 20 62 6f 78 2e   outside  the box.
0590  3c 2f 70 3e 0a 20 20 20  20 0a 20 20 20 20 3c 21   </p>·     ·    <!
05a0  2d 2d 20 49 6e 73 65 72  69 73 63 69 20 6c 27 69   -- Inser isci l'i
05b0  6d 6d 61 67 69 6e 65 20  2d 2d 3e 0a 20 20 20 20   mmagine  -->·    
05c0  3c 69 6d 67 20 73 72 63  3d 22 2f 73 74 61 74 69   <img src ="/stati
05d0  63 2f 63 68 61 72 69 7a  61 72 64 2e 6a 70 67 22   c/chariz ard.jpg"
05e0  20 61 6c 74 3d 22 43 68  61 72 69 7a 61 72 64 20    alt="Ch arizard 
05f0  49 6d 61 67 65 22 20 63  6c 61 73 73 3d 22 6c 6f   Image" c lass="lo
0600  67 6f 22 3e 0a 20 20 20  20 0a 20 20 20 20 3c 21   go">·     ·    <!
0610  2d 2d 20 46 6c 61 67 20  69 73 20 68 65 72 65 20   -- Flag  is here 
0620  69 6e 20 70 6c 61 69 6e  20 73 69 67 68 74 20 2d   in plain  sight -
0630  2d 3e 0a 20 20 20 20 3c  21 2d 2d 20 58 2d 46 6c   ->·    < !-- X-Fl
0640  61 67 3a 20 43 52 45 44  45 4e 54 49 41 4c 53 7b   ag: CRED ENTIALS{
0650  75 73 65 72 3a 38 61 66  61 38 34 37 66 35 30 61   user:8af a847f50a
0660  37 31 36 65 36 34 39 33  32 64 39 39 35 63 38 65   716e6493 2d995c8e
0670  37 34 33 35 61 7d 20 2d  2d 3e 0a 20 20 20 20 0a   7435a} - ->·    ·
0680  0a 0a 20 20 20 20 3c 64  69 76 20 63 6c 61 73 73   ··    <d iv class
0690  3d 22 66 6f 6f 74 65 72  22 3e 47 6f 6f 64 20 6c   ="footer ">Good l
06a0  75 63 6b 20 6f 6e 20 79  6f 75 72 20 61 64 76 65   uck on y our adve
06b0  6e 74 75 72 65 21 3c 2f  64 69 76 3e 0a 3c 2f 62   nture!</ div>·</b
06c0  6f 64 79 3e 0a 3c 2f 68  74 6d 6c 3e               ody>·</h tml>
```

```
Text item (text), 72 byte          Pacchetti: 27· visualizzati: 4 (14.8%)          Profilo: Default
```

Figure 2: Sniffing on Wireshark.

In the captured screen, the participant will notice a series of HTTP requests and responses. By examining the server response headers in detail, they will find a field called X-Flag, which contains the hidden message formatted as the required credentials.

The credentials, however, are slightly more complex to use as they are encoded with an MD5 hash and will appear as follows:

```
X-Flag:  CREDENTIALS{user:8afa847f50a716e64932d995c8e7435a}
```

When decoded, they will reveal:

```
CREDENTIALS{user:princess}
```

This information is essential to advance to the second part of the challenge. However, the participant must pay close attention: not all HTTP responses may contain the X-Flag field. Therefore, it is necessary to carefully analyze each packet to identify the correct one.

1.2 The Second Challenge

The second part of the challenge begins once the participant has successfully obtained the credentials from the first phase, the username and the password hash. At this point, the participant must use these credentials to access an Ubuntu system via SSH.

To do this, they will connect using the following command:

```
ssh user@IP_del_container
```

The IP address of the second Docker container, which hosts the Ubuntu system, is represented as `<IP_of_the_container>`, and in our case, it is **localhost -p 4849**. When the participant attempts to connect, they will be prompted to enter the password. This is the same password retrieved in the first phase and must be entered correctly to proceed.

Upon logging in, the participant will be presented with a terminal screen. Running the `ls` command will reveal a file named `flag.txt` in the user's home directory. However, if they attempt to view the file's content using commands like `cat`, `more`, or `less`, these commands will not work. This is because they have been disabled using a mechanism called "`alias`."

The participant must identify and resolve this issue. To do so, they can use the `unalias` command. The command to remove all aliases is:

```
unalias -a
```

Once the aliases are removed, standard commands like `cat`, `more`, and `less` will function correctly, allowing the participant to view the contents of the `flag.txt` file and complete the challenge.

The content of `flag.txt` contains the final message that confirms the successful completion of the challenge:

```
FLAG{ SEI DIVENTATO UN HACKER, questa è la flag :)  }
```

2 Capture the Flag

In this section, we solve various CTF challenges implemented by our colleagues from the Network Security course. For each challenge, a detailed explanation of the solution is provided, including a brief account of the trial-and-error process involved in arriving at the correct solution.

2.1 LANgo Unchained

The challenge provided a login page with fields for a username and password. Submitting random credentials returned a consistent "wrong credentials" response. The goal was to explore vulnerabilities in the password field to discover the flag.

Initial Observations

- **Access the Page:** Navigated to `http://localhost:5000`, which presented a simple login form.
- Attempted common credentials such as `admin/admin`, `test/password`, and empty fields. Each attempt returned the same `"wrong credentials"` message.
- Checked the page source for hidden fields or comments, but nothing unusual was found at this stage.

Identifying Command Injection

- **Input Testing:** When entering ls into both the username and password fields, the response was different:

  ```
  {
      "output": "app.py\nrequirements.txt\n",
      "password_command": "ls",
      "username": "ls"
  }
  ```

  This indicated that the `ls` command was executed, and the application returned the directory listing, which included `app.py` and `requirements.txt`. This suggested that command injection was possible through the password field.

- **Intuition:** The change in behavior indicated that user input was being passed to the system shell for execution. This opened the door to further exploration, such as reading files or listing hidden files.

Code Analysis

- **Extracting the Source Code:** To better understand the application's behavior, the source code of app.py was extracted using:
  - **Username:** `cat app.py`

- **Password:** `cat app.py`

This returned the full source code of the application. Key observations included:

- **Flag Location:** The flag was stored in `.flag/flag.txt` with the content `"1234"`.
- **Command Execution:** The password field was passed directly to `os.popen()` for execution:

```
output = os.popen(password).read()
```

  This confirmed that the password input was being used to execute arbitrary commands.

- **Flag Display Logic:** The flag was compared against a hardcoded value in the code, and if it matched, the user was redirected to a success page:

```
if password == FLAG_CONTENT:
    return redirect(url_for('success_page'))
```

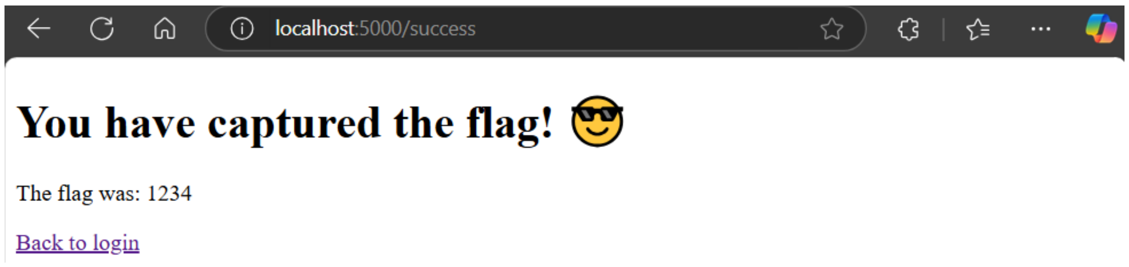
  If successful, the flag was displayed on this page.

Knowing all of this, the last step was to submit the right password to trigger the success page.

Logging In with the Flag

- **Username:** Any value (e.g., `user`)
- **Password:** `1234`

This resulted in the following success message:



localhost:5000/success

You have captured the flag! 😎

The flag was: 1234

Back to login

Figure 3: Successful Capture of Group 12 Flag.

The Flag is : 1234

2.2 Gli Hackers

The challenge provided a web application running in a Docker container. The goal was to access the flag, which was in the format `NETSECstring`. The hint provided was: *"It's pouring..."*, which indicated the possibility of input manipulation through a vulnerability.

Initial Observations

- **Access the Page:** Navigated to `http://localhost:8080`, which presented a to-do list.
- The application allowed for basic user interaction with the list, but no immediate vulnerabilities were visible.

Identifying Command Injection

Recalling the hint "It's pouring...", We hypothesized that the application might be vulnerable to command injection. The theory was tested by entering a semicolon `;` followed by the `ls` command in the input fields of the application. This is a standard approach for testing command injection vulnerabilities. The input was as follows:

```
; ls
```

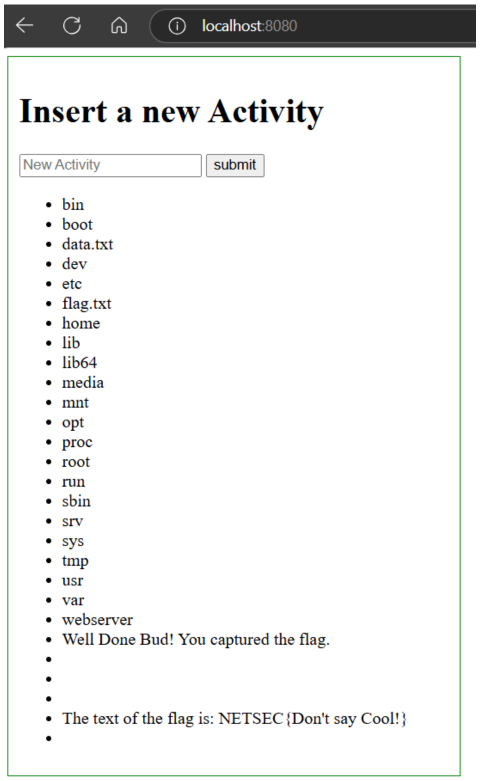
This printed a list of items present in the directory such as `bin`, `dev`,... and `flag.txt`.

Flag Discovery

To get the flag we decided to test the input:

```
; cat flag.txt
```

And that resulted in the following success message:



Insert a new Activity

```
• bin
• boot
• data.txt
• dev
• etc
• flag.txt
• home
• lib
• lib64
• media
• mnt
• opt
• proc
• root
• run
• sbin
• srv
• sys
• tmp
• usr
• var
• webserver
• Well Done Bud! You captured the flag.
•
•
•
• The text of the flag is: NETSEC{Don't say Cool!}
•
```

Figure 4: Successful Capture of Group 20010db8 Flag.

The Flag is : NETSEC{Don't say Cool!}

2.3 Knullektivet

The goal of this challenge was to exploit an SQL Injection vulnerability in the `/login` endpoint of a web application and retrieve the flag stored in the database.

Initial Observations

- **Access the Page:** Navigated to `http://localhost:5000`, which instructed navigating to `/login`.
- Initial testing with valid credentials `(admin:password)` returned:

```
{
"message": "Invalid credentials.",
"status": "failure"
}
```

- This indicated the app was verifying the `username` and `password` inputs, likely using a SQL query.

Identifying Command Injection

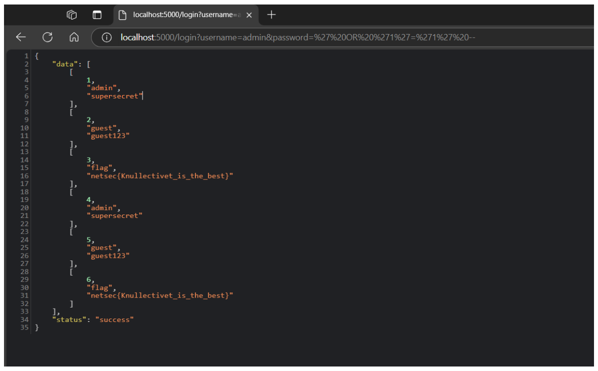
Testing for SQL Injection was performed by sending:

`http://localhost:5000/login?username=admin&password=' OR '1'='1' --`

`' OR '1'='1' --` bypassed the password validation by injecting SQL code that always evaluated to true, while the `--` commented out the rest of the query, effectively ignoring the actual password logic.

Flag Discovery

The payload worked, granting access as the admin user. This confirmed an SQL Injection vulnerability existed in the `/login` endpoint.
And that resulted in the following success message:



```
localhost:5000/login?username=admin&password=%27%20OR%20%271%27=%271%27%20--
```

```
{
    "data": [
        [
            1,
            "admin",
            "supersecret"
        ],
        [
            2,
            "guest",
            "guest123"
        ],
        [
            3,
            "flag",
            "netsec{Knullectivet_is_the_best}"
        ],
        [
            4,
            "admin",
            "supersecret"
        ],
        [
            5,
            "guest",
            "guest123"
        ],
        [
            6,
            "flag",
            "netsec{Knullectivet_is_the_best}"
        ]
    ],
    "status": "success"
}
```

Figure 5: Successful Capture of Group 69 Flag.

The Flag is : netsec{Knullectivet_is_the_best}

2.4 Salsiccia e Friarelli

The challenge provided a login page with fields for a username and password. Submitting random credentials returned a consistent ”wrong credentials” response, suggesting the presence of input validation. The goal was to explore vulnerabilities in the password field and the authentication process, using techniques like SQL injection, steganography and AES decryption. Through these methods, we were able to uncover multiple clues that guided us through the challenge, ultimately leading to the discovery of the flag.

Initial Observations

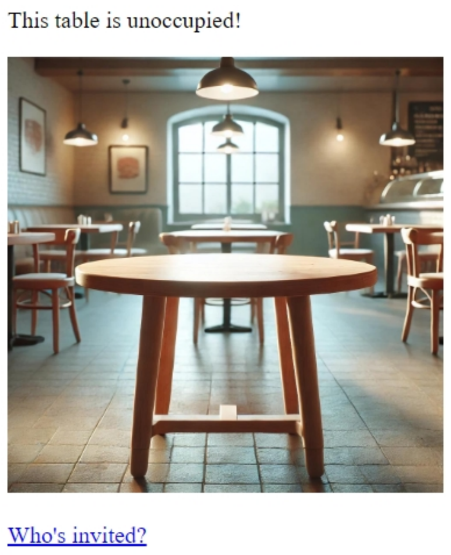
- **Access the Page:** Navigated to `http://localhost:5000`, which presented a simple login form.
- The first hint provided was to use `HINT` as both the username and the password.
- Upon successful login, the following message appeared :

```
Welcome! This is team Salsiccia e Friarelli’s CTF challenge

This is your first hint: try SQL Injection attacks
```

SQL Injection Attempt

- We attempted a common SQL injection payload: `’ OR ’1’=’1’ --`.
- This allowed us to bypass the login page and access a new page with the message:



This table is unoccupied!

Who's invited?

- Clicking on the ”Who’s invited?” button triggered a download of the table image, which was saved as `db_us3r5`.

- We analyzed the downloaded file using **OpenStego**, a steganography tool, to check for hidden messages.

Extracting the User Database

OpenStego applied on the table image revealed a hidden file called `userdatabase`, which contained the following text:

```
    USER DATABASE
# USERNAME PASSWORD
2 HINT HINT
3 Oznerol Oznerol
4 Annavoigairam Annavoigairam
5 hINt@02 hINt@02
```

Knowing all of this, we attempted to login with each of the usernames and passwords found in the userdatabase file.

Logging In with "Oznerol"

By entering `Oznerol` as both the username and the password, the page displayed the message:

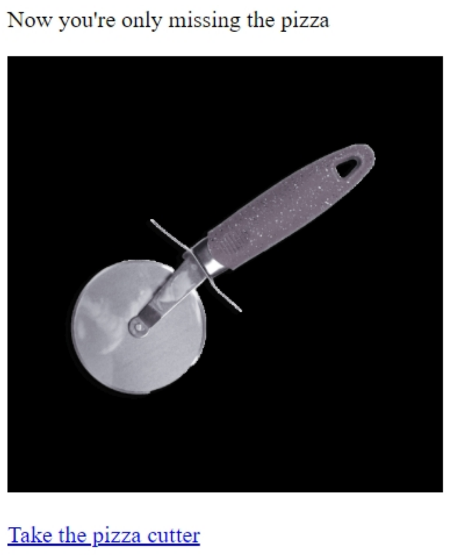
```
    Welcome!
This is an interesting challenge

To go on you must solve the following riddle.
The answer is composed of two words...
who knows how they could be used...

"We are two.
The first is grey the second is red.
The first can be picked and the second worn on a sleeve.
They say to either use the first or follow the second
but either is fine to choose your pizza
Who are we?"
```

We analyzed the riddle and deduced that the answer was *brain* and *heart* based on the clues provided:

- "The first is grey" refers to the brain (grey matter).
- "The second is red" refers to the heart (associated with red blood).
- "The first can be picked and the second worn on a sleeve" refers to common expressions about the brain and heart.



Now you're only missing the pizza

Take the pizza cutter

Logging in with "Annavoigairam"

- The page displayed
- Clicking the "Take the pizza cutter" button triggered a download of an image named `5ym_AES_k3y`.
- We applied steganographic decryption to this file and obtained a new file named `key.txt` containing the following text:

```
        ●Ç¾ïÓÉ::;vD¼cV¯FÀèV×Ìœ8o_"ü@
```

Logging in with "hINt@02"

- The page displayed the message:

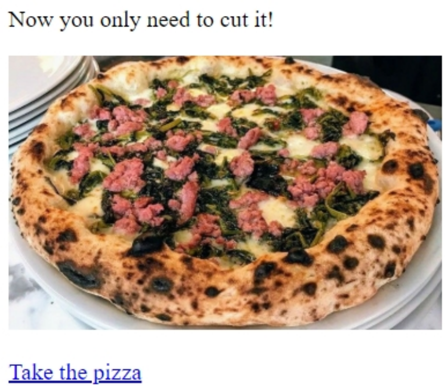
```
        Good job!
Now that you know who's coming let me give you some advice:

[HINT02]: Oznerol likes riddles but usually his answers
relate to one's body
```

- This confirmed that the correct answer to the riddle was indeed *brain* and *heart*. We then used these answers to proceed with the next step.

Final Login Attempt: "Brain" and "Heart"

- We tried logging in with the answer to the riddle, this successfully granted us access to the next stage with the message:


Now you only need to cut it!

Take the pizza

- Clicking the "Take the pizza" button triggered a download of an image named `5ym_AES_f14g`.
- We applied the decryption method to the image, this resulted in the extraction of a file named `encrypted_message`, containing the following:

```
Qö¦≫ä>¿ E≫½ÕqÑ krõñ‘¿‡"U\öÒèxMîY?úé>●ëŸüN
```

Flag Decryption

After successfully obtaining the encrypted message from the file `encrypted_message`, we proceeded with decrypting the content. This file had been extracted from an image called `5ym_AES_f14g`, which led us to believe that the message was encrypted using AES encryption. Upon further inspection, we noticed that the `encrypted_message` contained two distinct sections separated by spaces: the first seemed to be the initialization vector (IV), and the second appeared to be the encrypted ciphertext. Based on this, we deduced that the encryption mode used was likely AES in CBC mode.
We followed these steps to decrypt the message:

- We created a Python script to handle the decryption. The script reads the key from `key.txt` and the encrypted message (which includes both the IV and ciphertext) from `encrypted_message.txt`.
- The IV is extracted as the first 16 bytes, followed by the actual ciphertext.
- Using the `pycryptodome` library, we initialized the AES cipher in CBC mode with the extracted key and IV. We then decrypted the ciphertext, ensuring to remove any padding using the `unpad` function to restore the original message.
- Finally, the decrypted message was written into a new file, `decrypted_output.txt`.

The final decrypted message revealed the flag. This confirmed that the decryption process was successful and that we had completed the challenge!

The Flag is : netsec{m4ng14t1_5t4_p1z24}

3 Conclusions

The activity of designing and solving CTF challenges proved to be an effective tool for deepening knowledge of cybersecurity concepts. The opportunity to engage in a "hands-on" exercise encouraged active and enjoyable learning, combining theory and practice to tackle real-world problems. These experiences helped develop technical skills and awareness of good security practices.

References

[Has24] Hashcat Project. Hashcat - advanced password recovery, 2024. Accessed: December 2024. Available at: https://hashcat.net/hashcat/.

[Leg24] Helder Legrandin. Pycryptodome: A self-contained cryptographic library for python, 2024. Accessed: December 2024 Available at: https://www.pycryptodome.org/.

[MD524] MD5Online. Md5online: Free and fast md5 encryption, 2024. Accessed: December 2024.

[Ope23] OpenAI. Chatgpt: Conversational ai model, 2023. Accessed: December 2024. Available at: https://chat.openai.com.

[Ope24] OpenStego. Openstego: A steganography tool, 2024. Accessed: December 2024.

[Pec24] Tommaso Pecorella. Network security course slides, 2024.

[You21] YouTube. Bash alias ctf challenge built w/ docker. https://www.youtube.com/watch?v=gSoNSgzp5Yo, dec 2021. [Online; accessed December 2024].