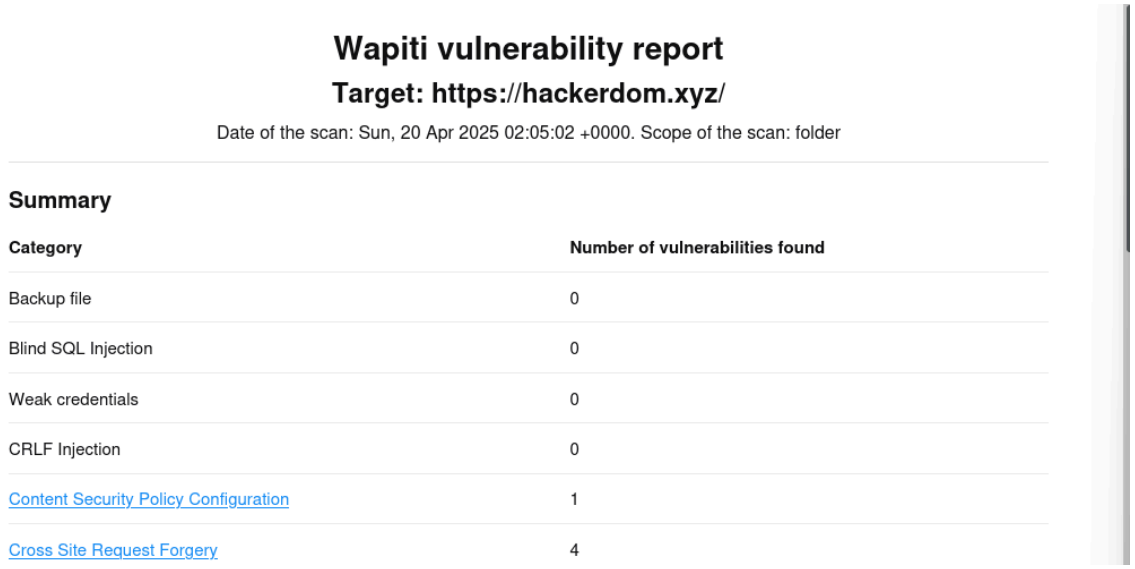
Vulnerability Report – Wapiti Scan of https://hackerdom.xyz

Scan Date: April 20, 2025
Tool Used: Wapiti 3.0.4
Scope: Full folder scan of the website
Total URLs/Form scanned: 100



Wapiti vulnerability report

Target: https://hackerdom.xyz/

Date of the scan: Sun, 20 Apr 2025 02:05:02 +0000. Scope of the scan: folder

Summary

Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	1
Cross Site Request Forgery	4

Vulnerabilities Found and Fixes

---

1. Content Security Policy (CSP) Not Set

Vulnerability:
The website does not use a Content-Security-Policy header, which helps prevent XSS and data injection.

Fix:
Add the following to .htaccess:

Header set Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self';"

Alternative Fix:
Install the **CSP Ninja** WordPress plugin.

---

2. Missing CSRF Protection

Cross Site Request Forgery

Description

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

Vulnerability found in /

Description | HTTP Request | cURL command line

```
Lack of anti CSRF token
```

Vulnerability found in /

Description | HTTP Request | cURL command line

```
Lack of anti CSRF token
```

Vulnerability found in /login/

Description | HTTP Request | cURL command line

```
Lack of anti CSRF token
```

Vulnerability found in /register/

Description | HTTP Request | cURL command line

```
Lack of anti CSRF token
```

Vulnerabilities Found (4):

- **Contact form (/) lacks CSRF token**
- **POST to /login/ lacks CSRF token**
- **POST to /register/ lacks CSRF token**

Fix (in custom PHP):

Solutions

Check if your framework has built-in CSRF protection and use it. If framework does not have built-in CSRF protection add CSRF tokens to all state changing requests (requests that cause actions on the site) and validate them on backend.

In the form:

```
<?php
session_start();
$_SESSION['csrf_token'] = $_SESSION['csrf_token'] ?? bin2hex(random_bytes(32));
?>
<input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
```

In the form handler:

```
session_start();
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("CSRF validation failed.");
}
```

WordPress Plugin Option:
Use **WP Cerber** or **Wordfence** to protect WordPress forms.

---

3. Missing HTTP Security Headers

HTTP Secure Headers

Description

HTTP security headers tell the browser how to behave when handling the website's content.

Vulnerability found in /

Description | HTTP Request | cURL command line

```
X-Frame-Options is not set
```

Vulnerability found in /

Description | HTTP Request | cURL command line

```
X-XSS-Protection is not set
```

Vulnerability found in /

Description | HTTP Request | cURL command line

```
X-Content-Type-Options is not set
```

Vulnerability found in /

Description | HTTP Request | cURL command line

```
Strict-Transport-Security is not set
```

Vulnerabilities Found:

- **X-Frame-Options not set**
- **X-XSS-Protection not set**
- **X-Content-Type-Options not set**
- **Strict-Transport-Security not set**

Fix (in .htaccess):

Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-XSS-Protection "1; mode=block"
Header always set X-Content-Type-Options "nosniff"

Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"

Plugin Option:
Install **HTTP Headers plugin** and configure all missing headers.

---

4. HttpOnly Cookie Flag Missing

Solutions
While creation of the cookie, make sure to set the HttpOnly Flag to True.

Vulnerability:
The cookie asp_transient_id was missing the HttpOnly flag.

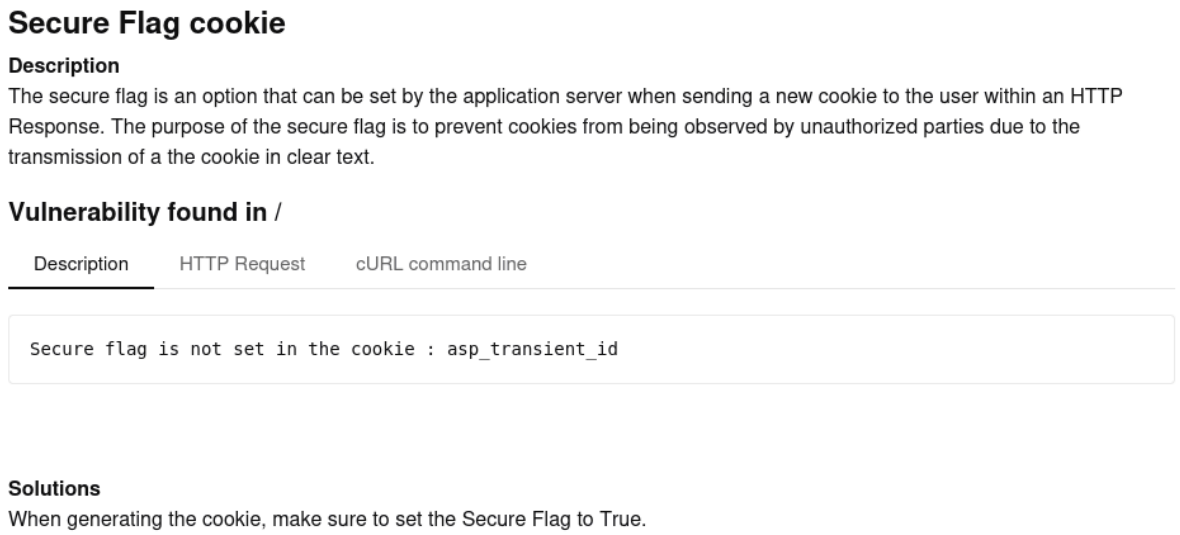
Fix:

```
setcookie("asp_transient_id", $value, [
  'secure' => true,
  'httponly' => true,
  'samesite' => 'Strict'
]);
```

Plugin Option:
Use a Really Simple SSL plugin to enforce secure cookie settings.

---

5. Secure Cookie Flag Missing



Secure Flag cookie

Description

The secure flag is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being observed by unauthorized parties due to the transmission of a the cookie in clear text.

Vulnerability found in /

Description | HTTP Request | cURL command line

```
Secure flag is not set in the cookie : asp_transient_id
```

Solutions

When generating the cookie, make sure to set the Secure Flag to True.

Vulnerability:
The cookie asp_transient_id was also missing the Secure flag.

Fix:
Same as above — ensure the cookie is set with secure => true in custom PHP.

No Other Vulnerabilities Detected

Test Category	Status
SQL Injection	Safe
Cross-Site Scripting (XSS)	Safe
Open Redirects	Safe
Command Injection (RCE)	Safe
Server-Side Request Forgery	Safe
Path Traversal	Safe
Backup Files / Configs	Safe
XML External Entity (XXE)	Safe

Conclusion

The Wapiti scan helped identify critical misconfigurations related to security headers, CSRF protection, and cookie flags. These vulnerabilities do not represent immediate exploitation risk, but if left unpatched, they can expose the system to more serious attacks.

Applying the recommended .htaccess configurations and updating your custom PHP pages with CSRF tokens and secure cookie flags will significantly improve the site's overall security posture.

References

- OWASP Security Headers Guide
- Mozilla Content Security Policy Docs
- WordPress Plugins: HTTP Headers, Wordfence, Really Simple SSL, WP Cerber