

NetFlowInsight

Introduction

The program, NetFlowInsight, is a web-based application designed to analyze network traffic and provide insights into captured packet data. It allows users to upload .pcap files, analyze them for potential threats, and manage notes on their analysis. The application integrates with external APIs (e.g., Groq and VxAPI) and uses Zeek to carve files from the uploaded .pcap files to perform file analysis stored within the .pcap files and generate reports.

The primary purpose of the program is to assist network administrators and security analysts in automating of identifying malicious activities or anomalies in their network traffic.

Functions

1. File Analysis:

- Users can upload .pcap files for analysis.
- The application processes the files and generates detailed reports using external APIs.

2. Note Management:

- Users can create, edit, and delete notes related to their analysis.

3. API Key Management:

- Users can manage their API keys for external integrations.

User Interface Description

1. File Upload and Analysis

- **Purpose:** Allows users to upload .pcap files for analysis.
- **Interface:**
 - A file upload form where users can select .pcap files.

- Validation ensures only .pcap files are accepted.
- Results are displayed after analysis, including a summary and detailed report.

2. Notes Management

- **Purpose:** Enables users to manage notes related to their analysis.
- **Interface:**
 - A list of notes displayed on the "View Notes" page.
 - Options to add, edit, or delete notes.
 - Inline editing with input sanitization to prevent XSS attacks.

3. API Key Management

- **Purpose:** Allows users to manage their API keys.
- **Interface:**
 - A masked display of the current API key.
 - An "Edit" button to update the API key.
 - Validation to ensure the API key is not empty.
 - CSRF protection for secure updates.

4. User Authentication

- **Purpose:** Ensures secure access to the application.
- **Interface:**
 - Login and signup forms with password.
 - Rate limiting to prevent brute force attacks.

5. Dashboard

- **Purpose:** Provides an overview of uploaded files and analysis results.
- **Interface:**
 - A list of uploaded files with timestamps.

Structure of the Program

1. Backend

1. **Framework:** Flask
2. **Key Components:**
 - **Routes:**
 - `/upload`: Handles file uploads.
 - `/view_notes`: Displays and manages notes.
 - `/update_api_key`: Updates the user's API key.
 - `/login` and `/signup`: Handles user authentication.
 - `/file_analysis_result`: To see the results of the analyzed files
 - **Database Models:**
 - User: Stores user information, including API keys.
 - Notes: Stores user-created notes.
 - PcapLoc: Tracks uploaded .pcap files.
 - FileAnalysis: Analysis of each file carved from the uploaded pcap file
 - **External Integrations:**
 - Groq API for natural language processing of analysis results.
 - VxAPI for file scanning and threat detection on Hybrid Analysis
 - Zeek for carving payloads from the uploaded .pcap files

2. Frontend

- **Technologies:** HTML, CSS, JavaScript
- **Key Features:**
 - **Dynamic Forms:**
 - File upload form with validation.
 - Inline note editing with sanitization.

- **AJAX Requests:**

- Used for updating notes and API keys without reloading the page.

System Architecture

The System uses an Ubuntu Docker container, which holds the NetFlowInsight app instance and the Zeek instance. The container also holds an instance of the sqlite database. The app directly communicates with the Groq and Hybrid Analysis through their python api. The following diagram gives a visual representation of the system architecture.

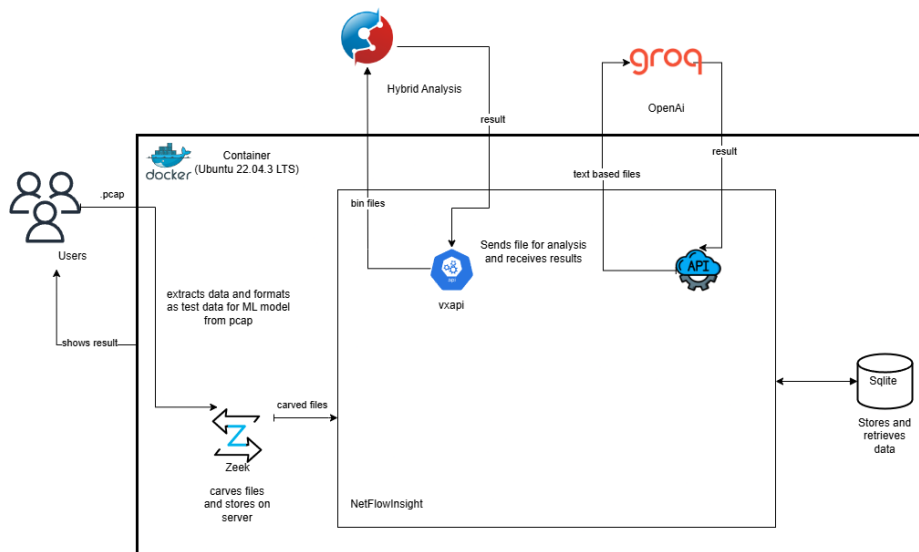


Figure 1: System Architecture

User guide

1. Signing up:

- Navigate to the “Signup” button from the navigation bar
- Fill up the signup form with the required details
- Click the submit button to create a profile

NetFlowInsight Login Sign Up

Sign up

Email Address

First Name

Last Name

Password

Confirm Password

Submit

Figure 2: Signup page

2. Logging In:

- Open the NetFlowInsight application in your preferred web browser.
- Enter your email address and password in the provided login fields.
- Click on the "Login" button to access your account.

Login

Email Address

Password

Login

Figure 3: Login page

3. Editing Groq api key:

- After logging in, you'll be on the homepage. Navgate to the "Profile" section in the navigation bar
- Click on the "Edit" button beside the "Groq API key" section.
- Input the Groq api key before starting the analysis since the api key is a requirement to run the analysis

Your Profile

First Name:	Meftahul
Last Name:	Islam
Email:	meftahul.islam@tuni.fi
Groq API Key:	<input type="text" value="gsk_"/> Edit
Edit API Key:	<input type="text" value="gsk_"/>

SaveCancel

Figure 4: Profile page for editing api-key

4. Uploading Pcap Files:

- Go to “NetflowInsight page” from the navigation bar. This page will have an upload field.
- Click on the "Choose File" button to select the Pcap file you want to analyze.
- Once the file is selected, click on the "Upload" button to initiate the analysis process.
- Wait for the analysis to complete. The duration may vary depending on the size of the file and the complexity of the analysis.

Logged in successfully!

Upload PCAP File

Select PCAP File:

No file chosen

Your Uploaded Files

Figure 5: Uploading pcap files

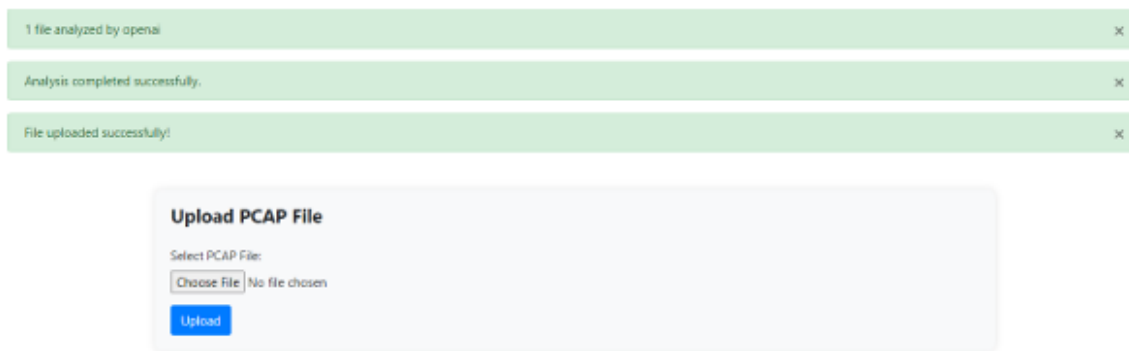


Figure 6: Upload confirmation

5. Viewing Analysis Results:

- Once the analysis is complete, navigate to the Analysis tab in the navigation bar and expand it.

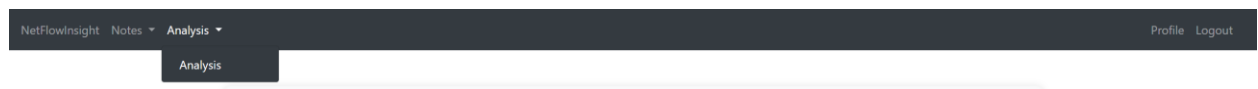


Figure 7: Navigating to Analysis

Explore the analysis results of files that have been extracted from your uploaded pcap file. Gain detailed information, insights, and verdicts related to the extracted files.

6. Note Keeping:

- Utilize the "Notes" feature from the navigation bar to create and store notes related to the results of your analyzed pcap files.
- Write down important information, ideas, to-do lists, or any relevant notes to keep track of your analysis findings.
- Save and organize your notes within the application for easy reference and retrieval.



Figure 8: Navigating to Notes

7. Log Out:

- To log out of your account, click on the "Log Out" button or sign out option in the application's navigation menu.

Functions and outputs

File Analysis Results:

- **Description:** This field presents the results of the analysis performed on the extracted files.
- **Contents:** It includes information such as file type, hashes, detected threats (if any) and any other relevant metadata extracted during the analysis process.
- **Purpose:** The file analysis results help users identify and understand the characteristics, potential risks, and anomalies associated with the extracted files from the uploaded pcap file.

Below is a screenshot taken from the application after the file analysis process and with result generated

Pcap File	Date Uploaded	Extracted File Hash	File Type	Result
sample.pcap	2023-05-26 01:06	63fa7603d63bad0857e2d28e344996821120ae58fc585fee6d11a210d01b2db1	plain	This code is written in jQuery and is part of a cycle plugin which allows users to cycle through elements on a webpage. The code allows for customizing the cycling, such as setting the speed, the height, and the timeout. It also provides transitions such as fading between elements. This code is not malicious.
		895815640fe855a180d0cf9769153e5fc00d2520d44af034fa6cc6fb615ee6af	png	This is a png file

Figure 9: Output demonstration

Notes:

- **Description:** This output allows users to view their own notes and observations regarding the file analysis results for each .pcap files.

- **Contents:** Users can view text-based notes, comments, or additional information relevant to their specific requirements or findings that they had created previously along with the date they were created.
- **Purpose:** User notes serve as a personalized section where users can record their observations, insights, or actions taken based on the analysis results. It enables collaboration, documentation, and facilitates future reference.

Below is an example of the output of the Notes functionality

Your Notes

Pcap File	Notes
capture.pcap	<div> <div> <div>&lt;script>>alert(1)&lt;/script>></div> <div>2025-05-09 11:58</div> <div>✎ ✕</div> </div> <div> <div>Add a new note</div> <div>Add Note</div> </div> </div>

Figure 10: Notes functionality demonstration

Edit api-key:

- **Description:** This field allows users to input and manage their Groq API key for enabling external integration.
- **Contents:** It includes the user-provided Groq API key, which can be added, edited, or removed at any time to manage access to Groq's inference capabilities.
- **Purpose:** The Groq API key integration enables the application to connect with Groq's services, allowing for analysis of text based files carved from the pcap files.

Below is an example of the output of the editing api keys functionality

The screenshot shows a web form titled "Your Profile" with a blue header. The form contains the following fields and controls:

- First Name:** Meftahul
- Last Name:** Islam
- Email:** meftahul.islam@tuni.fi
- Open API Key:** A text input field with a blue "Edit" button next to it.
- Edit API Key:** A text input field containing the text "gsk_".
- Buttons:** At the bottom left, there are two buttons: a green "Save" button and a grey "Cancel" button.

Figure 11: Updating api-key demonstration

Security Mechanisms

- **Authentication:**
 - Managed using Flask-Login.
- **Input Validation:**
 - Sanitization of user inputs using [bleach](#) and JavaScript.
- **Rate Limiting:**
 - Prevents abuse of endpoints using Flask-Limiter.
- **File Validation:**
 - Ensures only .pcap files are accepted for upload.
- **CSRF Protection:**
 - Ensures secure communication between the client and server.

OWASP Top 10:

1. **A01:2021 - Broken Access Control:** Validated file paths, restricted access to user-specific data, and ensured proper authorization checks.
2. **A03:2021 - Injection:** Sanitized file names and validated input.
3. **A04:2021 - Insecure Design:** Added rate limiting to sensitive routes.

4. **A05:2021 - Security Misconfiguration:** Ensured secure file handling and added proper error handling.
5. **A09:2021 - Security Logging and Monitoring Failures:** Added logging for critical actions.

User Authentication

The Flask-Login library is used to manage user sessions and enforce authentication. Ensures that only authenticated users can access protected routes.

```
@login_required
def home():
    return render_template("home.html", user=current_user)
```

Figure 12: User authentication validation for functions

File Upload Validation:

```
if not file or not allowed_file(file.filename):
    flash('Please include a .pcap file', category='error')
```

Figure 13: file upload type validation

Ensured that only files with the .pcap extension are accepted by checking the file extension using the allowed_file function.

Secure File Storage

```
filename = secure_filename(file.filename)
```

Figure 14: Secure file storage

Used secure_filename from werkzeug.utils to sanitize the uploaded file's name. This prevents directory traversal attacks or injection of malicious filenames.

This addresses A03:2021 - Injection.

Directory Traversal Protection

```
file_path = os.path.normpath(file_path)
real_path = os.path.realpath(file_path)

if not real_path.startswith(user_directory) or not os.path.exists(
    real_path):
    flash('Access denied: Invalid file path or file does not exist')
    return redirect(url_for('views.home'))
```

Figure 15: Directory traversal protection

Normalized and resolved the file path to its absolute path using `os.path.normpath` and `os.path.realpath`.

Checked if the resolved path starts with the user's directory and ensured the file exists.

This prevents attackers from accessing files outside the allowed directory using path traversal techniques (e.g., `../../etc/passwd`).

This addresses A01:2021 - Broken Access Control and A05:2021 - Security Misconfiguration.

Rate Limiting

```
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

limiter = Limiter(get_remote_address, app=None)

@views.route('/upload', methods=['POST'])
@login_required
@limiter.limit("10 per minute")
def upload():
```

Figure 16: Rate limiting

Added rate limiting to prevent abuse of sensitive routes like file uploads and API key updates.

This mitigates brute force attacks and denial-of-service (DoS) attacks.

This addresses A04:2021 - Insecure Design and A09:2021 - Security Logging and Monitoring Failures.

API Key Update Security

```
new_api_key = request.json.get('api_key')

try:
    user = User.query.get(current_user.id)
    if user:
        user.api_key = new_api_key
        db.session.commit()
    else:
        raise ValueError('User not found! Unknown error!')
except Exception as e:
    flash(f'{e}', category='error')
```

Figure 17: Api key update for valid users

Validated that the API key belongs to the logged-in user before updating it in the database.

Added error handling to ensure that invalid API key updates are logged and handled gracefully.

This addresses A01:2021 - Broken Access Control.

File Analysis Results

```
user = User.query.options(
    joinedload(User.pcap_loc)
    .joinedload(PcapLoc.file_analysis)
    .joinedload(FileAnalysis.file_result),
    joinedload(User.pcap_loc)
).filter_by(id=current_user.id).first()
```

Figure 18: User authentication for file analysis results

Ensured that only the logged-in user's file analysis results are fetched from the database.

This prevents unauthorized access to other users' data.

This addresses A01:2021 - Broken Access Control.

Logging and Monitoring

```
import logging

logging.basicConfig(filename='app.log', level=logging.INFO)

# Example: Log file upload success
flash(f'File uploaded successfully!', category='success')
logging.info(f"File uploaded by user {current_user.id}: {file_path}")
```

Figure 19: Logging and monitoring

Added logging for critical actions like file uploads, downloads, and API key updates.

This ensures that suspicious activities can be monitored and audited.

This addresses A09:2021 - Security Logging and Monitoring Failures.

Input Validation

```
if not file or not allowed_file(file.filename):
    flash('Please include a valid .pcap file', category='error')
    return redirect(url_for('views.home'))
```

Figure 20: Input validation

Validated user input to ensure only .pcap files are accepted.

This prevents malicious input from being processed.

This addresses A03:2021 - Injection and A05:2021 - Security Misconfiguration.

CSRF Tokens:

CSRF tokens are validated for all POST requests using `validate_csrf()`.

```
csrf_token = request.headers.get('X-CSRFToken')
validate_csrf(csrf_token)
```

Figure 21: CSRF token validation

XSS:

Input Sanitization with bleach

```
note = clean(request.form.get('note')) # Sanitize input
new_note = clean(request.json.get('note')) # Sanitize input for
```

Figure 22: XSS protection

Password Hashing

Passwords are hashed using `werkzeug.security.generate_password_hash()` with the `pbkdf2:sha256` algorithm

```
password=generate_password_hash(password1, method='pbkdf2:sha256')
```

Figure 23: Password hashing

Password Strength Enforcement

The `PasswordPolicy` from the `password_strength` library is used to enforce strong passwords.

```

policy = PasswordPolicy.from_names(
    length=8,
    uppercase=1,
    numbers=1,
    special=1,
    nonletters=1,
)
password_issues = policy.test(password1)

```

Figure 24: Password strength check

SQL Injection Prevention

SQLAlchemy ORM is used for database queries, which automatically parameterizes queries to prevent against SQL injections

```

user = User.query.filter_by(email=email).first()

```

Figure 25: Sql injection protection

Secure Execution of External Commands

The subprocess.run() function is used with capture_output=True, text=True, and check=True

This ensures that external commands are executed securely without invoking a shell, which mitigates the risk of command injection.

```

command = ['/myenv/bin/python', vxapi_path, 'scan_file', self._item_path, 'all']

# Execute the command securely
result = subprocess.run(command, capture_output=True, text=True, check=True)

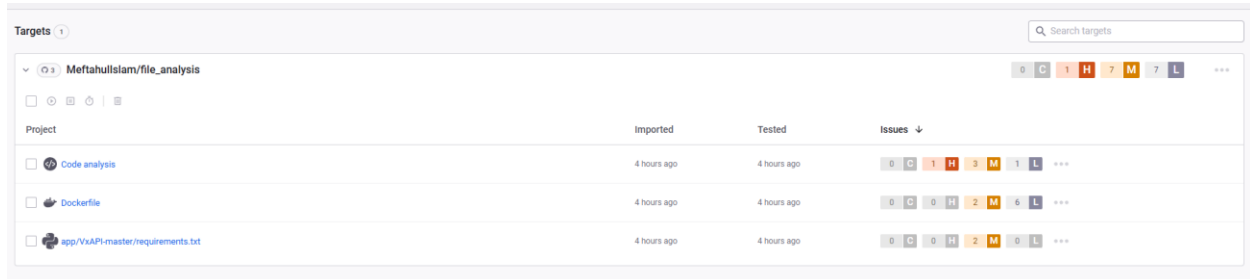
```

Figure 26: Secure execution of commands on shell

Security Testing results

Integrated the project to a CI/CD pipeline using Jenkins and used Snyk, Trivy and OWASP ZAP in pipeline for testing. The outputs are given below with the key findings.

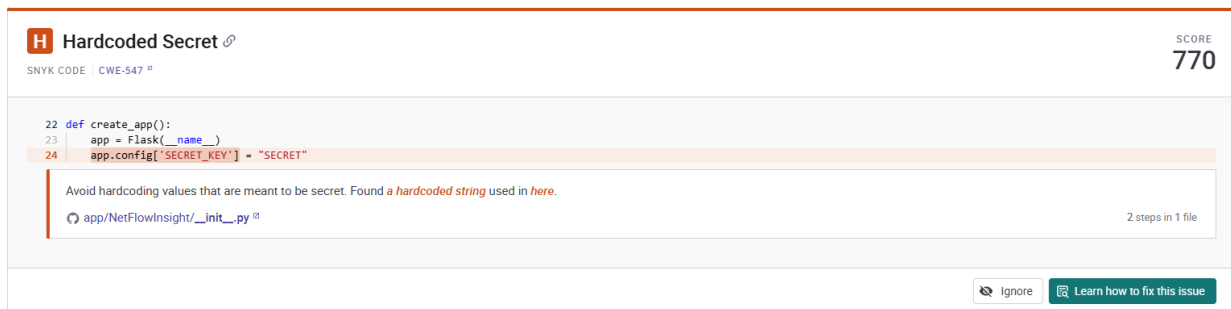
Snyk report:



Project	Imported	Tested	Issues
<input type="checkbox"/> Code analysis	4 hours ago	4 hours ago	0 C 1 H 3 M 1 L ***
<input type="checkbox"/> Dockerfile	4 hours ago	4 hours ago	0 C 0 H 2 M 6 L ***
<input type="checkbox"/> app/ViaAPI-master/requirements.txt	4 hours ago	4 hours ago	0 C 0 H 2 M 0 L ***

Figure 27: Snyk results

High score:



H Hardcoded Secret [🔗](#)
SNYK CODE | CWE-547 [🔗](#)
SCORE 770

```
22 def create_app():
23     app = Flask(__name__)
24     app.config['SECRET_KEY'] = "SECRET"
```

Avoid hardcoding values that are meant to be secret. Found a **hardcoded string** used in [here](#).
[🔗](#) app/NetFlowInsight/__init__.py [🔗](#) 2 steps in 1 file

[🗑️ Ignore](#) [🔍 Learn how to fix this issue](#)

Figure 28: Snyk results (high)

Found one High scored vulnerability. This is left because this is a development environment.

For a fix, we could pass it as a secret using an .env file in our docker run process.

Trivy report:

```
meftah0is/file_analysis:latest (ubuntu 24.04)
=====
Total: 140 (UNKNOWN: 0, LOW: 71, MEDIUM: 69, HIGH: 0, CRITICAL: 0)
```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
binutils	CVE-2017-13716	LOW	affected	2.42-4ubuntu2.5		binutils: Memory leak with the C++ symbol demangler routine in libiberty https://avd.aquasec.com/nvd/cve-2017-13716
binutils-common						
binutils-x86-64-linux-gnu						
coreutils	CVE-2016-2781			9.4-3ubuntu6		coreutils: Non-privileged session can escape to the parent session in chroot https://avd.aquasec.com/nvd/cve-2016-2781
cpio	CVE-2023-7216	MEDIUM		2.15+dfsg-1ubuntu2		CPIO: extraction allows symlinks which enables Remote Command Execution https://avd.aquasec.com/nvd/cve-2023-7216

Figure 29: Trivy report

Vulnerability Summary Report for meftah0is/file_analysis:latest (ubuntu 24.04)

Total Vulnerabilities: 690

- UNKNOWN: 0
- LOW: 71
- MEDIUM: 69
- HIGH: 0
- CRITICAL: 0

Notable Vulnerabilities:

- **binutils (CVE-2017-13716):** Memory leak in the C++ symbol demangler routine, which affects the **binutils** package (LOW severity).
- **coreutils (CVE-2016-2781):** Non-privileged session escape issue in **chroot** (LOW severity).
- **cpio (CVE-2023-7216):** Allows symlink extraction that can enable remote command execution (MEDIUM severity).
- **git (CVE-2024-52005):** An issue where the sideband payload is passed unfiltered to the terminal, which could lead to code injection or information leakage (MEDIUM severity).

- **gnupg (CVE-2022-3219)**: Denial of service via compressed packets in **gnupg** (LOW severity).
- **krb5-locales (CVE-2025-3576)**: Kerberos RC4-HMAC-MD5 checksum vulnerability enabling message spoofing via MD5 collisions (MEDIUM severity).
- **libavahi-client3 (CVE-2024-52615, CVE-2024-52616)**: Issues related to constant source ports and predictable transaction IDs in **avahi**, which may lead to network communication vulnerabilities (MEDIUM severity).

ZAP scanning report:

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	9
Missing Anti-clickjacking Header	Medium	8
Sub Resource Integrity Attribute Missing	Medium	14
Cookie without SameSite Attribute	Low	7
Insufficient Site Isolation Against Spectre Vulnerability	Low	13
Permissions Policy Header Not Set	Low	11
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	11
X-Content-Type-Options Header Missing	Low	9
Authentication Request Identified	Informational	2
Information Disclosure - Suspicious Comments	Informational	1
Modern Web Application	Informational	8
Non-Storable Content	Informational	1
Session Management Response Identified	Informational	11
Storable and Cacheable Content	Informational	9
Storable but Non-Cacheable Content	Informational	2
User Controllable HTML Element Attribute (Potential XSS)	Informational	11

Figure 30: OWASP ZAP report

Summary of Scores

- **High:** None
- **Medium:** 3 issues (CSP, Anti-clickjacking, SRI)
- **Low:** 7 issues (Cookies, Spectre, Permissions, Server info, X-Content-Type)
- Informational: 8 findings

Manual Testing

XSS:





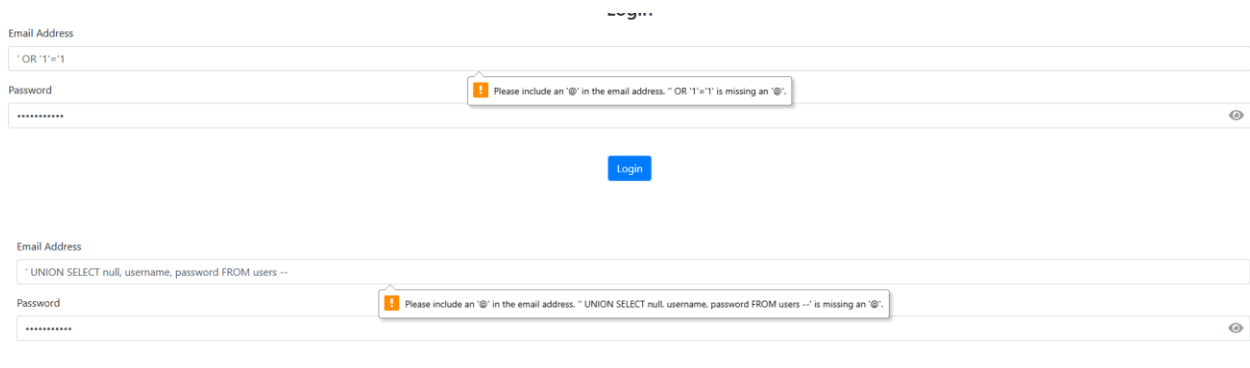
capture.pcap	dsgsadgsadg	2025-05-02 06:12		
	<script>alert(1)</script>	2025-05-09 06:18		

Figure 31: Security testing (XSS)

Performed basic reflected and stored XSS attacks using `<script>alert(1)</script>`. The sanitization mechanism prevented any xss attempt on the application.

SQL injection:



Email Address

`' OR '1'='1`

Password

Login

Email Address

`' UNION SELECT null, username, password FROM users --`

Password

Figure 32: Security Testing (sql injection)

Performed basic SQL injection attacks using `'OR '1'='1'` on input forms. The SQL injection attempts were not registered because of the security mechanisms put in place.

Suggestions for improvement

- Using HTTPS in deployment.
- Adding Role Based access as an admin panel for overall monitoring of the logs
- Adding an sftp server as a sandbox for storing malwares.

- Using PostgreSQL rather than SQLite
- Fix the container security issues according to Trivy report
- Add API Integration for Threat Intelligence.
- Optimize performance for handling .pcap files larger than 5MB, potentially through streaming parsing, multi-threading, or background processing to avoid UI and system slowdowns.

AI Usage:

ChatGPT (GPT-4-turbo) was used during the coding (mainly for the purpose of error resolution) and documentation process.