

TryHackMe Corridor Lab – VAPT Report

Prepared By: Nirmiti Dhawade

Role: Red Team Intern

Platform: TryHackMe

Submission: Internship Report (The Cyber Ledger)

Date: November 2025

1. Introduction

The TryHackMe **Corridor Lab** simulates a lightweight web-based challenge where the objective is to identify hidden information, extract encoded values, and enumerate the application using foundational cybersecurity techniques. This report outlines the systematic methodology followed during the assessment, including reconnaissance, scanning, inspection, and hash-cracking activities.

The engagement followed a structured, industry-aligned workflow combining reconnaissance, analysis, and exploitation steps. Each stage includes a designated screenshot placeholder for reference.

2. Scope of Assessment

- Target: TryHackMe *Corridor Lab*
- Type: Web Application Enumeration & Basic Exploitation
- Skills Applied:
 - Network Scanning
 - Web Inspection
 - Hash Cracking (MD5)
 - CyberChef Operations
 - On-page Flag Identification

3. Methodology

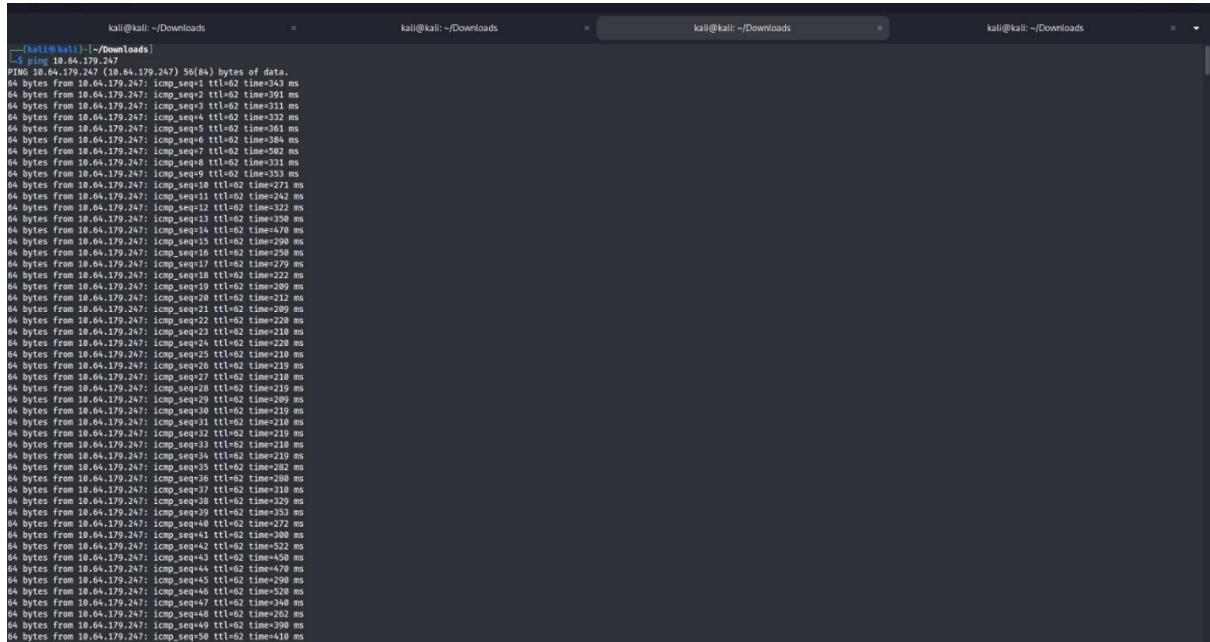
The assessment followed a linear, practical approach:

1. Connectivity Testing (Ping)
2. Service Enumeration (Nmap)
3. Web Interface Review (Port 80)
4. Hash Extraction
5. Hash Cracking (CrackStation)
6. Verification via CyberChef

7. Manual Inspection for Hidden Flag

4. Connectivity Validation (Ping Test)

To ensure the target was reachable, the **ping** utility was executed. Successful responses indicated that the host was active and accessible for further assessment.



The image shows four terminal windows from the Kali Linux environment, all displaying the output of a ping command to the IP address 10.64.179.247. The output is identical across all four windows, showing a series of ICMP echo replies. Each reply consists of 64 bytes (68 bytes of data) with a TTL of 62 and a time between 103 ms and 508 ms. The sequence numbers (seq) range from 1 to 50. The timestamp for each reply is also present.

```
ping 10.64.179.247
PING 10.64.179.247 (10.64.179.247) 68(84) bytes of data.
64 bytes from 10.64.179.247: icmp_seq=1 ttl=62 time=103 ms
64 bytes from 10.64.179.247: icmp_seq=2 ttl=62 time=103 ms
64 bytes from 10.64.179.247: icmp_seq=3 ttl=62 time=111 ms
64 bytes from 10.64.179.247: icmp_seq=4 ttl=62 time=332 ms
64 bytes from 10.64.179.247: icmp_seq=5 ttl=62 time=361 ms
64 bytes from 10.64.179.247: icmp_seq=6 ttl=62 time=384 ms
64 bytes from 10.64.179.247: icmp_seq=7 ttl=62 time=339 ms
64 bytes from 10.64.179.247: icmp_seq=8 ttl=62 time=331 ms
64 bytes from 10.64.179.247: icmp_seq=9 ttl=62 time=353 ms
64 bytes from 10.64.179.247: icmp_seq=10 ttl=62 time=271 ms
64 bytes from 10.64.179.247: icmp_seq=11 ttl=62 time=242 ms
64 bytes from 10.64.179.247: icmp_seq=12 ttl=62 time=230 ms
64 bytes from 10.64.179.247: icmp_seq=13 ttl=62 time=358 ms
64 bytes from 10.64.179.247: icmp_seq=14 ttl=62 time=478 ms
64 bytes from 10.64.179.247: icmp_seq=15 ttl=62 time=298 ms
64 bytes from 10.64.179.247: icmp_seq=16 ttl=62 time=258 ms
64 bytes from 10.64.179.247: icmp_seq=17 ttl=62 time=279 ms
64 bytes from 10.64.179.247: icmp_seq=18 ttl=62 time=277 ms
64 bytes from 10.64.179.247: icmp_seq=19 ttl=62 time=227 ms
64 bytes from 10.64.179.247: icmp_seq=20 ttl=62 time=209 ms
64 bytes from 10.64.179.247: icmp_seq=21 ttl=62 time=212 ms
64 bytes from 10.64.179.247: icmp_seq=22 ttl=62 time=209 ms
64 bytes from 10.64.179.247: icmp_seq=23 ttl=62 time=220 ms
64 bytes from 10.64.179.247: icmp_seq=24 ttl=62 time=218 ms
64 bytes from 10.64.179.247: icmp_seq=25 ttl=62 time=228 ms
64 bytes from 10.64.179.247: icmp_seq=26 ttl=62 time=218 ms
64 bytes from 10.64.179.247: icmp_seq=27 ttl=62 time=219 ms
64 bytes from 10.64.179.247: icmp_seq=28 ttl=62 time=219 ms
64 bytes from 10.64.179.247: icmp_seq=29 ttl=62 time=210 ms
64 bytes from 10.64.179.247: icmp_seq=30 ttl=62 time=210 ms
64 bytes from 10.64.179.247: icmp_seq=31 ttl=62 time=209 ms
64 bytes from 10.64.179.247: icmp_seq=32 ttl=62 time=210 ms
64 bytes from 10.64.179.247: icmp_seq=33 ttl=62 time=216 ms
64 bytes from 10.64.179.247: icmp_seq=34 ttl=62 time=219 ms
64 bytes from 10.64.179.247: icmp_seq=35 ttl=62 time=282 ms
64 bytes from 10.64.179.247: icmp_seq=36 ttl=62 time=288 ms
64 bytes from 10.64.179.247: icmp_seq=37 ttl=62 time=310 ms
64 bytes from 10.64.179.247: icmp_seq=38 ttl=62 time=293 ms
64 bytes from 10.64.179.247: icmp_seq=39 ttl=62 time=353 ms
64 bytes from 10.64.179.247: icmp_seq=40 ttl=62 time=272 ms
64 bytes from 10.64.179.247: icmp_seq=41 ttl=62 time=300 ms
64 bytes from 10.64.179.247: icmp_seq=42 ttl=62 time=322 ms
64 bytes from 10.64.179.247: icmp_seq=43 ttl=62 time=103 ms
64 bytes from 10.64.179.247: icmp_seq=44 ttl=62 time=478 ms
64 bytes from 10.64.179.247: icmp_seq=45 ttl=62 time=298 ms
64 bytes from 10.64.179.247: icmp_seq=46 ttl=62 time=528 ms
64 bytes from 10.64.179.247: icmp_seq=47 ttl=62 time=348 ms
64 bytes from 10.64.179.247: icmp_seq=48 ttl=62 time=339 ms
64 bytes from 10.64.179.247: icmp_seq=49 ttl=62 time=398 ms
64 bytes from 10.64.179.247: icmp_seq=50 ttl=62 time=418 ms
```

5. Service Enumeration with Nmap

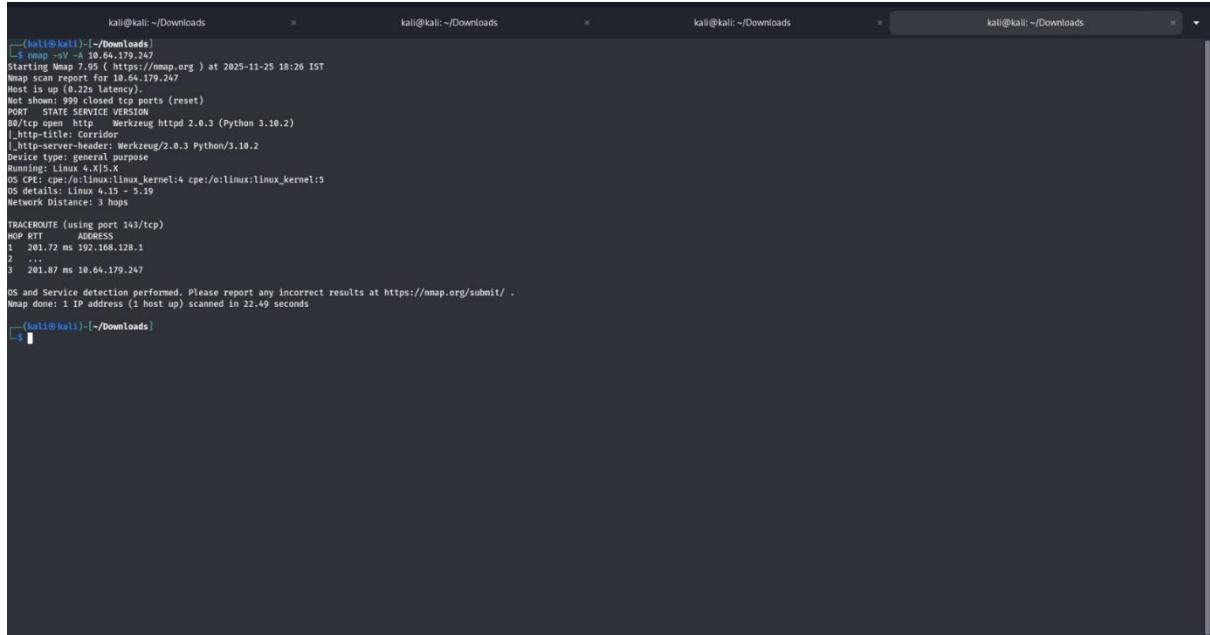
A detailed scan was conducted using:

```
nmap -sV -A <target-IP>
```

Key Observations:

- Port **80** was open.
- Detected service: HTTP
- No secondary high-risk attack surface was identified, confirming a single-page challenge structure.

This step validated that the challenge was entirely web-based.



The screenshot shows four terminal windows from a Kali Linux environment. Each window displays the output of an Nmap scan against the IP address 10.64.179.247. The scan results indicate that port 80 is open and serves an HTTP website. The website is identified as 'Corridor' with the Werkzeug/2.0.3 Python/3.10.2 stack. The system is running Linux 4.15. The traceroute command shows three hops between the scanner and the target host. The OS detection section notes that the target is a Linux kernel version 4.15. The scan took 22.49 seconds and involved one host.

```
kali㉿kali:~/Downloads
[~] nmap -sV -A 10.64.179.247
Starting Nmap 7.80 ( https://nmap.org ) at 2025-11-25 18:28 IST
Nmap scan report for 10.64.179.247
Host is up (0.22s latency).
Not shown: 999 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
80/tcp    open  httpd   Apache httpd 2.0.0.3 (Python 3.10.2)
|_http-title: Corridor
|_http-server-header: Werkzeug/2.0.3 Python/3.10.2
Device type: general purpose
Running: Linux 4.15|5.19
OS: Linux 4.15 - 5.19
Network Distance: 3 hops

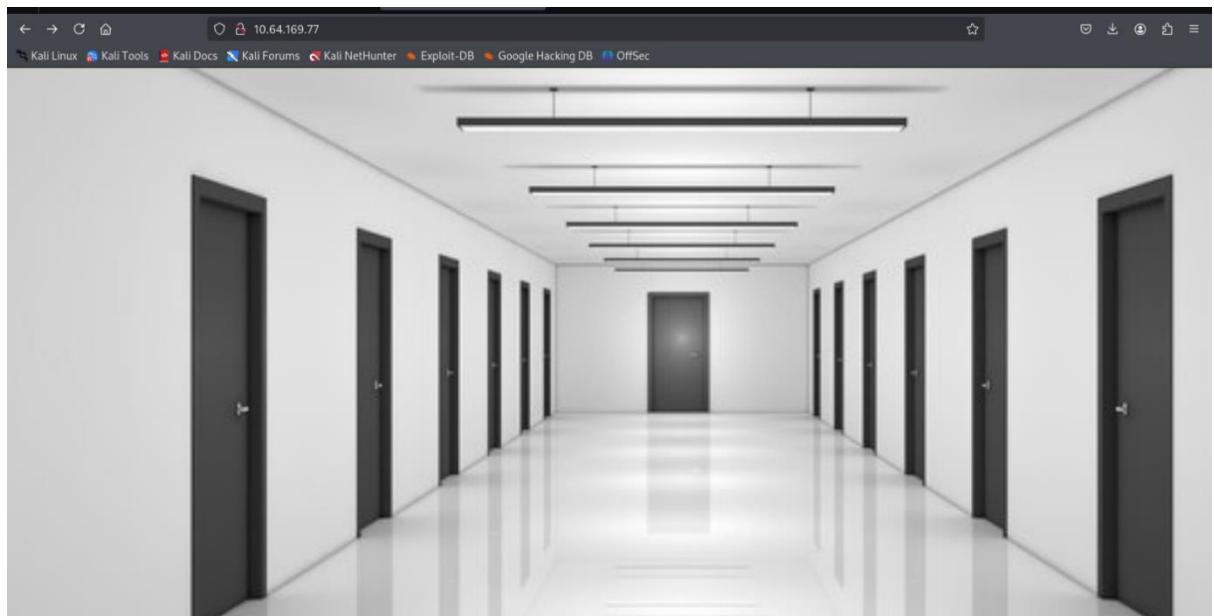
TRACEROUTE (using port 143/tcp)
HOP RTT     ADDRESS
1  201.72 ms 192.168.328.1
2  ...
3  201.87 ms 10.64.179.247

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 22.49 seconds
[~]
```

6. Web Access and Initial Analysis

Accessing the IP on port 80 displayed the corridor-style interface with numbers/inputs that hinted toward hidden encoded information.

No direct functionality was visible, so further inspection was required.



7. Hash Identification and Extraction

During interaction with the page, a series of encoded strings (MD5 hashes) were identified. These appeared to correspond to corridor door inputs.

Two key hashes were extracted:

- **Hash 1 → Result: 1**
- **Hash 2 → Result: 13**

These were cracked using external resources.

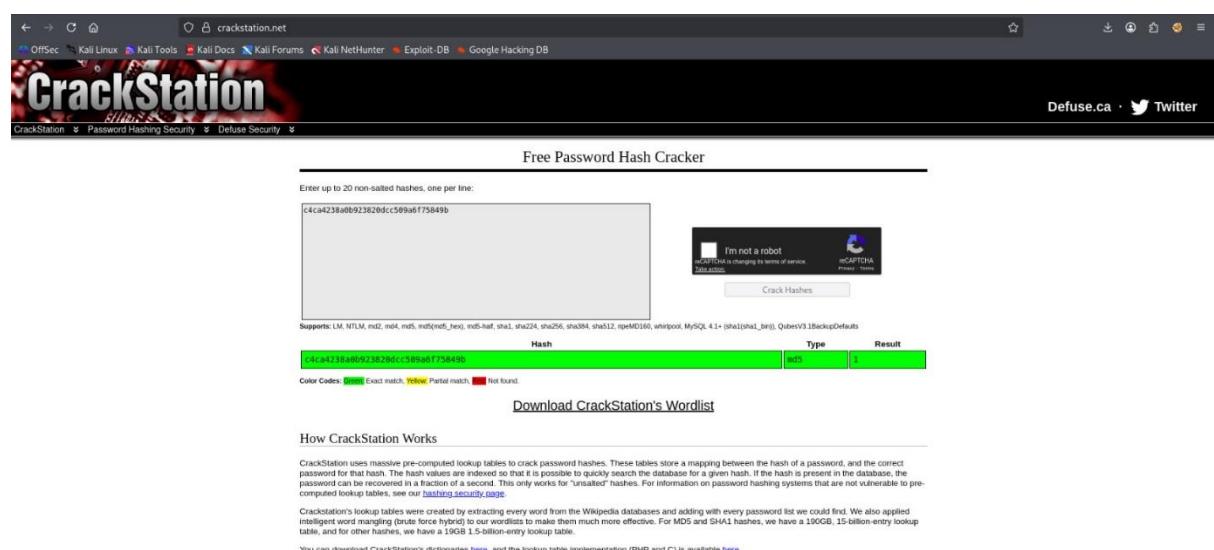
8. Cracking with CrackStation

The extracted MD5 hashes were tested using **CrackStation**, allowing verification of the intended numeric values.

Cracked Results:

- Hash 1 → **1**
- Hash 2 → **13**

This validated the numeric door sequence.



The screenshot shows the CrackStation homepage with the title "Free Password Hash Cracker". A text input field contains the MD5 hash "c4ca4238ae0b923828dc5b9a6775649b". Below the input field is a reCAPTCHA verification box. To the right of the input field is a "Crack Hashes" button. Below the input field, the text "Enter up to 20 non-salted hashes, one per line." is displayed. A table below the input field shows the cracked result: Hash "c4ca4238ae0b923828dc5b9a6775649b" has a Type of "MD5" and a Result of "1". The table also includes columns for "Hash", "Type", and "Result". At the bottom of the page, there is a section titled "How CrackStation Works" with a detailed explanation of how the service uses pre-computed lookup tables to crack password hashes.

CrackStation

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

c51ce418c124a10e0db5e4b997fc2af39

I'm not a robot
reCAPTCHA is changing its terms of service.
reCAPTCHA Privacy Policy Terms

Crack Hashes

Hash: c51ce418c124a10e0db5e4b997fc2af39 | Type: MD5 | Result: 13

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

Download CrackStation's Wordlist

How CrackStation Works

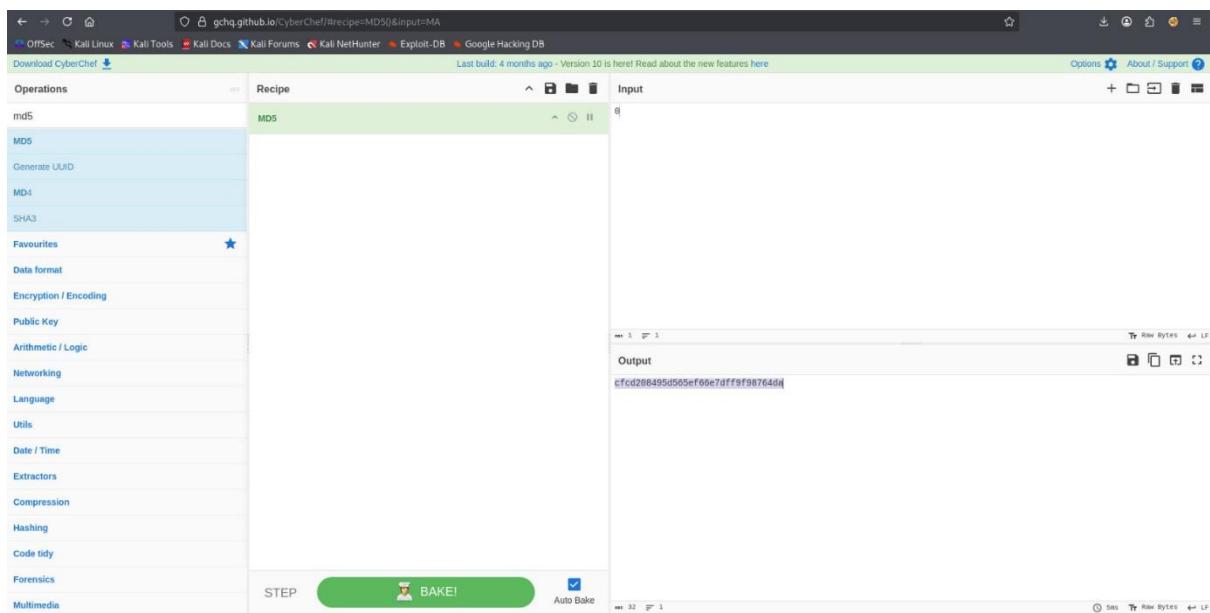
CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password that made it. The hash values are stored so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our [hashing security page](#).

CrackStation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries [here](#), and the lookup table implementation (PHP and C) is available [here](#).

9. Hash Verification using CyberChef

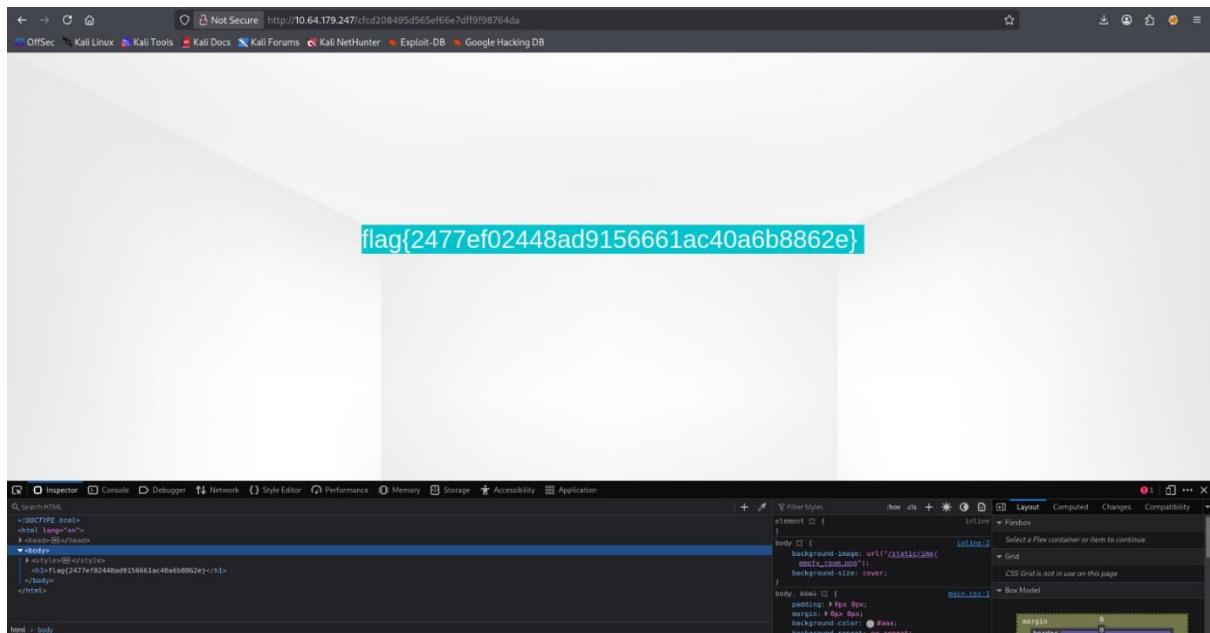
To ensure accuracy, the hash values were re-validated using **CyberChef**. By applying MD5 operations, the output matched the CrackStation result, establishing consistency.



10. Manual Web Inspection for Hidden Flag

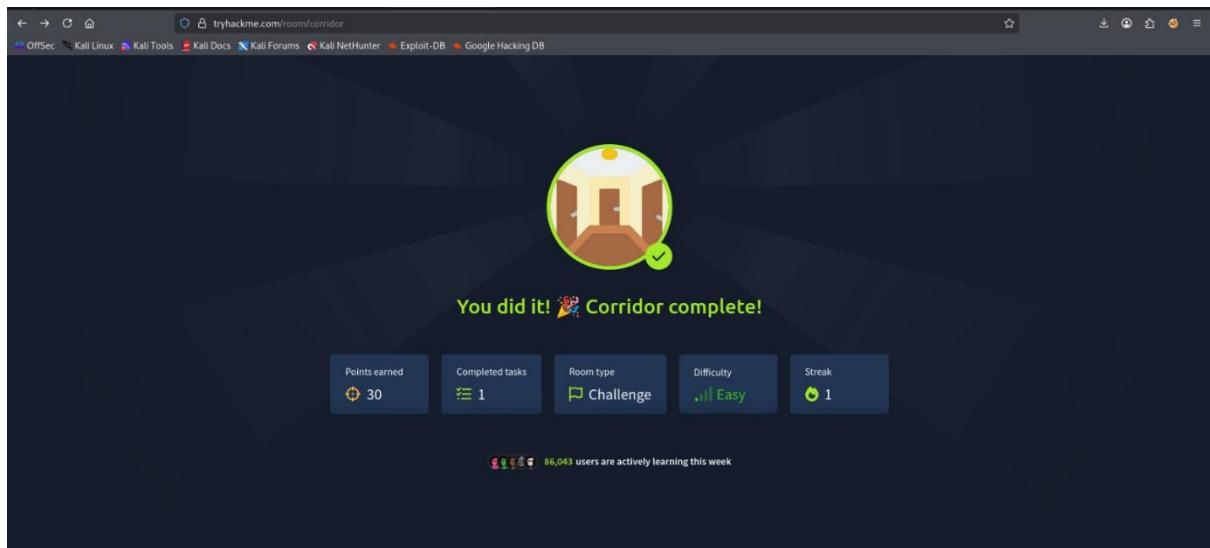
Final validation involved manually inspecting the webpage source using browser **Inspect Element**. The hidden flag was discovered within an HTML <title> attribute inside the page body.

This confirmed the conclusion of the challenge.



11. Completion Confirmation

The challenge platform confirmed successful completion.



12. Conclusion

The Corridor Lab demonstrated foundational skills essential for web-based assessments—reconnaissance, scanning, hash analysis, and source-code inspection. The structured approach helped uncover hidden values and flags efficiently. This methodology aligns with standard VAPT practices used in professional engagements.

13. References

- TryHackMe: Corridor Room
- CrackStation Hash Cracking Tool
- CyberChef (GCHQ)
- Nmap Official Documentation
- Browser Developer Tools (Inspect Element)
- Linux Networking Utilities (Ping)