Task Report

IIK-3100

NTNU

Norwegian University of Science and Technology

Prepared On:

03-Nov-25

Prepared By:

Sofiya Yasim Ibrahim Ali

syali@stud.ntnu.no

Table of contents

Scope of work

This report contains documentation of all the tasks completed in the course IIK3100 Home Task.

The main goal of the work was to understand how common security weaknesses appear in real systems and how they can be found and fixed in a safe and ethical way.

Each task focused on using different tools and techniques to identify security issues, understand why they occur, and learn how to prevent them.
The report includes short explanations, methods, and flags from each challenge, showing the steps I took and what I learned from each exercise.

The goal of this report is to show how ethical hacking methods can be applied in a safe and responsible environment to better understand common vulnerabilities and how to prevent them.

Task Summary

The following tasks were completed as part of the IIK-3100 hometask:

Home Tasks: All tasks except *NTNU SSH* and *Alone in Svalbard (Updated 2)* were completed.

Network Home Tasks: All tasks except *Cry With A Smile* were completed.

Web Home Tasks: All tasks were completed.

Cryptography Home Tasks: All tasks except *I love DES* were completed.

Each category covers a different part of ethical hacking. Home tasks focused on OSINT and information gathering. Network tasks explored scanning and service identification,
Web tasks involved exploiting and understanding web application vulnerabilities,
and Cryptography tasks covered decoding and decryption techniques. Each task section includes a short description, how the vulnerability was discovered, how it was exploited, and possible remediation steps to prevent similar issues.

CTF username: SSS CTF email: syali@stud.ntnu.no

Home Tasks

OSINT Theory By AI 10

Answered the questionnaire and received the flag: **Flag{osint101}**

The questionnaire was about principles of ethical hacking and OSINT. I learned differences between white hat and black hat hacking, understood the risks involved in ethical hacking, and recognized how OSINT gathers public data using advanced search techniques. The flag was given to me automatically after submitting the answers.

Technical Information Gathering Theory by AI 10

Answered the questionnaire and received the flag: **flag{ti9801}**

The Information Gathering questionnaire was about the difference between passive and active techniques, and how tools like WHOIS, DNS lookups, and banner grabbing are used to collect data about a target. It explored how ethical hackers operate within defined scopes and use tools like Burp Suite and Nikto to test web applications. I also matched common cybersecurity tools to their functions and understood how to report vulnerabilities responsibly. The flag was given to me after submitting the completed form.

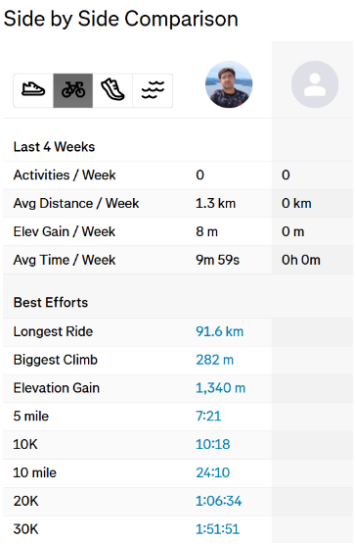
Longest Ride 50

- **Description**

  Find a user's longest ride via public profile information on Strava.

- **Vulnerability Discovery**

  I searched in Strava for "Muhammad Mudassar Yamin" and opened the public profile. The profile page exposes ride statistics and public activities. Screenshot shows the profile overview.



- **Vulnerability Exploitation**

  On the profile overview I found the "Longest ride" metric showing **91.6 km**. I captured a screenshot of the profile with the metric visible.

- **Vulnerability Remediation**

  Users should set activity visibility to “private” or “followers only” to avoid leaking personal activity statistics.

Image Meta Data Analysis 50

• Description

Extract metadata from a PNG image to find software and version used to edit it.

• Vulnerability Discovery

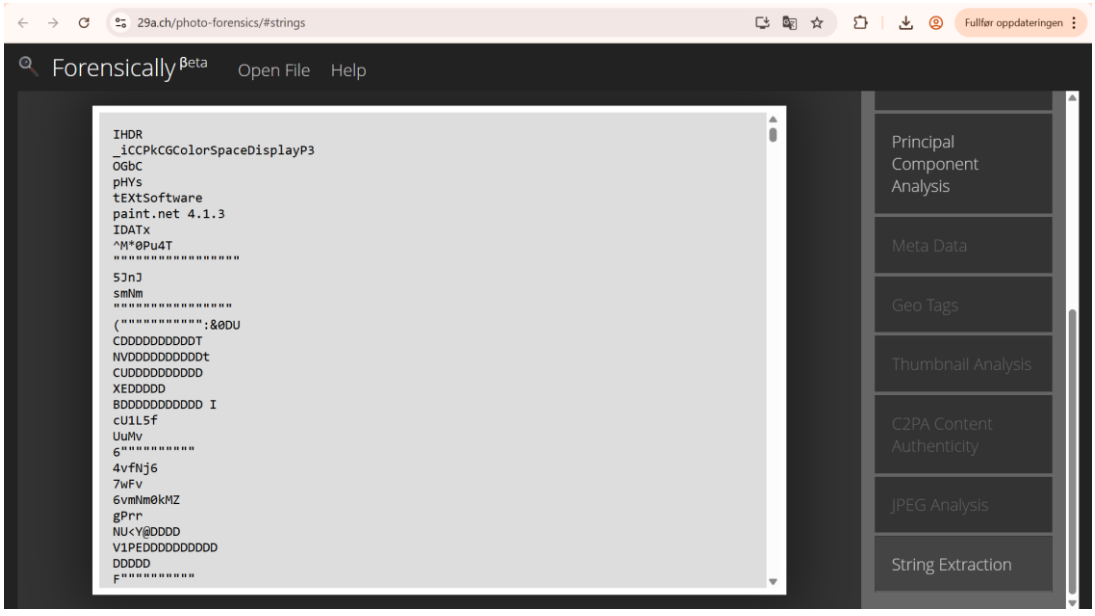
I uploaded the PNG to an online forensic tool (29a.ch string extraction). The tool lists embedded metadata and strings. Screenshot shows the tool output.

• Vulnerability Exploitation

Using the String Extraction feature I found an editor signature and flag: **Paint.net 4.1.3**. I saved a screenshot with the extracted string visible.

• Vulnerability Remediation

Strip metadata when publishing images, use tools that remove EXIF/embedded strings, and avoid leaving unnecessary build/editor info in distributed files.



Bucket of Info 50

• Description

Find which Amazon S3 bucket NTNU/Inspera uses.

• Vulnerability Discovery

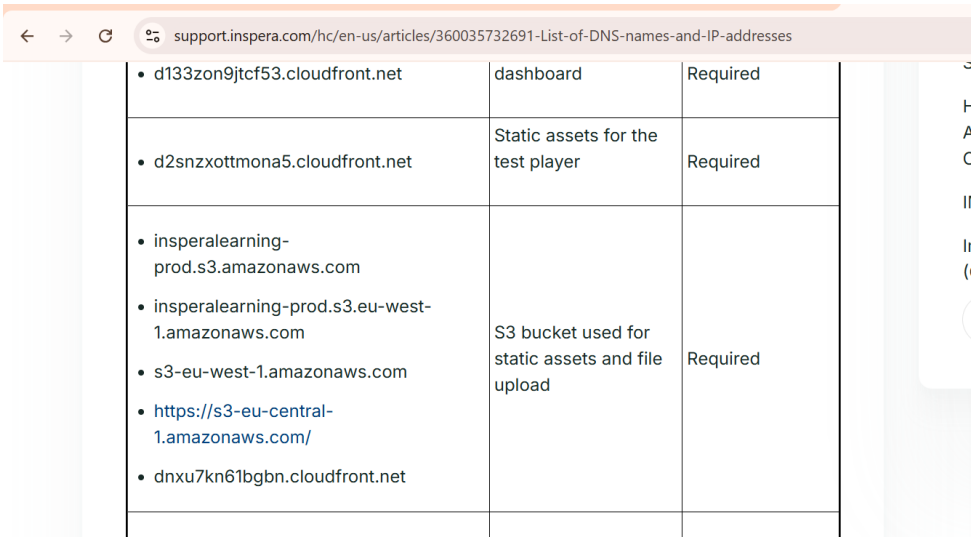
I searched for Inspera learning documentation and support pages. The support page listed DNS names and services including a bucket name. One was mentioned as the one used for static assets and file uploads. Screenshot shows the support page mention.

• Vulnerability Exploitation

Using the exact bucket name from documentation I submitted the string as the challenge answer: **insperalearning-prod.s3.eu-west-1.amazonaws.com** since this matched the hint.

• Vulnerability Remediation

Avoid exposing internal bucket names in public documentation. Use least privilege bucket policies and restrict public access to only necessary objects.



DNS Lady 50

• **Description**

Find a woman's name used as a nameserver for the domain capturethetgc.win.

• **Vulnerability Discovery**

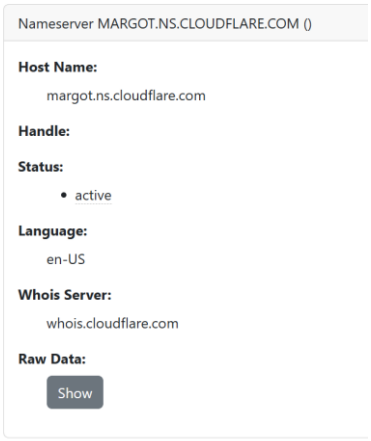
I performed a domain lookup via RDAP (rdap.org) for capturethetgc.win. RDAP displayed the domain registration details including nameservers. Screenshot shows RDAP output.

• **Vulnerability Exploitation**

From the RDAP nameserver list I found a nameserver containing the name **Margot**, which matched the challenge hint. I captured a screenshot of the nameserver line.

• **Vulnerability Remediation**

Be cautious when using personal names in public DNS hostnames; use neutral hostnames and avoid exposing personal data in infrastructure naming.



I am Well known 50

• **Description**

Find a security.txt file on a known subdomain.

• **Vulnerability Discovery**

I already knew about the math subdomain capquiz.math.ntnu.no since I use it. I tried the path /security.txt. Screenshot shows the path tested.

• **Vulnerability Exploitation**

I accessed https://capquiz.math.ntnu.no/security.txt and found the file. The path itself was the flag: **capquiz.math.ntnu.no/security.txt**. I took a screenshot showing the file contents.

• **Vulnerability Remediation**

If sensitive info is in security.txt, remove it or restrict access. Use security.txt only for safe contact info and avoid exposing internal links or secrets.

Network Home tasks

Network Scanning Theory by AI 10

Simple, I just answered the questionnaire and received the flag: **flag{ctf7382}**

The Network Reconnaissance questionnaire was about how scanning techniques like TCP Connect and SYN Stealth are used to identify devices, ports, and services on a network. I explored how reconnaissance helps assess vulnerabilities and how tools like Nmap support this process. The questionnaire also covered ways to avoid detection by IDS/IPS systems and emphasized the importance of legal and ethical compliance. The flag was given to me after submitting the form.

Finding MySql 50

Description

The goal was to identify which port was running an insecure MySQL service on 10.212.174.238.

Vulnerability Discovery

Using **Ubuntu (WSL)** on my Windows laptop, I performed a full TCP scan to discover open ports and services with nmap. I used the command: sudo nmap -p- 10.212.174.238

The scan output listed 3306/tcp open mysql, confirming that MySQL was running on port **3306**.

Vulnerability Exploitation

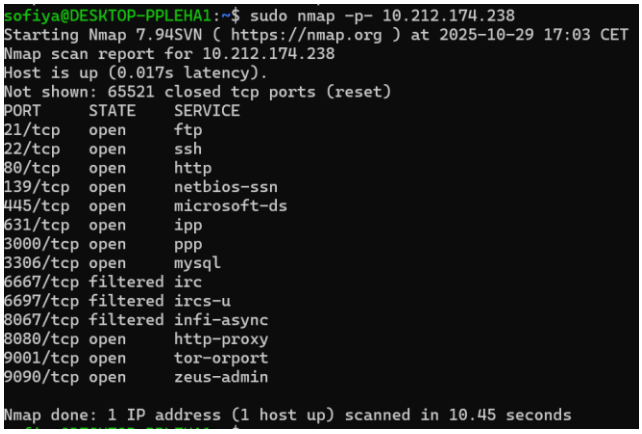
This was a discovery based task. Once the port was identified, the flag value was the port number 3306.

Vulnerability Remediation

Restrict MySQL access to trusted hosts.

Use strong authentication and disable remote root login.

Keep the MySQL server up to date to avoid known exploits.



Vulnerability Identification 50

- **Description**

  The task was to find which port in scope was vulnerable to **CVE-2013-6891**.

- **Vulnerability Discovery**

  I researched the CVE on cvedetails.com, which revealed that it affects the CUPS printing service that uses the Internet Printing Protocol (IPP) on port **631**.

- **Vulnerability Exploitation**

  Since the task only required identifying the vulnerable port, the flag was 631.

- **Vulnerability Remediation**

  Update to a patched version of CUPS.

  Restrict access to port 631 allowing only trusted hosts to connect. (Use firewall rules)

  Monitor logs for unusual activity related to printing services.

What feedback? 50

- **Description**

  The goal was to analyze home.pcapng file and find the feedback submitted by an anonymous user.

- **Vulnerability Discovery**

  I opened the file home.pcapng in Wireshark and filtered for HTTP POST requests using: http.request.method == "POST" One request to /api/feedback stood out.

- **Vulnerability Exploitation**

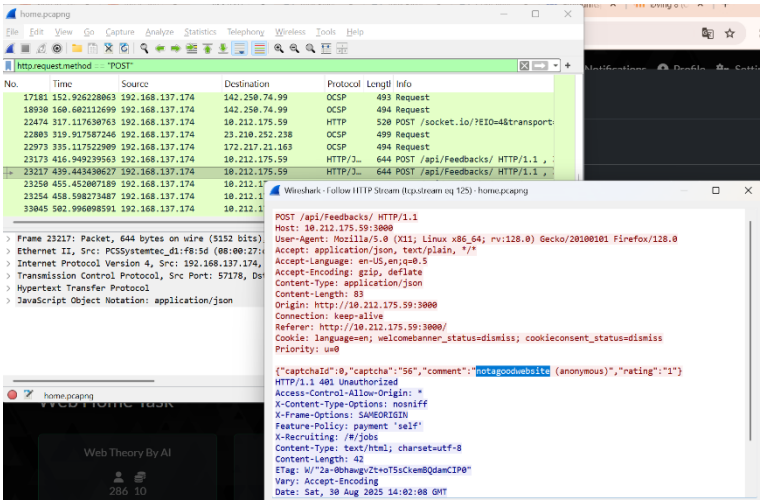
  I followed the HTTP stream and found a JSON body containing "user": "anonymous" and "comment": "notgoodwebsite". This confirmed the anonymous feedback, and the flag was the comment string **notgoodwebsite**.

- **Vulnerability Remediation**

  Sanitize and validate user input on web forms.

  Avoid storing sensitive data in plaintext network traffic.

  Use HTTPS to protect data in transit.



Tea Cups 50

- **Description**

  The challenge asked for the port number associated with the CUPS service.

- **Vulnerability Discovery**

  I researched the CUPS (Common UNIX Printing System) and confirmed it uses the Internet Printing Protocol (IPP) on TCP port 631.

- **Vulnerability Exploitation**

  By matching the protocol with the default port, I confirmed the flag: **631**.

- **Vulnerability Remediation**

  Keep CUPS updated.

  Restrict CUPS access to internal systems.

  Monitor printing services for exposure or misuse.

Web Home Tasks

Web Theory By AI 10

Answered the questionnaire and received the flag: **flag{web0980}**

The Web Application Security questionnaire was about common threats like broken authentication and cross-site scripting, and how to mitigate them using techniques such as input validation and secure session management. I explored the OWASP Top Ten and its role in helping developers prioritize security risks. The questionnaire also covered best practices for cloud security, legal compliance like GDPR and PCI-DSS, and tools for monitoring and vulnerability management. The flag was given to me automatically after submitting the completed form.

Multi Encoder 20

• **Description**

Decode data that had been encoded multiple times.

• **Vulnerability Discovery**

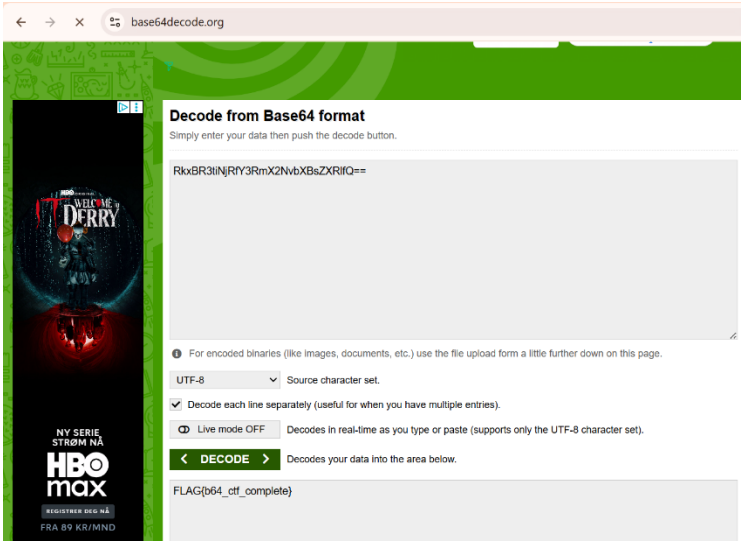
I recognized the encoded string pattern (Base64) due to its characters and padding =. I used an online Base64 decoder and repeatedly decoded the string around 10 times.

• **Vulnerability Exploitation**

After several decoding rounds, the final output displayed the flag: **FLAG{b64_ctf_complete}.**

• **Vulnerability Remediation**

Using cryptography exercise. Encoding is not encryption, so sensitive data should use strong cryptographic algorithms.



Variable Detective 40

• **Description**

Find a flag hidden in the page's JavaScript variables.

• **Vulnerability Discovery**

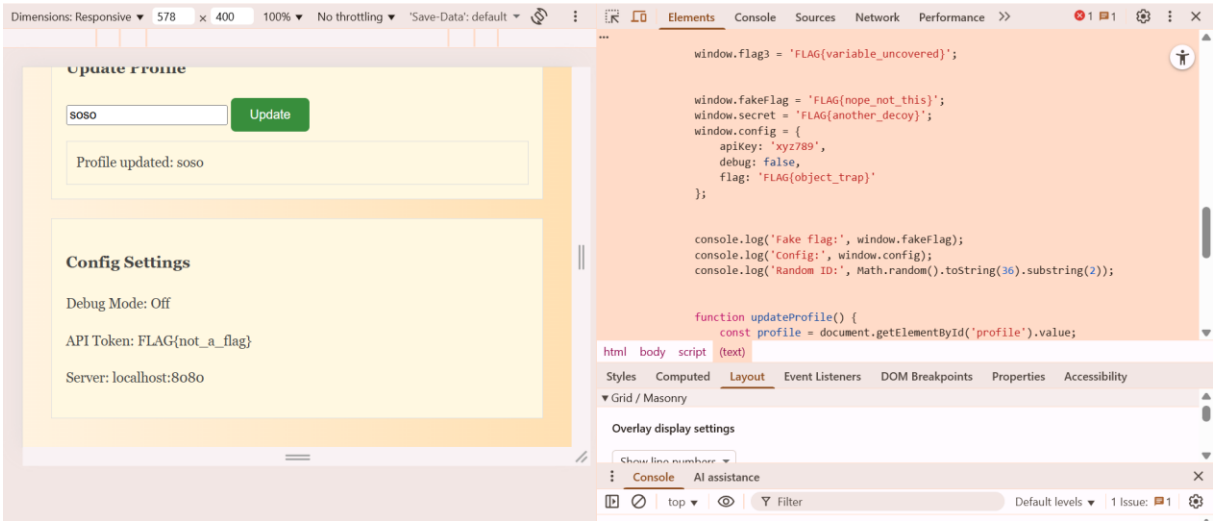
I inspected the page's HTML/JavaScript using DevTools. A variable stored directly in the page contained a string in the format of a flag.

• **Vulnerability Exploitation**

The hidden variable revealed the flag: **FLAG{variable_uncovered}.**

• **Vulnerability Remediation**

Never store or expose sensitive data such as keys or flags in client-side code.



Function Caller 40

- **Description**

  The challenge hinted that a function could reveal the flag.

- **Vulnerability Discovery**

  I examined the page's JavaScript functions and found one named revealFlag() that contained a return statement with a flag string.

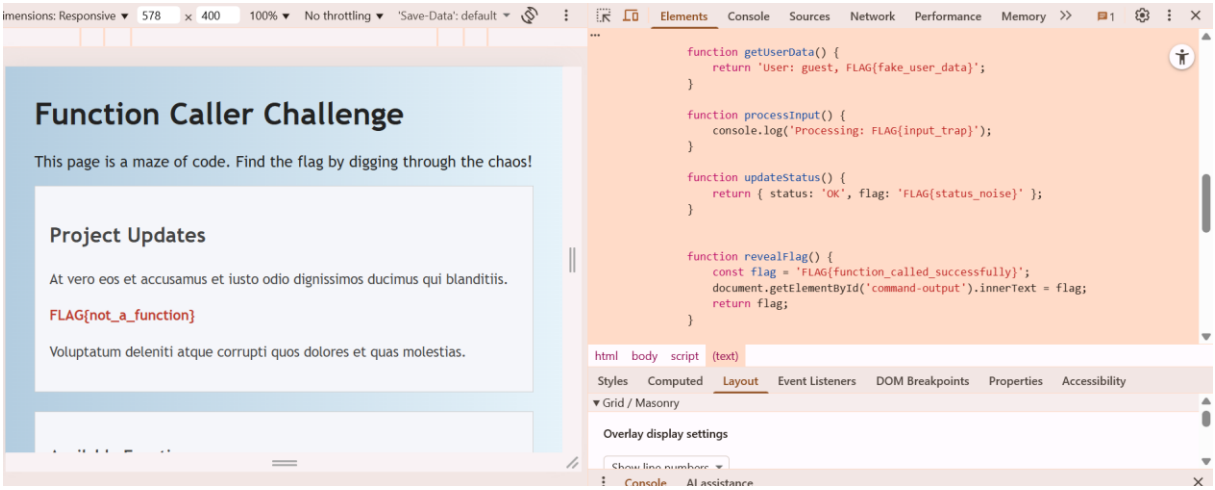
- **Vulnerability Exploitation**

  In the browser console, I executed the function and it returned the flag:

  FLAG{function_called_successfully}.

- **Vulnerability Remediation**

  Do not include sensitive flags or credentials in exposed JavaScript code.



```
> revealFlag()
< 'FLAG{function_called_successfully}'
>
```

Parameter Tampering 40

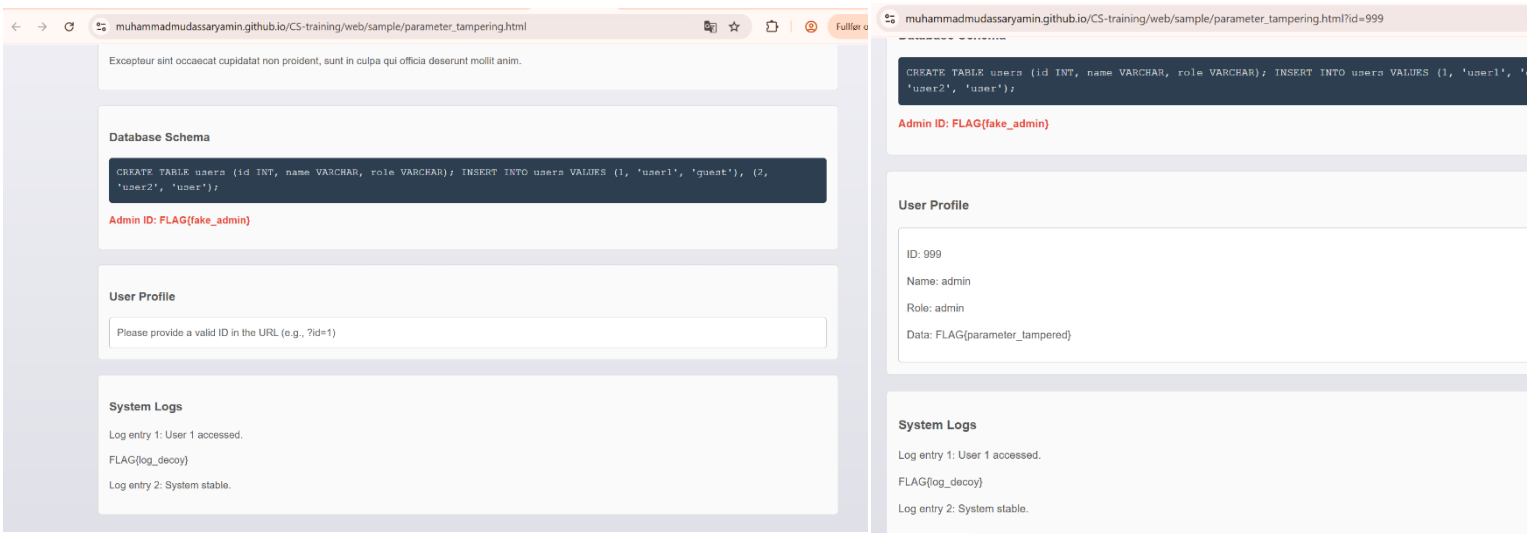


Figure 1:Snapshots before and after adding "?id=999" on the url. Check the User Profile

- **Description**

  Exploit weak access control where user data is loaded directly from URL parameters.

- **Vulnerability Discovery**

  While analyzing the page parameter_tampering.html, I noticed that it displayed user information based on the id parameter in the URL. This suggested that the page might load data directly from JavaScript without validating user privileges.

- **Vulnerability Exploitation**

  By changing the URL from /parameter_tampering.html to /parameter_tampering.html?id=999, like the id in the source code (look at the screenshot) the page displayed the admin profile along with the flag: **FLAG{parameter_tampered}.**

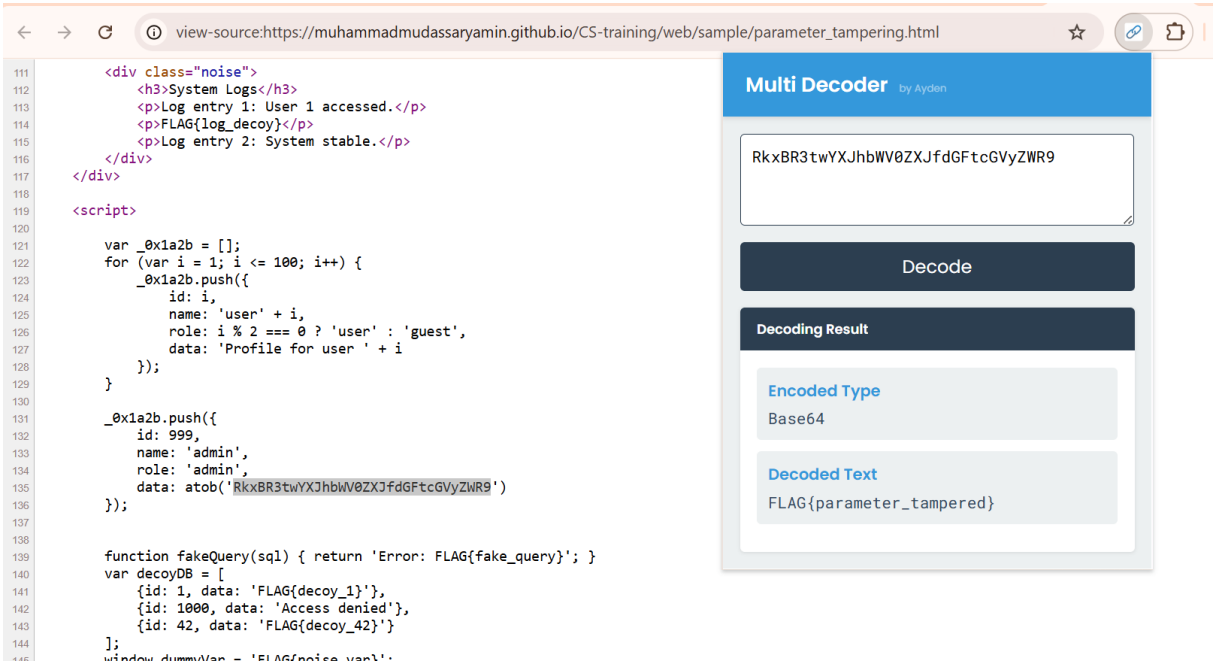
  I also inspected the page source and found an encrypted version of the same flag, which I successfully decoded using a Chrome multi-decoder tool.

- **Vulnerability Remediation**

  Validate all user input server-side and enforce strict access control.

  Do not rely on client-side logic or parameters for authorization.

  Use session-based or token-based authentication to protect user data.



SQLI 40

- **Description**

  Identify and exploit a SQL injection vulnerability.

- **Vulnerability Discovery**

  I explored “Tools” on the navigation bar and selected Task 1: SQL User Search. When I tested the input field with alex', it caused an SQL error, confirming the page was vulnerable to SQL injection.

- **Vulnerability Exploitation**

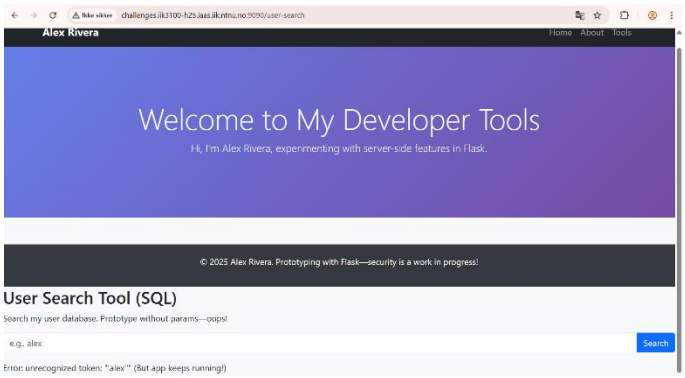
  I used the payload ' OR 1=1--, which bypassed the filter and returned all users. This revealed, a user named “secret” with the flag: **CTF{LEETUSER_1337}.**

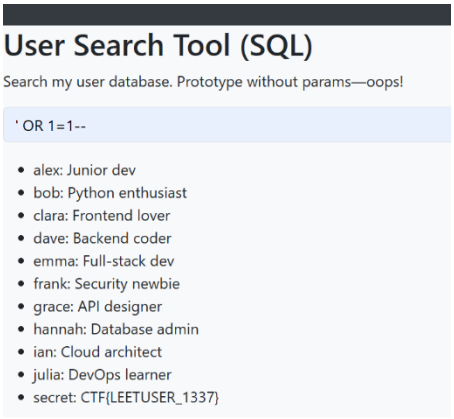
- **Vulnerability Remediation**

  Use parameterized queries or prepared statements.

  Validate user input.

Avoid displaying database errors in production.





Local File Inclusion (LFI) 40

- **Description**

  Exploit a Local File Inclusion vulnerability to access a restricted file.

- **Vulnerability Discovery**

  In Task 3: File Viewer, the page loaded files using a URL parameter like ?file=welcome.txt. The hint on the home page suggested a possible inclusion vulnerability.

- **Vulnerability Exploitation**

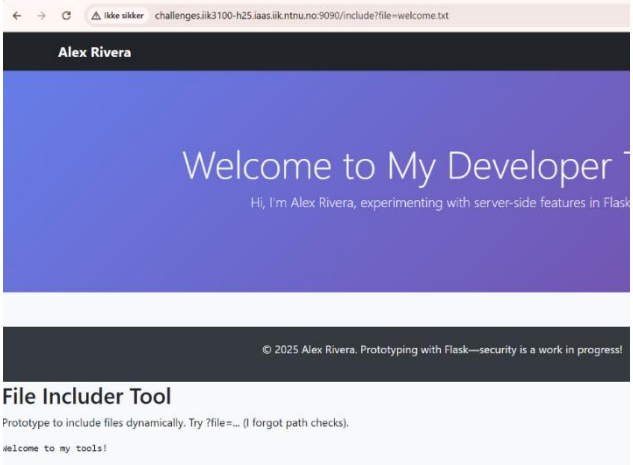
  By changing the parameter to ?file=secret_flag.php, the server executed the restricted file. This revealed the flag: **CTF{LFI_is_fun}.** This worked because the backend used the filename directly from the URL without restrictions.

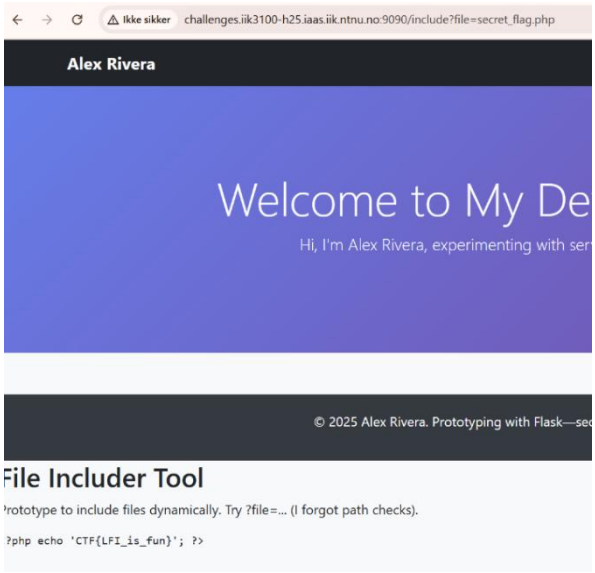
- **Vulnerability Remediation**

  Restrict file inclusion to a whitelist of allowed files.

  Sanitize user supplied file paths.

  Disable remote file inclusion in PHP configurations.





Path Traversal 40

- **Description**

  Exploit a path traversal vulnerability to access sensitive files.

- **Vulnerability Discovery**

  On Task 3: File Viewer, the URL was:

  http://challenges.iik3100-h25.iaas.iik.ntnu.no:9090/file-view?path=welcome.txt.

- **Vulnerability Exploitation**

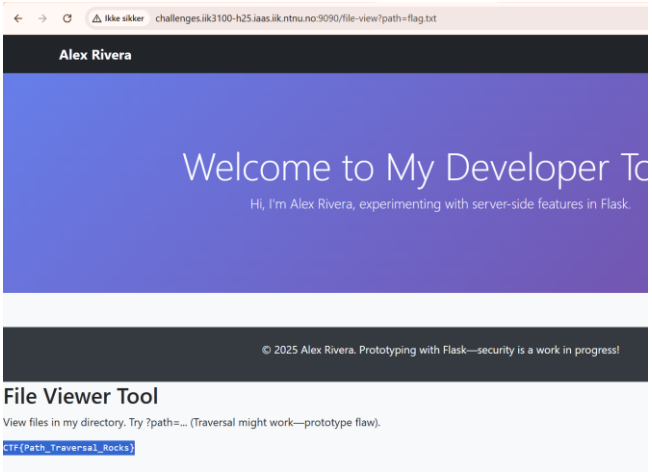
  I figured I could get to the flag.txt by changing the path parameter from welcome.txt to flag.txt, and then the page displayed the hidden flag file with flag: **CTF{Path_Traversal_Rocks]** This occurred because the backend directly used the file path without validation. The website directly uses the path parameters to read files without validation or restrictions, so just by changing the parameter I requested a different file.

- **Vulnerability Remediation**

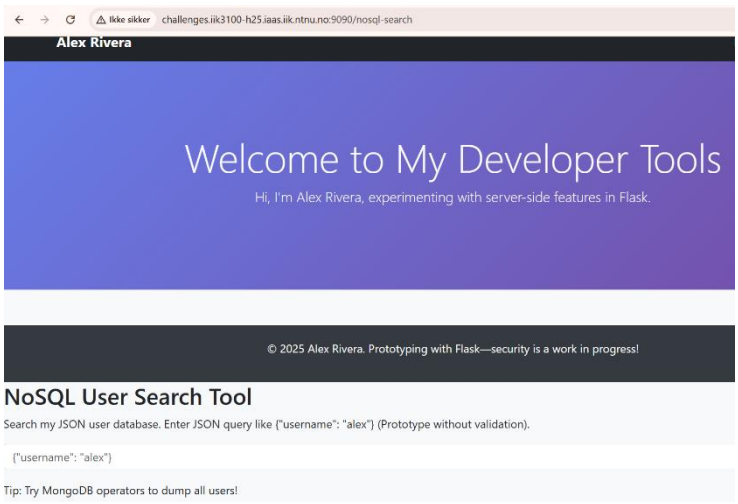
  Sanitize input by removing ../ patterns.

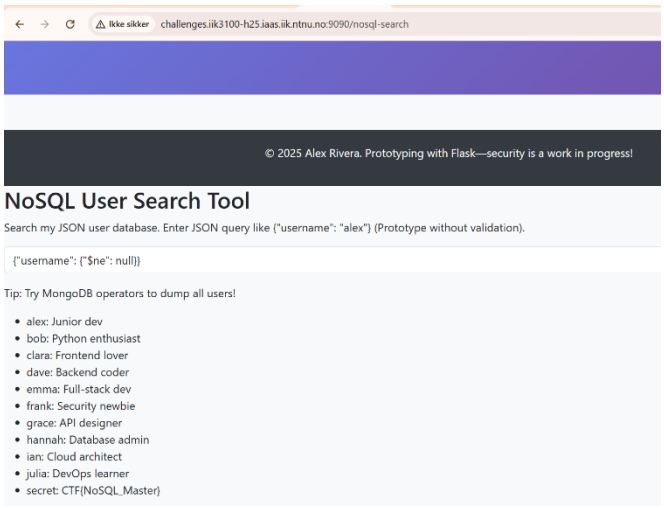
  Restrict file access to specific directories.

  Enforce server-side access control checks.



NoSQLI 40





- **Description**

  Exploit a NoSQL injection vulnerability to extract data.

- **Vulnerability Discovery**

  In Task 4: NoSQL Search, a hint suggested using MongoDB operators. The search input accepted JSON directly.

- **Vulnerability Exploitation**

  I entered the payload {"username": {"$ne": null}} into the search box, which used the $ne operator to match all users where the username is not null. The server returned a list of users, and the flag: **CTF{NoSQL_Master}** with the “secret” user. It worked because the backend accepted raw JSON queries without validation.

- **Vulnerability Remediation**

  Validate and sanitize JSON input.

  Use object schema validation and query parameterization.

  Avoid directly parsing user input as database queries.

Broken Web 50

- **Description**

  Find a hidden flag in the broken website.

- **Vulnerability Discovery**

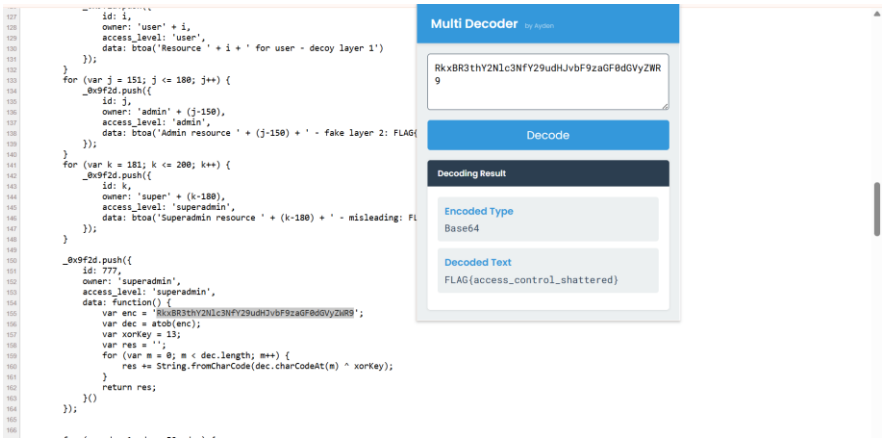
  After checking multiple fake flags on the website, I inspected the HTML source and found an encoded Base64 string associated with id:777.

- **Vulnerability Exploitation**

  I used a multi-decoder Chrome plugin to decode the string and found the encoded type base64 and decoded text which was the flag: **FLAG{access_control_shattered}** as you can see on the screenshot.

- **Vulnerability Remediation**

  Avoid storing sensitive information in client accessible source code. Always secure ID-based access with proper backend validation.



Eggmarket Egg-FI 50

Description

Find the flag on the egg website in a hidden file.

Vulnerability Discovery

The view egg endpoint used a query parameter img=yoshi.png to load images dynamically.

Vulnerability Exploitation

I noticed the image was loaded through a query parameter, so I tested if the site was vulnerable to Local File Inclusion by changing the img value. I changed the URL to img=../../flag.txt. and the clicked on “Download Image,” and I got access to the flag file with flag:

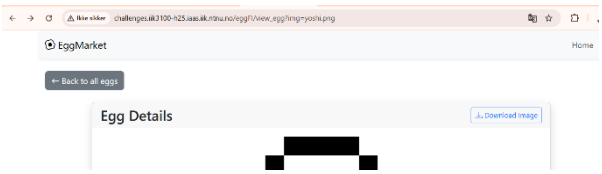
iik3100{l0c4l_f1l3_1nclus10n_vuln3r4b1l1ty}.

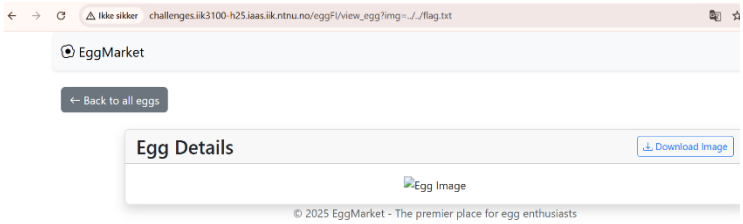
Vulnerability Remediation

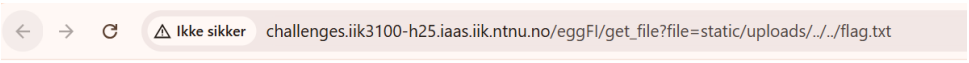
Restrict file paths and enforce input sanitization.

Use server-side mapping for static assets.

Disable directory traversal patterns in code.








```
iik3100{l0c4l_f1l3_1nclus10n_vuln3r4b1l1ty}
```

Eggmarket Ticketchecker 50

Description

Craft a payload that can steal the admin's cookies when they visit the provided URL.

Vulnerability Discovery

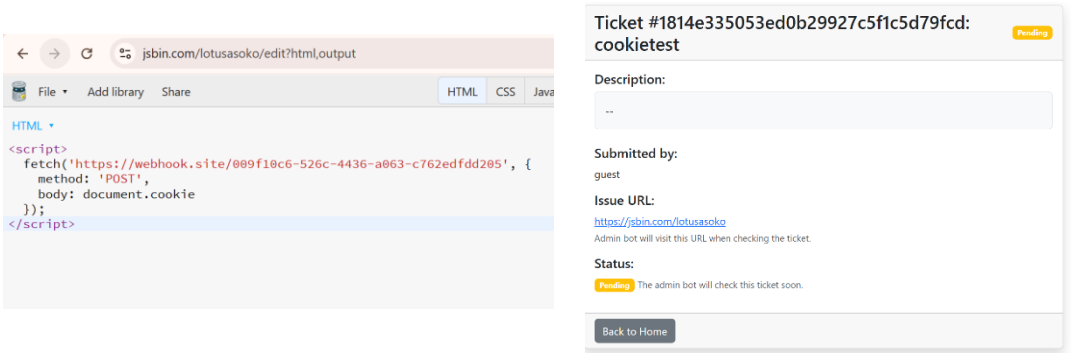
Based on the challenge description, I understood that the admin bot would open the URL I submitted. This made me think it might be possible to run JavaScript in the admin's browser and access their cookies which suggested a potential for client-side exploitation if the bot executed JavaScript on external pages.

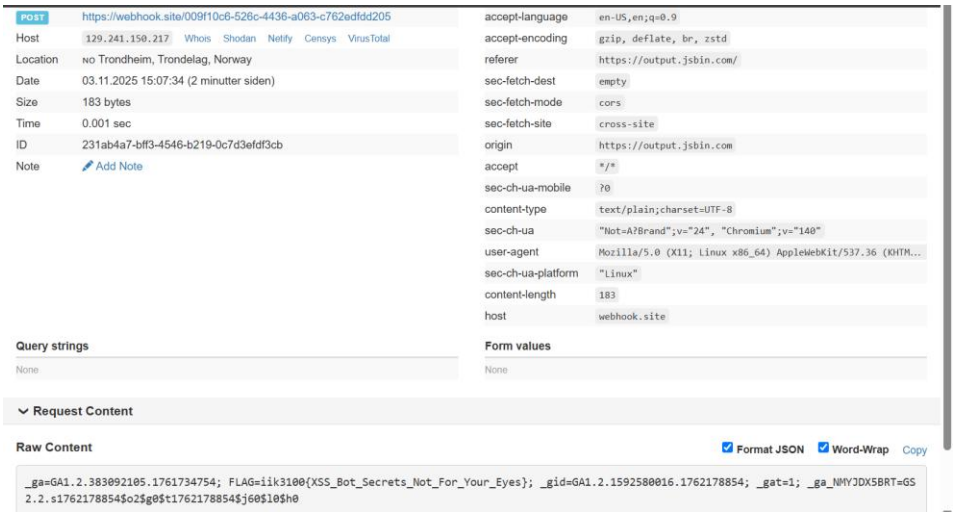
Vulnerability Exploitation

I created a JSBin page (https://jsbin.com) containing a payload that sent `document.cookie` to my webhook via a POST request. I submitted a ticket with the JSBin URL in the "Issue URL" field. When the admin bot visited the page, the script executed successfully, and the bot's cookies were captured in my webhook log. As you can see on the screenshot, I found the flag: **iik3100{XSS_Bot_Secrets_Not_For_Your_Eyes}** at the bottom of the page under Request Content.

Vulnerability Remediation

To prevent this vulnerability, the application should sanitize or restrict external URLs submitted via the ticket form. The admin bot should be sandboxed or run in a hardened environment that blocks JavaScript execution or disables access to cookies.





Eggmarket Shopparameter 50

- **Description**

  Exploit client-side validation flaws to purchase an item without sufficient credits.

- **Vulnerability Discovery**

  By inspecting the HTML code in Developer Tools, I saw the “Buy Flag” button was disabled and a hidden field showed credits=100.

- **Vulnerability Exploitation**

  I removed the disabled attribute and changed the credits value from 100 to 1000. After submitting the form, the server accepted the request without checking my actual balance and gave me the flag: **iik3100{client_side_validation_fail}.** The challenge was solved by exploiting weak server-side validation and trusting client-side input.

- **Vulnerability Remediation**

Validate all critical logic server-side, not in the client.

Never trust hidden fields or disabled inputs for access control.

Double check balances and permissions on every transaction.

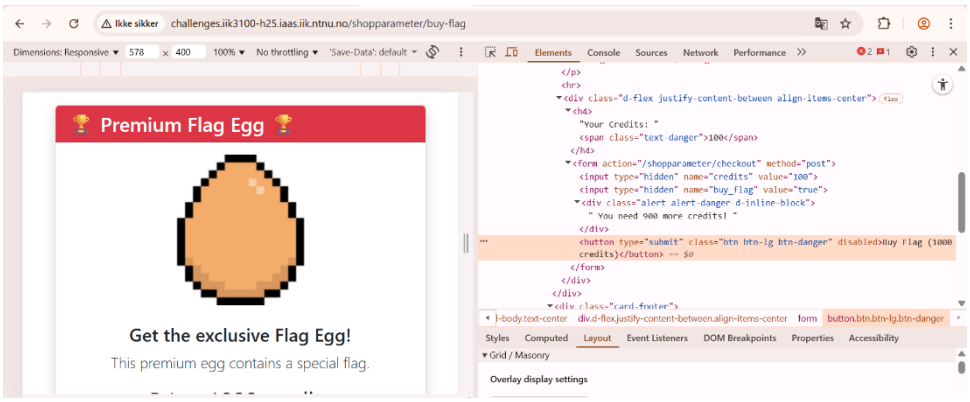
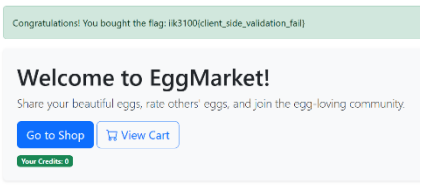


Figure 2: HTML code before

Figure 3:HTML code after



Eggmarket database 50

- **Description**

  Exploit SQL injection vulnerabilities in the login and user search features.

- **Vulnerability Discovery**

  The login form was vulnerable to SQL injection. I verified this by entering ' OR 1=1--, which bypassed authentication and logged me in as admin.

  Vulnerability Exploitation

  After login, I used the search function with a UNION-based payload:

  ' UNION SELECT null,null,(SELECT value FROM secrets LIMIT 1),null --.

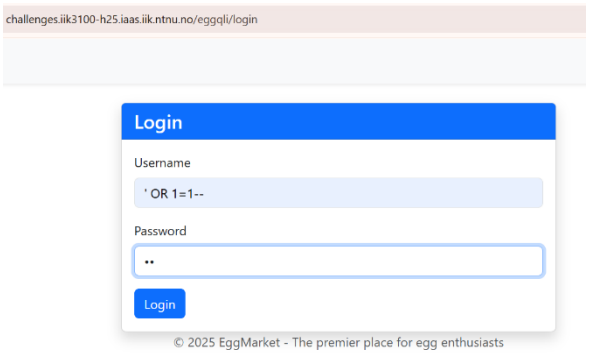
  This SQL query added a fake row to the search results and pulled one value from a secrets table, which could contain the flag. The flag**: iik3100{sqli_kung_fo_master}** appeared in the email column, confirming the injection worked.

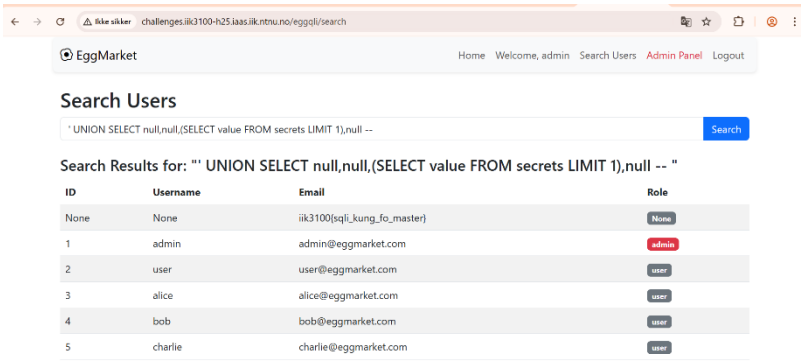
- **Vulnerability Remediation**

Implement prepared statements for SQL queries.

Filter and escape user input.

Limit database privileges and sanitize all parameters.





Free Purchase 50

- **Description**

Exploit a logic flaw in the pricing system to buy an item for free.

• Vulnerability Discovery

After bypassing authentication and logging in with ' OR 1=1--, I edited HTML fields in DevTools and tried changing the html discount field to 100%, but got the error that the discount was invalid when trying to purchase, so the discount field had a limit (50%) and both frontend and backend blocked it, but the price field could be freely changed.

• Vulnerability Exploitation

I set the price field to 0 while keeping discount at 0. The purchase succeeded, returning the flag: **CTF{task_3_logic_flaw}.**

• Vulnerability Remediation

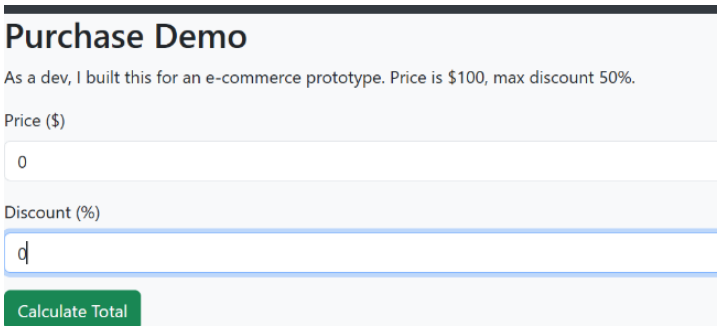
Validate business logic on the server side.

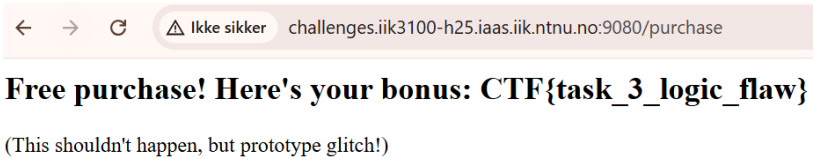
Never rely on client-side price or discount controls.

Implement server integrity checks before finalizing transactions.

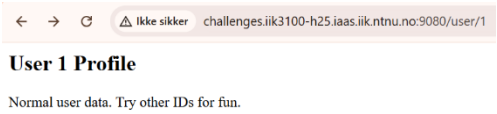
Figure 5: HTML code before

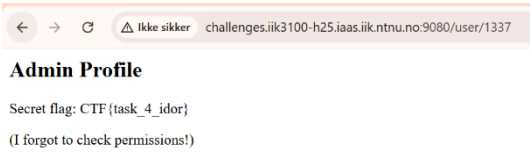
Figure 4:HTML code after





Leet User 50





- **Description**

  Exploiting an Insecure Direct Object Reference (IDOR) vulnerability to access unauthorized data.

- **Vulnerability Discovery**

  The challenge hinted that user IDs were predictable. I navigated to /user/1 and /user/1337 manually.

- **Vulnerability Exploitation**

  Starting from /user/1, which showed a normal user message, I tried accessing /user/1337 a very basic leetspeak number often used. That URL led to the Admin Profile, and flag: **CTF{task_4_idor}.** The home page showed “I forgot to check permissions!” which confirmed

that this was an IDOR vulnerability (Insecure Direct Object Reference). By directly accessing the admin's user ID without proper authorization checks, I was able to retrieve the flag.

- **Vulnerability Remediation**

  Implement proper access control for all resources.

  Use session-based authorization instead of URL parameters.

  Avoid exposing predictable resource ID's.

XSS or Something Else 50

• Description

Identify whether the search field was vulnerable to XSS or another issue exposing sensitive data.

• Vulnerability Discovery

I attempted to solve the challenge by injecting various XSS payloads into the search field, such as <script>alert(1)</script> and <img src=x onerror=alert(1)>, as the task suggested that the input might be vulnerable. However, none of the scripts were executed and the input was reflected but HTML-escaped when I looked at the HTML code, so the browser treated it as plain text rather than executable code

• Vulnerability Exploitation

Although script injection did not work, I inspected the application manually through the browser's Developer Console and tried to lookup document.cookie. This revealed that a flag was stored as a cookie value: **CTF{task_1_reflected_xss}.** Even though no code was executed, sensitive data was exposed through client-side storage once again.

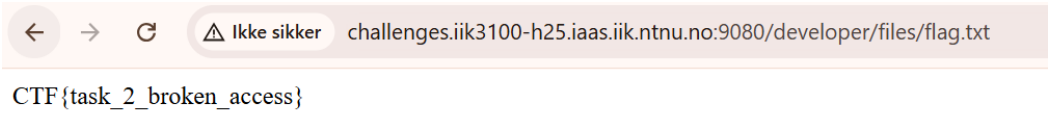
• Vulnerability Remediation

Avoid storing sensitive data (like flags or tokens) in client-side cookies.

Implement proper output encoding and Content Security Policy (CSP).

Validate and sanitize user input before reflecting it in HTML responses.

Broken Access Control 50



- **Description**

Exploit missing access restrictions to access protected files (flag.txt) on the web server.

- **Vulnerability Discovery**

A hint on the Tasks page mentioned that "Task 2 is located in /developer/files" and that protection was "forgotten." This suggested that the directory might be accessible without authentication.

- **Vulnerability Exploitation**

By navigating directly to /developer/files/flag.txt, I accessed the file without needing any credentials. The file contained the flag: **CTF{task_2_broken_access}.** This confirmed the absence of proper access control mechanisms on the server.

- **Vulnerability Remediation**

Enforce authentication and authorization checks on all resources.

Use server-side role validation to restrict sensitive paths.

Regularly audit web server directories for exposure.

Cryptography Home Task

What the logo is hiding? 20

- **Description**

  Extract hidden data embedded inside an image file using steganography.

• **Vulnerability Discovery**

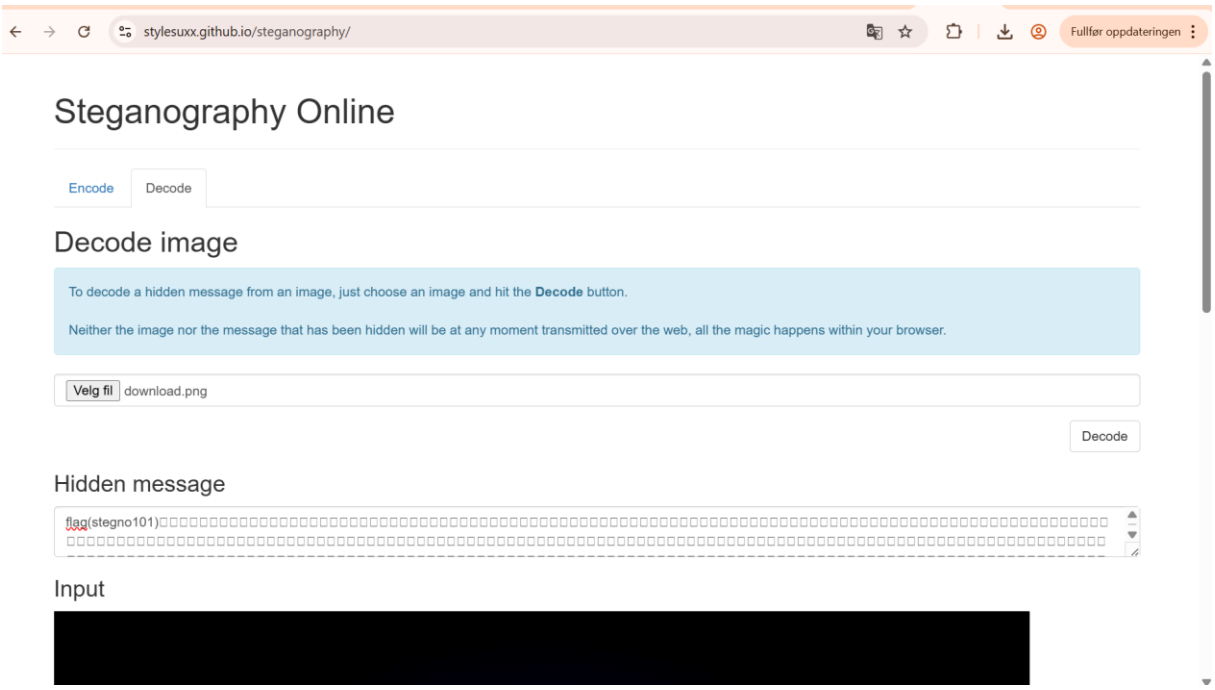
I used the online tool https://stylesuxx.github.io/steganography/ to analyze the provided image (download.png).

• **Vulnerability Exploitation**

By running the Decode function in the tool, I extracted a hidden message embedded in the image pixels. The decoded output revealed the flag: **flag{stegno101}.**

• **Vulnerability Remediation**

To prevent information leakage, remove hidden data or metadata from images before sharing them publicly.



Figur 1: https://stylesuxx.github.io/steganography/

Figur2: download.png

Breaking very week RSA 60

- **Description**

  Decrypt a weak RSA message using a small modulus and exponent.

• Vulnerability Discovery

Using the RSA example code from GeeksforGeeks, I replaced the public key values (e = 3, n = 62615533) and ciphertext (5472482). I then added a small loop to factorize n into its two prime numbers.

• Vulnerability Exploitation

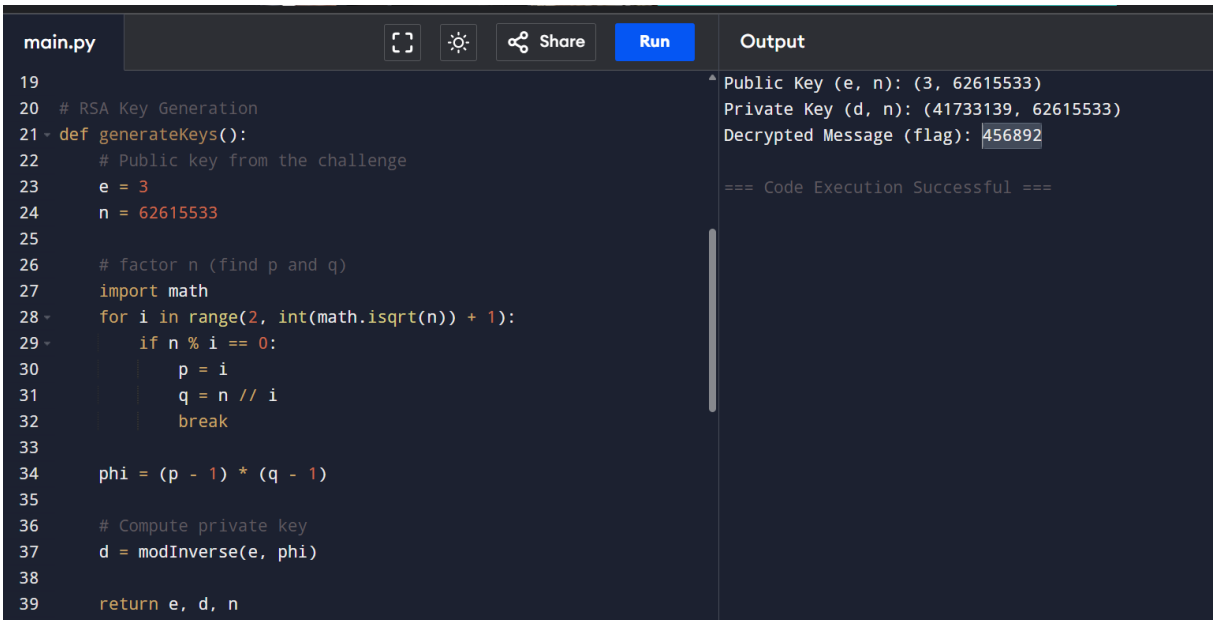
Once the primes were identified, I computed the private key d and decrypted the message, revealing the flag: **456892**.

• Vulnerability Remediation

Use RSA keys of at least 2048 bits.

Avoid small public exponents like e = 3.

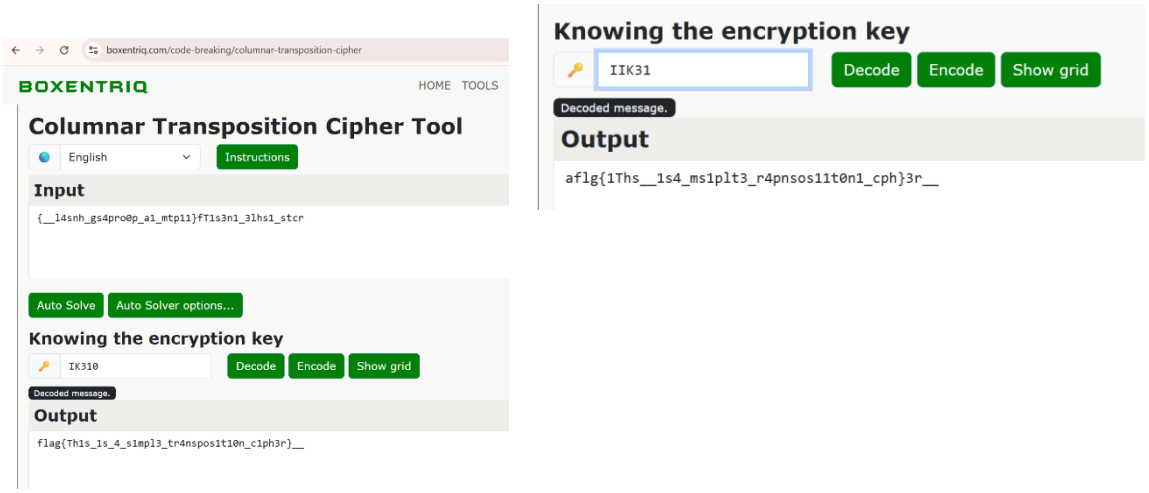
Generate keys using secure, vetted cryptographic libraries.



The two snapshots show the edited parts of the code from the Python code RSA sample:

https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/

Old Crypto Transposition 60



- **Description**

Decrypt a transposition cipher using course-related hints.

- **Vulnerability Discovery**

I used the Boxentriq Transposition Cipher Solver online tool. The hint mentioned a 5-character numeric variation of the course code, so I tested several keys such as “IIK31” and “IK310.”

- **Vulnerability Exploitation**

Since the hint was 5 numeric variations of a course code I figured it was this course IIK3100, so I first tried IIK31, but as you can see on the snapshot, the output gave almost the correct answer, but it said aflg instead of flag so I figured I was close. I put in the encrypted message, with an encryption key, using the key IK310, the output text was then correctly formed, revealing the flag: **flag{Th1s_1s_4_s1mpl3_tr4nspos1t10n_c1ph3r}.**

- **Vulnerability Remediation**

Avoid using classical ciphers like transposition for data protection. Use modern encryption algorithms (AES, RSA) for confidentiality and integrity.

SUM: 1380 POINTS