



agentschmitz

Penetrationstest-Bericht

Automatisierter KI-Sicherheitstest

ZIELSYSTEM

Juice Shop

<http://127.0.0.1:8091>

DATUM

5. Oktober 2026

Gefundene Schwachstellen **20**



● **1** Kritisch

● **7** Hoch

● **5** Mittel

● **6** Niedrig

● **1** Info

GESAMTRISIKO: KRITISCH

VERTRAULICH

Scanner-IP 85.214.252.135 · Hosted in Germany · ID d247b9ca



Inhaltsverzeichnis

1. Zusammenfassung	1
2. Geltungsbereich	2
3. Methodik	3
4. Schwachstellen	4
5. Fazit	23



Penetrationstest-Report

Ziel: <http://127.0.0.1:8091> (OWASP Juice Shop, ermittelte Version 20.2.0, Node.js/Express)

Testtyp: Autorisierter Penetrationstest – explorative Vollflächenerkundung

Durchführung: Ein freiarbeitender, autonomer Pentesting-Agent (keine festen Missionen; alle erreichbaren Oberflächen wurden untersucht)

Datum: 05.10.2026

Klassifizierung: Vertraulich – nur für Auftraggeber bestimmt

Genehmigungsstatus: Autorisiert

Zusammenfassung

Im Rahmen eines autorisierten Penetrationstests wurde die Webanwendung unter <http://127.0.0.1:8091> vollständig untersucht. Es handelt sich nach Fingerprinting um einen **OWASP Juice Shop** (Version 20.2.0) auf Node.js/Express.

Das Gesamtrisiko wird als **KRITISCH** eingestuft. Es wurden **20 Schwachstellen** identifiziert, **sämtlich mit Status „confirmed“** und durch echte Anwendungsantworten belegt. Eine WAF war **nicht** im Pfad (Canary-Probe unauffällig); daher konnte jeder Befund end-to-end verifiziert werden. Es existieren **keine** nicht verifizierbaren Befunde.

Schwere	Anzahl
Kritisch	1
Hoch	7
Mittel	5
Niedrig	6

Informationell	1
Gesamt	20

Kernfeststellungen:

- 1. Vollständige Authentifizierungsumgehung:** Der Login-Endpoint (`/rest/user/login`) ist durch unauthentifizierte SQL-Injection komplett kompromittierbar. Ohne Passwortkenntnis wird ein gültiger Admin-JWT ausgestellt – die Anwendung ist damit vollständig übernehmbar.
- 2. Kritische Datenexposition:** Das JWT bettet den (ungesalzenen, trivial knackbaren) MD5-Passwort-Hash des Administrators ein. Das `/ftp`-Verzeichnis stellt ohne Authentifizierung eine KeePass-Datenbank, vertrauliche M&A-Dokumente und per Null-Byte-Bypass beliebige weitere Dateien bereit.
- 3. Systemisch fehlende Zugriffskontrolle:** Order-Tracking, Admin-Konfiguration, Benutzer-, Karten-, Beschwerde- und Feedback-APIs sind unzureichend geschützt; fremde Warenkörbe und Bestellungen fremder Kunden sind auslesbar. In den Feedback-Daten liegt eine **live Wallet-Seed-Phrase** – ein unmittelbar verwertbares Secret.
- 4. Injections über mehrere Paradigmen:** Neben klassischer SQL-Injection (Login, Produktsuche) wurde eine serverseitige JavaScript-Injection (`$where`) im Order-Tracking bestätigt, die alle Bestellungen exfiltriert.

Die Befunddichte und -schwere ist konsistent mit dem Charakter des Ziels als absichtlich verwundbarer Trainingsanwendung; für einen produktiven Einsatz wären die beschriebenen Mängel jedoch unmittelbar und umfassend zu beheben (siehe Fazit).

Geltungsbereich

Element	Beschreibung
Primärhost	127.0.0.1, Port 8091/TCP (HTTP)
Anwendung	OWASP Juice Shop v20.2.0 (SPA + REST-Backend)
Untersuchte Oberflächen	Web-UI, REST-APIs (<code>/rest/*</code> , <code>/api/*</code> , <code>/api/v1</code> , <code>/api/v2</code>), File-Server (<code>/ftp</code> , <code>/ftp/quarantine</code>), statische Assets (<code>/assets</code>), API-Dokumentation (<code>/api-docs</code>), Monitoring (<code>/metrics</code>), Metadaten (<code>/robots.txt</code> , <code>/.well-known/security.txt</code>)

Authentifizierung während des Tests	Start ohne Authentifizierung; Admin-Session wurde ausschließlich über den bestätigten SQLi-Bypass erlangt
WAF	Keine erkannt (detected: false, keine Signale; Canary-Probe lieferte unauffällige 200-Baseline)
Außerhalb des Geltungsbereichs	Nicht identifizierte Dienste/Ports; physische und soziale Angriffe; Ausfallszenarien (DoS)

Methodik

Der Test wurde von **einem einzigen freiarbeitenden, autonomen Pentesting-Agenten** durchgeführt. Dieser arbeitete ohne vorgegebene Missionen und erkundete selbstständig alle erreichbaren Angriffsflächen in folgenden Phasen:

- 1. Reconnaissance & Fingerprinting:** HTTP-Erreichbarkeit, Titel, Security-Header, Server-Banner, `robots.txt`, `security.txt`, Verzeichnis- und Endpoint-Enumeration (`/ftp`, `/api`, `/api-docs`, `/metrics` u. a.).
- 2. Angriffsflächen-Mapping:** Katalogisierung aller Endpunkte, Dateiserver-Whitelist-Logik, API-Strukturen, Swagger-Spec (`/api-docs/swagger-ui-init.js`), Fehlerseitenverhalten.
- 3. WAF-Erkundung:** Canary-Probes; Ergebnis: **kein WAF im Pfad** – alle Tests liefen direkt gegen die Ursprungsanwendung.
- 4. Schwachstellensuche & Exploitation:** Manuelle Injection-Tests (SQL, serverseitiges JS), Authentifizierungs-/Autorisierungstests (IDOR, fehlende Ownership-Prüfung), Datei-Offenlegung (Extension-Whitelist-Bypass, Null-Byte), JWT-Analyse (Payload-Dekodierung), Konfigurations- und Informationslecks, CAPTCHA-/Anti-Automationslogik, Datei-Upload.
- 5. Verifikation:** Jeder Befund wurde durch reproduzierbare HTTP-Anfragen mit vollständiger Antwortdokumentation bestätigt. Ein Befund (Datei-Upload) ist in der **Ausführung** (Stored XSS) tentativ, da die Verifikationsabfragen SPA-Fallback-HTML statt der SVG-Bytes lieferten; die fehlende Upload-Validierung selbst ist durch echte App-Antworten (204) belegt.

Qualitätsstandard: Kein Befund beruht auf Spekulation; alle Evidence-Auszüge stammen aus tatsächlichen Antworten des Zielsystems.

Schwachstellen

Bestätigte Schwachstellen

Kritisch

F-01 – SQL-Injection im Login-Endpoint !' vollständige Admin-Authentifizierungsumgehung

Feld	Wert
Schwere	Kritisch
CWE	CWE-89
Endpunkt	POST /rest/user/login
Status	Bestätigt

Beschreibung:

Der Login-Endpoint verkettet Benutzereingaben ungeschützt in die SQL-Abfrage. Eine Tautologie-Injection im E-Mail-Feld authentifiziert den Angreifer als ersten Benutzer (id=1, Rolle **admin**), ohne dass ein Passwort bekannt sein muss. Der ausgestellte JWT gewährt Zugriff auf alle Admin-APIs (/api/Users, /api/Cards, /rest/user/authentication-details, /rest/order-history usw.) und ermöglicht damit die vollständige Übernahme der Anwendung. Der im Token enthaltene Hash **0192023a7bbd73250516f069df18b500** ist ein ungesalzener MD5 und entspricht dem bekannten schwachen Passwort **admin123**.

Nachweis (Evidence):

http

POST /rest/user/login

```
{"email":"","or 1=1--","password":"x"}
```

!' 200 OK

```
{"authentication":{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...","bid":1,"umail":"admin@juice-sh.op"}}
```

JWT-Payload (dekodiert):

```
{"data":{"id":1,"email":"admin@juice-sh.op",  
"password":"0192023a7bbd73250516f069df18b500",  
"role":"admin","isActive":true,...},"bid":1}
```

Auswirkung:

Vollständige Übernahme der Anwendung ohne Passwortkenntnis: Zugriff auf alle Admin- und CRUD-Endpunkte, das vollständige Benutzerverzeichnis, Zahlungsdaten, Bestellhistorien und die Anwendungskonfiguration. In Kombination mit F-02 ergibt sich zusätzlich die Kompromittierung des Klartext-Admin-Passworts.

Empfehlung:

Parametrisierte Abfragen/Prepared Statements (Sequelize-Platzhalter) statt String-Konkatenation; Authentifizierungslogik von der Datenabfrage trennen und Passwort serverseitig prüfen; Passwort-Hashes mit Salz und langsamem KDF (bcrypt/argon2) speichern.

Hoch

F-02 – Admin-Passwort-Hash im JWT-Payload offengelegt (Excessive Data Exposure)

Feld	Wert
Schwere	Hoch
CWE	CWE-200
Endpunkt	POST /rest/user/login (JWT-Payload)
Status	Bestätigt

Beschreibung:

Das nach dem Login ausgestellte JWT bettet die gesamte Benutzerzeile in den Payload ein, einschließlich des Passwort-Hashes (ungesalzener MD5, trivial knackbar) und interner Felder. Jeder Token-Leak (Logs, Referrer, XSS) kompromittiert damit unmittelbar die Zugangsdaten. Passwortmaterial darf niemals in JWTs enthalten sein.

Nachweis (Evidence):

```
Base64-Payload-Segment des JWT dekodiert:  
{  
  "data":  
  {  
    "...",  
    "password": "0192023a7bbd73250516f069df18b500",  
    "role": "admin",  
    "...",  
    "bid": 1,  
    "..."  
  }  
}
```

Auswirkung:

Offline-Crack des ungesalzenen MD5-Hashes (bestätigt: **admin123**) !' Klartext-Passwort des Administrators; jeder Token-Leak führt zur vollständigen Kontoübernahme, unabhängig von Token-Ablaufmechanismen.

Empfehlung:

JWT-Payload auf minimale Claims (**sub**, **role**, **exp**) beschränken; keine sensiblen Datenbankfelder serialisieren; starke, gesalzene Hashing-Verfahren einsetzen.

F-03 – Unauthentifizierte SQL-Injection in der Produktsuche

Feld	Wert
Schwere	Hoch
CWE	CWE-89
Endpunkt	GET /rest/products/search
Status	Bestätigt

Beschreibung:

Der Suchparameter wird ungefiltert in die SQL-**WHERE**-Klausel konkateniert. Eine Tautologie liefert statt gefilterter Treffer den vollständigen Produktkatalog inklusive nicht gelisteter Artikel. Der Endpoint erfordert keine Authentifizierung.

Nachweis (Evidence):

```
GET /rest/products/search?q=%27%20or%20%271%27%3D%271
! ' 200 {"status":"success","data":[...]}
Vollständiger Katalog: Apple Juice, Apple Pomace, Banana Juice, Basil Smoothie, ...
50+ Einträge inkl. nicht gelisteter Artikel
```

Auswirkung:

Datenextraktion über den ungeschützten Endpoint; mittels UNION-/Boolean-basierter Payloads sind weitergehende Datenbankabfragen (z. B. Benutzertabelle) plausibel. Offenlegung nicht öffentlich gelisteter Produkte.

Empfehlung:

Parametrisierte Queries mit ORM-Platzhaltern; Eingabevalidierung für Suchparameter; keine SQL-Fehlerdetails an Clients.

F-04 – Serverseitige JavaScript-Injection (`\$where`) im Order-Tracking exfiltriert alle Bestellungen

Feld	Wert
Schwere	Hoch
CWE	CWE-94
Endpunkt	GET /rest/track-order/<id>
Status	Bestätigt

Beschreibung:

Der Pfadparameter des Order-Tracking wird in eine marsdb-**\$where**-Klausel interpoliert, die als JavaScript ausgewertet wird (der Stack-Trace belegt die **\$where**-Auswertung). Eine **||**-Tautologie gibt alle Bestellungen zurück, einschließlich fremder Kunden mit E-Mail, Bestellinhalt und Gesamtsumme. Beliebiger JavaScript-Code im **\$where**-Kontext öffnet zudem den Weg zur weitergehenden Datenextraktion.

Nachweis (Evidence):

```
GET /rest/track-order/1'%7C%7C'1'%3D%3D'1
```

```
!' 200 mit ALLEN Bestellungen, u. a.:
{"orderId":"fe01-2a68d285297105c7","email":"d*m*","totalPrice":30.92,"products":[...]}
Kontrollprobe:
GET /rest/track-order/1'%20or%20'1'='1
!' 500 SyntaxError, Stack: "at Object.$where
(/juice-shop/node_modules/marsdb/dist/DocumentMatcher.js:419:23) ...
at /juice-shop/build/routes/trackOrder.js:50:29"
```

Auswirkung:

Massenexfiltration sämtlicher Bestelldaten (PII, Warenkörbe, Summen) ohne Authentifizierung; beliebiger JS-Code im **\$where**-Kontext ermöglicht potenziell weiterreichende Extraktion aus der Datensammlung.

Empfehlung:

Benutzereingaben niemals in **\$where**-/JS-Ausdrücke interpolieren; exakte, typsichere ID-Abfrage verwenden; Eingabevalidierung auf das erwartete ID-Format (z. B. Hashid-Muster).

F-05 – IDOR: Unauthentifzierter Zugriff auf beliebige Order-Tracking-Einträge

Feld	Wert
Schwere	Hoch
CWE	CWE-639
Endpunkt	GET /rest/track-order/<id>
Status	Bestätigt

Beschreibung:

Der Order-Tracking-Endpoint führt keine Authentifizierung und keine Ownership-Prüfung durch. Beliebige Order-IDs (kleine sequenzielle Werte oder anderweitig erlangte Hashids) lassen sich von jedermann tracken; dabei werden Kunden-E-Mail (teilweise maskiert), Bestellinhalt, Summe und Lieferstatus offengelegt.

Nachweis (Evidence):

```
GET /rest/track-order/1 (ohne Authorization-Header)
!' 200 {"status":"success","data":[{"orderId":"1"}]}
GET /rest/track-order/5267-89af6629b3a7958c (ohne Authorization-Header)
```

```
! ' 200 {"orderId":"5267-89af6629b3a7958c","email":"*dm*n@j**c*-sh.*p",
"totalPrice":8.96,"products":[{"quantity":3,"name":"Apple Juice (1000ml)",...}],
"eta":"5","delivered":false}
```

Auswirkung:

PII- und Bestelldaten fremder Kunden sind durch Erraten von IDs oder Auslesen aus anderen Quellen durch beliebige Dritte zugänglich (Datenschutz-/DSGVO-Relevanz).

Empfehlung:

Authentifizierung erzwingen und Ownership verifizieren (Order gehört zum aufrufenden Benutzer); nicht erratbare, kryptografische Order-IDs verwenden; PII nur minimal zurückgeben.

F-06 – Null-Byte-%00-Extension-Whitelist-Bypass am `/ftp`-FileServer !' beliebige Datei-Offenlegung

Feld	Wert
Schwere	Hoch
CWE	CWE-436
Endpunkt	GET /ftp/<datei>%2500.md
Status	Bestätigt

Beschreibung:

Die Whitelist des fileServer prüft nur die Dateieindung nach dem URL-kodierten Null-Byte: Direkte Anfragen auf nicht freigegebene Dateien werden mit 403 abgewiesen, während %2500.md die Prüfung passiert und die Originaldatei ausgeliefert wird. Betroffen sind Backups (`package.json.bak`, `coupons_2013.md.bak`), kompilierter Krypto-Code (`encrypt.pyc`) und sonstige Dateien im `/ftp`-Verzeichnis. (Pfad-Traversal mit `../` wurde korrekt mit 403 blockiert; der Null-Byte-Trick genügt jedoch für die Offenlegung.)

Nachweis (Evidence):

```
GET /ftp/eastere.gg                !' 403 "Error: Only .md and .pdf files are
allowed!"
GET /ftp/eastere.gg%2500.md        !' 200 (Dateiinhalt inkl. Base64-Blob)
GET /ftp/coupons_2013.md.bak%2500.md !' 200 (Coupon-Daten)
```

```
GET /ftp/encrypt.pyc%2500.md           !' 200 (Python-Bytecode der
Verschlüsselungsroutine)
GET /ftp/package.json.bak%2500.md      !' 200 (express-jwt 0.1.3, sanitize-html 1.4.2,
marsdb ~0.6)
```

Auswirkung:

Beliebige Dateien im `/ftp`-Verzeichnis sind unabhängig von der Endungs-Whitelist herunterladbar. Dependency-Manifeste und Bytecode ermöglichen präzises Vulnerability-Matching und Code-Analyse für Folgeketten.

Empfehlung:

Dateiendungsprüfung auf dem dekodierten, kanonischen Pfad durchführen; Null-Bytes in Pfaden ablehnen; Backup-Dateien aus auslieferbaren Verzeichnissen entfernen und sensible Dateien außerhalb des Webroots halten.

F-07 – Sensible Dateien ohne Authentifizierung über ``/ftp`` abrufbar (KeePass-Vault, vertrauliche Dokumente)

Feld	Wert
Schwere	Hoch
CWE	CWE-548
Endpunkt	GET /ftp, GET /ftp/incident-support.kdbx
Status	Bestätigt

Beschreibung:

Das `/ftp`-Verzeichnis ist unauthentifiziert mit Verzeichnislisting erreichbar und enthält direkt abrufbare sensible Dateien: eine binäre KeePass-Datenbank (`incident-support.kdbx`, vollständig heruntergeladen), vertrauliche M&A-Dokumente, die verschlüsselte Ankündigung sowie Malware-Quarantäne-Verknüpfungen. `robots.txt` (`Disallow: /ftp`) bestätigt den Pfad zusätzlich. Eine Kompromittierung des Vaults würde Incident-Response-Credentials liefern.

Nachweis (Evidence):

```
GET /ftp                               !' 200 Verzeichnislisting (auch /ftp/quarantine)
GET /ftp/incident-support.kdbx         !' 200 application/octet-stream (vollständig
heruntergeladen)
```

```
GET /ftp/acquisitions.md !' 200 "# Planned Acquisitions > This document
is confidential! Do not distribute!"
GET /ftp/announcement_encrypted.md !' 200
GET /ftp/quarantine/juicy_malware_windows_64.exe.url !' 200
robots.txt: "User-agent: * Disallow: /ftp"
```

Auswirkung:

Offline-Angriff auf die Vault-Passphrase !' Kompromittierung von Incident-Response-Credentials;
Vertraulichkeitsverletzung bei M&A-Dokumenten; Malware-Verknüpfungen als weitere Angriffsvektoren.

Empfehlung:

Verzeichnislisting deaktivieren, sensible Dateien aus dem Webroot entfernen, Vault und vertrauliche
Dokumente mit Zugriffs kontrolle schützen; `robots.txt` nicht als Zugriffsschutz betrachten.

F-08 – Unauthentifizierter Zugriff auf die Admin-Anwendungskonfiguration

Feld	Wert
Schwere	Hoch
CWE	CWE-306
Endpunkt	GET /rest/admin/application-configuration
Status	Bestätigt

Beschreibung:

Ein als „admin“ bezeichneter Konfigurations-Endpoint ist ohne jegliche Authentifizierung lesbar und liefert die vollständige Anwendungskonfiguration: OAuth-ClientId mit autorisierten Redirects, Chatbot-Modell, XSS-Bonus-Payload, interne Challenge-URLs, `baseUrl` und Kontakt-E-Mails. Die internen Details erleichtern weitergehende Angriffe (Redirect-Missbrauch, gezielte XSS-Payloads, LLM-Angriffsfläche).

Nachweis (Evidence):

```
GET /rest/admin/application-configuration (ohne Authorization-Header)
!' 200 vollständige Config-JSON, u. a.:
googleOAuth.clientId = "1005568560502-...apps.googleusercontent.com"
authorizedRedirects = [http://localhost:3000, http://127.0.0.1:3000, ...]
chatbot.model = "gemma4:e4b"
```

```
challenges.xssBonusPayload = "<iframe ...soundcloud...>"
baseUrl = "http://localhost:3000"
```

Auswirkung:

Detailliertes internes Wissen über OAuth-Konfiguration, Redirect-Ziele, Chatbot-Modell und vorbereitete XSS-Payloads senkt die Schwelle für Folgekompromittierungen erheblich.

Empfehlung:

Admin-Endpunkte hinter Authentifizierungs-/Autorisierungs-Middleware schützen; nur die für den Client notwendige, reduzierte Konfiguration ausliefern; keine sicherheitsrelevanten Interna preisgeben.

Mittel

F-09 – IDOR: Cross-User-Zugriff auf fremde Warenkörbe über sequenzielle IDs

Feld	Wert
Schwere	Mittel
CWE	CWE-639
Endpunkt	GET /rest/basket/<id>
Status	Bestätigt

Beschreibung:

Warenkorb-IDs sind sequenziell und der Endpoint prüft nicht, ob der Warenkorb zum Aufrufer gehört. Mit einem beliebigen gültigen Token (hier Admin-JWT mit `bid=1`) wurde der Warenkorb von UserId 2 (`jim`) vollständig inkl. Produkte und Mengen gelesen. Eine unauthentifizierte Anfrage auf `/rest/basket/2` wurde mit 401 abgewiesen – der Kernbefund (fehlende Ownership-Prüfung für beliebige gültige Token) ist durch die App-Antwort dennoch eindeutig bewiesen.

Nachweis (Evidence):

```
GET /rest/basket/2 (mit Admin-JWT, bid=1 / UserId 1)
! 200 {"data":{"id":2,"UserId":2,...,"Products":[{"name":"Raspberry Juice (1000ml)",
```

```
...,"BasketItem":{"BasketId":2,"quantity":2}}]]}}
GET /rest/basket/1 (mit Admin-JWT)
!' 200 voller Warenkorb von UserId 1 inkl. aller Items und Mengen
GET /rest/basket/2 (ohne Token) !' 401
```

Auswirkung:

Jeder authentifizierte Benutzer kann Warenkörbe fremder Benutzer (Produkte, Mengen, Coupon-Status) auslesen – Verletzung der Vertraulichkeit und des Tenant-Prinzips.

Empfehlung:

Serverseitigen Ownership-Check erzwingen (`Basket.UserId === authentifizierter Benutzer`); Zugriffe auf fremde IDs mit 403 ablehnen.

F-10 – Excessive Data Exposure auf Benutzer-APIs (vollständiges Benutzerverzeichnis, Passwort-/TOTP-Felder)

Feld	Wert
Schwere	Mittel
CWE	CWE-200
Endpunkt	GET /api/Users, GET /rest/user/authentication-details
Status	Bestätigt

Beschreibung:

Die Benutzer-APIs geben weit mehr Daten zurück als erforderlich: `/api/Users` listet mit Admin-JWT 24 Benutzer mit E-Mails, Rollen (`admin/deluxe/customer/accounting`), `deluxeTokens` und `lastLoginIp`; `/rest/user/authentication-details` liefert zusätzlich dedizierte `password`- und `totpSecret`-Felder (hier maskiert) sowie `lastLoginTime`. In Kombination mit der unauthentifzierten Login-SQLi (F-01) ergibt sich eine vollständige Konto-/PII-Kompromittierungskette.

Nachweis (Evidence):

```
GET /api/Users (mit Admin-JWT)
!' 200, 24 Benutzer: E-Mails, Rollen, deluxeTokens, lastLoginIp "123.456.789"
Beispiel: ciso@juice-sh.op !' deluxeToken
"d715c2c75d4a42d3825a050e0a0163c1959b51165373f17bd8eed7b1e05bf20d"
GET /rest/user/authentication-details
```

```
!' 200 mit Feldern "password" und "totpSecret" (maskiert
"*****") plus lastLoginTime
```

Auswirkung:

Vollständiges Benutzerverzeichnis inkl. Rollen und Deluxe-Tokens; die Serialisierung von Passwort-/TOTP-Feldern ist ein struktureller Designfehler, der bei jedem Serialisierungsfehler zur Secret-Offenlegung wird.

Empfehlung:

API-Antworten auf minimal notwendige Felder beschränken (Whitelist-Serialisierung); Passwort-/TOTP-Felder niemals serialisieren; Admin-Endpunkte zusätzlich hart autorisieren.

F-11 – PII-, Finanz- und Secret-Daten über CRUD-APIs offengelegt (Cards, Complaints, Feedbacks, Challenges)

Feld	Wert
Schwere	Mittel
CWE	CWE-200
Endpunkt	GET /api/Cards, /api/Complaints, /api/Feedbacks, /api/Challenges
Status	Bestätigt

Beschreibung:

Breit zugängliche CRUD-APIs legen mit Admin-JWT gespeicherte Kartendaten (maskiert), Kundenbeschwerden mit UserId-Verknüpfung, sämtliches Feedback sowie interne Challenge-/Flag-Metadaten offen. Im Feedback steht eine **live Wallet-Seed-Phrase** – ein unmittelbar verwertbares Secret. Der Admin-Token ist wiederum unauthentifiziert via Login-SQLi (F-01) erlangbar.

Nachweis (Evidence):

```
GET /api/Cards      !' 200 [{"fullName":"Administrator","cardNum":"*****4368","expMonth":2,"expYear":2081}, {"... ****8108}]
GET /api/Complaints !' 200 Kundenbeschwerde mit UserId-Verknüpfung
GET /api/Feedbacks  !' 200 inkl. Wallet-Seed-Phrase: "purpose betray marriage blame crunch monitor spin slide donate sport lift clutch"
```

GET /api/Challenges !' 200 interne Challenge-/Flag-Metadaten (inkl. ALCHEMY_API_KEY-Status)

Auswirkung:

Die Seed-Phrase ermöglicht direkten Wallet-Zugriff (finanzieller Schaden möglich); Kartendaten, Beschwerden (PII) und interne Metadaten sind massenweise auslesbar.

Empfehlung:

Dateminimierung und feldbasierte Autorisierung pro API; Secrets niemals in Feedback-/Kommentarfeldern speichern bzw. bei Ausgabe maskieren; Challenge-Metadaten nicht über Produkt-APIs exponieren.

F-12 – CAPTCHA-Antwort von API offengelegt (Anti-Automation wirkungslos)

Feld	Wert
Schwere	Mittel
CWE	CWE-804
Endpunkt	GET /rest/captcha
Status	Bestätigt

Beschreibung:

Der CAPTCHA-Endpoint liefert die Klartext-Antwort zusammen mit der Challenge aus. Die Anti-Automatisierungskontrolle bietet damit keinen Schutz und lässt sich skriptbasiert lösen – praktisch bestätigt, da der automatisierte Feedback-POST (201) genau diese Antwort verwendete.

Nachweis (Evidence):

```
GET /rest/captcha
!' 200 {"captchaId":0,"captcha":"10-3-10","answer":"-3"}
```

Auswirkung:

Automatisiert erzeugbare Massen-Einträge (Feedback, Beschwerden) !' Spam, Datenpollution und Ressourcenmissbrauch; alle darauf aufbauenden Rate-Limit-Ansätze sind wirkungslos.

Empfehlung:

Antwort niemals an den Client ausliefern; serverseitige Verifikation mit zeitlich begrenzten, einmaligen Challenges; ggf. externen CAPTCHA-Dienst nutzen.

F-13 – Unbeschränkter Datei-Upload akzeptiert skripthaltige SVGs (Stored-XSS-Fläche)

Feld	Wert
Schwere	Mittel
CWE	CWE-434
Endpunkt	POST /file-upload
Status	Bestätigt (Konfidenz der Ausführung: tentativ)

Beschreibung:

Der Upload-Endpoint akzeptiert mehrfach skripthaltige SVG-Multipart-Uploads ohne Zurückweisung (204) – die fehlende Inhaltsvalidierung ist durch echte App-Antworten belegt. Die Persistenz/Ausführung als Stored XSS konnte jedoch nicht bestätigt werden, da die Verifikationsanfragen SPA-Fallback-HTML statt der SVG-Bytes lieferten; der Befund bleibt daher in der Auswirkung tentativ. Ein WAF war nicht beteiligt (die Canary-Probe lieferte eine unauffällige 200-Baseline).

Nachweis (Evidence):

```
POST /file-upload (multipart, SVG mit <script>-Payload) !' 204 (mehrfach, auch bei "Real multipart upload")
GET /assets/public/images/uploads/t7.svg      !' 200, jedoch SPA-Fallback-HTML statt SVG-Bytes
GET /assets/public/images/uploads/p6xss.svg  !' 200, jedoch SPA-Fallback-HTML statt SVG-Bytes
```

Auswirkung:

Wird die Datei unter ihrem Namen als `image/svg+xml` ausgeliefert, führt der Aufruf der URL zu Script-Ausführung im Kontext der Origin (Stored XSS). Die Ausführung konnte im Test nicht abschließend verifiziert werden.

Empfehlung:

Inhaltsvalidierung und Allow-Listing für Upload-Typen; SVG sanitisieren oder als `text/plain` ausliefern; Uploads außerhalb des ausführbaren Webroots bzw. über isolierte Origin ausliefern; Dateinamen serverseitig neu vergeben.

Niedrig

F-14 – Unauthentifizierte Sicherheitsfragen-Enumeration (Benutzer-Enumeration)

Feld	Wert
Schwere	Niedrig
CWE	CWE-200
Endpunkt	GET /rest/user/security-question
Status	Bestätigt

Beschreibung:

Der Endpoint verrät ohne Authentifizierung, ob eine E-Mail registriert ist, und offenbart die zugehörige Sicherheitsfrage. Dies ermöglicht Benutzer-Enumeration sowie gezielte Angriffe auf den Passwort-Reset bzw. Social Engineering.

Nachweis (Evidence):

```
GET /rest/user/security-question?email=admin@juice-sh.op (ohne Auth)
! ' 200 {"question":{"id":2,"question":"Mother's maiden name?","..."}}
```

Auswirkung:

Registrierte Accounts sind identifizierbar; die Sicherheitsfrage unterstützt gezielte Reset- und Social-Engineering-Angriffe.

Empfehlung:

Uniforme Antworten unabhängig von der Registrierung erteilen; Sicherheitsfrage erst nach verifizierter Identität und mit Rate-Limiting preisgeben; Lockout-Mechanismen ergänzen.

F-15 – Versions-Offenlegung über unauthentifizierten Admin-Endpoint

Feld	Wert
Schwere	Niedrig
CWE	CWE-200
Endpoint	GET /rest/admin/application-version
Status	Bestätigt

Beschreibung:

Der Admin-Versions-Endpoint ist unauthentifiziert abrufbar; Fehlerseiten bannen zusätzlich Framework- und Versionsinformationen. Dies erlaubt präzises Fingerprinting für Vulnerability-Matching.

Nachweis (Evidence):

```
GET /rest/admin/application-version (ohne Auth) !' 200 {"version":"20.2.0"}  
Fehlerseiten-Banner: "OWASP Juice Shop (Express ^4.22.1)"
```

Auswirkung:

Gezieltes CVE-Matching gegen die exakt ermittelte Versionsmatrix (App, Express, npm-Dependencies aus F-06).

Empfehlung:

Versions-Endpoint authentifizieren; Versionsinformationen und Framework-Banner aus Client-sichtbaren Antworten entfernen.

F-16 – Ausführliche Fehlerseiten mit internen Stack-Traces und Dateisystempfaden

Feld	Wert
Schwere	Niedrig

CWE	CWE-209
Endpunkt	Diverse (z. B. GET /ftp/coupons_2013.md.bak)
Status	Bestätigt

Beschreibung:

Fehlerantworten (403/401/500) rendern vollständige Stack-Traces mit internen Dateisystempfaden und Bibliotheksdetails. Dies offenbart interne Routen, Bibliotheksversionen und die Codestruktur und beschleunigt die weitere Ausnutzung erheblich.

Nachweis (Evidence):

```
403/401/500-Antworten mit vollen Stack-Traces, z. B.:
"at verify (/juice-shop/build/routes/fileServer.js:68:18)"
"/juice-shop/node_modules/serve-index/index.js:149:39"
"/juice-shop/build/lib/insecurity.js:218:5"
"UnauthorizedError: error:1E08010C:DECODER routines::unsupported"
```

Auswirkung:

Detailliertes internes Wissen über Routing, Bibliotheken und Codestruktur; direkte Unterstützung der Befunde F-04 und F-06.

Empfehlung:

Generische Fehlerseiten an Clients ausliefern; detaillierte Logs ausschließlich serverseitig führen; **errorhandler** in Produktion deaktivieren.

F-17 – Fehlende Security-Header und Wildcard-CORS

Feld	Wert
Schwere	Niedrig
CWE	CWE-693
Endpunkt	Global (z. B. GET /)
Status	Bestätigt

Beschreibung:

Antworten enthalten weder Content-Security-Policy, Strict-Transport-Security, Referrer-Policy noch Permissions-Policy; zudem ist `access-control-allow-origin: *` gesetzt und `cache-control: public, max-age=0`. Die fehlenden Header verstärken die bestätigten Injection-/XSS-Befunde, und Wildcard-CORS erlaubt jedem Origin das Lesen von API-Antworten.

Nachweis (Evidence):

```
Response-Header auf GET /:  
strict-transport-security:      (fehlt)  
content-security-policy:       (fehlt)  
referrer-policy:               (fehlt)  
permissions-policy:            (fehlt)  
access-control-allow-origin: *  
cache-control: public, max-age=0
```

Auswirkung:

Ohne CSP bleibt injiziertes Script ausführbar (Hebel für F-13); eine beliebige Dritt-Origin kann per CORS API-Antworten (z. B. mit Token) lesen; cachebare sensible Antworten erhöhen Leak-Risiken.

Empfehlung:

CSP, HSTS, Referrer-Policy und Permissions-Policy setzen; CORS auf explizit vertrauenswürdige Origins beschränken; sensible API-Antworten nicht cachebar ausliefern.

F-18 – Öffentlich exponierte Swagger/OpenAPI-Dokumentation der B2B-Order-API

Feld	Wert
Schwere	Niedrig
CWE	CWE-200
Endpunkt	GET /api-docs/
Status	Bestätigt

Beschreibung:

Die Swagger-UI und das vollständige OpenAPI-3.0-Spec (in `swagger-ui-init.js`) sind öffentlich abrufbar: Server `/b2b/v2`, `POST /orders` inkl. Schemas und Beispielen mit `couponCode` sowie

customerReference-Formaten. Die Dokumentation erweitert die Angriffsfläche und leakt interne Order-/Coupon-Formate für Business-Logic-Angriffe.

Nachweis (Evidence):

```
GET /api-docs/ !' 200 Swagger-UI
GET /api-docs/swagger-ui-init.js !' 200 vollständiges OpenAPI-3.0-Spec
server: /b2b/v2; POST /orders; orderLinesData-Beispiel mit couponCode "pes[Bh.u*t"
```

Auswirkung:

Vollständige API-Karte für Angreifer; interne Coupon-/Referenzformate ermöglichen gezielte Business-Logic-Missbrauchstests (z. B. Coupon-Injection in orderLinesData).

Empfehlung:

API-Dokumentation in Produktion deaktivieren oder hinter Authentifizierung stellen; keine internen Format-/Coupon-Beispiele in öffentlich zugänglichen Specs.

F-19 – Unauthentifizierter Prometheus-`/metrics`-Endpoint

Feld	Wert
Schwere	Niedrig
CWE	CWE-200
Endpoint	GET /metrics
Status	Bestätigt

Beschreibung:

Der Prometheus-Endpoint ist unauthentifiziert abrufbar und offenbart Prozess-/Node-Metriken (Speicher, File-Deskriptoren, Event-Loop-Lag), HTTP-Request-Zähler nach Statusklassen, LLM-Token-Counter und Startup-Dauern. Dies ermöglicht Infrastruktur-Fingerprinting, Traffic-Analyse und die Identifikation interner Komponenten (LLM-Integration).

Nachweis (Evidence):

```
GET /metrics !' 200 text/plain
Prozess-/Node-Metriken (Speicher, FDs, Event-Loop-Lag);
```

http_requests_count (2XX:50, 3XX:34, 4XX:6, 5XX:4);
LLM-Token-Counter; Startup-Dauern (z. B. datacreator 20.62s)

Auswirkung:

Recon über Last, Komponenten und Fehlerquoten; Bestätigung einer LLM-Integration als zusätzliche Angriffsfläche.

Empfehlung:

`/metrics` nur intern (separater Port/Netzwerk oder IP-Allowlist) exponieren; Authentifizierung für Monitoring-Endpunkte erzwingen.

Informationell

F-20 – Versteckte servergerenderte Seite und öffentlich exponiertes `/assets/private``-Verzeichnis

Feld	Wert
Schwere	Informationell
CWE	CWE-538
Endpunkt	<code>/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg</code>
Status	Bestätigt

Beschreibung:

Der Base64-Blob in `/ftp/eastere.gg` (via Null-Byte-Bypass, F-06, erhalten) dekodiert zu einem versteckten, servergerenderten Pfad außerhalb der SPA. Die Seite „Welcome to Planet Orangeuze“ ist unauthentifiziert erreichbar und lädt direkt Assets aus `/assets/private` – nicht-öffentliche Assets und eine undokumentierte Route sind öffentlich exponiert.

Nachweis (Evidence):

```
Base64-Blob aus /ftp/eastere.gg%2500.md dekodiert !' versteckter Pfad
GET /the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg
!' 200 "Welcome to Planet Orangeuze"
Lädt: /assets/private/three.js, OrbitControls.js, dat.gui.min.js, stats.min.js,
orangemap2k.avif, starry_background.avif
```

Auswirkung:

Undokumentierte Route und als privat gekennzeichnete Assets sind öffentlich abrufbar; geringes direktes Risiko, aber Hinweis auf unvollständige Trennung von öffentlichem und internem Static-Content.

Empfehlung:

Versteckte Routen entfernen oder authentifizieren; `/assets/private` außerhalb des öffentlichen Static-Roots ausliefern.

Nicht verifizierbar – durch WAF blockiert

Es existieren keine Befunde in dieser Kategorie.

Im Rahmen dieses Tests wurde **kein WAF erkannt** (`detected: false`, keine Blockierungs-Signale; die Canary-Probe lieferte eine unauffällige 200-Baseline). Sämtliche 20 Befunde wurden daher **direkt gegen die Ursprungsanwendung end-to-end verifiziert** und tragen durchgängig den Status „confirmed“. Ein Re-Test aus einer WAF-Whitelist-Quell-IP oder gegen das Origin ist zur Absicherung dieser Ergebnisse **nicht erforderlich**. Sollten in Folgeumgebungen (z. B. Staging/Produktion mit vorgeschaltetem WAF) Befunde mit Status `unverifiable_waf` auftreten, gelten diese ausdrücklich als **nicht entkräftet**: Die WAF verhinderte in diesem Fall lediglich die Validierung, und ein Re-Test aus einer Whitelist-Quell-IP oder direkt gegen das Origin wäre zwingend erforderlich.

Fazit

Der autorisierte Penetrationstest von `http://127.0.0.1:8091` ergab ein **kritisches Gesamtrisiko** mit 20 vollständig bestätigten Schwachstellen (1× kritisch, 7× hoch, 5× mittel, 6× niedrig, 1× informationell). Da kein WAF im Pfad war und keine Befunde unverifizierbar blieben, ist die Aussagekraft des Tests für den untersuchten Geltungsbereich **abschließend**.

Zentrale Angriffskette (bestätigt):

Unauthentifizierte SQL-Injection am Login (F-01) !' Admin-JWT inkl. MD5-Passwort-Hash (F-02) !' Klartext-Admin-Passwort (**admin123**) !' Zugriff auf alle Admin-/CRUD-APIs inkl. Benutzerverzeichnis, Kartendaten und einer live Wallet-Seed-Phrase (F-10, F-11). Parallel ermöglichen **/ftp**-Bypass und -Exposure (F-06, F-07) die Kompromittierung des KeePass-Incident-Response-Vaults, und die **\$where**-Injection (F-04) exfiltriert sämtliche Kundebestellungen.

Priorisierte Behebungs-Roadmap:

Priorität	Maßnahmen	Befunde
Sofort (0–7 Tage)	SQL-Injection beseitigen (parametrisierte Queries), JWT-Payload auf minimale Claims reduzieren, /ftp -FileServer sichern (Whitelist-Bypass, Listing, sensible Dateien entfernen), Admin-Endpunkte authentifizieren	F-01, F-02, F-03, F-06, F-07, F-08
Kurzfristig ("d 30 Tage)	Ownership-Prüfungen für Orders/ Warenkörbe erzwingen, API-Antworten auf Minimalfelder beschränken (Whitelist-Serialisierung), Seed-Phrase und Secrets aus Feedback entfernen, CAPTCHA-Antwort nicht ausliefern, Upload-Validierung/ SVG-Sanitizing, \$where -Interpolation entfernen	F-04, F-05, F-09, F-10, F-11, F-12, F-13
Mittelfristig	Security-Header (CSP, HSTS, Referrer-/Permissions-Policy) setzen, CORS auf vertrauenswürdige Origins beschränken, generische Fehlerseiten, Versions-/Swagger-/ Metrics-Exposure beseitigen, versteckte Routen und /assets/ private entfernen	F-14 bis F-20

Positive Feststellungen: **X-Frame-Options:** **SAMEORIGIN** und **X-Content-Type-Options:** **nosniff** sind gesetzt; Pfad-Traversal (**../**) wurde korrekt blockiert; unauthentifizierte Warenkorbzugriffe werden mit 401 abgewiesen. Diese Teilmaßnahmen zeigen, dass grundlegende Kontrollen teilweise vorhanden sind – sie reichen jedoch angesichts der systemischen Injection- und Autorisierungsdefizite bei Weitem nicht aus.

Kontext: Das Ziel wurde als OWASP Juice Shop v20.2.0 identifiziert – eine absichtlich verwundbare Trainingsanwendung. Die Befunde bestätigen die intendierten Schwachstellenklassen und eignen sich hervorragend als Lern- und Regressionsszenario. Für einen produktionsnahen Einsatz wären sämtliche oben genannten Maßnahmen zwingend umzusetzen, bevor die Anwendung einem realen Bedrohungsmodell ausgesetzt wird.

Ende des Reports – Vertraulich