Security Assessment Report

Author: Piyush Bansal
Date: 23 / Nov / 2025
Target Environment: Local OWASP Juice Shop instance (Node.js + npm)
Testing Methodology: Manual testing + OWASP Top 10 mapping

# 1. Overview

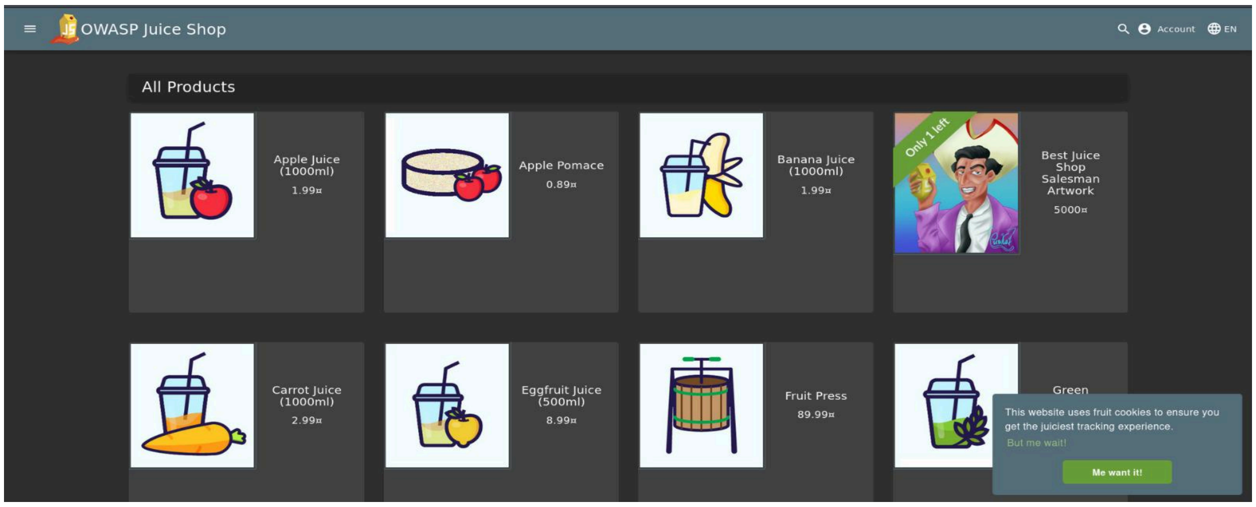
This document contains findings from a security assessment of the OWASP Juice Shop web application. The goal of this test was to identify common vulnerabilities such as Injection, Broken Authentication, XSS, and others from the OWASP Top 10 list.

# 2. Scope of Testing

### In-Scope Components:
- Juice Shop website (local: http://localhost:3000)
- Login, registration, search bar, product pages
- Client-side frontend
- API network requests

### Out of Scope:
- Denial-of-service/stress testing
- Modifying backend code
- Outbound attacks

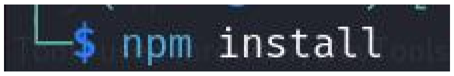
# 3. Tools Used
- Browser Developer Tools (Network / Console)
- Manual Payload Testing
- Burp Suite (didn't use because of misconfiguration and proxy problem)
- OWASP ZAP
- Nmap / Nikto



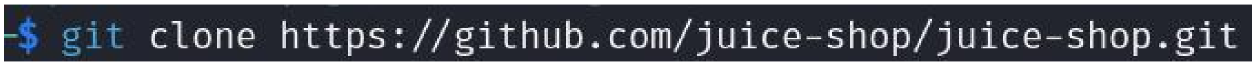
This website uses fruit cookies to ensure you get the juiciest tracking experience.

Setup :

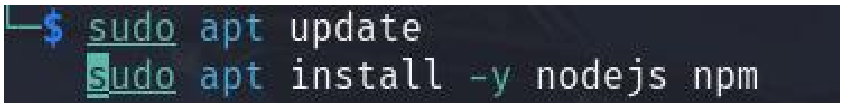
- Installed Node.js + npm
- Cloned Owasp Juice Shop Github Repository
- Started Using npm start
- Accessed https://localhost:3000



```
└─$ npm install
```

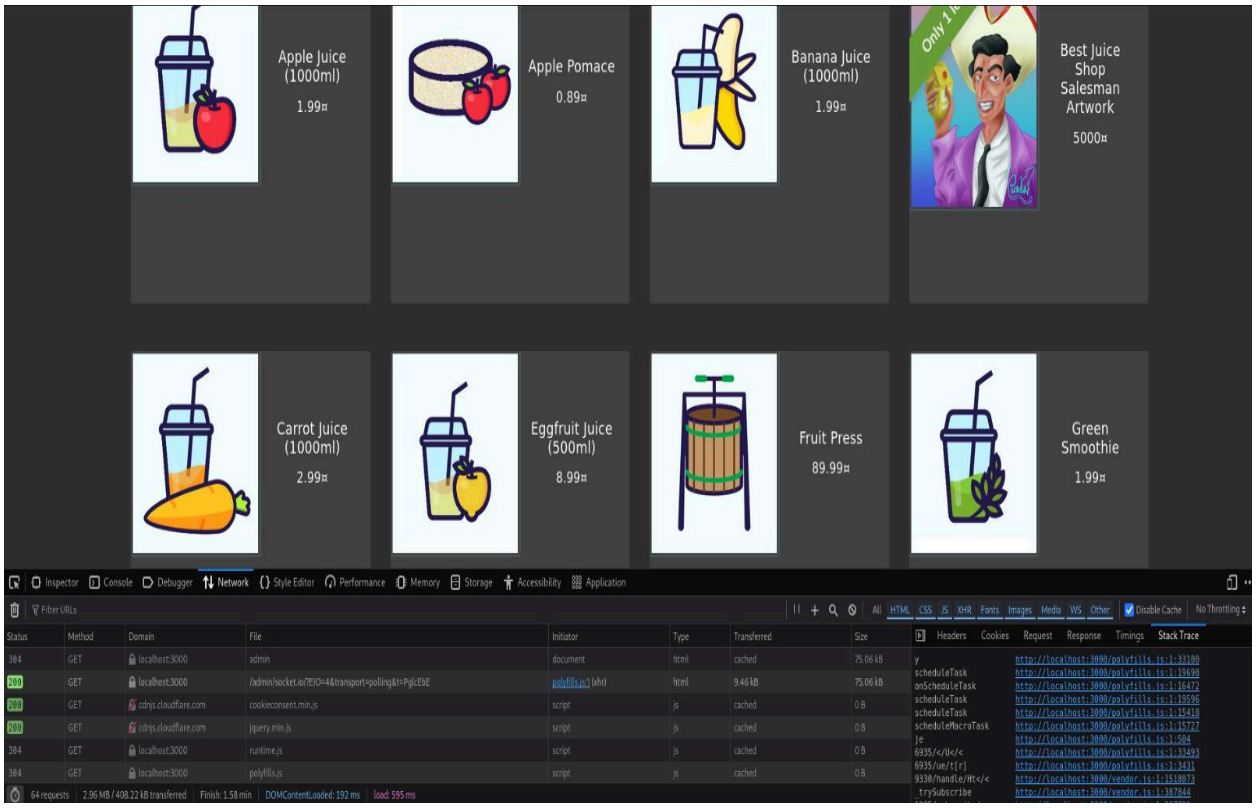


```
-$ git clone https://github.com/juice-shop/juice-shop.git
```



```
└─$ sudo apt update
   sudo apt install -y nodejs npm
```

Initial Observation: Explored the application manually, listed important pages, identified api communications, and observed request/response behaviors



Status	Method	Domain	File	Initiator	Type	Transferred	Size
304	GET	localhost:3000	admin	document	html	cached	75.06 kB
200	GET	localhost:3000	/admin/socket.io/?EIO=4&transport=polling&t=PglcEbE	polyfills.js:1 (xhr)	html	9.46 kB	75.06 kB
200	GET	cdnjs.cloudflare.com	cookieconsent.min.js	script	js	cached	0 B
200	GET	cdnjs.cloudflare.com	jquery.min.js	script	js	cached	0 B
304	GET	localhost:3000	runtime.js	script	js	cached	0 B
304	GET	localhost:3000	polyfills.js	script	js	cached	0 B

64 requests | 2.96 MB / 408.22 kB transferred | Finish: 1.58 min | DOMContentLoaded: 192 ms | load: 595 ms

```
y                     http://localhost:3000/polyfills.js:1:33100
scheduleTask          http://localhost:3000/polyfills.js:1:19698
onScheduleTask        http://localhost:3000/polyfills.js:1:16472
scheduleTask          http://localhost:3000/polyfills.js:1:19596
scheduleTask          http://localhost:3000/polyfills.js:1:15418
scheduleMacroTask     http://localhost:3000/polyfills.js:1:15727
je                    http://localhost:3000/polyfills.js:1:504
6935/</U</<           http://localhost:3000/polyfills.js:1:33493
6935/ue/t[r]          http://localhost:3000/polyfills.js:1:3431
9330/handle/Ht</<     http://localhost:3000/vendor.js:1:1518073
_trySubscribe         http://localhost:3000/vendor.js:1:387844
```

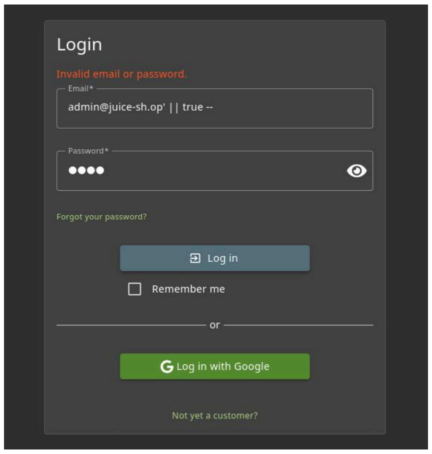
\

Vulnerability 1 : Improper Input Handling ([object Object] Error Leak)

Description : When testing the login form with special characters or injection-like payloads (e.g., ', {"$ne":""}), the frontend displays: [object Object] This indicates the application is exposing internal JavaScript error objects instead of sanitizing output.

.

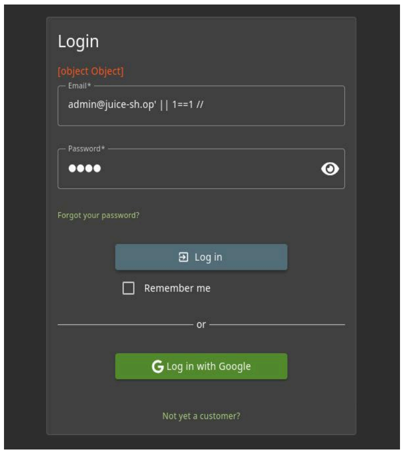
Impact :
- Minor information disclosure.
- Helps an attacker understand how the frontend handles failed login attempts.
- Could assist in crafting more precise injection payloads.



Invalid email or password.

Email*: admin@juice-sh.op' | | true --

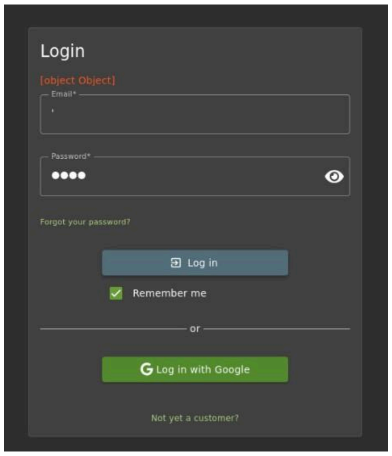
Password*: ●●●●



[object Object]

Email*: admin@juice-sh.op' | | 1==1 //

Password*: ●●●●



[object Object]

Email*: '

Password*: ●●●●

Remember me ✓

Testing :
-payloads used: username: ‘ password 123
-Input in Email field : ‘ or admin@juice-sh.op’ || 1==1 // or admin@juice-sh.op’ || true –
-Output : [object Object] however this only appears on the screen (UI), not in the network response

OWASP Mapping : A05: Security Misconfiguration
Severity : Low

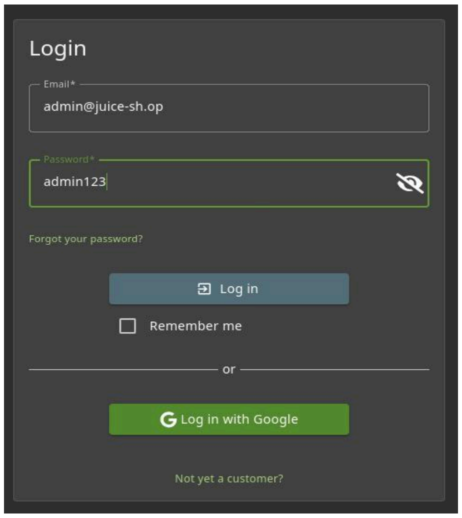
Remediation :
1. Robust Input Validation
Assume all input is malicious and use an "accept known good" validation strategy.
Implement Strict Validation: Ensure all inputs conform to the expected type, length, format, and range of values. Reject any input that does not strictly conform to specifications or transform it into a valid form.
Use Whitelisting: Define and use a list of acceptable inputs rather than trying to filter out known bad inputs.
Validate All Input Sources: Treat input from all sources as untrusted, including form data, cookies, HTTP headers, URL parameters, and data from external systems.
Context-Specific Validation: Apply validation specific to the context in which the data will be used (e.g., file names, database queries, HTML output) to prevent attacks like SQL injection or cross-site scripting (XSS).

2. Secure Error Handling
Proper error handling provides a meaningful, generic message to the user while logging detailed diagnostics for administrators, and provides no useful information to an attacker.
Generic User Messages: Configure the application to display simple, non-descriptive error messages to end-users (e.g., "An unexpected error occurred. Please try again later.")

Vulnerability 2 : Default Admin Credentials

Description : OWASP Juice Shop ships with public default credentials : admin@juice-sh.op and admin123. These provide full administrative access.

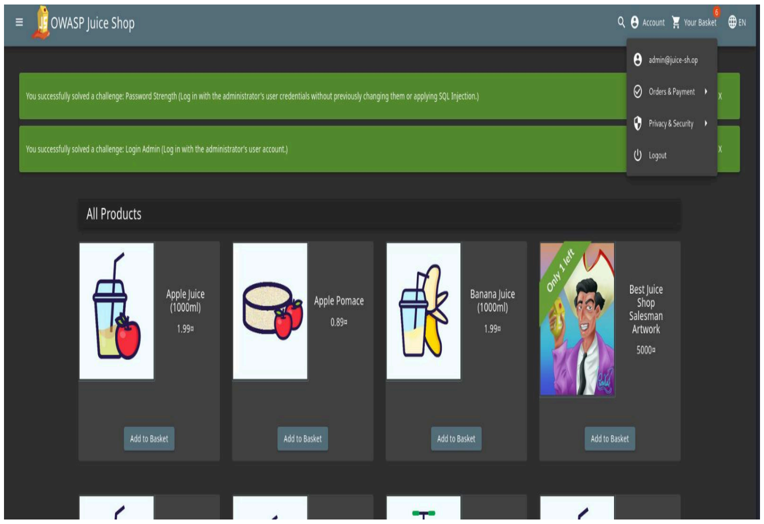
Impact :
- Full administrator access.
- Ability to modify products, view user data, access restricted endpoints.



Login

Email*
admin@juice-sh.op

Password*
admin123



You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.)

You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

Testing:
1. Navigate to Login Page.
2. Use the above credentials.
3. Access the admin panel.

OWASP Mapping: A07: Identification and Authentication Failures*
Severity: Critical

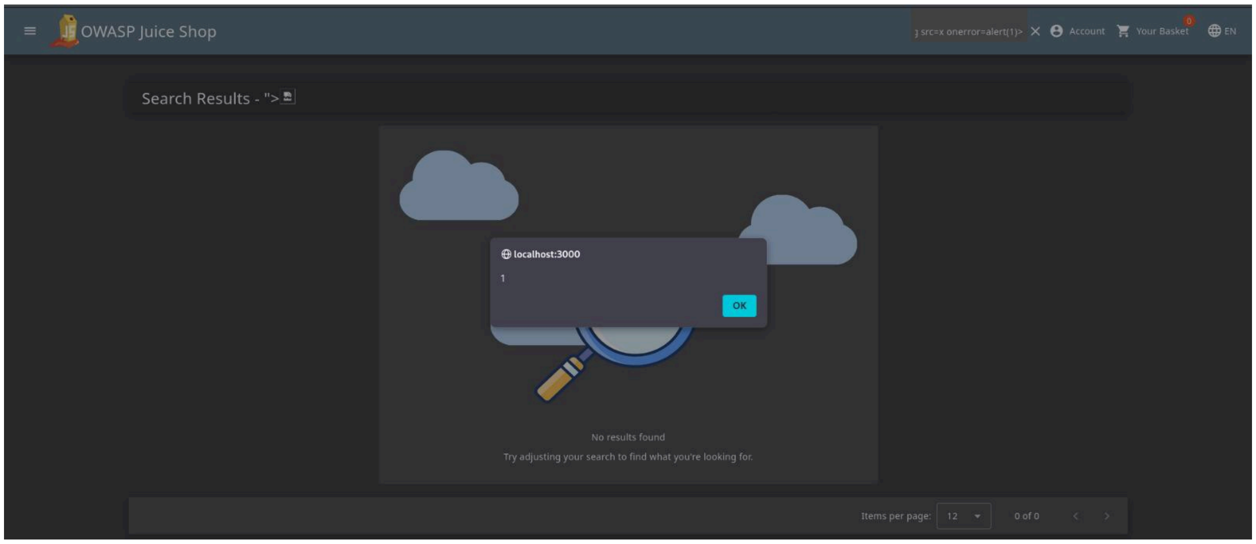
Remediation :
- ❖ Change credentials: Immediately change all default usernames and passwords on all affected systems and devices.
- ❖ Use strong, unique passwords: Create new passwords that are long, complex (including a mix of numbers, letters, and symbols), and unique to each system. Avoid using weak passwords based on dictionary words or easily guessed information.
- ❖ Disable unnecessary accounts: If default accounts are not required for a specific function, disable or remove them.
- ❖ Enforce a strong password policy: Implement policies that require users to create strong passwords, and periodically update them.
- ❖ Implement multi-factor authentication (MFA): Require more than just a password to log in, adding a significant layer of security.
- ❖ Limit access: Restrict access to systems and accounts with default credentials to only the necessary personnel.
- ❖ Use password managers: Store new, unique passwords in a secure password manager or a Privileged Access Management (PAM) tool to manage them safely.

Vulnerability 3 : Client-side reflected XSS

Description : The search functionality in OWASP Juice Shop is vulnerable to Reflected XSS. User-supplied input is dynamically injected into the page without proper HTML or JavaScript sanitization.

When a malicious payload is entered in the search bar, the browser executes arbitrary JavaScript.

Payload Used : “><img src=x onerror=alert(1)>



Test 1 : Login Email Field
Payloads tested : ', "><img src=x onerror=alert(1)>
Behavior: Form validates input; error returned “Invalid email or password”, no DOM injection.
Result : Not vulnerable.

Test 2 : Global Search Bar (VULNERABLE)
Payload: "><img src=x onerror=alert(1)>
Behavior: A JavaScript alert popup is triggered, confirming reflected XSS.

Impact:
- Session hijacking
- Account takeover via cookie theft
- Redirecting users to malicious sites
- JS execution in user’s browser

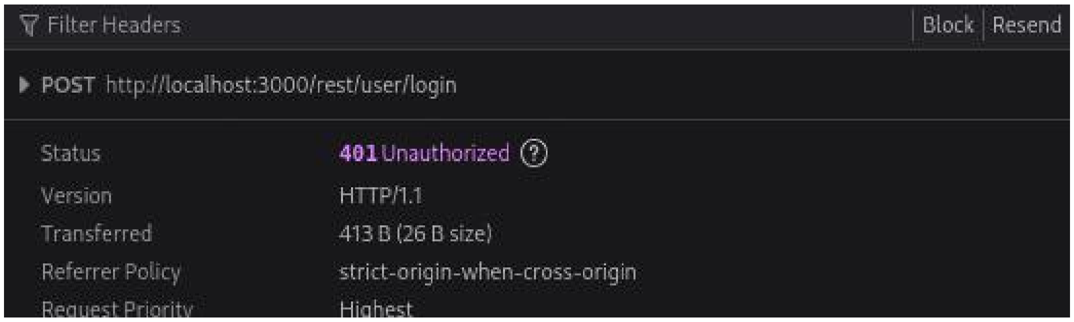
OWASP Category: A03: Injection → Cross-Site Scripting (XSS)
Severity : High

Remediation :
-Apply server-side and client-side input sanitization.
-HTML-encode special characters (<, >, ", ').
-Use libraries such as DOMPurify for input cleaning.
-Avoid inserting user input directly into the DOM without escaping.

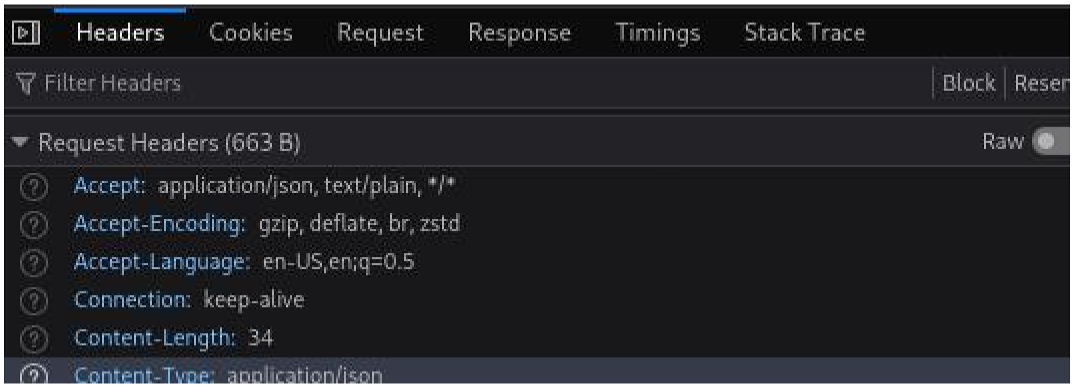
Vulnerability 4 : Missing CSRF protection on Login Endpoint

Description : During testing on the /rest/user/login endpoint, it was observed that the web application does not implement CSRF protection mechanisms such as CSRF tokens, SameSite cookie restrictions, or origin validation.

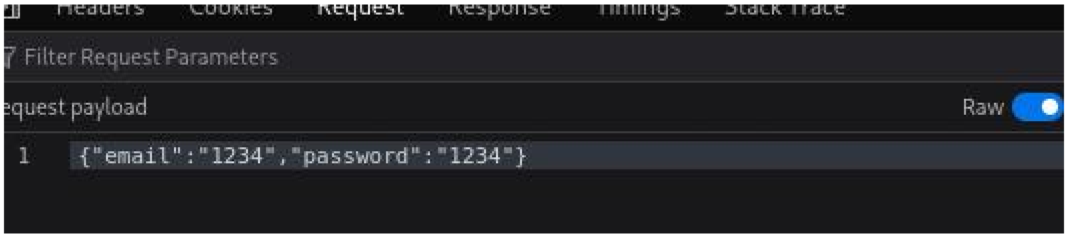
However, due to the endpoint requiring:Content-Type: application/json modern browsers send a CORS preflight request, preventing an attacker-controlled HTML form from automatically submitting the malicious POST request. This limits real-world exploitability even though protection is not implemented.



```
POST http://localhost:3000/rest/user/login

Status              401 Unauthorized
Version             HTTP/1.1
Transferred         413 B (26 B size)
Referrer Policy     strict-origin-when-cross-origin
Request Priority    Highest
```



```
Request Headers (663 B)
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Length: 34
Content-Type: application/json
```



```
equest payload
{"email":"1234","password":"1234"}
```

Impact : Although the endpoint lacks CSRF protection, the requirement for application/json prevents traditional CSRF attacks because:
-Browsers do not allow cross-site POST requests with JSON bodies without a CORS preflight request.
-Attacker-controlled CSRF payloads cannot set Content-Type: application/json.
-the server does not allow non-JSON bodies, preventing exploitation.

Therefore:
✔ Vulnerability exists (missing CSRF controls)
⚠ Exploitability is limited

If the server accepted text/html or x-www-form-urlencoded, full account takeover via CSRF would be possible.

Remediation:

-Implement full CSRF protection using one of the following:
-Synchronizer Token Pattern (server-generated CSRF token)
-Double Submit Cookie Pattern
-Set authentication cookies with:

OWASP Category : A01: Broken Access Control
Severity : Medium

Nikto Scan Findings Report

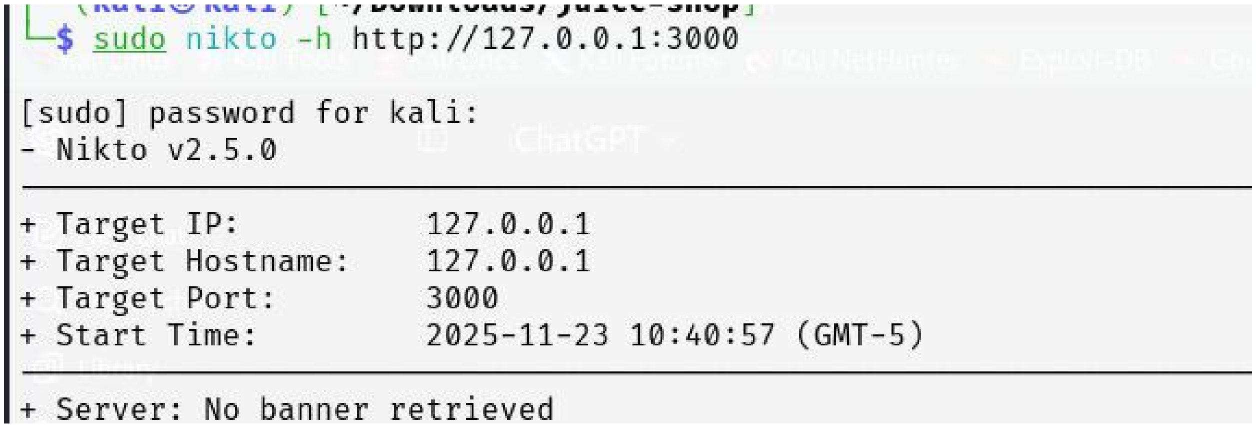
Introduction : Nikto is an open-source web server vulnerability scanner that performs automated tests to find security weaknesses, such as outdated software, dangerous files, and server misconfigurations

1. Server Header Disclosure

Finding: Nikto was unable to retrieve a server banner.

Explanation: While this may reduce information exposure, it should be verified that no sensitive information is unintentionally being disclosed.

OWASP Mapping: A05 – Security Misconfiguration



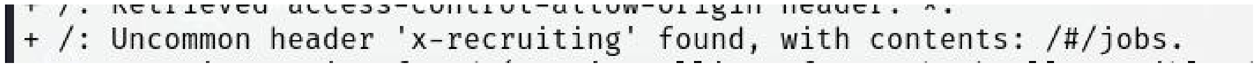
```
└─$ sudo nikto -h http://127.0.0.1:3000

[sudo] password for kali:
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    127.0.0.1
+ Target Port:        3000
+ Start Time:         2025-11-23 10:40:57 (GMT-5)
---------------------------------------------------------------------------
+ Server: No banner retrieved
```

2. Uncommon Header Detected

Finding: Uncommon header x-recruiting found with contents: /#/jobs.

Explanation:Custom headers may reveal unnecessary metadata or internal information about the application.

OWASP Mapping: A05 – Security Misconfiguration



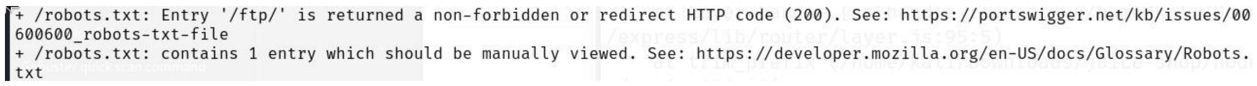
```
+ /: Uncommon header 'x-recruiting' found, with contents: /#/jobs.
```

3. robots.txt Exposure

Findings:
- /ftp/ directory returned code 200.
- robots.txt contains entries that should be manually reviewed.

Explanation: Sensitive directories should not be listed in robots.txt if they are meant to be private.

OWASP Mapping: A05 – Security Misconfiguration



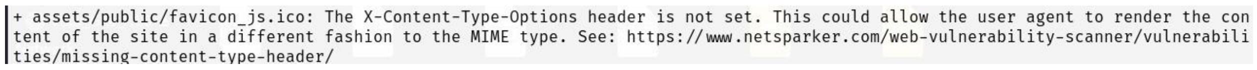
```
+ /robots.txt: Entry '/ftp/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00
600600_robots-txt-file
+ /robots.txt: contains 1 entry which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.
txt
```

4. Missing Security Header

Finding: X-Content-Type-Options header not set for favicon_js.ico.

Explanation: This allows browsers to MIME-sniff content, which can lead to certain attacks.

OWASP Mapping: A06 – Security Misconfiguration



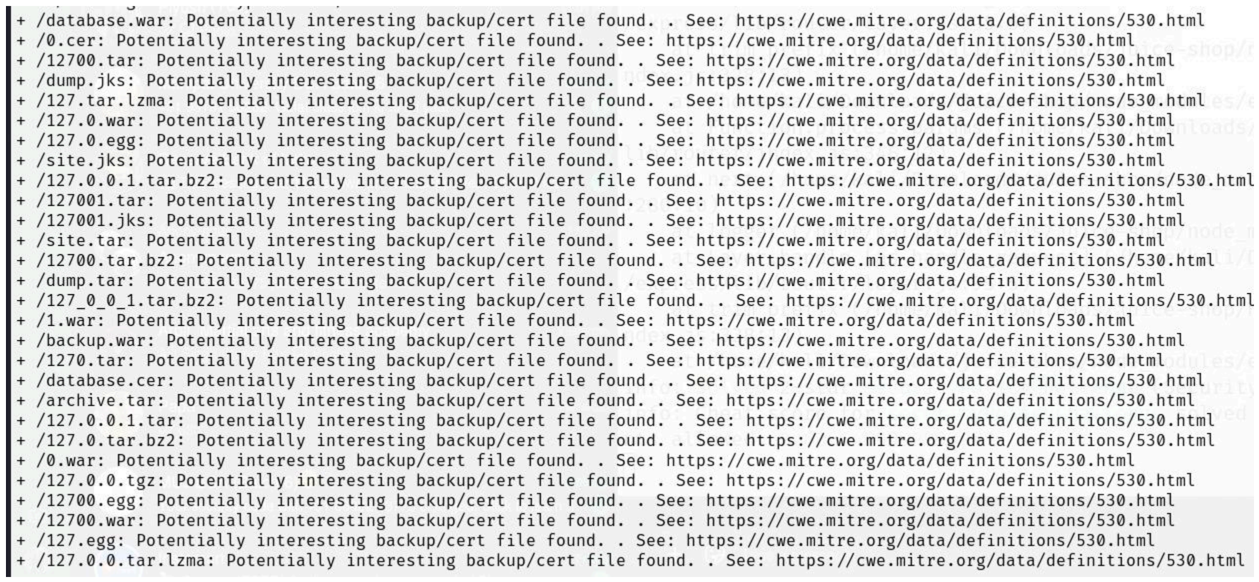
```
+ assets/public/favicon_js.ico: The X-Content-Type-Options header is not set. This could allow the user agent to render the con
tent of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabili
ties/missing-content-type-header/
```

5. Sensitive Backup and Certificate Files Accessible

Finding: Multiple sensitive files were detected, including:
* .war
* .cer
* .jks
* .pem
* .tar, .tgz, .bz2, .lzma archives

Explanation: These files may contain sensitive information such as credentials, certificate keys, or source code.

OWASP Mapping: A01 – Broken Access Control



```
+ /database.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /0.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /dump.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.egg: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /site.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.1.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127001.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127001.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /site.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /dump.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127_0_0_1.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /1.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /backup.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /1270.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /database.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /archive.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.1.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /0.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.egg: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.egg: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
```

Summary and Recommendations

Key Issues Identified:

* Exposure of sensitive directories and backup files
* Missing security headers
* Presence of uncommon or unnecessary headers

Recommended Remediation:

* Remove or secure publicly accessible backup/certificate files
* Limit directory exposure and update robots.txt appropriately
* Add missing security headers (e.g., X-Content-Type-Options)
* Patch or update vulnerable WordPress plugins
* Conduct a full access control review across the application

This document summarizes all key findings from the Nikto scan and provides OWASP mappings for clear risk categorization.