IBSERVICE REPORT

SUBJECT:	awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad	DATE:	27.02.2025	↓

General Information about the Target
Target IP: 224.0.0.1
Target Hostname: awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion
Target Port: 80
Server: nginx
The main web resource ("/") redirects to the login page: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login

Discovered Vulnerabilities and Observations

2.1 **Description**
Content Security Policy (CSP) is an additional layer of protection that helps to detect and prevent certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.
Vulnerability found in /
Description: CSP is not configured.
HTTP request:

```
GET / HTTP/1.1
Host:
awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion
```

Severity: High (The lack of CSP makes the site vulnerable to XSS attacks and malicious code injection).
Solution
Configure Content Security Policy by adding the HTTP header Content-Security-Policy and specifying allowed resource loading sources.
Resources:
Mozilla: Content Security Policy (CSP)

OWASP: Content Security Policy Cheat Sheet

2.2 **Secure HTTP Headers**

Description

HTTP security headers tell the browser how to handle website content.

Vulnerabilities found in /

Missing X-XSS-Protection

HTTP request (cURL):

```
GET / HTTP/1.1
Host:
awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.on
ion
```

Severity: Medium (May increase the risk of XSS attacks).

2.3 **Missing Strict-Transport-Security (HSTS)**

HTTP request:

```
GET / HTTP/1.1
Host:
awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.on
ion
```

Severity: High (Without HSTS, "downgrade attacks" and traffic interception are possible).

Solution

Configure security headers:

Enable X-XSS-Protection: 1; mode=block

Enable Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

Resources:

OWASP: HTTP Security Headers

KeyCDN: Hardening Your HTTP Security Headers

2.4 **Secure Flag in Cookies**

Description

The Secure flag is an option that the server can set when sending cookies. It prevents cookies from being leaked over unencrypted HTTP connections.
Vulnerability found in /
Missing Secure flag in cookie: EWsMVjIa4EGHKKWIZNC2
HTTP request:

```
GET / HTTP/1.1
Host:
awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion
```

Severity: High (Cookies can be intercepted when transmitted over unencrypted HTTP connections).
Solution
Configure the server to always set the Secure flag when generating cookies.
Example in PHP:

```
setcookie("session", $value, [
    'secure' => true,  // Only transmit over HTTPS
    'httponly' => true,  // Disallows cookie access via JS
    'samesite' => 'Strict'  // Prevents CSRF attacks
]);
```

Resources:
[OWASP: Session Management Cheat Sheet]

2.5 **X-Frame-Options Header Not Set**
This allows an attacker to embed the site in an iframe on a malicious page (Clickjacking).
Severity: Medium
Solution
Add the X-Frame-Options header with one of the following values:
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
X-Frame-Options: ALLOW-FROM https://example.com

2.6 **Missing X-Content-Type-Options Header**
Description

The absence of the X-Content-Type-Options header allows browsers to interpret MIME types in an unsafe manner (MIME sniffing).
Severity: Medium
Solution
Add the X-Content-Type-Options: nosniff header to the HTTP response from the server:
X-Content-Type-Options: nosniff

Directories

/ - 302 (Redirect)
/search/ - 307 (Redirect)
/index.php - 307 (Redirect)
/assets/ - 403 (Access Forbidden)
/assets/img/ - 403 (Access Forbidden)
/assets/products/ - 403 (Access Forbidden)
/assets/img/icons/ - 403 (Access Forbidden)
/assets/img/icons/search/ - 403 (Access Forbidden)
/affiliate/ - 307 (Redirect)
/assets/css/ - 403 (Access Forbidden)
/assets/img/icons/flags/ - 403 (Access Forbidden)
/assets/small/ - 403 (Access Forbidden)
/assets/avatars/ - 403 (Access Forbidden)
/pgp/ - 307 (Redirect)
/assets/img/icons/sun/ - 403 (Access Forbidden)
/administration/ - 307 (Redirect)
/administration/products/ - 307 (Redirect)
/administration/chat/ - 307 (Redirect)
/assets/img/icons/sell/ - 403 (Access Forbidden)

Recommendations for Fixing Vulnerabilities
Configuring HTTP Security Headers
Add and correctly configure the X-Frame-Options, X-Content-Type-Options, Content-Security-Policy, X-XSS-Protection, and Strict-Transport-Security headers in the web server (nginx) configuration.
Follow the official recommendations from Mozilla, OWASP, and Netsparker for configuring each of these headers.

Securing Cookies

Set the Secure flag for all cookies, especially those used for authentication or transmitting sensitive information.

IBSERVICE REPORT

SUBJECT:	awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad	DATE:	28.02.2025	↓

General Information about the Target

Target IP: 224.0.0.1
Target Hostname: awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion
Target Port: 80
Server: nginx
The main web resource ("/") redirects to the login page: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login

Discovered Vulnerabilities and Observations

2.1 Possible SQL Injection Vulnerability

1. Summary A possible SQL injection vulnerability has been identified in the password reset functionality of the web application. The vulnerability is confirmed by controlling query execution time through SQL payloads, indicating that unsanitized input is being processed by the database.

2. Affected Endpoint

Request:

```
POST http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login HTTP/1.1

host: awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:135.0) Gecko/20100101 Firefox/135.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

Referer: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login

Content-Type: application/x-www-form-urlencoded
```

```
content-length: 163

Origin: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion

Connection: keep-alive

Cookie: EWsMVjIa4EGHKKWIZNC2=59737353299788899ed1dd4fed2ec2d1

Upgrade-Insecure-Requests: 1

Priority: u=0, i
7if6ttcnzvP9u6hF5Fia=case+randomblob%2810000000%29+when+not+null+then+1+else+1+end+&usern
ame=servers&password=123123123&session_duration=15&captcha=YOLB3&honeypot=
```

- **HTTP Method:** POST
- **URL:** http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/reset-password
- **Vulnerable Parameters:** captcha, 7if6ttcnzvP9u6hF5Fia, default_language, withdrawal_pin, honeypot

3. Vulnerability Details The injection is confirmed using time-based SQL injection techniques:

- **Payload 1:** case randomblob(100000000) when not null then 1 else 1 end
  - Execution time: **718 ms**
- **Payload 2:** case randomblob(1000000000) when not null then 1 else 1 end
  - Execution time: **2,269 ms**
- **Original Query Execution: 463 ms**

These results confirm that query execution time is influenced by the input values, which is indicative of a time-based SQL injection vulnerability.

4. Evidence

- The application returned an HTTP 200 OK response, which suggests that the payload was executed successfully.
- The observed response delay aligns with the expected time-based SQL injection behavior.

5. Risk Assessment

- **Impact:** High
- **Likelihood:** High
- **Affected Systems:** Database backend, user authentication system
- **Potential Exploitation:** An attacker can exploit this vulnerability to execute arbitrary SQL queries, retrieve sensitive data, modify database content, or escalate privileges.

6. Mitigation Recommendations To prevent SQL injection, implement the following best practices:

1. **Use Prepared Statements:** Ensure all SQL queries use parameterized queries or prepared statements. Example in Python:
2. `cursor.execute("SELECT * FROM users WHERE username = ?", (username,))`
3. **Server-Side Input Validation:**
   - Perform strict type checking for all input parameters.
   - Use an allowlist of expected input values.
4. **Escape User Input:** If using a database driver that does not support prepared statements, ensure all user input is properly escaped before being concatenated into a SQL query.
5. **Apply the Principle of Least Privilege:**
   - Restrict database user permissions to the minimum necessary.
   - Avoid using privileged accounts such as sa or db-owner.
6. **Use Web Application Firewalls (WAFs):** Deploy a WAF to monitor and block malicious SQL injection attempts.

7. Conclusion This vulnerability represents a significant security risk and should be remediated immediately. Implementing prepared statements, strict input validation, and least-privilege principles will mitigate the risk of SQL injection attacks.

8. References

- OWASP SQL Injection Prevention Cheat Sheet
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command

2. Additional Security Findings

2.1 Content Security Policy (CSP) Header Not Set

Risk Level: **Medium**

Description: The Content Security Policy (CSP) header is missing. CSP helps mitigate various attacks, including Cross-Site Scripting (XSS) and data injection attacks, by restricting the sources from which content (JavaScript, CSS, images, etc.) can be loaded.

Remediation: Ensure that the web server, application server, or load balancer is configured to set a strong Content-Security-Policy header.

References:

- [MDN Introduction to CSP]
- [OWASP CSP Cheat Sheet]
- [W3C CSP Specification]
- [Web.dev Guide to CSP]
- [CanIUse CSP Support]
- [Content-Security-Policy.com]

2.2 Security Technology Identified: reCAPTCHA

Risk Level: **Informational**

Description: The application utilizes Google's reCAPTCHA, which is a security mechanism designed to differentiate between human users and automated bots.

Recommendation:

- **While reCAPTCHA is beneficial, it should not be solely relied upon for security. Additional mechanisms, such as rate limiting and bot detection, should be implemented**

IBSERVICE REPORT

SUBJECT:	awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad	DATE:	01.03.2025	↓

General Information about the Target
Target IP: 224.0.0.1
Target Hostname: awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion
Target Port: 80
Server: nginx
The main web resource ("/") redirects to the login page: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login

Discovered Vulnerabilities and Observations

Detailed Vulnerability Report

1) X-Content-Type-Options Header Missing

URL:
http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/robots.txt
Parameter: x-content-type-options
Description: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing it to be interpreted as a different content type than declared. This issue also applies to error pages (401, 403, 500, etc.), which may still be affected by injection vulnerabilities.
Risk: Medium
Solution: Ensure the application/web server sets the Content-Type header appropriately and includes X-Content-Type-Options: nosniff for all responses.
References:

- **Microsoft Documentation**
- **OWASP Security Headers**

2) Information Disclosure via Base64 Encoding

URL: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/
Evidence:

```
iVBORwØKGgoAAAANSUhEUgAAAMgAAAA8CAIAAACsOWLGAAAACXBIWXMAAA7EAAAOxAGVKw4bAA
AEjØ1EQVR4nO1cba7jIAzkSXuj3ome6fVMzZmyP9JHXJwQY2y+4miFRlVKbDMeprRvf/zv+loW
55xzzj8ehg2LYOf82/+u4...
```

Other Info: Detected `\x89PNG\r\n\x1A` header indicating a PNG file.
Risk: Low
Solution: Manually verify that Base64-encoded data does not expose sensitive information. Ensure such data cannot be leveraged to exploit other vulnerabilities.
References:

- OWASP Information Leakage

3) Authentication Request Identified

URL:
http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/auth/login
Parameter: username
Other Info:

```
userParam=username

userValue=

passwordParam=password
```

Risk: Medium
Solution: Ensure that authentication endpoints are properly secured against brute force attacks and that sensitive data is transmitted securely using HTTPS.

4) Sec-Fetch-Dest Header is Missing

URL:
http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/robots.txt
Parameter: Sec-Fetch-Dest
Description: The Sec-Fetch-Dest header is missing, which can affect how the browser processes and requests resources. This header helps prevent cross-site leaks by specifying how the requested resource should be used.
Risk: Low
Solution: Ensure that Sec-Fetch-Dest is included in request headers.
References:

- MDN: Sec-Fetch-Dest

- MDN: Sec-Fetch-Site
- MDN: Sec-Fetch-Mode

5) User Agent Fuzzer

Parameter: User-Agent
Description: Variations in response based on different User-Agent strings indicate that different content is served to different user agents. This can reveal hidden functionality or security flaws when responses differ for specific user agents.
Risk: Medium
Solution: Implement proper request validation and ensure that user-agent-based filtering does not expose unintended content or behaviors.

Technology Identified

URL: http://awazonhndi7e5yfaobpk7j2tsnp4kfd2xa63tdtzcg7plc5fka4il4ad.onion/
Detected Tech: Nginx
`CPE: cpe:2.3:a:f5:nginx:*:*:*:*:*:*:*:*`