

# StuRa

StuRa Project Report  
**Abdullah Almamun**  
**Giyosiddin Abdumalikov**

June 2024

This report is submitted to the  
**Chair of Practical Computer Science/Software Systems Engineering**  
**Brandenburg University of Technology, Germany**

**Examiners:**

Prof. Dr. rer. nat. Leen Lambers  
Dr. Lucas Sakizloglou

# **Contents**

## **1. Introduction**

- 1.1. Overview of the project
- 1.2. Objectives of the security analysis
- 1.3. Scope of the analysis
- 1.4. Scope of Engagement

## **2. Methodologies**

- 2.1. Description of methodologies used
- 2.2. Tools and frameworks employed
- 2.3. Security Levels (Severity CVSS 3.1 Score)

## **3. Findings**

- 3.1 Threat modeling
- 3.2. Static Code Analysis
- 3.3. Dynamic Testing
  - 3.3.1. OWASP ZAP
  - 3.3.2 Burp Suite and Nessus
- 3.4. Penetration Testing

## **4. Conclusion**

- 4.1. Summary of overall security posture

## **5. Appendices**

- 5.1. Additional resources and references

# **1. Introduction**

## **1.1. Overview of the Project**

The security analysis presented herein pertains to StuRa a web application developed for facilitating student interactions, including semesterticket refund requests and service inquiries within the context of the StuRa of BTU Cottbus-Senftenberg. This application serves as a pivotal platform for managing sensitive student data and operational activities crucial to the administrative functions of the organization.

## **1.2. Objectives of the Security Analysis**

The primary objectives of this comprehensive security analysis are multifold. Firstly, it aims to identify potential vulnerabilities within StuRa that may compromise the confidentiality, integrity, and availability of sensitive student information and operational functionalities. Secondly, the analysis seeks to assess the current security posture of the application, utilizing a combination of static code analysis, dynamic testing methodologies, and threat modeling. Lastly, the report endeavors to provide actionable recommendations for mitigating identified vulnerabilities and strengthening the overall security framework of StuRa.

## **1.3. Scope of the Analysis**

The scope of this security analysis encompasses a thorough examination of StuRa using both static and dynamic testing approaches. Static code analysis was conducted using SonarQube to scrutinize the source code for programming errors, security vulnerabilities, and deviations from best coding practices. Dynamic testing involved the utilization of OWASP ZAP, Burp Suite, and Nessus to simulate real-world attack scenarios, identify vulnerabilities during runtime, and assess network-level security configurations. Additionally, threat modeling was employed to systematically identify potential threats, evaluate their likelihood and potential impact on the application, and recommend prioritized mitigation strategies.

## **1.4. Scope of Engagement**

The assessment and penetration testing activities were scoped to include:

- Examination of the web application accessible at <http://localhost:8000> and redirecting ports
- Evaluation of the email services hosted at <http://localhost:8025>

# **2. Methodologies**

## **2.1. Description of Methodologies Used**

The security analysis was underpinned by a structured approach involving multiple methodologies:

- **Static Code Analysis:** Leveraging Semgrep to analyze the source code statically, identifying vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), inadequate input validation, and insufficient error handling.
- **Dynamic Testing:** Utilizing OWASP ZAP, Burp Suite, and Nessus to perform dynamic testing and vulnerability scanning, detecting vulnerabilities in real-time through simulated attacks and comprehensive network-level assessments.

- Threat Modeling: Employing a systematic approach to identify and prioritize potential threats to StuRa, evaluating their impact on the application's security posture, and proposing targeted mitigation strategies.

## **2.2. Tools and Frameworks Employed**

- The analysis utilized industry-standard tools and frameworks tailored to each phase of the assessment:
- OWASP ZAP: Employed for web application security scanning and detection of vulnerabilities such as .htaccess information leaks, missing Content Security Policy (CSP) headers, and potential sensitive information leaks via redirects.
- Burp Suite: Utilized for intercepting and analyzing HTTP traffic, identifying vulnerabilities including missing HttpOnly flags on cookies and server information leakage via "X-Powered-By" headers.
- Nessus: Applied for network-based vulnerability scanning, complementing the dynamic testing efforts by identifying vulnerabilities across the network infrastructure.
- Snyk: Used for static code analysis to detect security vulnerabilities, programming errors, and license issues, ensuring adherence to secure coding practices.

## **2.3. Security Levels (Severity CVSS 3.1 Score)**

The severity ratings based on the Common Vulnerability Scoring System (CVSS) 3.1 are summarized as follows:

- Critical (9-10): No vulnerabilities of critical severity were identified that could potentially grant unauthorized administrative-level access or compromise high-value organizational data.
- High (7-8.9): No vulnerabilities categorized as high severity were uncovered, which could otherwise allow access to critical data with specific prerequisites.
- Medium (4-6.9): No vulnerabilities of medium severity were found that might require complex conditions or user privileges for exploitation.
- Low (0.1-3.9): No vulnerabilities of low severity were detected, indicating minimal impact on organizational operations.
- Info: No informational vulnerabilities, which could improve overall security posture, were identified.

## 3. Findings

### 3.1 Threat modeling

Threat modeling is a critical technique in which we detect and analyze potential security threats and vulnerabilities to our online application. Using frameworks such as STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), we may actively examine vulnerabilities related to our application for the BTU Cottbus-Senftenberg StuRa. This technique enables us to apply relevant security controls and protections early in the development process. For example, we implement strong authentication procedures to avoid identity spoofing and encrypt important data to reduce information leak risks. By including threat modeling into our development procedures, we improve our application's overall resistance to a variety of potential threats, protecting student data and keeping user trust. Our threat modeling in STRIDE will be described below.

#### STRIDE Threat Modeling Analysis

##### 1. Spoofing Identity (S)

Threat: An attacker could impersonate a legitimate student user to gain unauthorized access to the application.

Mitigation: Implement strong authentication mechanisms such as multi-factor authentication (MFA) for user logins. Ensure proper session management to prevent session hijacking.

##### 2. Tampering with Data (T)

Threat: Attackers might modify data in transit or in the database, such as altering refund requests or other service requests.

Mitigation: Use HTTPS to encrypt data in transit. Implement input validation and output encoding to prevent injection attacks (SQL injection, XSS). Apply integrity checks to data where appropriate.

##### 3. Repudiation (R)

Threat: A user (or attacker) could deny performing actions within the application, such as denying they requested a refund or submitted a service request.

Mitigation: Implement logging and auditing mechanisms to track user activities and actions taken within the application. Ensure logs are tamper-proof and easily accessible for review.

##### 4. Information Disclosure (I)

Threat: Sensitive information (like personal details, financial information related to refunds) could be exposed to unauthorized parties.

Mitigation: Apply principle of least privilege, ensuring users can only access information necessary for their roles. Encrypt sensitive data at rest and in transit. Conduct regular security assessments and vulnerability scans.

##### 5. Denial of Service (D)

Threat: An attacker could attempt to overload the application or its components, rendering it unavailable to legitimate users.

Mitigation: Implement rate limiting, CAPTCHA, and other anti-DoS mechanisms. Use scalable infrastructure to handle spikes in traffic. Monitor performance metrics and set up alerts for abnormal activity.

##### 6. Elevation of Privilege (E)

Threat: Attackers might exploit vulnerabilities to gain higher privileges than intended, allowing unauthorized access to administrative functions or sensitive data.

Mitigation: Implement strict access controls and segregation of duties. Use the principle of least privilege for user accounts and roles. Regularly review and update access permissions.

## 3.2 Static Code Analysis

In the assessment of the StuRa web application, static code analysis was performed to detect security vulnerabilities and ensure code quality. The tool employed for this analysis was Snyk, a robust platform known for its comprehensive security testing capabilities.

### Tools and Methods:

#### Snyk

The **snyk test** command was used to scan the project's dependencies for open-source vulnerabilities and license issues. This tool identifies potential risks within third-party libraries and frameworks, ensuring that all components comply with security standards and licensing policies.

```
(kali@kali)-[~/tmp/stura-forms]
$ ./snyk-linux test --all-projects

Testing /home/kali/tmp/stura-forms ...

Organization: simple402
Package manager: composer
Target file: composer.lock
Project name: laravel/laravel
Open source: no
Project path: /home/kali/tmp/stura-forms
Licenses: enabled

✓ Tested 165 dependencies for known issues, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.

Testing /home/kali/tmp/stura-forms ...

Organization: simple402
Package manager: npm
Target file: package-lock.json
Project name: package.json
Open source: no
Project path: /home/kali/tmp/stura-forms
Licenses: enabled

✓ Tested /home/kali/tmp/stura-forms for known issues, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.

Testing /home/kali/tmp/stura-forms ...

Organization: simple402
Package manager: composer
Target file: test-idp/composer.lock
Project name: laravel/laravel
Open source: no
Project path: /home/kali/tmp/stura-forms
Licenses: enabled

✓ Tested 98 dependencies for known issues, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.

Testing /home/kali/tmp/stura-forms ...

Organization: simple402
Package manager: npm
Target file: test-idp/package-lock.json
Project name: package.json
Open source: no
Project path: /home/kali/tmp/stura-forms
Licenses: enabled

✓ Tested /home/kali/tmp/stura-forms for known issues, no vulnerable paths found.

Next steps:
- Run `snyk monitor` to be notified about new related vulnerabilities.
- Run `snyk test` as part of your CI/test.

Tested 4 projects, no vulnerable paths were found.
```

## Snyk Code Test

The **snyk code test** command was utilized to perform static code analysis, which examines the source code for security vulnerabilities and coding flaws. This command leverages Snyk's extensive vulnerability database and advanced detection algorithms to uncover potential threats early in the development lifecycle.

```
(kali㉿kali)-[~/tmp/stura-forms]
$ ./snyk-linux code test

Testing /home/kali/tmp/stura-forms ...

x [Low] Use of Password Hash With Insufficient Computational Effort
Path: database/factories/Dticket/DticketExcludeFactory.php, line 20
Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

x [Low] Use of Password Hash With Insufficient Computational Effort
Path: database/factories/UserFactory.php, line 24
Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

x [Medium] Use of Hardcoded Credentials
Path: lang/en/validation.php, line 27
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/en/validation.php, line 166
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/en/validation.php, line 198
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/en/auth.php, line 7
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/de/auth.php, line 7
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/de/validation.php, line 27
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/de/validation.php, line 166
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/de/validation.php, line 197
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

x [Medium] Use of Hardcoded Credentials
Path: lang/de/validation.php, line 198
Info: Do not hardcode passwords in code. Found a hardcoded password used in variable.

✓ Test completed

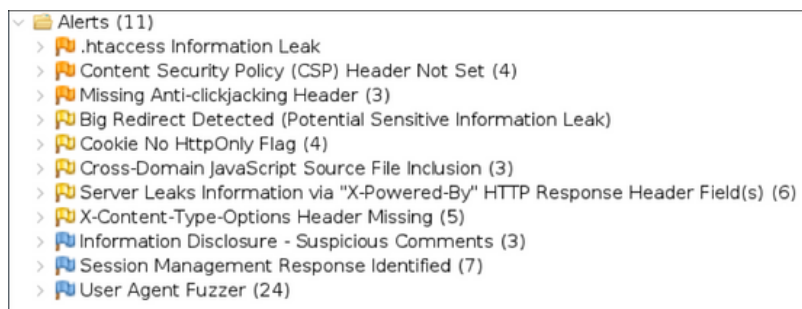
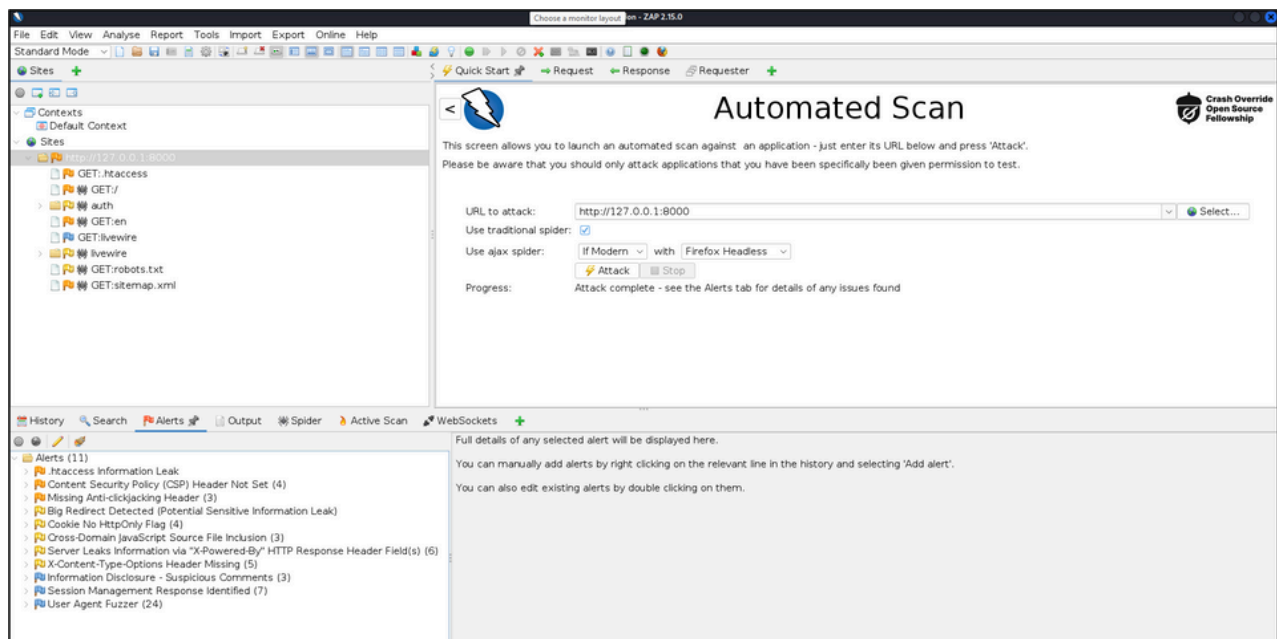
Organization:    simple402
Test type:       Static code analysis
Project path:    /home/kali/tmp/stura-forms

Summary:

11 Code issues found
9 [Medium] 2 [Low]
```

## 3.3. Dynamic Testing Findings

### 3.3.1. OWASP ZAP



#### .htaccess Information Leak

- URL: `http://127.0.0.1:8000/.htaccess`
- Description: OWASP ZAP flagged the presence of .htaccess files accessible via HTTP, which are typically used in Apache servers to override global server configuration settings on a per-directory basis. Exposure of these files can divulge critical server configurations, including URL rewrite rules, potentially aiding attackers in crafting more targeted exploits.
- Security Level: Medium
- Remediation: To mitigate this vulnerability, it is recommended to restrict access to .htaccess files either by placing them in a directory not served by the web server or by configuring strict file permissions. Regular auditing and sanitization of .htaccess content are crucial to prevent inadvertent exposure of sensitive information.

```
<IfModule mod_rewrite.c>
<IfModule mod_negotiation.c>
  Options -MultiViews -Indexes
</IfModule>

RewriteEngine On

# Handle Authorization Header
RewriteCond %{HTTP:Authorization} .
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

# Redirect Trailing Slashes If Not A Folder...
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_URI} (.+)/$
RewriteRule ^ %1 [L,R=301]

# Send Requests To Front Controller...
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
</IfModule>
```



## Content Security Policy (CSP) Header Not Set

- URL: <http://127.0.0.1:8000>
- Description: The absence of a Content Security Policy (CSP) header on HTTP responses leaves the StuRa web application vulnerable to various types of attacks, notably Cross-Site Scripting (XSS) and data injection exploits. CSP is a security feature that helps mitigate these risks by specifying trusted sources from which browsers can load content, thereby preventing malicious script execution and unauthorized data manipulation.
- Security Level: Medium
- Impact: The lack of CSP increases the likelihood of successful XSS attacks, where malicious scripts can be injected into web pages, potentially compromising user data and application integrity.
- Remediation: Implementing a robust CSP configuration involves defining strict policies tailored to the application's needs. Common practices include specifying allowed sources for scripts, styles, and other resources via HTTP headers or meta tags. Regular testing and adjustment of CSP settings are essential to ensure effective protection against evolving threats.

## Missing Anti-clickjacking Header

- URL: <http://127.0.0.1:8000>
- Description: OWASP ZAP identified the absence of the X-Frame-Options or Content-Security-Policy frame-ancestors directive in HTTP headers, exposing the application to clickjacking attacks. Clickjacking, or UI redress attacks, occur when malicious actors trick users into interacting with hidden elements overlaid on legitimate web content, potentially leading to unintended actions or data exposure.
- Security Level: Medium
- Impact: Without proper framing protection, attackers can exploit user interactions to perform actions on behalf of the victim, bypassing authentication mechanisms or inducing inadvertent actions.
- Remediation: To mitigate clickjacking risks, it is advised to set the X-Frame-Options header to 'DENY' or 'SAMEORIGIN', instructing browsers to prevent rendering the page within a frame unless it originates from the same site. Alternatively, leveraging Content Security Policy's frame-ancestors directive allows fine-grained control over framing permissions, ensuring content is only embedded in trusted contexts.

```
HTTP/1.1 200 OK
Host: 127.0.0.1:8000
Connection: close
X-Powered-By: PHP/8.3.8
Content-Type: text/html; charset=UTF-8
Cache-Control: no-cache, private
Date: Mon, 24 Jun 2024 21:47:30 GMT
Set-Cookie: XSRF-TOKEN=eyJpdiI6IjFVa0Y5Q2I1S2RzRnRnpEenh1NFRkaEE9PSIsInZhbHVlIjoieZk52TWp4N3A5Z1ZEVDhRQzBuSnVpMGVZTHVUYWVWdnNTTUE4SG1SWhZ3Z01KVVlmeTBhdG00SEN4Tl1Ga0JBcEgvdnFQM2RaYnJwMVhkZ215c0JkdnZtS0lMVMhZeEEzM3YyLzF1VHphbVUzK1Yzd0J4OUd3Z2sxYVJ5U1BT0YiLCJtYmMiOiJmMmM1M2E1YmI0ZGUzNjIyN2RlNWE2ZmQ2YjF1NTdhOTBhNWE3NmIxMDc5N2U5YjE0YWEzZmQwODQ4MjA1Y2M4IiwidGFuIjoieIn0%3D; expires=Mon, 24 Jun 2024 23:47:30 GMT; Max-Age=7200; path=/; samesite=lax
Set-Cookie: laravel_session=
```

## Big Redirect Detected (Potential Sensitive Information Leak)

- URL: <http://127.0.0.1:8000/auth/saml2/redirect>
- Impact: Without proper framing protection, attackers can exploit user interactions to perform actions on behalf of the victim, bypassing authentication mechanisms or inducing inadvertent actions.
- Remediation: To mitigate clickjacking risks, it is advised to set the X-Frame-Options header to 'DENY' or 'SAMEORIGIN', instructing browsers to prevent rendering the page within a frame unless it originates from the same site. Alternatively, leveraging Content Security Policy's frame-ancestors directive allows fine-grained control over framing permissions, ensuring content is only embedded in trusted contexts.
- Description: The detection of a large response body in a redirect scenario indicates a potential information disclosure vulnerability. While redirect responses typically should not include body content, the presence of substantial data suggests inadvertent exposure of sensitive information such as session tokens or user credentials.
- Security Level: Low
- Impact: Information leakage through redirects can lead to privacy violations, exposing sensitive data to unauthorized parties and undermining user trust.
- Remediation: To address this vulnerability, server configurations should be reviewed and adjusted to ensure proper handling of redirect responses. Implementing secure redirect practices involves verifying that only necessary headers are included in redirections and that no sensitive data is inadvertently exposed.

```
HTTP/1.1 302 Found
Host: 127.0.0.1:8000
Connection: close
X-Powered-By: PHP/8.3.8
Cache-Control: no-cache, private
Date: Mon, 24 Jun 2024 21:47:30 GMT
Location:
http://localhost:8001?SAMLRequest=hZLNbsIwEITvPIXle7Bj8gNWCKJFVZfG0HaQy%2BVE5xiNbFp1kHt29eBInGo6MGX0ex%2Bs7t0ZL9NjQ6yBWx0FPtDImfPIJL3dqfX
8r0TYJFzaJjirtXcCFDAAtWgkcFvyzfxhxdmQ8n1rrcLNjdFyMcVvVRhJthXjqgxHk1EUUPcEzaGoJj6L4pgWhQj92A8qjF70bNfHLQN0cqnBcm2dRFng0chjQc58HsR8RF8xWrhQsg
t7rNpZu%2BeE1KYU9c6A5WNKfyy30A3Sm%2Bvfr%2BvjiZgN%2FneeZLT5scozmAbHvErdHQNbLdyPagSvm8Xv0NpUS4pREQtC2I0%2BtC1B84Txq8HdqT5vkvXI9kDjDcfofKI
ExgDR5dL2Wi8zUgvxGd6ZthL206hw19aqjle%2F7Y4CV2mKSJTuYg6SDHw%3D%3D6SigAlg=http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmldsig-more%23rsa-sha256Si
gnature=S1eSkeQILP%2BBSxEH0IFv%2FqzRSEKkKpD%2BpZ71mguUhc0r1DcCo4cPey4TwlmYp6p0f06EM2LrWmsGW03c3r%2Fxp66j3HYCys%2B5emLqx0ZHUv0XKny6orBYITS
x7cKTnDMR3MKYsW8mSAasKLXUe5Ga68%2FCp19sxcNG1VjueP707oPvyHZYccj2A80bbHpVYvYhov%2F0NII0306Q%2FTcL6KpArpxgYL4h%2FLdt dwmNIHmwU%2BVy9E1ZUwNN9FJ
g%2FFz6oRf12LR%2BBSQdUwKcuAQfy7SGPQ3G52Fm47RknWoA1g92HDjg0AaNUX0s7R36z%2BGZBFcCR6zf0dE2hrmfKcHuLgJPQVQUi1rTWS5JvL92LH5c7M15u0cB6GxyHFa%2F
fkU%2FNhrF4tHqccT7Y%2FH0N53Ltw%2BryVVLcuJ2thD%2Fwmm9dhw6a%2F8DmpGSHKRWnEL%2FhcFvKXowVtEfHqJKGzqE%2B12Bk7ckK92JgdaE06GCSqLYyBR01B30M7LF1Y
jE0oGngYBby%2BvEVLmnbssmluI5dN9FrC9soAk7ih37DKaFe3YtqXYS3hebyRDA0EMB242F5dx9Dhys2DsItHf0A1kk%2BxXB9FBU0nH7fXoGxVymHLtXW4Z6%2B8cxU6BmLmMt
aWxm3Qct%2B14IoxzUwuYHrix8aoB0CW0cqqafLJrz9aM3GnCns%3D
Content-Type: text/html; charset=UTF-8
Set-Cookie: XSRF-TOKEN=
eyJpdiI6InEzNEhVbGg5bWdldjB3ZkVmd3h0tE9PSiInZhbHVlIjoiaWJkZlJmVnN0p1Wkpkb1ArVDk0ZDBpNGoxeVZwaE12dVNIsmUraz1EwUhydDAvampCZkVX0ExrRTBIMmRLQV
hWVW90REhM2Sm10REh0FvXvcw0HE3YU0yZHVfO0Gx0B3RkZFBld2dqS2eN5QkVaSEfYmNhSemNjD320aEtGMHY5UTM1LCJtYWML01IzNDczZjNjNmV1ZWZmZTM1YWV1YTQ2ZWRLMDgw
NmM2NDZAZWJyZjI0MjNlNjU4N2E2ZmZlZTBhZnZlNDYwZjRlIiwidGFiIjoiaW9n%3D"; expires=Mon, 24 Jun 2024 23:47:30 GMT; Max-Age=7200; path=/; samesite=
lax
Set-Cookie: laravel_session=
```

## Cookie No HttpOnly Flag

- URL: <http://127.0.0.1:8000>
- Description: OWASP ZAP identified cookies without the HttpOnly flag set, allowing client-side scripts to access them. This oversight increases the vulnerability of session cookies to Cross-Site Scripting (XSS) attacks, where malicious scripts can steal cookie data and impersonate authenticated users.
- Security Level: Low
- Impact: The absence of HttpOnly flag on cookies facilitates session hijacking and other client-side attacks, compromising user privacy and application security.
- Remediation: It is recommended to enforce the HttpOnly flag on all cookies unless there is a specific requirement for client-side access. By restricting cookie access to server-side operations only, potential XSS vulnerabilities can be mitigated effectively. Regular security assessments should validate the correct implementation of HttpOnly across all application cookies.

```
HTTP/1.1 200 OK
Host: 127.0.0.1:8000
Connection: close
X-Powered-By: PHP/8.3.8
Content-Type: text/html; charset=UTF-8
Cache-Control: no-cache, private
Date: Mon, 24 Jun 2024 21:47:30 GMT
Set-Cookie: XSRF-TOKEN=eyJ3dDI6IjEwYV9yS2Q1BzZRpSnPEnhInFRkaEE9PSIsInZhbnVlIjoiaWZk52Twp4N3A5Z1ZEVDRhQzBzSnVpMGVZTHVUYVVvdnNTTUE4SGlSwNz3Z0lKVVlmeTBhdG00SEN4TllGa0JBCgEvZnFMQ2RaYnJwMvhKZ21ScDkdnZTS0lMvMhZeEEZm3YylzFiVhPhBVubUzKlyZdlJ4OUd3Z2xxyJV5U1BT50YlCJtyWMi0iJmMmmIMZEIYmIOZGUzNjIyNzRlNWEEZmQ2ZmYfjNTdh0TBHhWE3NmIxMc5N2U5YjE0YwEzZmQwDDQ4MjAlY2M4IiwidGFniIjoiaWln0%3D; expires=Mon, 24 Jun 2024 23:47:30 GMT; Max-Age=7200; path=/; samesite=Lax
Set-Cookie: laravel_session=eyJ3dDI6IjEwYV9yS2Q1BzZRpSnPEnhInFRkaEE9PSIsInZhbnVlIjoiaWZk52Twp4N3A5Z1ZEVDRhQzBzSnVpMGVZTHVUYVVvdnNTTUE4SGlSwNz3Z0lKVVlmeTBhdG00SEN4TllGa0JBCgEvZnFMQ2RaYnJwMvhKZ21ScDkdnZTS0lMvMhZeEEZm3YylzFiVhPhBVubUzKlyZdlJ4OUd3Z2xxyJV5U1BT50YlCJtyWMi0iJmMmmIMZEIYmIOZGUzNjIyNzRlNWEEZmQ2ZmYfjNTdh0TBHhWE3NmIxMc5N2U5YjE0YwEzZmQwDDQ4MjAlY2M4IiwidGFniIjoiaWln0%3D; expires=Mon, 24 Jun 2024 23:47:30 GMT; Max-Age=7200; path=/; httponly; samesite=Lax
```

## Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

- URL: <http://127.0.0.1:8000>
- Description: OWASP ZAP identified cookies without the HttpOnly flag set, allowing client-side scripts to access them. This oversight increases the vulnerability of session cookies to Cross-Site Scripting (XSS) attacks, where malicious scripts can steal cookie data and impersonate authenticated users.
- Security Level: Low
- Impact: The absence of HttpOnly flag on cookies facilitates session hijacking and other client-side attacks, compromising user privacy and application security.
- Remediation: It is recommended to enforce the HttpOnly flag on all cookies unless there is a specific requirement for client-side access. By restricting cookie access to server-side operations only, potential XSS vulnerabilities can be mitigated effectively. Regular security assessments should validate the correct implementation of HttpOnly across all application cookies.
- Description: Presence of the "X-Powered-By" HTTP header in server responses discloses information about the underlying technology stack, potentially aiding attackers in identifying and exploiting known vulnerabilities associated with specific server software or frameworks.
- Security Level: Low
- Impact: Disclosure of server details through "X-Powered-By" headers can facilitate targeted attacks, where adversaries leverage knowledge of server components to craft tailored exploits.
- Remediation: To mitigate this risk, organizations should consider disabling or removing "X-Powered-By" headers from HTTP responses. By minimizing exposure of server stack details, the attack surface available to potential adversaries is reduced, enhancing overall application security.

```
HTTP/1.1 200 OK
Host: 127.0.0.1:8000
Connection: close
X-Powered-By: PHP/8.3.8
Content-Type: text/html; charset=UTF-8
Cache-Control: no-cache, private
Date: Mon, 24 Jun 2024 21:47:30 GMT
Set-Cookie: XSRF-TOKEN=
eyJ3d16iFVa9y5Q2lBSzRpSnPEenh1NFRkaEE9PSIsInZhbnVlIjoizk52TWhp43A5ZlZEVDRhQzBzSnVpMGVZTHVUYVWVndNTTUE4SGlSwNz30ZlKVVlmeTBhdG00SEN4Tl1Ga0
bcEgVznrQM2RaYnJwMVNhkZ215cDJKdnZtS6LMVmhZeEzM3YyLzF1VhpHbVUzK1YzdJ40Ud3Z2sxYVJ5U1BTs0YlCJtyYm0iJmMmM1M2E1YmI0ZGuzNjIyN2RLNWE2ZmQ2YjF1
NTdhOTBhNWEN3NmIxMdc5N2U5YjE9YWEZmQwODQ4MjAlY2M4IiwidGFmIjoiaW03ZD; expires=Mon, 24 Jun 2024 23:47:30 GMT; Max-Age=7200; path=/; samesite=
lax
```

## X-Content-Type-Options Header Missing

- URL: http://127.0.0.1:8000
- Description: OWASP ZAP identified the absence of the X-Content-Type-Options header, which prevents browsers from performing MIME-sniffing. This security control is essential in mitigating risks associated with content-type confusion attacks, where browsers may incorrectly interpret content types and execute malicious scripts embedded in disguised files.
- Security Level: Low
- Impact: Without the X-Content-Type-Options header set to 'nosniff', browsers are vulnerable to MIME-sniffing attacks, potentially leading to XSS and drive-by download exploits.
- Remediation: Secure the application against MIME-sniffing vulnerabilities by configuring the X-Content-Type-Options header to 'nosniff' in all HTTP responses. This precaution ensures browsers strictly adhere to declared content types, mitigating the risk of malicious content execution.

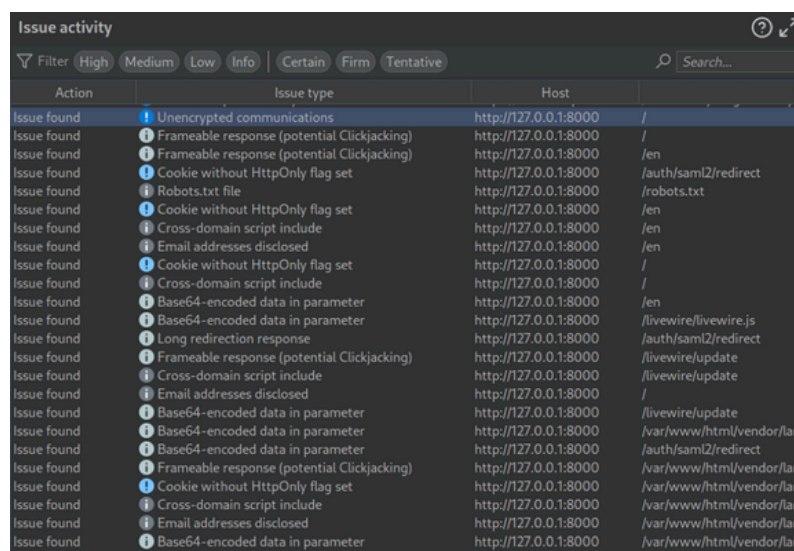
## 3.3.2 Burp Suite

Burp Suite is a comprehensive web application security testing tool that intercepts and modifies HTTP/S traffic, automates vulnerability scanning, performs attacks like SQL injection, cross-site scripting (XSS), and brute force, and allows detailed manual request analysis. It assesses session token security, handles various data encoding/decoding formats, compares data for discrepancies, and supports custom extensions. Additionally, it detects out-of-band vulnerabilities, discovers hidden content, manages session states, logs interactions, generates detailed reports, defines testing scope, visualizes web applications, and controls scan speed and request concurrency for optimal performance. These features make Burp Suite essential for both automated and manual web application security testing.

### Burp Suite

We chose Burp Suite and OWASP ZAP because together they offer a comprehensive suite of tools for testing web applications. Burp Suite excels in manual testing with its detailed interception and modification capabilities, while OWASP ZAP provides robust automated scanning features.

- Findings: Burp Suite yielded results consistent with OWASP ZAP's findings, Unencrypted communications, Cookie without HttpOnly flag set, Email addresses disclosed, and other security misconfigurations.
- Evidence: Refer to the attached screenshot from Burp Suite for detailed vulnerability instances detected during the assessment.



Action	Issue type	Host
Issue found	Unencrypted communications	http://127.0.0.1:8000
Issue found	Frameable response (potential Clickjacking)	http://127.0.0.1:8000
Issue found	Frameable response (potential Clickjacking)	http://127.0.0.1:8000
Issue found	Cookie without HttpOnly flag set	http://127.0.0.1:8000
Issue found	Robots.txt file	http://127.0.0.1:8000
Issue found	Cookie without HttpOnly flag set	http://127.0.0.1:8000
Issue found	Cross-domain script include	http://127.0.0.1:8000
Issue found	Email addresses disclosed	http://127.0.0.1:8000
Issue found	Cookie without HttpOnly flag set	http://127.0.0.1:8000
Issue found	Cross-domain script include	http://127.0.0.1:8000
Issue found	Base64-encoded data in parameter	http://127.0.0.1:8000
Issue found	Base64-encoded data in parameter	http://127.0.0.1:8000
Issue found	Long redirection response	http://127.0.0.1:8000
Issue found	Frameable response (potential Clickjacking)	http://127.0.0.1:8000
Issue found	Cross-domain script include	http://127.0.0.1:8000
Issue found	Email addresses disclosed	http://127.0.0.1:8000
Issue found	Base64-encoded data in parameter	http://127.0.0.1:8000
Issue found	Base64-encoded data in parameter	http://127.0.0.1:8000
Issue found	Frameable response (potential Clickjacking)	http://127.0.0.1:8000
Issue found	Cookie without HttpOnly flag set	http://127.0.0.1:8000
Issue found	Cross-domain script include	http://127.0.0.1:8000
Issue found	Email addresses disclosed	http://127.0.0.1:8000
Issue found	Base64-encoded data in parameter	http://127.0.0.1:8000

## Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			Total
		Certain	Firm	Tentative	
Severity	High	0	0	0	0
	Medium	0	0	0	0
	Low	1	4	0	5
	Information	16	12	0	28
	False Positive	0	0	0	0

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.



**Summary:** During our vulnerability assessment using OWASP ZAP, we identified a total of 11 alerts, categorized into 3 medium severity, 5 low severity, and 3 informative alerts. However, upon conducting a detailed assessment with Burp Suite, we discovered a total of 17 alerts, with nearly all matching those identified by OWASP ZAP. Specifically, Burp Suite categorized 16 alerts as informational and 1 as low severity.

## Nessus

Nessus for network vulnerability scanning and extensive security assessments.

### Findings

When we scanned the system with Nessus, it showed 16 alerts as Information on localhost on port 8000. Summary of the scanned host in the screenshot can be seen below.

Vulnerability assessment on <http://127.0.0.1:8000> by Nessus

Vulnerability assessment on http://127.0.0.1:8000 by Nessus

## 127.0.0.1



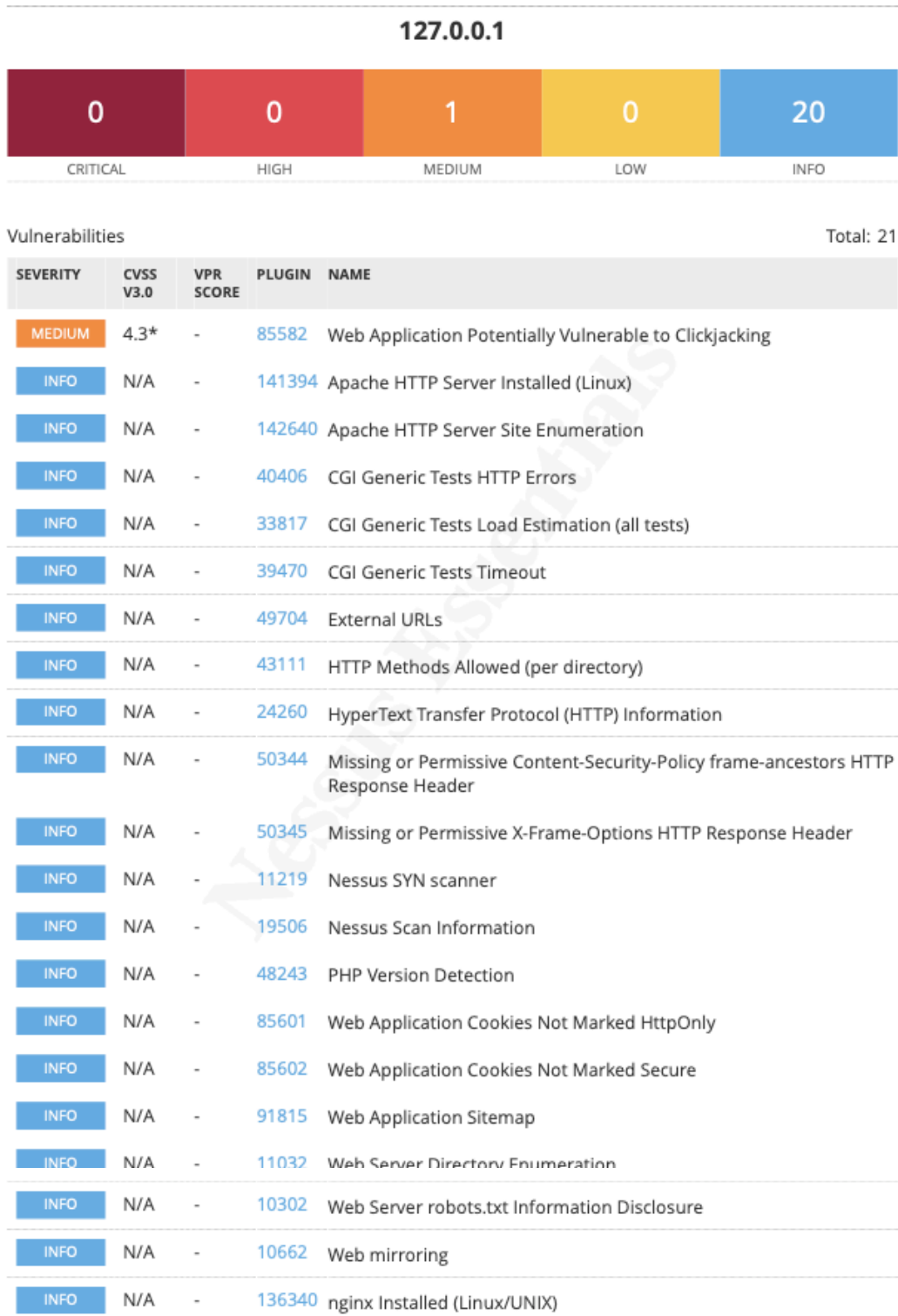
### Vulnerabilities

Total: 16

SEVERITY	CVSS V3.0	VPR SCORE	PLUGIN	NAME
INFO	N/A	-	141394	Apache HTTP Server Installed (Linux)
INFO	N/A	-	142640	Apache HTTP Server Site Enumeration
INFO	N/A	-	49704	External URLs
INFO	N/A	-	43111	HTTP Methods Allowed (per directory)
INFO	N/A	-	24260	HyperText Transfer Protocol (HTTP) Information
INFO	N/A	-	50344	Missing or Permissive Content-Security-Policy frame-ancestors HTTP Response Header
INFO	N/A	-	50345	Missing or Permissive X-Frame-Options HTTP Response Header
INFO	N/A	-	11219	Nessus SYN scanner
INFO	N/A	-	19506	Nessus Scan Information
INFO	N/A	-	48243	PHP Version Detection
INFO	N/A	-	85601	Web Application Cookies Not Marked HttpOnly
INFO	N/A	-	85602	Web Application Cookies Not Marked Secure
INFO	N/A	-	91815	Web Application Sitemap
INFO	N/A	-	11032	Web Server Directory Enumeration
INFO	N/A	-	10302	Web Server robots.txt Information Disclosure
INFO	N/A	-	136340	nginx Installed (Linux/UNIX)

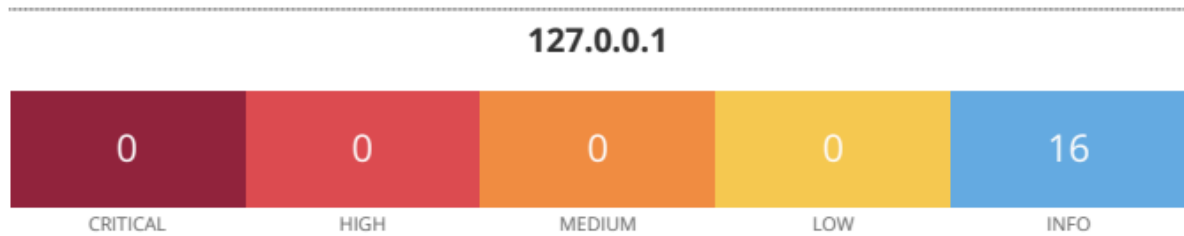
However, when port 8001 was scanned, Nessus identified more issues which the majority were Information as shown below. Most importantly, one alert categorized as Medium with CVSS score 4.3 was found out that refers to Clickjacking vulnerability. Clickjacking, also known as UI redress attack or user interface (UI) deception attack, is a technique used by malicious actors to trick users into clicking on something different from what they perceive they are clicking on.

#### Vulnerability assessment on http://127.0.0.1:8001 by Nessus



Moreover, when the email host on port 8025 was scanned, Nessus produced the same result as on port 8000. Namely, all the issues were Information.

Vulnerability assessment on http://127.0.0.1:8025 by Nessus



#### Vulnerabilities

Total: 16

SEVERITY	CVSS V3.0	VPR SCORE	PLUGIN	NAME
INFO	N/A	-	141394	Apache HTTP Server Installed (Linux)
INFO	N/A	-	142640	Apache HTTP Server Site Enumeration
INFO	N/A	-	49704	External URLs
INFO	N/A	-	43111	HTTP Methods Allowed (per directory)
INFO	N/A	-	24260	HyperText Transfer Protocol (HTTP) Information
INFO	N/A	-	50344	Missing or Permissive Content-Security-Policy frame-ancestors HTTP Response Header
INFO	N/A	-	50345	Missing or Permissive X-Frame-Options HTTP Response Header
INFO	N/A	-	11219	Nessus SYN scanner
INFO	N/A	-	19506	Nessus Scan Information
INFO	N/A	-	48243	PHP Version Detection
INFO	N/A	-	85601	Web Application Cookies Not Marked HttpOnly
INFO	N/A	-	85602	Web Application Cookies Not Marked Secure
INFO	N/A	-	91815	Web Application Sitemap
INFO	N/A	-	11032	Web Server Directory Enumeration
INFO	N/A	-	10302	Web Server robots.txt Information Disclosure
INFO	N/A	-	136340	nginx Installed (Linux/UNIX)



Summary: Similar to Burp Suite and OWASP ZAP, Nessus also identified common vulnerabilities within the StuRa web application, including PHP version detection, missing security headers like CSP and X-Frame-Options, external URLs, robots.txt and among other issues as shown in the screenshots.

In a nutshell, we utilized Nessus in order to obtain better results compared to OWASP ZAP and Burpsuite, nevertheless, all the vulnerability assessment tools gave almost the same results, meaning that, the majority of the issues were Information which has very little or no impact, ensuring the application secure. No critical and high level severity weaknesses were identified, however, only one issue on port 8001 showed that it is vulnerable to Clickjacking.

### 3.4 Penetration Testing Findings

The primary objective of the penetration testing conducted on the StuRa web application was to meticulously identify and assess potential security vulnerabilities. The aim was to ensure the application's resilience and robust functionality in the face of adversarial attempts. Following an exhaustive assessment encompassing both automated and manual testing methodologies, it was determined that no vulnerabilities were detected during the penetration testing phase. This outcome underscores the current strong security posture of the StuRa application.

#### Tools Used

A suite of specialized tools was employed during the penetration testing process, including:

- Kali Linux Tools for web application testing.
- Dirb for web content discovery and directory enumeration.
- Nikto for scanning web servers to uncover common security vulnerabilities.
- Wapiti for scanning web applications to identify potential security weaknesses.

#### reconnaissance

##### DIRB

We used DirB to brute-force discover hidden directories and files on the application, aiming to uncover interesting information. Below are the results from the tool.

```
(kali㉿kali)-[~]
$ dirb http://localhost:8000

DIRB v2.22
By The Dark Raver

START_TIME: Tue Jun 25 12:54:20 2024
URL_BASE: http://localhost:8000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

— Scanning URL: http://localhost:8000/ —
+ http://localhost:8000/.htaccess (CODE:200|SIZE:603)
+ http://localhost:8000/admin (CODE:302|SIZE:330)
+ http://localhost:8000/dashboard (CODE:302|SIZE:330)
+ http://localhost:8000/en (CODE:200|SIZE:4369)
+ http://localhost:8000/favicon.ico (CODE:200|SIZE:0)
+ http://localhost:8000/index.php (CODE:200|SIZE:4684)
+ http://localhost:8000/locale (CODE:405|SIZE:1033544)
+ http://localhost:8000/robots.txt (CODE:200|SIZE:24)
+ http://localhost:8000/up (CODE:200|SIZE:3516)

END_TIME: Tue Jun 25 13:00:22 2024
DOWNLOADED: 4612 - FOUND: 9
```

As shown, there are some interesting paths which are `/.htaccess`, `/locale`, `/robots.txt`

```
(kali㉿kali)-[~/btu_project/stura-forms/public]
$ dirb http://localhost:8001

DIRB v2.22
By The Dark Raver

START_TIME: Tue Jun 25 12:43:33 2024
URL_BASE: http://localhost:8001/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://localhost:8001/ ---
+ http://localhost:8001/.htaccess (CODE:200|SIZE:603)
+ http://localhost:8001/favicon.ico (CODE:200|SIZE:0)
+ http://localhost:8001/index.php (CODE:200|SIZE:4437)
+ http://localhost:8001/robots.txt (CODE:200|SIZE:24)
+ http://localhost:8001/up (CODE:200|SIZE:2126)

END_TIME: Tue Jun 25 12:44:33 2024
DOWNLOADED: 4612 - FOUND: 5
```

```
(kali㉿kali)-[~]
$ dirb http://localhost:8025

DIRB v2.22
By The Dark Raver

START_TIME: Tue Jun 25 12:54:28 2024
URL_BASE: http://localhost:8025/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: http://localhost:8025/ ---
+ http://localhost:8025/favicon.ico (CODE:200|SIZE:15406)
+ http://localhost:8025/proxy (CODE:400|SIZE:18)
+ http://localhost:8025/search (CODE:200|SIZE:641)

END_TIME: Tue Jun 25 12:54:31 2024
DOWNLOADED: 4612 - FOUND: 3
```

When we scanned ports 8001 / 8025, insensitive paths were found out.

## Nikto

To enhance our ability to uncover vulnerabilities and misconfigurations, we employed Nikto. This tool specializes in identifying weaknesses found primarily in web applications and servers.


When ports 8000, 8001, 8025 were scanned, Nikto identified some issues like unavailability of CSP, anti-clickjacking X-Frame-Options and Php version detection and other issues which can be useful information during exploitation.



```
(kali@kali)~/btu_project/stura-forms/public
$ nikto -h 127.0.0.1:8000
- Nikto v2.5.0

+ Target IP: 127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port: 8000
+ Start Time: 2024-06-25 12:54:10 (GMT-4)

+ Server: No banner retrieved
+ /: Cookie XSRF-TOKEN created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Retrieved x-powered-by header: PHP/8.3.8.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OPTIONS: Allowed HTTP Methods: GET, HEAD .
+ /.htaccess: Contains configuration and/or authorization information.
```




```
(kali@kali)~
$ nikto -h 127.0.0.1:8025
- Nikto v2.5.0

+ Target IP: 127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port: 8025
+ Start Time: 2024-06-25 13:00:09 (GMT-4)

+ Server: No banner retrieved
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ 8074 requests: 0 error(s) and 2 item(s) reported on remote host
+ End Time: 2024-06-25 13:00:32 (GMT-4) (23 seconds)

+ 1 host(s) tested
```



```
(kali@kali)~/btu_project/stura-forms/public
$ nikto -h 127.0.0.1:8001
- Nikto v2.5.0

+ Target IP: 127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port: 8001
+ Start Time: 2024-06-25 12:59:44 (GMT-4)

+ Server: No banner retrieved
+ /: Cookie XSRF-TOKEN created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Retrieved x-powered-by header: PHP/8.3.8.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OPTIONS: Allowed HTTP Methods: GET, HEAD, POST .
+ /.htaccess: Contains configuration and/or authorization information.
```

PHP version knowledge is particularly crucial because attackers are aware that different PHP versions can have various vulnerabilities. Understanding the PHP version running on a server helps both attackers and defenders assess potential security risks and vulnerabilities that could be exploited. Fortunately, we witnessed that PHP in this application is latest.

## Wapiti

We utilized another vulnerability scanning tool called Wapiti. It's designed to scan web applications for vulnerabilities like SQL injection, XSS (Cross-Site Scripting), command execution, and file inclusion issues. Wapiti actively investigates web applications to identify these common security weaknesses, helping to strengthen their overall security posture.

Finding of the tool is provided below.

From the results, we can see that potential vulnerabilities like SQL Injection, XSS, CSRF, LFI, SSRF, and command execution were not identified. However, Medium and Low severity vulns like CSP configuration, HTTP secure headers, HTTPOnly flag cookie, and Secure flag cookie were found out on ports 8000 and 8001.

### Wapiti vulnerability report





Target: <http://127.0.0.1:8000/>

Summary	
Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
<a href="#">Content Security Policy Configuration</a>	1
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Htaccess Bypass	0
<a href="#">HTTP Secure Headers</a>	4
<a href="#">HttpOnly Flag cookie</a>	2
Open Redirect	0
<a href="#">Secure Flag cookie</a>	2
SQL Injection	0
Server Side Request Forgery	0
Cross Site Scripting	0
XML External Entity	0
Internal Server Error	0
Resource consumption	0
Fingerprint web technology	0

## Wapiti vulnerability report

Target: http://127.0.0.1:8001/

### Summary

Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
<a href="#">Content Security Policy Configuration</a>	1 
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Htaccess Bypass	0
<a href="#">HTTP Secure Headers</a>	4 
<a href="#">HttpOnly Flag cookie</a>	2 
Open Redirect	0
<a href="#">Secure Flag cookie</a>	2 
SQL Injection	0
Server Side Request Forgery	0
Cross Site Scripting	0
XML External Entity	0
Internal Server Error	0
Resource consumption	0
Fingerprint web technology	0

When we scanned port 8025 which is Mailbox, Wapiti could not identify any issues as shown below.

## Wapiti vulnerability report

Target: <http://127.0.0.1:8025/>

### Summary

Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	0
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Htaccess Bypass	0
HTTP Secure Headers	0
HttpOnly Flag cookie	0
Open Redirect	0
Secure Flag cookie	0
SQL Injection	0
Server Side Request Forgery	0
Cross Site Scripting	0
XML External Entity	0
Internal Server Error	0
Resource consumption	0
Fingerprint web technology	0

Summary: from information gathering we collected less insensitive information about the system which do not lead to any potential attacks. Additionally, the data obtained from Dirb, Nikto and Wapiti were alike which we performed assessment in DAST.

## 4. Conclusion

### 4.1. Summary of overall security posture

We initiated our security assessment process by conducting a thorough threat modeling exercise at the outset. This step allowed us to identify potential threats and vulnerabilities systematically, forming the foundation of our subsequent testing phases.

Following the threat modeling, we implemented a series of security testing methodologies on the web application, specifically Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and penetration testing. Each of these methods served to uncover different types of vulnerabilities within the system:

1. **\*SAST\***: This involved analyzing the application's source code, byte code, and binaries for security vulnerabilities without executing the code. It provided insights into potential flaws in the application's codebase.
2. **\*DAST\***: This was performed by interacting with the application in its running state. It helped identify runtime vulnerabilities that could be exploited during the application's execution.
3. **\*Penetration Testing\***: This included simulated attacks on the application to identify security weaknesses that could be exploited by an adversary.

Throughout these testing phases, several tools were employed to identify potential security gaps. The results indicated a number of vulnerabilities categorized as Medium and Low. Importantly, these vulnerabilities did not pose significant threats to the system's integrity or functionality. Most of the identified weaknesses were related to information exposure, which, while important to address, do not constitute critical security threats.

During the penetration testing phase, our efforts were primarily focused on information gathering. This is due to the fact that we did not uncover any substantial vulnerabilities that could be exploited to compromise the system further. The absence of high-risk vulnerabilities and the inability to proceed beyond information gathering suggest that the web application demonstrates a robust security posture.

In summary, our comprehensive security testing—including threat modeling, SAST, DAST, and penetration testing—revealed that the web application is well-secured against potential threats. The findings underscore that while there are minor vulnerabilities, they do not significantly impact the overall security of the system. Therefore, the web application can be considered secure, with no evident critical vulnerabilities that would facilitate a system breach.



## 5. Appendices

### 5.1. Additional resources and references

1. [https://owasp.org/www-community/vulnerabilities/Use\\_of\\_hard-coded\\_password](https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password)
2. .htaccess Information Leak
3. Vulnerabilities .htaccess Information Leak
4. Content Security Policy (CSP) Header Not Set
5. Content Security Policy (CSP) Header Not Set
6. Missing Anti-clickjacking Header
7. Clickjacking Defense Cheat Sheet
8. Big Redirect Detected (Potential Sensitive Information Leak)
9. <https://owasp.org/www-community/HttpOnly#>
10. <https://cwe.mitre.org/data/definitions/1004.html>
11. Cookie without HttpOnly flag set
12. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)
13. X-Content-Type-Options Header Missing