Result Report

Juice-Shop

Saftiger Vertrieb GmbH

Adam Apfel
Obststraße 44
Safthausen

Deining, October 3, 2025

Project Number: 0001

Report Version 1.0

CONFIDENTIAL

Schutzpunkt GmbH
Schloßstraße 7 A
92364 Deining

Contact:
Max Bäumler
Mobile: +49 9184 8081 966
Mail: *max.baeumler@schutzpunkt.com*
Web: *www.schutzpunkt.com*

Managing Director: Max Bäumler
Registration Court Nuermberg: HR B 44700
VAT-ID: DE455642304

Table of Contents

1 Document Control

1.1 Team

Contact	Details	Role
Max Bäumler	Mobile: +49 9184 8081 966 Mail: max.baeumler@schutzpunkt.com	Lead Pentester

1.2 List of Changes

Version	Description	Date
0.1	Initial version	Oct 3, 2025
1.0	Finalization of the report	Oct 3, 2025

2 Executive Summary

The goal of this penetration test was to assess the security of web application **Juice-Shop**, identify vulnerabilities, and evaluate potential risks to the organization's critical assets. All activities were performed between **Tuesday, September 30, 2025** and **Thursday, October 2, 2025**. In total **3 person days** were used for this test. This assessment is part of a broader effort to ensure the ongoing security and resilience of the organization's systems and data.

Key Findings

- **Overall Risk Level**: Based on the test results, **the security level can be classified as moderate compared to similar tests for other customers**. In total **5 open vulnerabilities** were identified, with varying levels of risk.
- **Key Vulnerabilities**: The vulnerabilities **H1, M2,** could potentially lead to: Immediate financial losses due to vulnerable voucher codes. Furthermore, unauthorized access, data breaches, and operational disruptions due to unauthorized access to an administration interface.
- **Less risky vulnerabilities**: In our judgment, the remaining vulnerabilities are less risky, but should not be ignored. They could be exploited in certain scenarios, but their exploitation is less likely to cause immediate damage.

Recommendations

1. **Immediate Actions**: The vulnerabilities **H1, M2,** should be prioritized and addressed as soon as possible. As these represent the greatest risk according to the testers' assessment.
2. **Quick Wins**: The vulnerabilities **H1, M1, M3,** can be remedied with presumably little effort.
3. **Ongoing Security Improvement**: Regular security fixes, testing and reviews should be part of a continuous security improvement plan. Retesting key systems after remediation is highly recommended to ensure effectiveness.

Limitations and Scope of the Test

The penetration test was conducted within a defined scope, focusing on the web application Juice-Shop. The results are valid only for the period in which the test was conducted, and security postures can change over time. It is important to note that not all vulnerabilities may have been discovered, as the test is limited by the scope and the tight allocated timespan.

Conclusion

Overall, while the security level is generally regular, the identified vulnerabilities, especially the key vulnerabilities, require immediate attention to avoid potential exploitation. Regular penetration testing, along with prompt remediation of findings, is essential to maintaining a secure environment. By addressing the identified risks, the organization can significantly improve its defense against cyber threats and enhance the security of its digital assets.

2.1 Vulnerability Overview

In the course of this test unresolved vulnerabilities with the following criticality were identified: **1 High**, **3 Medium** and **1 Info** .

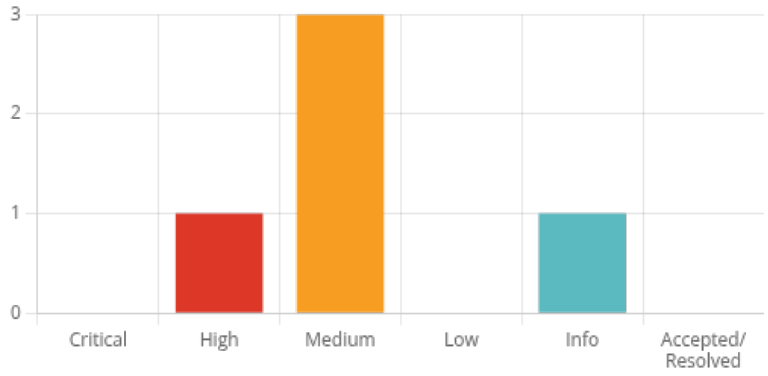

Figure 1 - Distribution of identified vulnerabilities

2.2 Identified Vulnerabilities

The table below provides an overview of the identified vulnerabilities.

#	CVSS	Description	IA	QW	State	Page
H1	8.6	Login Bypass via Error-based SQL Injection (SQLi)	‼	✳	Open	8
M1	6.1	Cross-Site Scripting (XSS) - Reflected		✳	Open	12
M2	5.3	Flaws in Discount Coupon Logic	‼		Open	16
M3	5.3	Directory Listing Enabled		✳	Open	21
I1	0.0	Information Disclosure via Stack Traces			Open	24

‼ IA = Immediate action. ✳ QW = Quick win.

3 General Conditions

In this section the general conditions of the entire engagement are documented.

3.1 Objective

The objective of this penetration test is identifying security weaknesses, misconfigurations, and potential exploitation paths which endanger confidentiality, integrity and availability of the web application **Juice-Shop**, while ensuring compliance with relevant security policies and best practices.

The goal is to cover as many vulnerabilities as possible within a given time frame while adhering to all rules of engagement agreed upon at the kick-off meeting to ensure minimal disruption to business operations.

3.2 Test Period

All activities were performed between **Tuesday, September 30, 2025** and **Thursday, October 2, 2025**. In toatl **3 person days** were used for this test.

3.3 Scope

The project scope defines exactly what is part of the test.

In Scope

The following applications are part of the scope

System	Description
https://juice-shop.lab	OWASP Juice-Shop (Testsystem)

Out of Scope

No subsections are explicitly excluded form the scope

3.4 User Accounts and Permissions

User accounts

No users were provided by the customer. However, there was a self-registration function. The following accounts were self-registered and used.

User	Role	Description
max.baeumler@schutzpunkt.com	Regular user	Self-registered

Coupon codes

The following coupon codes were also provided by the customer.

Code	Discount	Validity
q:<Irh7ZKp	10%	September 2025
pEw8ph7ZKu	10%	October 2025
pEw8ph7ZKu	15%	October 2025
pes[Ch7ZKp	10%	November 2025

3.5 IP Addresses

From these systems the attacks were performed.

System	Description
80.151.38.120	IP Address of our office in Deining

3.6 Test Basis and Approach

The test was conducted using the OWASP WSTG framework.

The following test approach was chosen:

- **Base of Information:** Grey-Box
- **Aggressiveness:** Deliberative
- **Coverage:** Limited
- **Approach:** Obviously
- **Access:** Network Connection
- **Origin:** External

3.7 Limitations

During the test no limitations occurred.

4 Findings

8.6 H1: Login Bypass via Error-based SQL Injection (SQLi)

Vector	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L*
State	Open
Immediate Action recommended	Yes ‼
Quick Win to Fix	Yes ✳
Tags	WSTG-INPV-05, CWE-89, ATT&CK-T1190
Affected Components	https://juice-shop.lab/rest/user/login

Summary

The application exposes database error information when processing untrusted input, enabling error-based SQL injection. Attackers can trigger SQL interpreter errors and use the returned error messages to bypass the login.

Impact

This allows any visitor to the website to log in as an administrator using this vulnerability.

Furthermore, disclosure of detailed database errors enables targeted inference of query structure and schema. Attackers can use error content to extract sensitive data, modify records, or bypass logical controls that depend on query outcomes. Persistent exposure of SQL errors increases the probability of full data compromise for affected tables and related entities.

Recommendation

Enforce safe handling of all user-controlled inputs and stop exposing database error details to end users.

Technical Description

Untrusted input is concatenated into SQL queries without parameterization, allowing malformed SQL to reach the database engine. When evaluation fails, the database emits detailed error messages that the application returns in its HTTP responses or UI, leaking internal query context. Error-based SQL injection leverages these disclosed errors to iteratively learn the structure of the underlying queries and schema. Disclosed content commonly includes SQLSTATE codes, driver names, exception class and stack traces, table or column identifiers, function names, and type mismatch details. These signals enable precision adjustments of input values, revealing columns, types, and constraints until meaningful data retrieval or modification becomes feasible. The root cause is string-based query construction and permissive error handling that surfaces low-level database exceptions to end users.

Secondary factors include inconsistent input validation and lack of enforced parameterized data access patterns across the codebase. The example below demonstrates a generic server response that reveals a database error without exposing environment-specific details.

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8

Error executing query on /items?id=<value>
SQLSTATE[42000]: Syntax error or access violation: TODO: actual DB error excerpt (e.g., column not found, type mismatch)
Driver: TODO: driver/version if disclosed | Query fragment: TODO: fragment if disclosed
```

Typical indicators of error-based SQL injection include database error codes in responses, leaked identifiers and data types, and traces linking directly to query construction paths. When present, these signals materially reduce uncertainty for an attacker, speeding the path to data compromise.

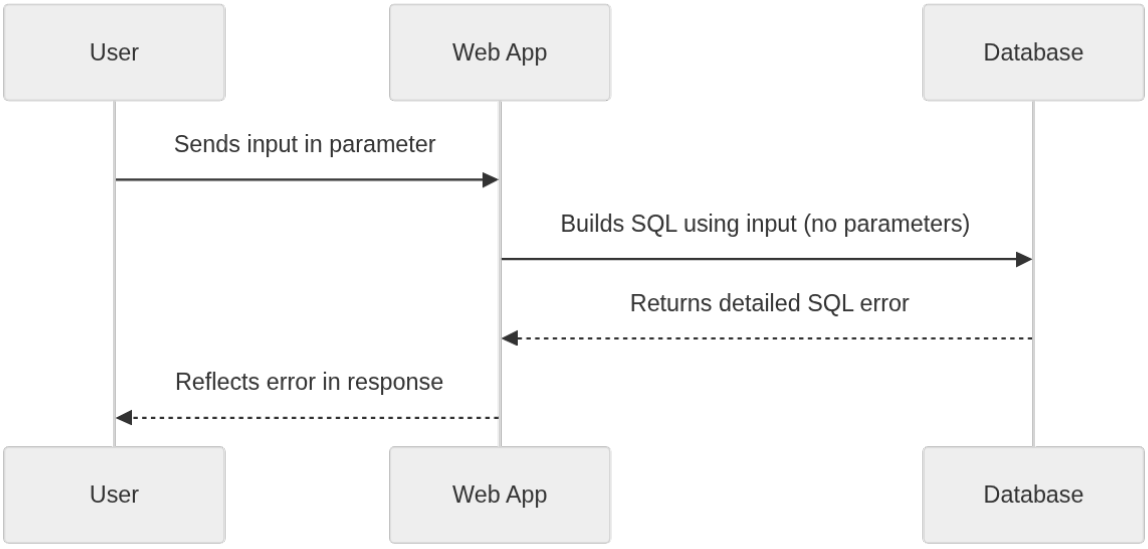

Figure 2 - Error-based SQLi signal flow

Evidence

In Figure 3, you can see that an SQL error is triggered if the email address contains a `'`.

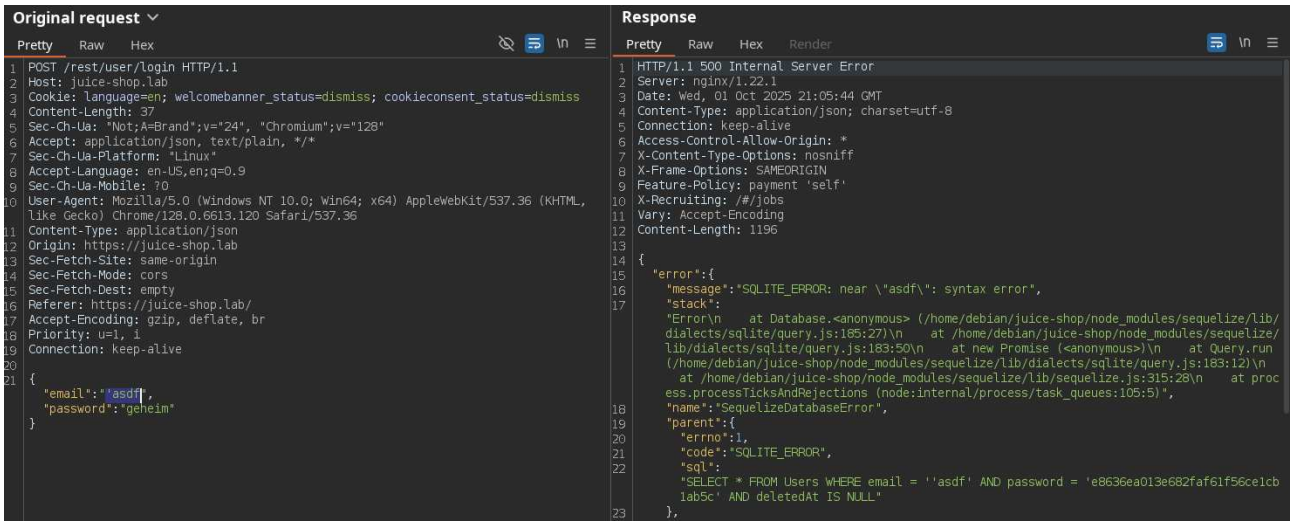

Figure 3 - SQL error in email

By cleverly choosing the username `' or 1 = 1; -- -`, an SQL injection occurs, allowing the login to be bypassed. Finally, you are logged in as an administrator (see Figure 3).

The following happens in the background when the query is assembled:

1. `'` closes the email address field.
2. Afterward the inserted SQL code is then used.
3. `or 1 = 1` is a true statement.
4. `;` terminates the SQL query.
5. The rest of the original statement is commented out with `--`.

```
SELECT * FROM Users WHERE email = '' or 1 = 1; -- -' AND password =
'e8636ea013e682faf61f56ce1cb1ab5c' AND deletedAt IS NULL
```

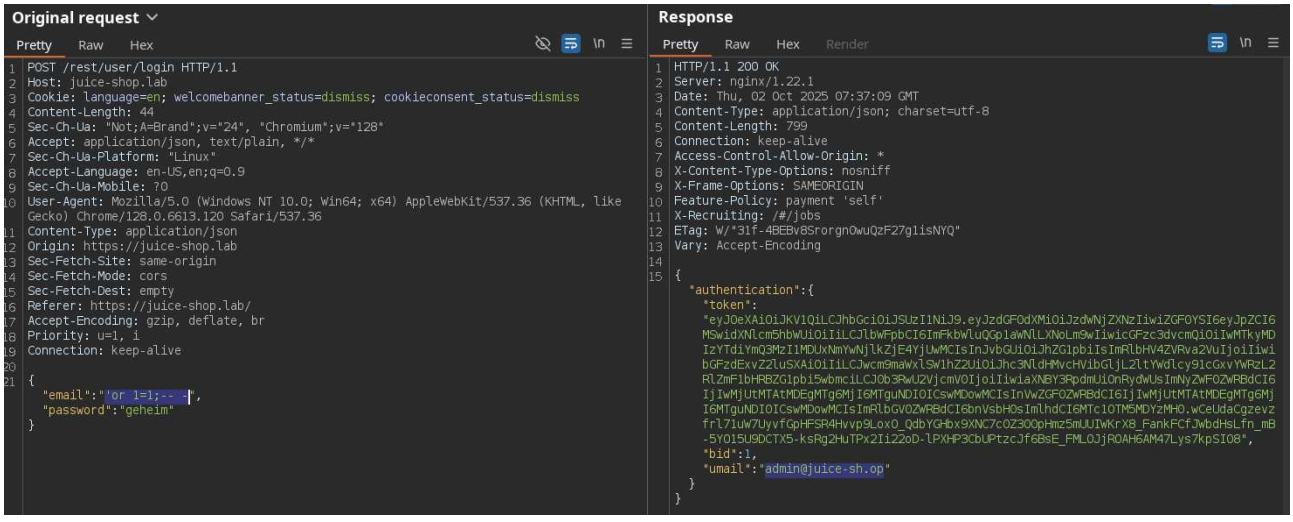

Figure 4 - SQL injection login bypass

This then allows successful access to the administration interface at *https://juice-shop.lab/#/administration* (see Figure 5). Here, for example, all other users can be viewed and reviews can be deleted.

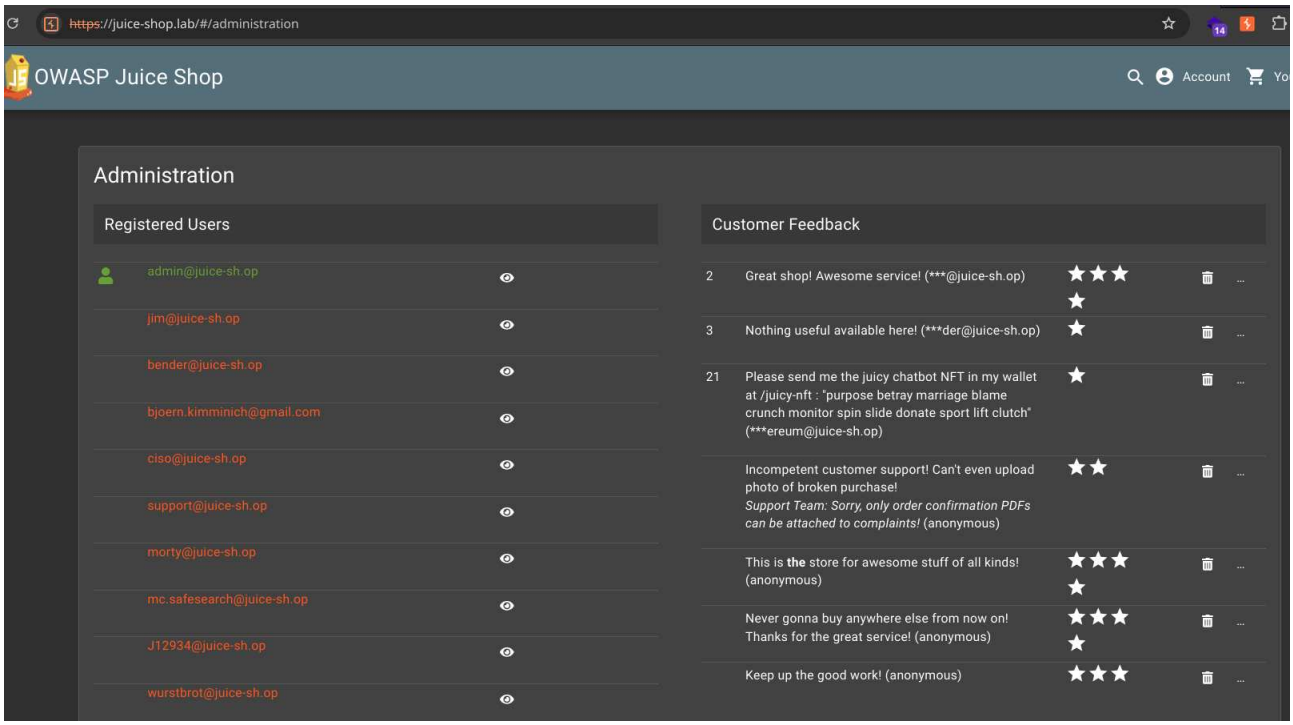

Figure 5 - Access to administration interface

Technical Recommendation

Replace string-concatenated SQL with prepared statements and bind variables for every query that uses user-controlled data. Enforce parameterization via a shared data access layer or ORM configuration and prohibit ad-hoc query construction in code reviews and CI checks. Implement generic error handling that maps database exceptions to standardized responses (e.g., HTTP 500) without revealing SQLSTATE, identifiers, or stack traces, and log full details server-side with correlation IDs. Disable verbose error pages and detailed exception messages in production configurations for the web framework and database drivers. Validate and normalize inputs according to strict type and length constraints before reaching the data layer, rejecting unexpected formats early. Inventory and refactor all endpoints that include user input in queries, adding tests to assert that errors are not reflected and that queries are parameterized. Add static analysis or lint rules to detect string-built SQL and enforce prepared statement usage across the codebase. Example (conceptual):

```
// Before: vulnerable
String sql = "SELECT * FROM items WHERE id = " + id_variable; // concatenation
stmt.executeQuery(sql);

// After: parameterized
PreparedStatement ps = conn.prepareStatement("SELECT * FROM items WHERE id = ?");
ps.setInt(1, id_variable);
ps.executeQuery();
```

References

- *https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection*
- *https://owasp.org/www-community/attacks/SQL_Injection*
- *https://cwe.mitre.org/data/definitions/89.html*
- *https://portswigger.net/web-security/sql-injection*

6.1 M1: Cross-Site Scripting (XSS) - Reflected

Vector	*CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N*
State	Open
Immediate Action recommended	No
Quick Win to Fix	Yes ✳
Tags	WSTG-CLNT-01, CWE-79
Affected Components	https://juice-shop.lab/#/search?q=

Summary

The application reflects user-supplied input into the response without proper context-aware output encoding. This enables an attacker to craft a link or request that executes attacker-controlled script in the victim's browser within the application. The issue is limited to transient responses and requires user interaction, but consequences can include account compromise, data exposure, and unauthorized actions performed via the victim's session.

Impact

Successful exploitation allows execution of attacker-controlled script in the victim's browser in the application. Potential outcomes include session misuse if tokens are accessible to script, unauthorized actions performed on behalf of the user, exposure of on-page sensitive data, and trusted UI manipulation that facilitates convincing in-origin phishing. The attack requires user interaction (for example, following a crafted link or submitting data), and impact can extend to any user who engages with the crafted input.

Recommendation

Apply strict context-aware output encoding for all reflected data and enforce a restrictive Content Security Policy if possible.

Technical Description

Reflected Cross-Site Scripting occurs when user-controlled data is immediately echoed back in a server response without appropriate output encoding for its rendering context. Typical sources include query parameters, path segments, form fields, or headers that are inserted into HTML, attributes, URLs, or inline scripts via string concatenation. The root cause is the absence or bypassing of context-aware output encoding and the use of unsafe rendering patterns that treat untrusted data as trusted markup or script. Framework auto-escaping may be disabled, misconfigured, or circumvented by unsafe templating constructs. Improper content handling, such as returning HTML with untrusted data while using a permissive response type, can further increase exposure, though the core flaw is the lack of correct encoding. A restrictive Content Security Policy (CSP) can limit impact but does not remediate the underlying issue. The example below shows a parameter reflected into HTML content without encoding, illustrating how untrusted input is rendered by the browser:

```
GET /search?q=<untrusted_input> HTTP/1.1
Host: TODO: host
User-Agent: TODO
Accept: text/html
```

```
<!doctype html>
<html>
  <body>
    <p>Search for: <untrusted_input></p>
  </body>
</html>
```

Evidence

- Affected endpoint and parameter:
  - URL: *https://juice-shop.lab/#/search*
  - Parameter: *q*

In Figure 6, you can see that the input `asdf` in the search field is mirrored in the Search Results context of the web application.

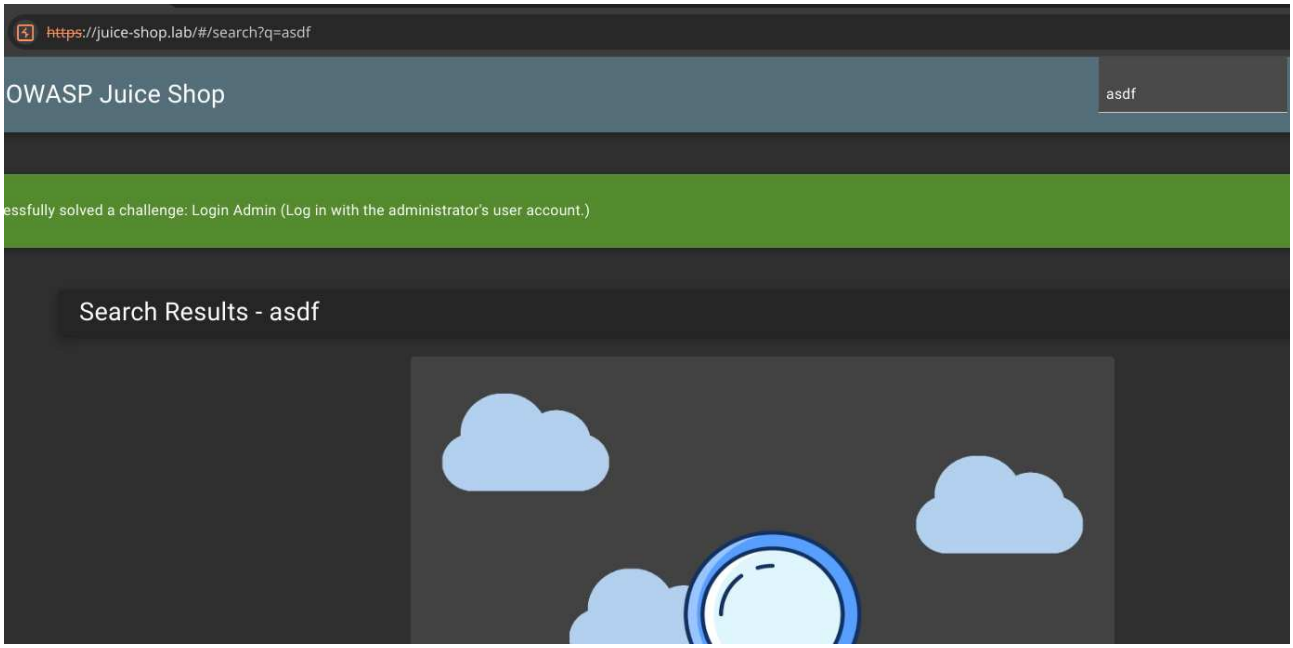

Figure 6 - Reflected input from search field

Entering the following JavaScript code displays a pop-up message confirming the execution of this code (see Figure 7).

```
<iframe%20src%3D“javascript:alert(‘Protection point%20XSS’)”>
```

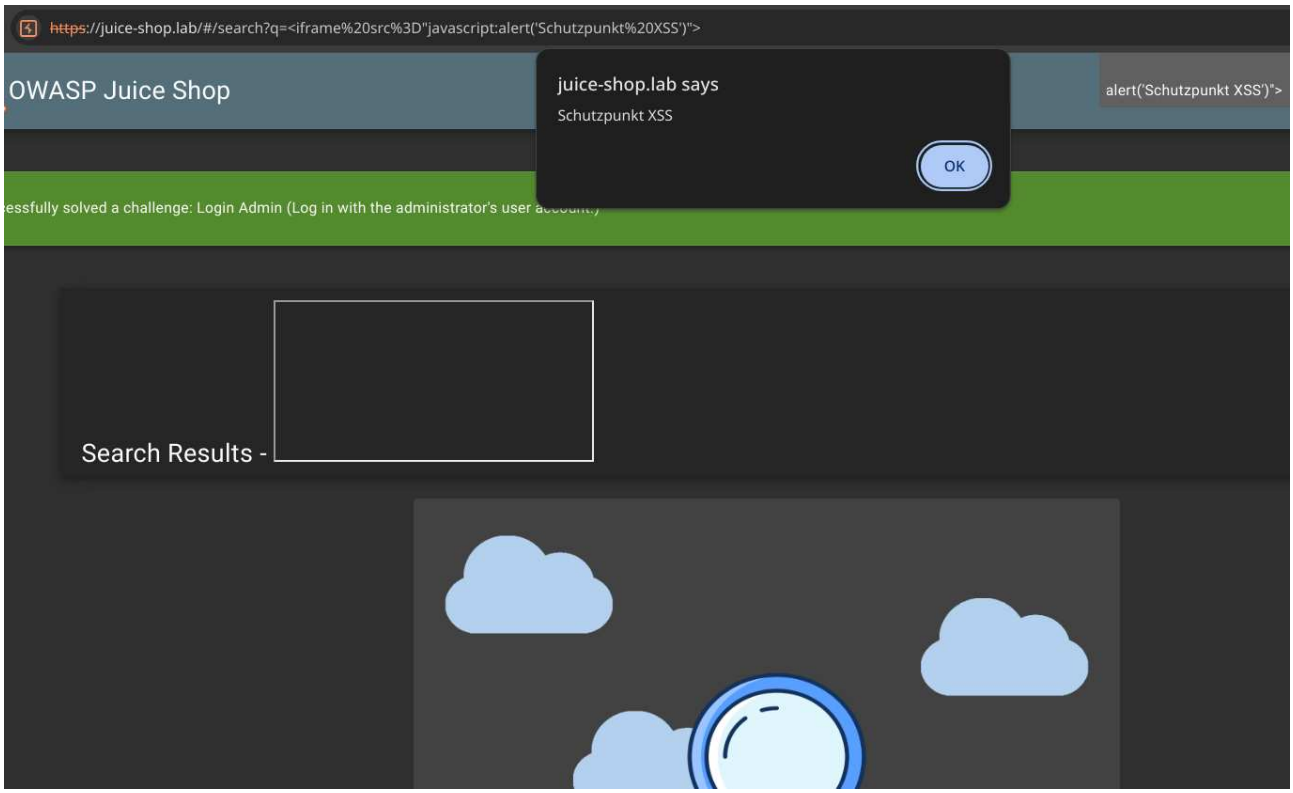

Figure 7 - Reflected XSS

Figure 8 shows the affected location in the frontend code of the application.

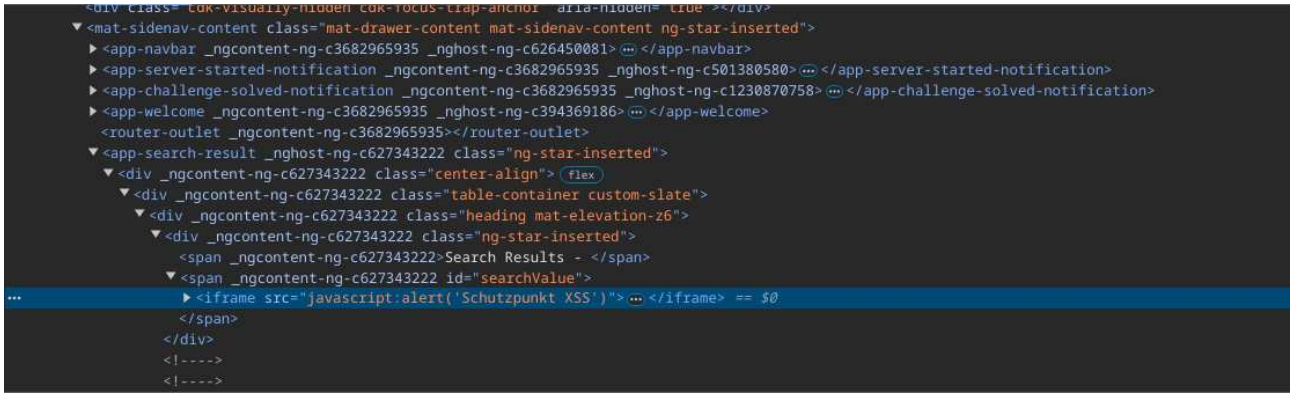

Figure 8 - Location in the code with XSS

Technical Recommendation

Identify every location where untrusted input is reflected into responses and ensure context-aware output encoding is applied before rendering. Use framework features that auto-escape by default, and avoid concatenating untrusted data into HTML, attributes, URLs, CSS, or JavaScript contexts. Apply the correct encoder for the specific context, such as HTML text, HTML attribute, URL parameter, or JavaScript string, instead of relying on generic or blacklist filters. Avoid rendering untrusted input as HTML; use safe templating and DOM APIs that handle text nodes rather than markup. Deploy a restrictive CSP that disallows inline script and limits script sources to vetted origins to reduce impact if an encoding gap remains. Add unit and integration tests that verify reflected fields are encoded and that dangerous characters are rendered harmless in their specific contexts.

Example (illustrative):

```
// Template or server-side rendering
String safe = org.owasp.encoder.Encode.forHtml(userInput);
out.print(safe);
```

References

- *https://owasp.org/www-community/attacks/xss/*
- *https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html*
- *https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting*
- *https://cwe.mitre.org/data/definitions/79.html*

5.3 M2: Flaws in Discount Coupon Logic

Vector	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N*
State	Open
Immediate Action recommended	Yes ‼️
Quick Win to Fix	No or to less Information
Tags	WSTG-BUSL-01, WSTG-BUSL-02, CWE-840, CWE-345, ATT&CK-T1190
Affected Components	https://juice-shop.lab/rest/basket/6/coupon/

Summary

The application's discount coupon logic allows unintended price reductions due to insufficient server-side validation and weak enforcement of redemption rules. Attackers can apply or reuse coupons beyond intended price constraints. The weakness is a design flaw in business rules and transactional enforcement, not a cosmetic issue.

Impact

Unauthorized discount application can lead to direct revenue loss, margin erosion, and distorted financial reporting. Inventory may be depleted at unintended price points, and promotional budgets and campaign analytics become unreliable. Abuse can cascade into fraud patterns such as arbitrage, resale, or account farming.

Preconditions

Received or collected one or more discount vouchers in order to derive a pattern.

Recommendation

Implement a new voucher concept or at least validate voucher rules comprehensively on the server side.

Technical Description

The discount system does not strictly validate and enforce business rules on the server, which enables misuse of coupon benefits. Typical gaps include missing checks for single-use limits, lack of binding a coupon to a specific account or order, absence of minimum threshold validation, improper handling of stacking rules, and failure to block negative or near-zero totals.

If coupon state is tracked client-side or inferred from mutable parameters, attackers can tamper with values or replay requests to achieve repeated discounts. Weak input validation may also permit unintended combinations.

Evidence

The customer provided the following coupon codes.

Code	Discount	Validity
q:<Irh7ZKp	10%	September 2025
pEw8ph7ZKp	10%	October 2025
pEw8ph7ZKu	15%	October 2025
pes[Ch7ZKp	10%	November 2025

Decoding

It is noticeable that these are similar in terms of the date and the percentage. The part `h7ZK` always seems to remain the same, which suggests that the coupons are encoded. This turned out to be Base85 encoding, as can be seen in Figure 9, and is structured as follows: `MMMYY-%%`, where `%` stands for the discount in percent.

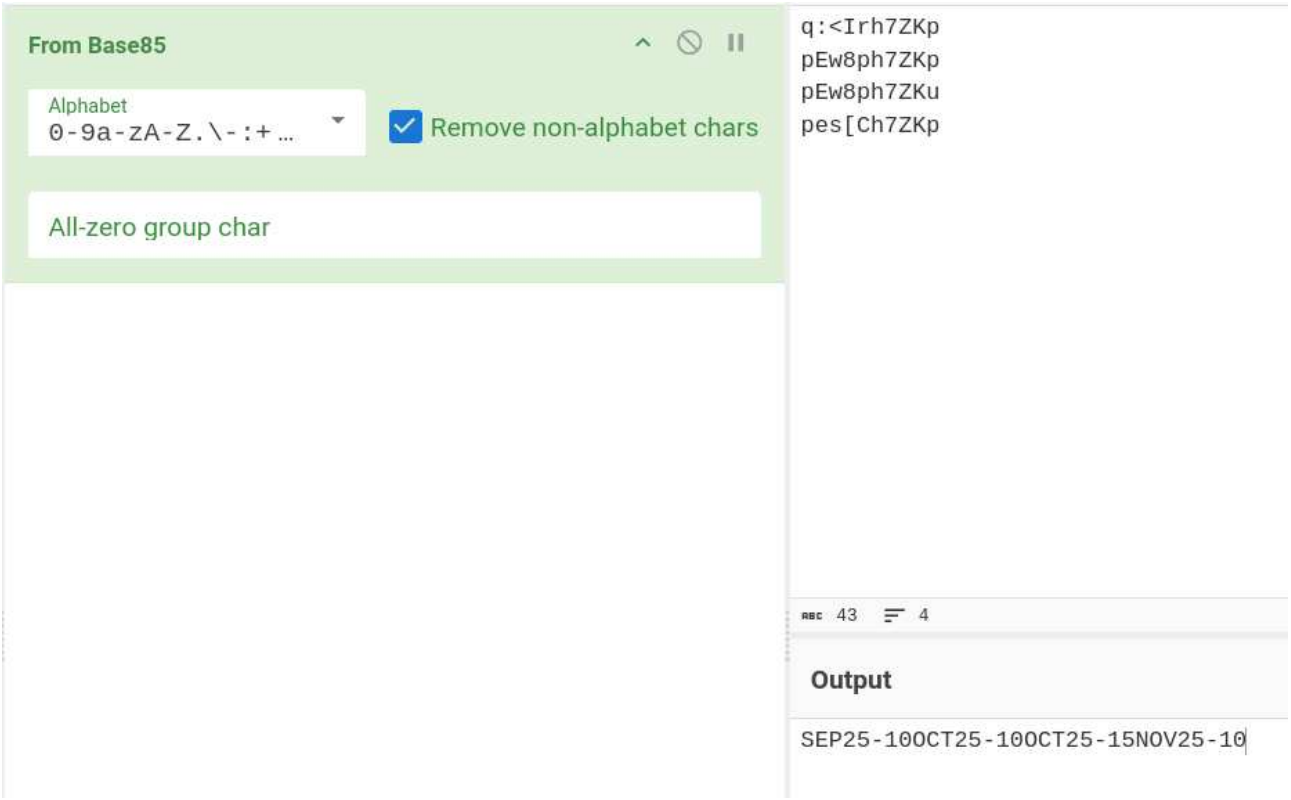

Figure 9 - Base85 decoded coupons

Creating your own coupon

A new coupon `OCT25-99` was then created (Figure 10). The value of this coupon is `pEw8ph7Z*G`

Figure 10 - Self-created coupon

Redeeming the coupon

The following request-response pair shows the attempt to redeem this self-generated coupon. The checkout overview shows that this was successful (see Figure 11).

Request

```
PUT /rest/basket/6/coupon/pEw8ph7Z*G HTTP/1.1
Host: juice-shop.lab
[...SNIP...]
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Linux"
Origin: https://juice-shop.lab
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.lab/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
```

```
Content-Length: 2

{}
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Thu, 02 Oct 2025 13:00:07 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 15
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
ETag: W/"f-EKshmF+cUf70Vv3BHGSC98QSEKM"
Vary: Accept-Encoding

{"discount":99}
```

Checkout-Page

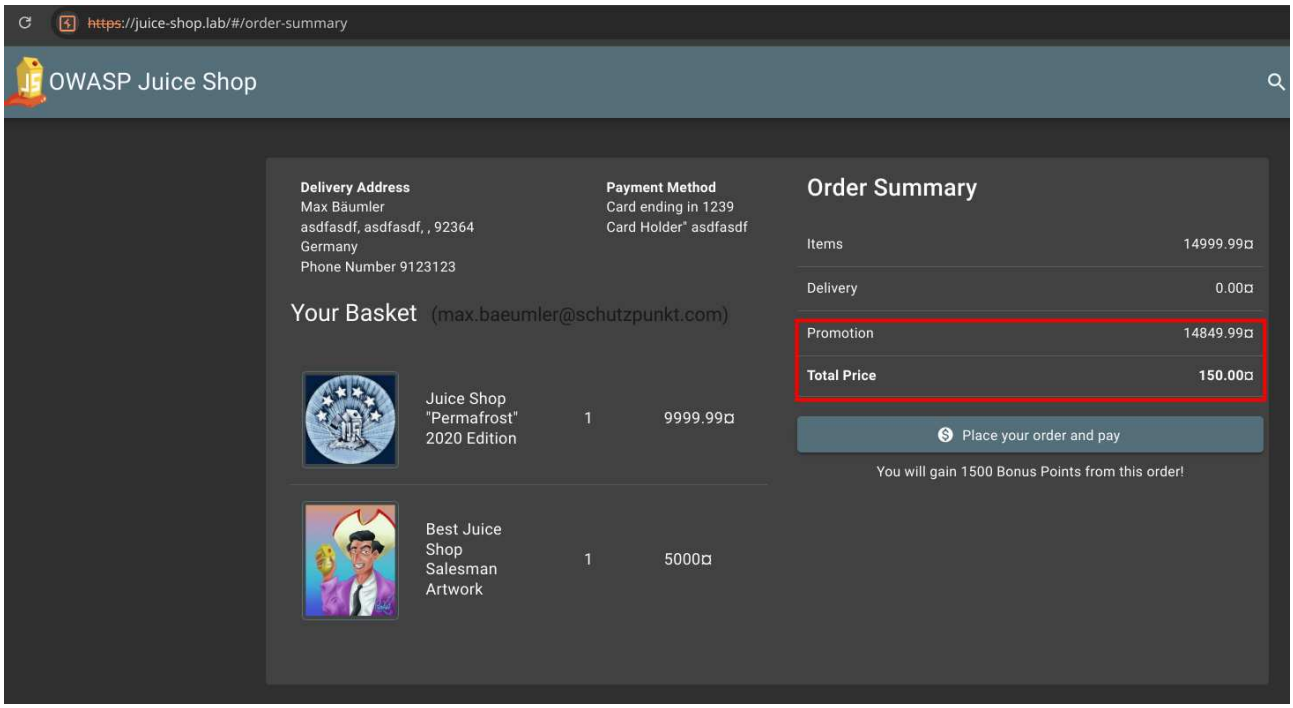

Figure 11 - Gained Discount of 99 %

Technical Recommendation

Verify on server side whether a coupon is actually permitted. For November, for example, check whether the coupon is actually exactly `NOV25-10` and do not calculate the discount from the text. However, it is recommended that the voucher logic be reimplemented, as the same problem can arise repeatedly due to carelessness. For example, if you want to grant a customer a voucher with a larger discount, this is currently not possible, but instead allows all users to obtain this discount fraudulently.

Validate and enforce all coupon constraints on the server before applying any price change, including eligibility (user, segment, geography), validity window, minimum order thresholds, item/category scope, stacking limits, and maximum discount. Bind coupon instances to a unique subject, such as user ID or order ID, and persist state with atomic transactions to ensure single-use or limited-use enforcement. Log all coupon lifecycle events with correlation IDs and monitor for anomalies such as unusually high redemption rates or rapid sequential uses. Add unit and integration tests for rule evaluation, idempotency, and concurrency to prevent regressions.

References

- *https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Business_Logic_Testing/01-Testing_for_business_logic*
- *https://cheatsheetseries.owasp.org/cheatsheets/Business_Logic_Security_Cheat_Sheet.html*
- *https://portswigger.net/web-security/logic-flaws*
- *https://owasp.org/Top10/A04_2021-Insecure_Design/*

5.3 M3: Directory Listing Enabled

Vector	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
State	Open
Immediate Action recommended	No
Quick Win to Fix	Yes ✳
Tags	CWE-548, CWE-200
Affected Components	https://juice-shop.lab/ftp

Summary

The web server is configured to allow directory listing on one or more public paths. This exposes the names and structure of files and subdirectories to any visitor, even without authentication. File listings increase information disclosure and make reconnaissance and targeted attacks easier.

Impact

Unrestricted directory browsing can reveal sensitive files, configuration remnants, backups, and credentials stored within web-accessible paths. Exposed filenames and structure enable faster reconnaissance, targeted brute-force against discovered assets, and discovery of overlooked administrative endpoints. Information leakage can aid subsequent attacks such as credential harvesting from configuration files, exploitation of outdated components identified in listings, and unauthorized download of proprietary content. If search engines index these listings, exposure persists beyond the immediate audience and increases long-term risk.

Recommendation

Disable directory listing on all publicly accessible web paths.

Technical Description

Directory listing occurs when the server returns an automatically generated index page for a directory that lacks a default index file such as index.html. It typically results from enabling features like Apache mod_autoindex, Nginx autoindex, or IIS Directory Browsing, or from default configurations that are not hardened. When enabled, a request to a directory path (for example, /assets/) yields a page listing files, sizes, timestamps, and subdirectories. These listings reveal internal file naming, build artifacts, backups, temporary files, and configuration fragments that were not intended to be publicly accessible. Attackers and automated crawlers can iterate directories to map the application structure and locate sensitive files faster. Publicly reachable listings require no authentication and may be cached or indexed by search engines, increasing unintended exposure. This behavior is often accidental and persists until directory indexing is explicitly disabled or a default index file is provided.

Example request and response demonstrating an enabled listing.

```
GET /assets/ HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html>
  <head><title>Index of /assets/</title></head>
  <body>
    <h1>Index of /assets/</h1>
    <a href="backup.zip">backup.zip</a>
    <a href=".env">.env</a>
    <a href="config.old">config.old</a>
  </body>
</html>
```

Evidence

- Observed URLs with directory listing enabled:
  - *https://juice-shop.lab/ftp*

As can be seen in Figure 12, several sensitive files can also be read under the above path, such as:

- incident-support.kdbx
- suspicious_errors.yml
- coupons_2013.md.bak

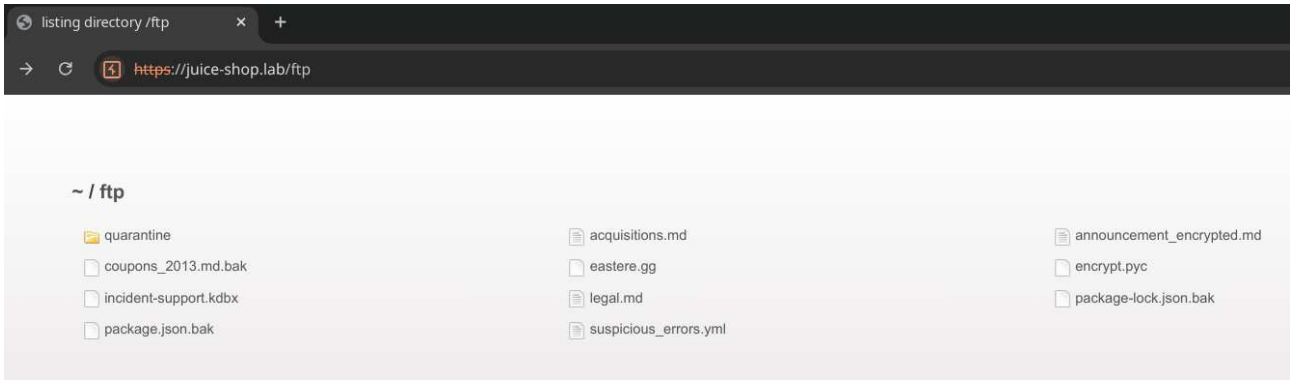

Figure 12 - Directory Listing Enabled

Technical Recommendation

Turn off directory indexing at the web server level for all public sites and applications, and enforce least exposure for any path that must remain accessible. In Apache HTTP Server, disable indexes globally or per directory by removing autoindex and using a restrictive Options directive.

```
# httpd.conf or .htaccess
Options -Indexes
# Ensure mod_autoindex is not loaded if not needed
# LoadModule autoindex_module modules/mod_autoindex.so  # comment or remove
```

In Nginx, disable autoindex for relevant locations or server blocks.

```
server {
  location / {
    autoindex off;
```

```
  }
}
```

In IIS, disable Directory Browsing at the site or application level, or enforce it via web.config.

```
<!-- web.config -->
<configuration>
  <system.webServer>
    <directoryBrowse enabled="false" />
  </system.webServer>
</configuration>
```

If a listing is required for a legitimate use case, restrict it to authenticated, role-limited users and isolate it from public routes. Validate that no sensitive files reside within web-accessible directories and deploy a default index page where appropriate to prevent auto-generated listings.

References

- *https://cwe.mitre.org/data/definitions/548.html*
- *https://owasp.org/Top10/A05_2021-Security_Misconfiguration*
- *https://httpd.apache.org/docs/current/en/mod/mod_autoindex.html*
- *https://nginx.org/en/docs/http/ngx_http_autoindex_module.html*

0.0 I1: Information Disclosure via Stack Traces

Vector	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N*
State	Open
Immediate Action recommended	No
Quick Win to Fix	No or to less Information
Tags	WSTG-ERRH-02, CWE-209
Affected Components	https://juice-shop.lab/rest/<anything>

Summary

The application exposes detailed stack traces to end users when errors occur. These traces reveal internal code paths, framework and library versions, configuration details, and file system locations. Such information enables efficient reconnaissance and can be used to craft targeted attacks. The behavior indicates improper error handling and debug or verbose settings in production.

Impact

Attackers can enumerate frameworks, versions, and libraries from stack trace content, enabling version-specific exploit selection. File paths and class names reveal application structure, aiding targeted probing of components and error-prone interfaces. Detailed exception messages may disclose configuration values and operational context, raising the likelihood of data exposure via secondary flaws.

Recommendation

Disable stack trace display in production and return generic error messages while logging detailed diagnostics server-side.

Technical Description

This finding occurs when unhandled exceptions or misconfigured error handlers return raw stack traces in HTTP responses or UI pages. Verbose error output is often enabled by default in development or debug modes and mistakenly left active in production. Frameworks commonly include default error pages that echo exception messages, class names, file paths, line numbers, and dependency versions. Returning these details to clients provides attackers with accurate insight into the internal architecture, technologies in use, and potential weak points. Causes include missing global exception handling, absent environment gating for debug flags, direct serialization of exceptions, and insufficient API error normalization. Typical signals include 500 responses with multiline traces, error pages that include framework branding and version numbers, and directory or file path disclosures. The issue is content leakage, not availability or authorization failure, but it directly improves the attacker’s ability to exploit other flaws. Preventing disclosure requires routing all errors through standardized handlers that emit generic messages while logging specifics server-side. The following illustrates a representative leaked response.

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8

java.lang.NullPointerException: Cannot read property 'id' of undefined
    at com.example.controllers.UserController.getUser(UserController.java:42)
    at
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1012)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:661)
/opt/app/services/user-service/src/main/java/com/example/controllers/UserController.java:42
Framework: Spring Boot 2.6.3
```

Evidence

If you call one of the REST API's unknown endpoints here *asdf* at the URL *https://juice-shop.lab/rest/*, you will receive a stack trace of the application (see Figure 13). This contains the following information:

- Version number + technology: *Express.js 4.21.0*
- Technology:*Angular.js*
- Path: `/home/debian/juice-shop`

Figure 13 - Stack trace at unknown REST endpoint

Technical Recommendation

Implement centralized exception handling that maps all unhandled errors to generic responses without stack or path details. Ensure environment gating disables debug or developer exception pages in production, e.g., `debug=false` and equivalent settings across all services. Standardize API error formats to a minimal schema, such as an opaque error code and correlation ID, and avoid echoing exception messages. Log full diagnostics, including stack traces, to server-side sinks with appropriate access controls and retention, not to client responses. Review web server and application framework configuration to ensure no default verbose error pages are exposed externally. Validate behavior with automated tests that assert absence of stack traces and internal paths in all error scenarios.

```
// Centralized error handling example (generic response)
app.use(function errorHandler(err, req, res, next) {
  const correlationId = generateId();
  logError({ correlationId, err }); // server-side log includes err.stack
  // Do not return err.stack or internal paths to client
  res.status(500).json({ error: "An unexpected error occurred.", correlationId });
});
```

References

- *https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Error_Handling/02-Testing_for_Stack_Trace*
- *https://cwe.mitre.org/data/definitions/209.html*
- *https://owasp.org/Top10/A05_2021-Security_Misconfiguration/*
- *https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html*

5 Disclaimer

This engagement has been conducted with the objective of identifying potential security vulnerabilities and providing actionable recommendations. However, it is important to note the following:

- **No Guarantee of Completeness:** While the test is designed to identify vulnerabilities within the scope, there is no guarantee that all vulnerabilities, threats, or risks have been discovered. The results should not be considered exhaustive, and new vulnerabilities may arise over time.
- **Time-and-Material Approach:** The engagement follows a time-and-material approach, where testing efforts are billed based on the amount of time spent, resources used, and the complexity of tasks performed. As such, the results should be viewed in the context of the testing period and resources allocated.
- **Temporary Validity of Results:** The findings and vulnerabilities identified are valid only for the period during which the test was conducted. Security postures can change quickly, and new vulnerabilities may arise after the test is completed.
- **Retesting and Continuous Improvement:** Retesting is always encouraged as it can uncover additional vulnerabilities that may not have been detected in the initial assessment or occur in the future. Security is an ongoing process, and frequent testing is vital for maintaining a strong security posture.
- **Protection Systems May Impact Results:** Active protection systems (such as Intrusion Detection Systems, Intrusion Prevention Systems, Firewalls, etc.) may impact the test results. These systems can interfere with the testing process, potentially leading to incomplete or misleading findings.
- **False Positives:** During the assessment, false positives may occur—cases where vulnerabilities are identified but later determined not to be exploitable or not present. In our practice, we share all information gathered, including potential false positives, as they might still provide useful insights. Prominent examples are findings based on the reported version number that got backports of security fixes or applications that that use vulnerable libraries but not the vulnerable components of them.

The results and recommendations provided are based on the understanding and scope of the testing, and it is advised that they be used as part of a broader, continuous security improvement process.

List of Figures

A Appendix

A.1 Additional files related to the report

The following files are provided separately with the report

- Processed results of the port scanner *nmap*
  - juice-shop.lab.html
  - juice-shop.lab.nmap
  - juice-shop.lab.gnmap
  - juice-shop.lab.xml
- Results of the vulnerability scanner *nuclei*
  - nuclie-juice-shop.json