Application Name:	OWASP Juice Shop
Version Number:	19.2.1
Assessment Type	Dynamic Application Security Assessment
Report Published Date:	02/05/2026
Scheduling ID	

1 INTRODUCTION AND BACKGROUND 2

2 OBJECTIVE 2

3 APPROACH 2

4 EXECUTIVE REPORT 3

5 TECHNICAL TESTING RESULTS 5

1. SQL Injection on Authentication Endpoint 5
2. *Full Database Extraction via SQLMap* 6
3. *Reflected / DOM-Based Cross-Site Scripting (XSS) in Search Input* 8
4. *Sensitive File Exposure on Web Server* 9
5. *Insecure Direct Object Reference (IDOR) — Arbitrary Wallet Deposit Manipulation* 11
6. *IDOR — Unauthorized Access and Modification of Other Users' Shopping Baskets* 12
7. *Passwords Stored Without Encryption / Cryptographic Hashing* 14
8. *Weak MD5 Password Hashing — Cryptographically Broken Algorithm* 15
9. *Unauthenticated / Unprotected Admin and Accounting Routes* 17
10. *Sensitive Internal Paths Exposed via main.js Bundle* 18
11. *Sensitive Discount Coupon Code Leaked via Chatbot* 19
12. *Missing X-Frame-Options / Clickjacking Vulnerability* 20

6 NEXT STEPS 23

7 VULNERABILITY REMEDIATION TIMELINE 23

8 CONTACTS AND DEMOGRAPHICS 24

Assessment Contacts 24

9 REPORT PROPERTIES 24

CLASSIFICATION 24

1 Introduction and Background

Application security assessment was performed on the OWASP Juice Shop application. The assessment period was from 28 Apr 2026 to 04 May 2026.

The OWASP Juice Shop is a publicly accessible demo online banking and e-commerce platform intentionally designed for security testing, training, and education. The application simulates real-world functionalities including user authentication, product ordering, wallet management, coupon redemption, and customer interactions. The system processes user-related information such as account credentials, order details, wallet balances, and session data.

NOTE: Due to the unavailability of a dedicated live environment and in adherence to security and ethical testing principles, all assessment activities — including payload delivery, route enumeration, and exploitation verification — were conducted exclusively against a locally hosted instance of OWASP Juice Shop. All findings, payloads, affected URLs, and evidence documented in this report are derived from the controlled local test environment and reflect the security posture of the equivalent production application architecture.

2 Objective

- Verify the implementation of security controls within the application in alignment with OWASP Top 10 (2021) and industry best practices.
- Identify, prioritize, and document security vulnerabilities discovered during dynamic testing.
- Provide actionable technical remediation recommendations to improve the overall security posture of the application.

3 Approach

- Performed application reconnaissance and mapped available endpoints and functionalities using browser DevTools and main.js bundle analysis.
- Conducted Dynamic Application Security Testing (DAST) against OWASP Juice Shop hosted in a local test environment (https://juice-shop.herokuapp.com — local instance).
- Utilized Burp Suite Community Edition for intercepting, modifying, replaying, and analyzing HTTP requests and responses.
- Performed manual vulnerability testing aligned with OWASP Top 10 (2021) categories — including Injection, Broken Access Control, Cryptographic Failures, Insecure Design, and Security Misconfiguration.
- Tested for SQL Injection, XSS, IDOR, CSRF, Security Header gaps, Business Logic Flaws, Session Management issues, and Cryptographic weaknesses.
- Analyzed findings based on exploitability, severity, and potential impact to the application and its users.
- Documented all vulnerabilities with technical description, proof-of-concept payloads, confirmed impact, and prioritized remediation recommendations.

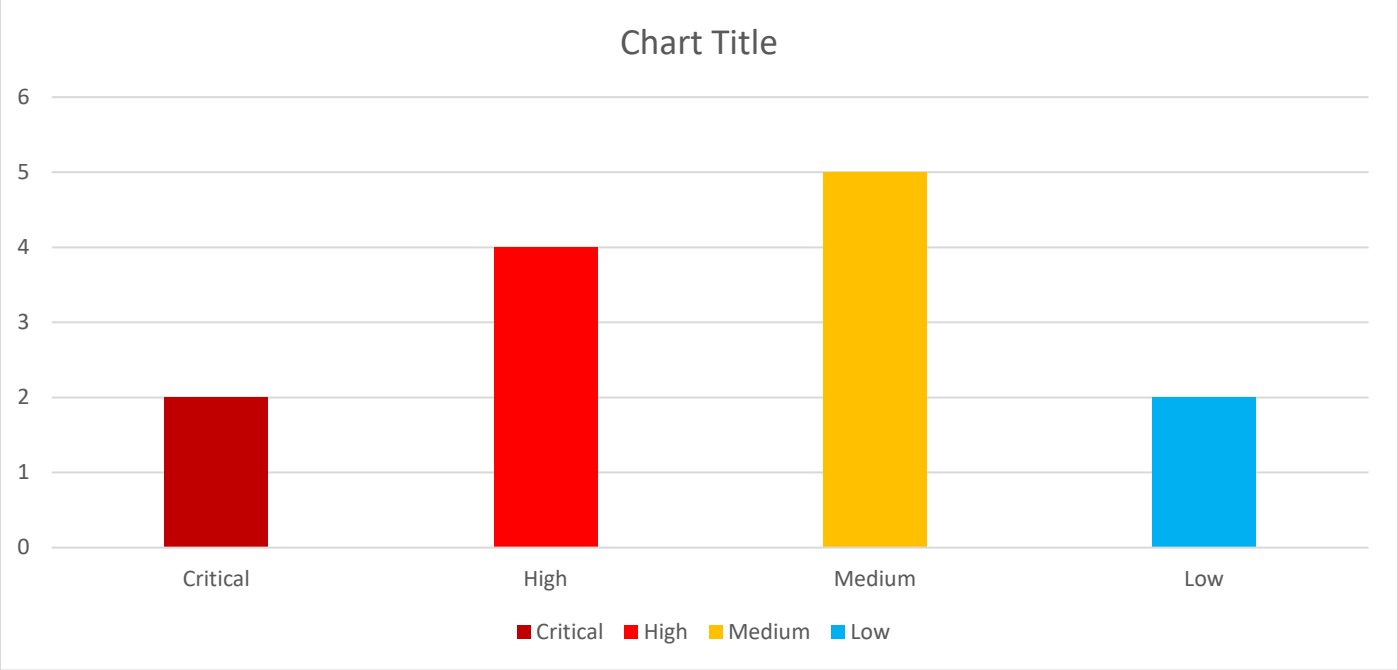
Constraints:

- Testing was performed exclusively against a local/offline instance of OWASP Juice Shop — no production or live customer-facing systems were accessed.
- Testing was limited to Burp Suite Community Edition capabilities — no automated authenticated scanning tools (Burp Pro, WebInspect) were used.
- All exploit payloads and route enumeration reflected in this report were validated solely in the controlled test environment.

4 Executive Report

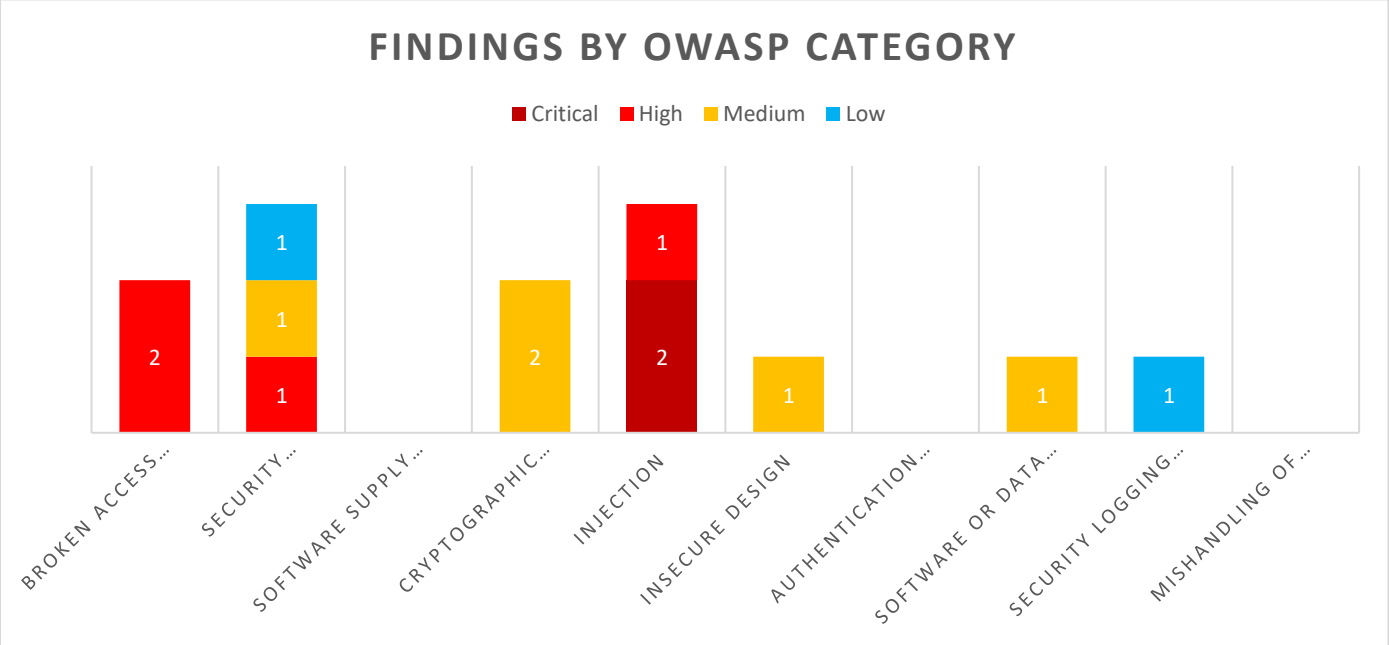
Executive summary of vulnerabilities identified during Dynamic Application Assessment of **OWASP JUICE SHOP**

Findings by Vulnerability Severity



Absence of a bar chart represents there are no vulnerabilities for that severity level

Findings by OWASP Category



List of Vulnerabilities

The full list of findings can be found in the Technical Testing Results section below.

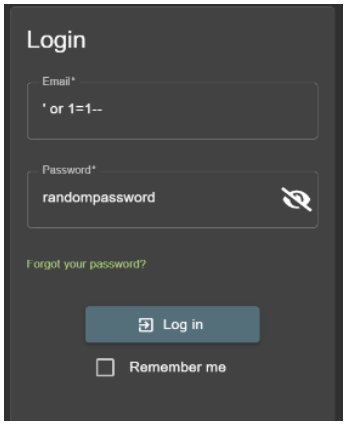
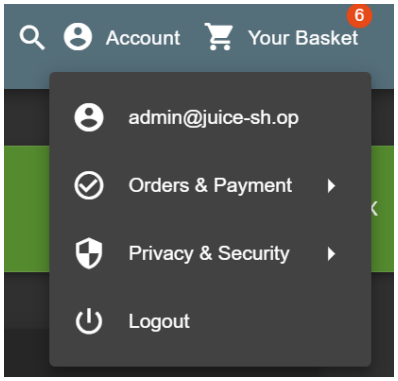
Sl. No.	eGRC Finding ID	Finding	Severity	# instances identified	Finding status	Target date for remediation
1		Injection	Critical	Web App		
2		Injection	Critical	Web App		
3		Injection	High	Web App		
4		Security Misconfiguration	High	Web App		

5		Broken Access Control	High	Web App		
6		Broken Access Control	High	Web App		
7		Cryptographic Failures	Medium	Web App		
8		Cryptographic Failures	Medium	Web App		
9		Security Misconfiguration	Medium	Web App		
10		Software & Data Integrity Failures	Medium	Web App		
11		Insecure Design	Medium	Web App		
12		Security Misconfiguration	Low	Web App		
13		Security Logging & Monitoring Failures	Low	Web App		

5 Technical Testing Results

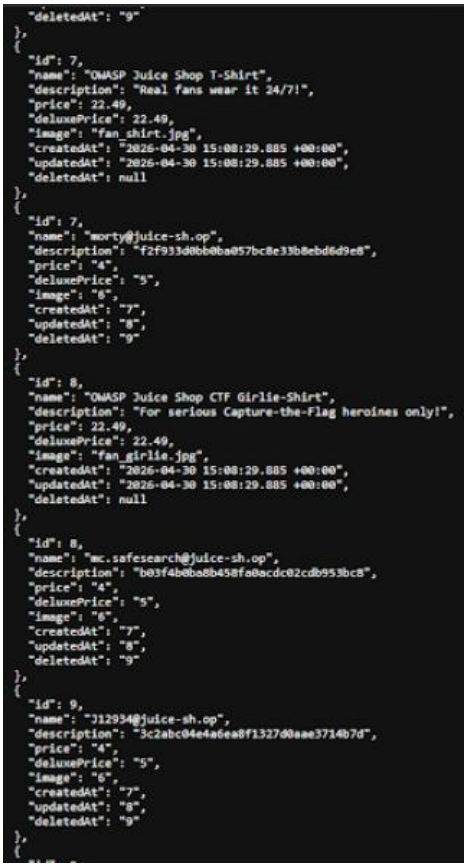
1. SQL Injection on Authentication Endpoint

Risk Rating	Critical
Category	Injection
Description	• The login endpoint fails to use parameterized queries or prepared statements, allowing attacker-controlled input to be interpreted as SQL syntax by the backend database engine. • Classic bypass payloads such as ' OR 1=1-- can be injected into the username or password field to manipulate the WHERE clause of the authentication query. • The application returns verbose SQL error messages or differential HTTP responses, confirming injection context and enabling blind/error-based enumeration. • Authentication logic relies entirely on database query result evaluation, meaning a manipulated true-result query grants access without valid credentials. • No WAF, input sanitization, or prepared statement layer intercepts the malicious payload before it reaches the database engine.
Impact	• Complete authentication bypass — attacker can log in as any user including admin without knowing credentials. • If stacked queries are supported, attacker can execute DDL/DML commands — DROP, INSERT, UPDATE on any table. • Full database enumeration possible — schema, tables, user credentials, PII exfiltration via UNION-based or blind injection. • Privilege escalation to admin account leads to full application compromise.
Recommendation	1. Enforce parameterized queries / prepared statements across all SQL-interacting code paths — never concatenate user input into query strings. 2. Use an ORM (e.g., Sequelize, TypeORM) with query binding to eliminate raw query construction. 3. Implement server-side input validation — whitelist allowed character sets for username/password fields. 4. Suppress SQL error messages in production; return generic HTTP 401 responses only. 5. Deploy a WAF rule set (e.g., ModSecurity CRS) to detect and block known SQLi patterns at the perimeter.
Finding Id	
Target Date	
Affected URLs	

Evidences	 
Management response	djaGDJGjfakj

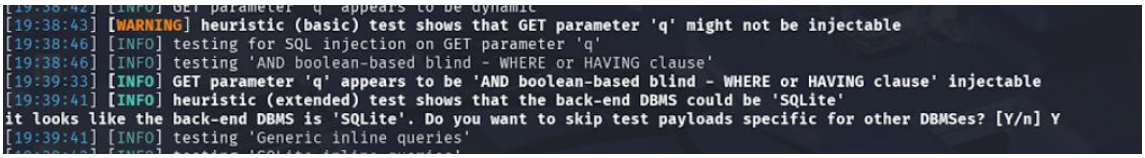
2. Full Database Extraction via SQLMap

Risk Rating	Critical
Category	Injection
Description	• The injectable authentication endpoint is exploitable by automated tooling (SQLMap) capable of enumerating the entire database schema and extracting all table contents without manual payload crafting. • SQLMap's automated detection of injection type (error-based, time-based blind, UNION-based) allows efficient extraction even against partially hardened endpoints. • The database stores plaintext or weakly hashed credentials (noted in Finding #10), meaning extracted dumps immediately yield usable account credentials for lateral movement. • No rate limiting, account lockout, or anomaly detection on the login endpoint was observed to deter automated exploitation attempts. • The combined effect of SQLi + plaintext passwords results in complete confidentiality breach of all stored user and application data.
Impact	• Complete exfiltration of all user records — PII, email addresses, order history, payment references. • Credential stuffing attacks possible using extracted email-password pairs across other platforms. • Business logic data (orders, coupons, pricing) exposed, enabling fraud. • Regulatory breach — GDPR/PCI-DSS violations if payment or personal data is compromised.
Recommendation	1. Complete exfiltration of all user records — PII, email addresses, order history, payment references. 2. Credential stuffing attacks possible using extracted email-password pairs across other platforms. 3. Business logic data (orders, coupons, pricing) exposed, enabling fraud. 4. Regulatory breach — GDPR/PCI-DSS violations if payment or personal data is compromised.

Finding Id	
Target Date	
Affected URLs	
Evidences	


```
  "deletedAt": "9"
},
{
  "id": 7,
  "name": "OWASP Juice Shop T-Shirt",
  "description": "Real fans wear it 24/7!",
  "price": 22.49,
  "deluxePrice": 22.49,
  "image": "fan_shirt.jpg",
  "createdAt": "2026-04-30 15:08:29.885 +00:00",
  "updatedAt": "2026-04-30 15:08:29.885 +00:00",
  "deletedAt": null
},
{
  "id": 7,
  "name": "morty@juice-sh.op",
  "description": "f2f933d0bb0ba057bc8e33b8ebd6d9e8",
  "price": "4",
  "deluxePrice": "5",
  "image": "6",
  "createdAt": "7",
  "updatedAt": "8",
  "deletedAt": "9"
},
{
  "id": 8,
  "name": "OWASP Juice Shop CTF Girlie-Shirt",
  "description": "For serious Capture-the-Flag heroines only!",
  "price": 22.49,
  "deluxePrice": 22.49,
  "image": "fan_girlie.jpg",
  "createdAt": "2026-04-30 15:08:29.885 +00:00",
  "updatedAt": "2026-04-30 15:08:29.885 +00:00",
  "deletedAt": null
},
{
  "id": 8,
  "name": "mc.safesearch@juice-sh.op",
  "description": "b03f4b0ba8b458fa0acdc02cdb953bc8",
  "price": "4",
  "deluxePrice": "5",
  "image": "6",
  "createdAt": "7",
  "updatedAt": "8",
  "deletedAt": "9"
},
{
  "id": 9,
  "name": "J12934@juice-sh.op",
  "description": "3c2abc04e4a6ea8f1327d0aae3714b7d",
  "price": "4",
  "deluxePrice": "5",
  "image": "6",
  "createdAt": "7",
  "updatedAt": "8",
  "deletedAt": "9"
},
{
```

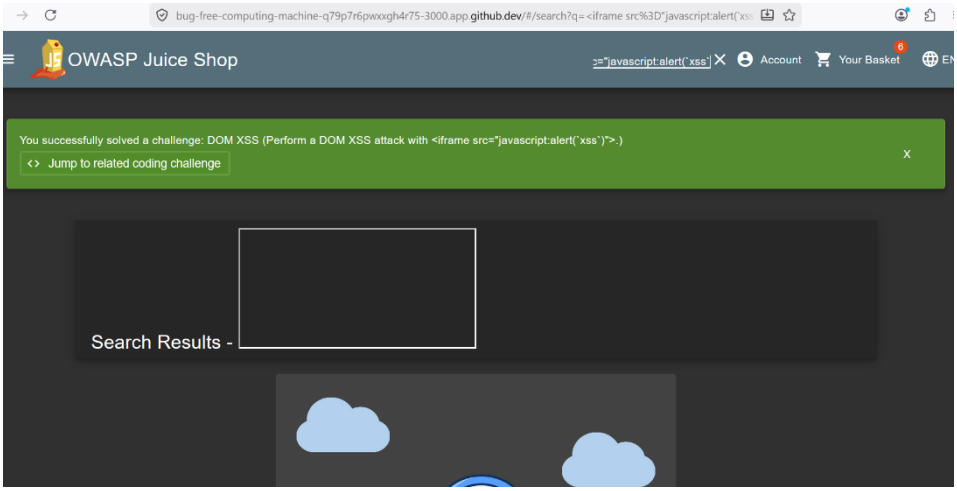
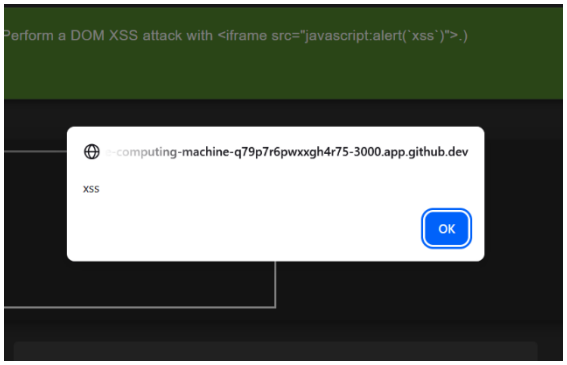

admin@juice-sh.op	0192023a7bbd73250516f069df18b500
jim@juice-sh.op	e541ca7ecf72b8d1286474fc613e5e45
bender@juice-sh.op	0c36e517e3fa95aabf1bbffc6744a4ef
bjoern.kimminich@gmail.com	6edd9d726cbdc873c539e41ae8757b8c
ciso@juice-sh.op	861917d5fa5f1172f931dc700d81a8fb
support@juice-sh.op	3869433d74e3d0c86fd25562f836bc82
morty@juice-sh.op	f2f933d0bb0ba057bc8e33b8ebd6d9e8
mc.safesearch@juice-sh.op	b03f4b0ba8b458fa0acdc02cdb953bc8
J12934@juice-sh.op	3c2abc04e4a6ea8f1327d0aae3714b7d
wurstbrot@juice-sh.op	9ad5b0492bbe528583e128d2a8941de4
amy@juice-sh.op	030f05e45e30710c3ad3c32f00de0473
bjoern@juice-sh.op	7f311911af16fa8f418dd1a3051d6810
bjoern@owasp.org	9283f1b2e9669749081963be0462e466
chris.pike@juice-sh.op	10a783b9ed19ea1c67c3a27699f0095b
accountant@juice-sh.op	963e10f92a70b4b463220cb4c5d636dc
uvogin@juice-sh.op	05f92148b4b60f7dacd04cceebb8f1af
demo	fe01ce2a7fbac8fafaed7c982a04e229
john@juice-sh.op	00479e957b6b42c459ee5746478e4d45
emma@juice-sh.op	402f1c4a75e316afec5a6ea63147f739
stan@juice-sh.o	e9048a3f43dd5e094ef733f3bd88ea64
ethereum@juice-sh.op	2c17c6393771ee3048ae34d6b380c5ec
testing@juice-sh.op	b616a64605a07941fbd31868aea3b54b



Management response	djaGDJGjfakj

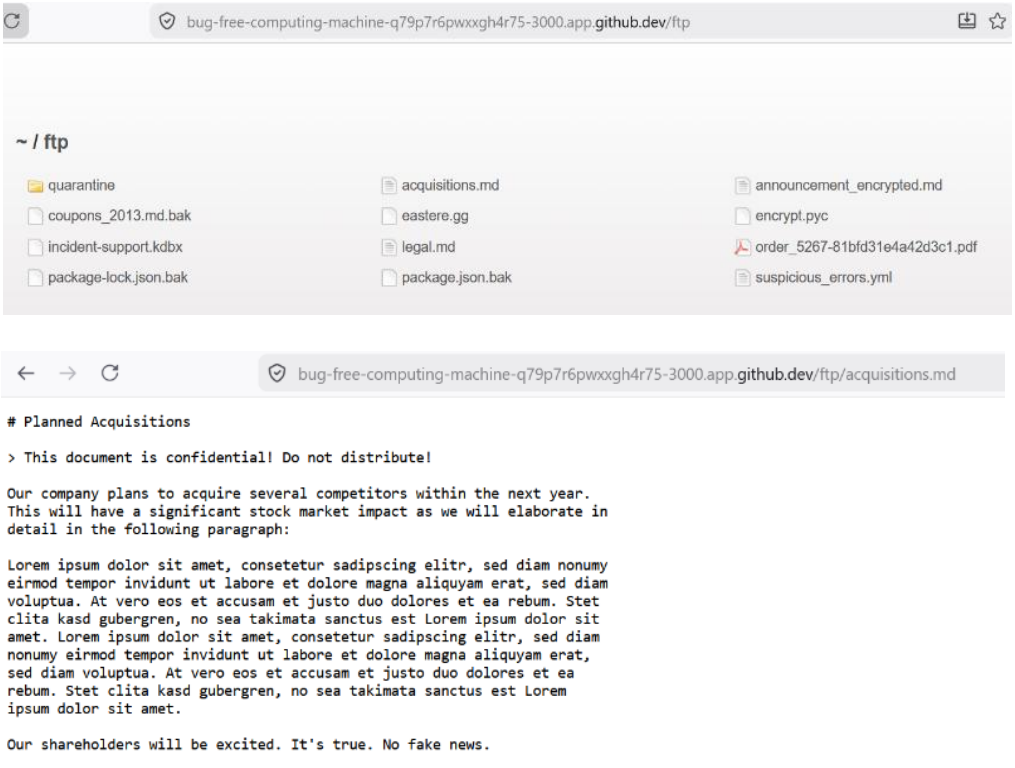
3. Reflected / DOM-Based Cross-Site Scripting (XSS) in Search Input

Risk Rating	High
Category	Injection
Description	• The search input field reflects user-supplied data directly into the HTML DOM without output encoding or sanitization, enabling JavaScript execution in the victim's browser context. • Confirmed payloads: <iframe src="javascript:alert(`xss`)"> executes embedded JavaScript via the iframe's src attribute; Google demonstrates open redirect / phishing vector through anchor injection. • The application does not implement a Content Security Policy (CSP) header to restrict script execution sources, leaving no browser-level mitigation against injected scripts. • Input is processed client-side via Angular/React without proper DOM sanitization, suggesting DOM-based XSS in addition to reflected XSS. • No HttpOnly or SameSite cookie attributes were observed, making session token theft via document.cookie exfiltration feasible.
Impact	• Session hijacking via JavaScript document.cookie exfiltration to attacker-controlled server. • Credential harvesting through injected fake login forms overlaid on the legitimate page. • Keylogging, clipboard theft, and webcam/mic access via injected scripts in modern browser APIs. • Open redirect payload enables convincing phishing attacks leveraging the trusted juice-shop domain.
Recommendation	1. Encode all user-supplied output using context-aware encoding (HTML entity encoding for HTML context, JS escaping for script context) before rendering in the DOM. 2. Implement a strict CSP header: Content-Security-Policy: default-src 'self'; script-src 'self' to block inline and external script execution. 3. Use Angular's built-in DomSanitizer or React's JSX auto-escaping — never bypass with innerHTML or dangerouslySetInnerHTML. 4. Set HttpOnly and SameSite=Strict on all session cookies to prevent JavaScript-based theft. 5. Implement an allowlist-based URL validation for redirect parameters to prevent open redirect exploitation.
Finding Id	

Target Date	
Affected URLs	
Evidences	 
Management response	djaGDJGjfakj

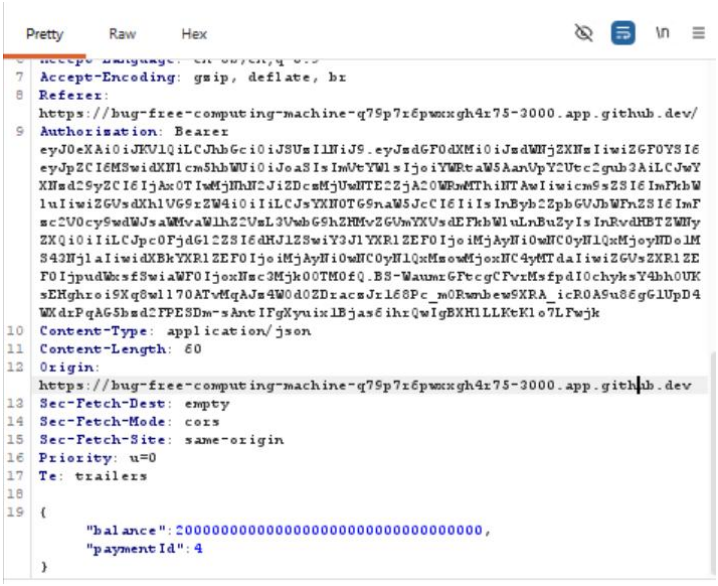
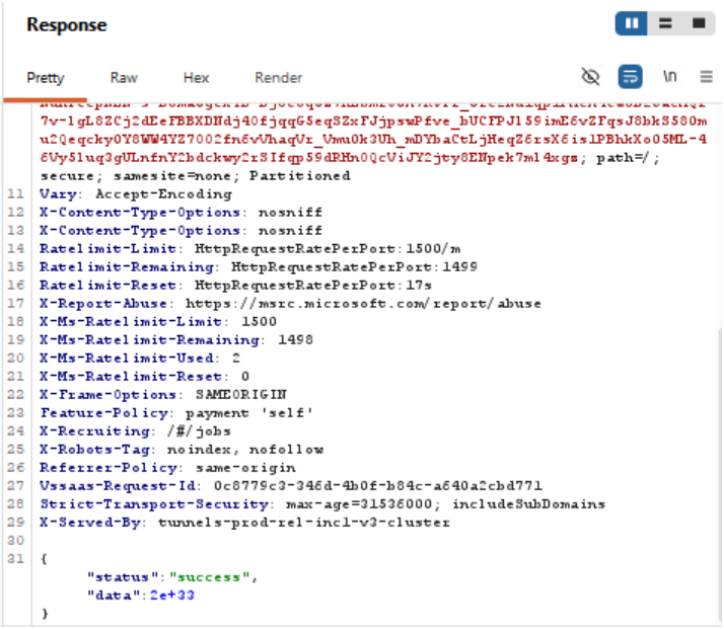
4. Sensitive File Exposure on Web Server

Risk Rating	High
Category	Security Misconfiguration
Description	• The web server exposes sensitive files (e.g., configuration files, backup files, or internal documents) via direct URL access without authentication or access control enforcement. • Improper directory listing configuration or absence of deny rules in server configuration (nginx/Apache) allows unauthenticated traversal and retrieval of non-public assets. • Sensitive paths discovered via the main.js bundle (Finding #8) likely guided enumeration to locate these files — confirming that front-end asset leakage directly enables server-side exploitation. • No HTTP authentication challenge, authorization header check, or IP restriction is enforced on the sensitive resource endpoints. • The exposed files may contain API keys, DB connection strings, internal IPs, or user data depending on the file type accessed.

Impact	• Disclosure of internal infrastructure details aids attacker reconnaissance for targeted exploitation. • If config files are exposed, credentials or API keys embedded in them may enable lateral movement to connected services. • Backup files may contain older source code revealing deprecated but exploitable logic. • Compliance violations — inadvertent exposure of PII or financial data.
Recommendation	1. Disable directory listing at the web server level (Options -Indexes for Apache; autoindex off for nginx). 2. Enforce deny rules for non-public file extensions (.bak, .config, .env, .log) via server config or .htaccess. 3. Move all sensitive files outside the web root entirely — they should never be served by the HTTP server. 4. Implement authentication and RBAC checks at the middleware layer for any endpoint serving internal resources. 5. Perform regular automated scans (e.g., Nikto, Gobuster) as part of CI/CD to detect newly exposed paths before production deployment.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

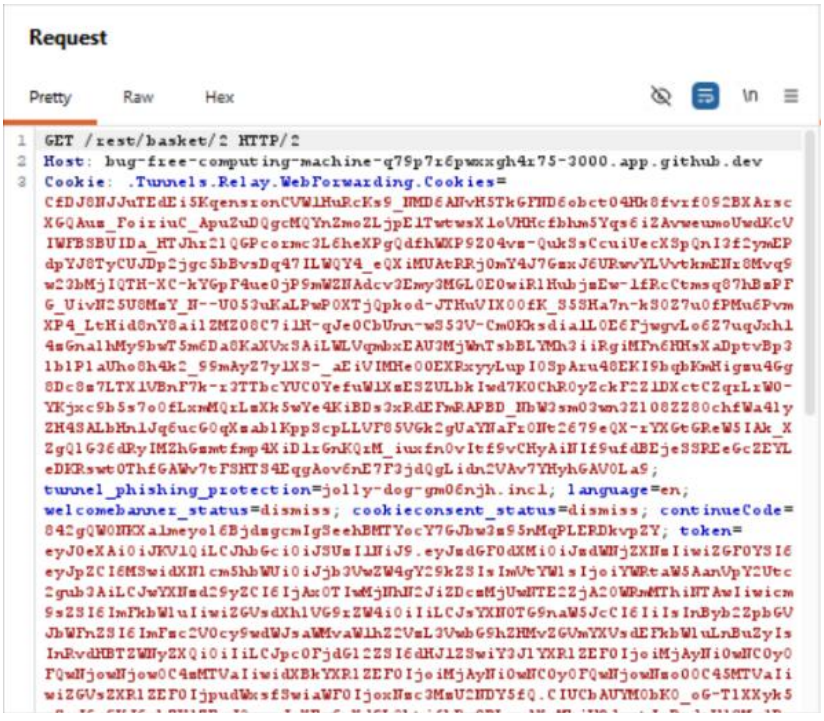
5. Insecure Direct Object Reference (IDOR) — Arbitrary Wallet Deposit Manipulation

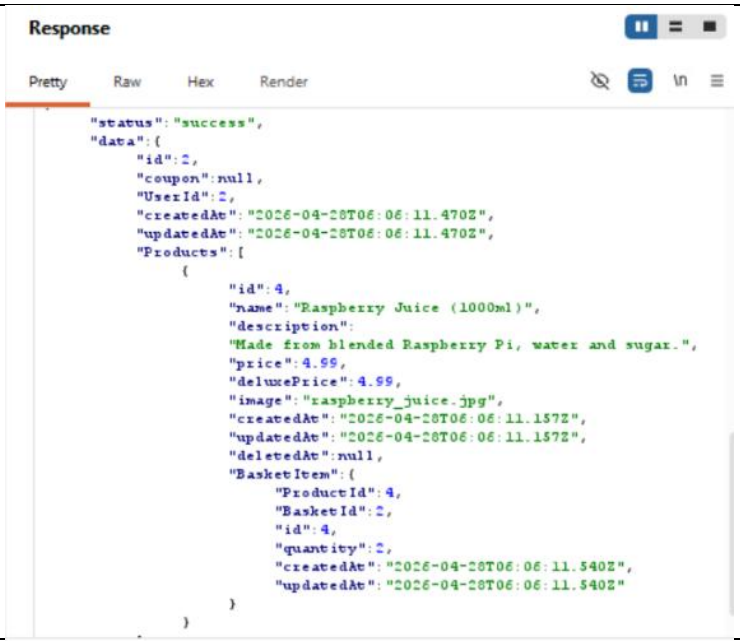
Risk Rating	High
Category	Broken Access Control
Description	• The API endpoint responsible for wallet/credit deposits accepts an attacker-controlled amount parameter in the request body without server-side validation or business logic enforcement on the permissible value range. • An authenticated user can intercept the HTTP request (via Burp Suite or browser DevTools) and modify the amount field to an arbitrary positive integer — including extremely large values — which the server processes and credits without verification. • No server-side check enforces that the deposit amount corresponds to an actual payment transaction or payment gateway confirmation token, indicating the payment flow and balance update are decoupled. • The vulnerability may also permit negative amounts, enabling arbitrary deduction from other accounts if the object reference allows cross-account targeting via a user ID parameter. • Absence of an HMAC signature or anti-tampering mechanism on the transaction payload allows trivial parameter tampering without invalidating the request.
Impact	• Financial fraud — attacker can credit arbitrary amounts to their own wallet without any real monetary transaction. • Potential for cross-account manipulation if user ID is part of the controllable request payload. • Business revenue loss and platform integrity compromise. • Undermines payment provider integration — transactions not reconciled with actual payment gateway events.
Recommendation	1. Never trust client-supplied monetary values — the server must independently verify the transaction amount against a payment gateway event ID (e.g., Stripe PaymentIntent, Razorpay order ID) before updating balances. 2. Implement strict server-side business logic validation — define and enforce allowed value ranges (minimum/maximum deposit amounts). 3. Tie balance updates to confirmed webhook callbacks from the payment provider with HMAC signature verification. 4. Enforce RBAC on all financial endpoints — ensure users can only modify objects scoped to their own authenticated session. 5. Log all transaction attempts with user ID, IP, amount, and outcome — alert on anomalous deposit patterns.
Finding Id	
Target Date	
Affected URLs	

Evidences	 
Management response	djaGDJGjfakj

6. IDOR — Unauthorized Access and Modification of Other Users' Shopping Baskets

Risk Rating	High
Category	Broken Access Control
Description	• The basket/cart API endpoints accept a basket ID as a path or query parameter (e.g., /api/BasketItems/1) and perform no ownership validation — any authenticated user can substitute another user's basket ID to view or modify it. • The server does not validate that the basketId in the request corresponds to the authenticated user's session — it returns or mutates the resource based solely on the supplied ID.

	• Basket IDs appear to be sequential integers, making enumeration trivial — an attacker can iterate from ID 1 to N to access all users' baskets without any resistance. • Write access to another user's basket allows item injection (adding products to their cart, potentially triggering unwanted purchases if one-click checkout exists) or item deletion (denial of service to their shopping session). • This is a classic IDOR vulnerability arising from the absence of object-level authorization checks at the API layer.
Impact	• Privacy violation — attacker can view all items in any user's basket, inferring shopping behaviour and preferences. • Cart tampering — attacker can add/remove items from another user's basket, potentially causing unintended purchases. • If basket contains saved payment method references, indirect exposure of payment data.
Recommendation	1. Enforce object-level authorization on every basket API call — validate that the basketId being accessed belongs to the authenticated user's session/JWT before processing the request. 2. Use indirect references (UUIDs or session-scoped tokens) instead of sequential integer IDs to prevent enumeration. 3. Implement a centralized authorization middleware (e.g., CASL, OPA) that enforces ownership checks across all resource-level API operations. 4. Add automated IDOR detection tests to the API test suite — assert that cross-account resource access returns HTTP 403.
Finding Id	
Target Date	
Affected URLs	
Evidences	



Management response	djaGDJGjfakj

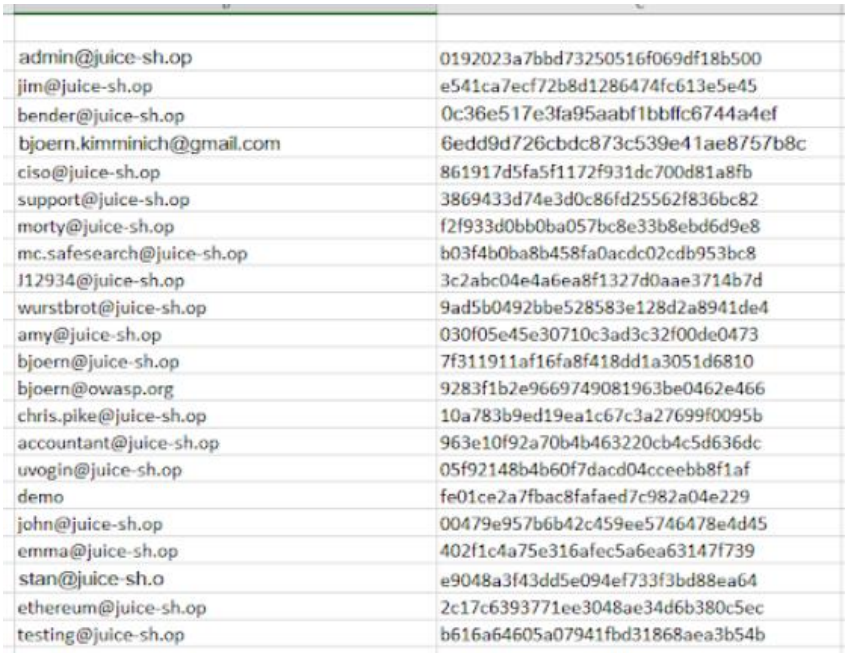
7. Passwords Stored Without Encryption / Cryptographic Hashing

Risk Rating	Medium
Category	Cryptographic Failures
Description	• The user registration endpoint accepts and processes POST requests with empty or null values for fields marked as required (e.g., email, password, username), indicating that field-presence validation exists only on the client side and is absent at the API/server layer. • By intercepting the registration request and clearing required field values (via Burp Suite or curl), an attacker can create malformed user accounts with empty identifiers that may break downstream business logic or data integrity constraints. • Absence of server-side validation violates the principle of defense-in-depth — client-side validation is a UX aid, not a security control, and is trivially bypassed by any HTTP client or proxy tool. • Empty field accounts may bypass authentication flows if the login query matches on empty strings or NULL values, depending on ORM behavior and query construction. • This indicates a systemic design gap — if registration lacks validation, other forms (profile update, checkout) may have similar omissions.
Impact	• Creation of ghost accounts with empty identifiers that may pollute user databases and break analytics/reporting. • Potential authentication bypass if empty-field accounts can be used to log in without credentials. • Downstream application errors or crashes if null values propagate to services expecting valid email/username formats.
Recommendation	1. Implement comprehensive server-side input validation using a validation library (e.g., Joi, Yup, express-validator) for all registration and user-input endpoints —

	validate field presence, type, length, and format independently of client-side checks. 2. Enforce database-level NOT NULL constraints and unique constraints on critical columns (email, username) as a last line of defense against invalid data insertion. 3. Return descriptive HTTP 400 (Bad Request) responses with field-level error details when validation fails — this aids developers without exposing internal logic. 4. Conduct a validation audit across all POST/PUT endpoints to identify other missing server-side checks systemically. 5. Add integration tests that send malformed/empty payloads to all input-accepting endpoints and assert HTTP 400 responses.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

8. Weak MD5 Password Hashing — Cryptographically Broken Algorithm

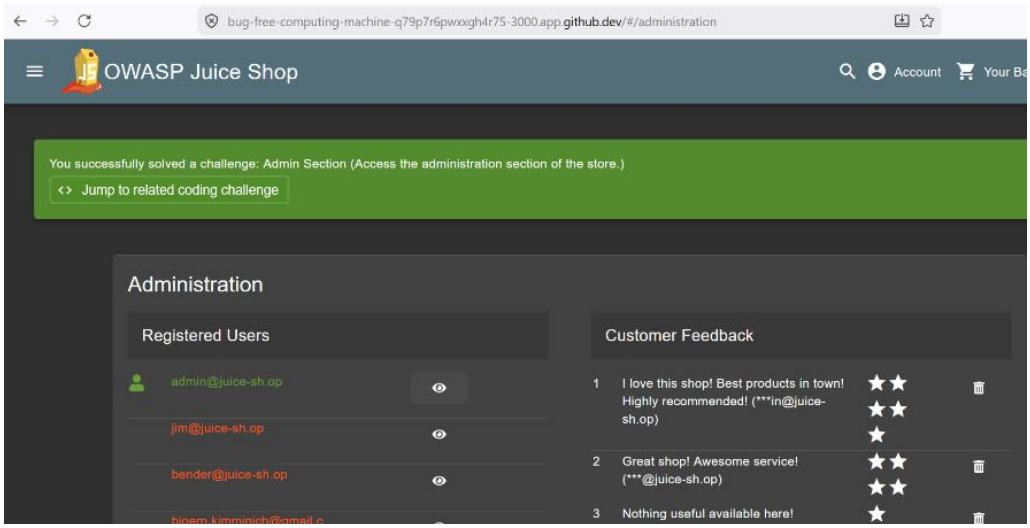
Risk Rating	Medium
Category	Cryptographic Failures
Description	• User passwords for certain accounts within the application are hashed using the MD5 algorithm — a cryptographically broken hash function that was deprecated for security use in 1996 and is entirely unsuitable for password storage in any context. • MD5 produces a fixed 128-bit output with no salting applied per-user, making the stored hashes immediately vulnerable to precomputed rainbow table lookups — entire MD5 hash databases covering billions of common passwords are publicly available online (e.g., CrackStation, HashKiller). • MD5 can be computed at approximately 10–20 billion hashes per second on a single modern GPU (RTX 4090 class), meaning an exhaustive brute-force attack against all 8-character lowercase alphanumeric passwords completes in under 60 seconds.This indicates a systemic design gap — if registration lacks validation, other forms (profile update, checkout) may have similar omissions.

Impact	• All MD5-hashed passwords are trivially reversible via publicly available rainbow tables — any extracted hash can be cracked within seconds using free online tools, requiring no specialized hardware. • An attacker who obtains even a single valid MD5 hash (e.g., via SQL injection) can immediately authenticate as that user without needing to interact with the application. • Passwords cracked from MD5 hashes are usable for credential stuffing against banking, email, and social media platforms — the real-world financial and personal impact extends far beyond the application itself.
Recommendation	1. Replace all MD5 password hashes immediately with bcrypt (cost factor ≥ 12), Argon2id (recommended by OWASP as the first choice for new systems), or scrypt — these are the only algorithms acceptable for password storage per current NIST guidance. 2. Generate a unique, cryptographically random per-user salt (minimum 16 bytes / 128 bits) using a CSPRNG such as crypto.randomBytes() in Node.js — the salt must be stored alongside the hash but must not be derived from user attributes. 3. Implement an on-login upgrade path — when a user authenticates successfully and a legacy MD5 hash is detected, immediately re-hash with the new algorithm and overwrite the stored value in the database. 4. Force a mandatory password reset notification for all accounts with MD5-hashed passwords — treat all such passwords as already compromised and require users to choose new credentials.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

admin@juice-sh.op	0192023a7bbd73250516f069df18b500
jim@juice-sh.op	e541ca7ecf72b8d1286474fc613e5e45
bender@juice-sh.op	0c36e517e3fa95aabf1bbffc6744a4ef
bjoern.kimminich@gmail.com	6edd9d726cbdc873c539e41ae8757b8c
ciso@juice-sh.op	861917d5fa5f1172f931dc700d81a8fb
support@juice-sh.op	3869433d74e3d0c86fd25562f836bc82
morty@juice-sh.op	f2f933d0bb0ba057bc8e33b8ebd6d9e8
mc.safesearch@juice-sh.op	b03f4b0ba8b458fa0acdc02cdb953bc8
J12934@juice-sh.op	3c2abc04e4a6ea8f1327d0aae3714b7d
wurstbrot@juice-sh.op	9ad5b0492bbe528583e128d2a8941de4
amy@juice-sh.op	030f05e45e30710c3ad3c32f00de0473
bjoern@juice-sh.op	7f311911af16fa8f418dd1a3051d6810
bjoern@owasp.org	9283f1b2e9669749081963be0462e466
chris.pike@juice-sh.op	10a783b9ed19ea1c67c3a27699f0095b
accountant@juice-sh.op	963e10f92a70b4b463220cb4c5d636dc
uvogin@juice-sh.op	05f92148b4b60f7dacd04cceebb8f1af
demo	fe01ce2a7fbac8fafaed7c982a04e229
john@juice-sh.op	00479e957b6b42c459ee5746478e4d45
emma@juice-sh.op	402f1c4a75e316afec5a6ea63147f739
stan@juice-sh.o	e9048a3f43dd5e094ef733f3bd88ea64
ethereum@juice-sh.op	2c17c6393771ee3048ae34d6b380c5ec
testing@juice-sh.op	b616a64605a07941fbd31868aea3b54b

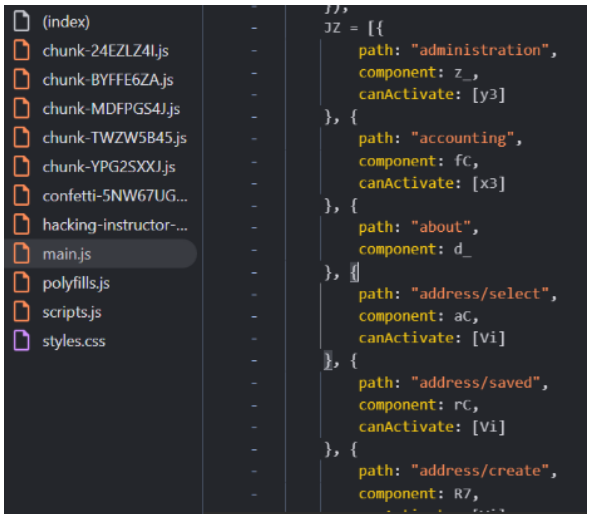
9. Unauthenticated / Unprotected Admin and Accounting Routes

Risk Rating	Medium
Category	Security Misconfiguration
Description	• The /administration and /accounting routes are directly accessible via URL without any server-side authorization check enforcing admin-level role membership. • Route protection appears to rely solely on client-side Angular route guards (e.g., canActivate), which are trivially bypassed by directly navigating to the URL or manipulating client-side state. • The routes are discoverable via the bundled main.js file where Angular router configurations expose all registered paths in plaintext. • Once accessed, the administration panel likely exposes user management, order management, and configuration capabilities that should be restricted to privileged roles. • No HTTP 401/403 response is returned when a non-admin authenticated or unauthenticated user accesses these paths, confirming the absence of server-side enforcement.
Impact	• Any user — authenticated or not — can access admin functionality, enabling user data manipulation, order tampering, or configuration changes. • Ability to create/delete users, reset passwords, or escalate privileges without administrative credentials. • Complete loss of application integrity if admin controls are abused at scale.
Recommendation	1. Implement server-side authorization middleware that validates JWT/session claims for role = admin on every request to /administration and /accounting API endpoints — client-side guards alone are insufficient. 2. Return HTTP 401 for unauthenticated requests and HTTP 403 for authenticated non-admin requests to sensitive routes. 3. Apply the principle of least privilege — admin routes should only be accessible from internal IPs or via VPN in production environments. 4. Conduct a full route audit and map each route to its required permission scope in a centralized access control matrix.
Finding Id	
Target Date	
Affected URLs	

Evidences	
Management response	djaGDJGjfakj

10. Sensitive Internal Paths Exposed via main.js Bundle

Risk Rating	Medium
Category	Security Misconfiguration
Description	• The Angular production build (main.js) is served unobfuscated and contains plaintext Angular router configurations, exposing all registered application routes including privileged paths such as /administration and /accounting. • The bundle may additionally contain hardcoded API endpoint URLs, internal service references, feature flags, environment-specific configurations, or developer comments left in source. • Attackers can retrieve main.js without authentication, parse it using browser DevTools or tools like js-beautify, and enumerate the full attack surface of the application in minutes. • This constitutes a reconnaissance amplifier — every other vulnerability in this report was easier to discover because the route map was publicly available in the frontend bundle. • Dead code, deprecated routes, or legacy API endpoints included in the bundle but not linked in the UI remain exploitable if left accessible server-side.
Impact	• Drastically reduces attacker effort for endpoint enumeration and attack surface mapping. • Exposes admin routes, internal APIs, and potentially hardcoded secrets to unauthenticated users. • Enables targeted exploitation of discovered endpoints — as demonstrated in Findings #4 and #7.
Recommendation	1. Enable production-grade minification and tree-shaking in the Angular build pipeline — use ng build --configuration production with buildOptimizer: true. 2. Apply JavaScript obfuscation tooling (e.g., javascript-obfuscator) as a post-build step to obscure route strings and identifiers.

	3. Remove all dead code, unused routes, and developer comments from production builds via linting rules and build-time analysis. 4. Critically: server-side access control must be the primary defense — obscuring routes is a defense-in-depth measure, not a substitute for proper authorization. 5. Audit the bundle for hardcoded secrets using tools like truffleHog or gitleaks integrated into CI/CD pipelines.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

11. Sensitive Discount Coupon Code Leaked via Chatbot

Risk Rating	Medium
Category	Insecure Design
Description	• The application's chatbot exposes valid promotional coupon codes to any authenticated user through conversational prompts, without verifying entitlement, user role, or triggering conditions. • The chatbot response logic likely matches user input against hardcoded or database-stored patterns and returns coupon values when triggered — with no rate limiting or one-time-use enforcement visible at the discovery stage. • Coupon codes are treated as secrets but are stored or referenced in a context (chatbot response templates or knowledge base) accessible to the application layer without confidentiality controls. • This represents an insecure design flaw — business-sensitive data (discount codes) is accessible through an unintended channel without appropriate access controls. • If coupon codes are not single-use or rate-limited, the leaked code can be reused unlimited times by any user who discovers the trigger phrase.

Impact	• Unlimited discount abuse — any user can claim promotional codes not intended for them, causing direct revenue loss. • Coupon codes shared publicly (social media, forums) amplify the financial impact at scale. • Undermines marketing campaign integrity and budget control.
Recommendation	1. Redesign chatbot response logic to never return raw coupon codes — use entitlement checks before surfacing any promotional data. 2. Implement coupon codes as single-use, time-bound tokens tied to a specific user account upon issuance. 3. Store coupon codes with access controls — only the redemption API should query coupon values, not the chatbot response engine. 4. Add rate limiting and anomaly detection on coupon redemption endpoints to detect bulk abuse. 5. Conduct a review of all chatbot response templates and knowledge base entries for inadvertent sensitive data exposure.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

12. Missing X-Frame-Options / Clickjacking Vulnerability

Risk Rating	Low
Category	Injection

Description	• The application does not return X-Frame-Options or Content-Security-Policy: frame-ancestors headers in HTTP responses, allowing any external domain to embed the application inside an <iframe>. • An attacker can host a malicious page with a transparent or opaque iframe overlay positioned over legitimate UI elements (e.g., checkout button, account settings), tricking users into performing unintended actions. • Combined with the absence of CSRF tokens (if applicable), iframe-based attacks can execute state-changing operations on behalf of the authenticated victim. • The attack requires no server-side vulnerability — it exploits the browser's default iframe rendering behavior in the absence of protective headers. • Sensitive pages including the administration panel (/administration) are equally framing-unprotected, increasing severity of the clickjacking attack surface.
Impact	• Victim unknowingly performs authenticated actions (purchases, profile changes, fund transfers) triggered by attacker's overlaid UI. • Admin clickjacking can lead to unauthorized configuration changes or privilege grants. • Low user awareness — attack requires no malware, just visiting an attacker-controlled page.
Recommendation	1. Add X-Frame-Options: DENY or X-Frame-Options: SAMEORIGIN to all HTTP responses at the server/middleware level. 2. Implement CSP frame-ancestors directive: Content-Security-Policy: frame-ancestors 'none' for full protection (supersedes X-Frame-Options in modern browsers). 3. Configure the header globally in the web server config (nginx/Apache) rather than per-route to prevent omission on sensitive endpoints. 4. Use automated security header scanners (e.g., securityheaders.com, OWASP ZAP) in CI/CD pipeline to catch missing headers before deployment.
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj

13. Publicly Accessible Application Logs Exposing Internal System Information

Risk Rating	Low
Category	Security Logging & Monitoring Failures
Description	The application exposes server log files that can be accessed without authentication. These logs contain internal request details including endpoints, timestamps, IP addresses, and user activity.

Impact	• Enables attackers to:
	  ○ Map internal API endpoints
	  ○ Identify admin routes
	  ○ Analyze user behavior
	• Assists in further attacks (reconnaissance phase)
	• May expose sensitive data if logging increases in future

Recommendation	1. Restrict access to log files (authentication required) 2. Store logs outside web root 3. Sanitize sensitive data before logging 4. Implement proper access control for log storage
Finding Id	
Target Date	
Affected URLs	
Evidences	
Management response	djaGDJGjfakj