CyberTestify

Automated Security Scan Report

ornek.com · Active Verification Bundle

CyberTestify Security Scan — Completed

Report No: **CT-ÖRNEK-715E**

Verification Code: **4B2E-B284**

This report is not a formal penetration test / certification.

Table of Contents

TARGET	PACKAGE	REPORT NO
ornek.com	Active Verification Bundle	CT-ÖRNEK-715E

1. Executive Summary

OVERALL ASSESSMENT — High Risk

In the checks that were run, the highest risk was detected in the Injection (SQLi/XSS) Verification area (an injection vulnerability was PROVEN via active verification.); priority remediation/verification is recommended. 8 unique pages/endpoints were scanned, a total of 149 requests were sent to 19 input points. SSRF/RCE are time-based/indirect. Each check is reported separately below.

⚡ Management Decision

2 high/critical indicator(s) requiring priority attention were found; an urgent remediation plan is recommended. Medium/low items can be addressed via planned improvement.

- **Overall risk level: High** — the highest risk is in the **Injection (SQLi/XSS) Verification** area (an injection vulnerability was PROVEN via active verification.).
- **Scope honesty:** This package focuses on the **external surface that requires no authentication** — publicly accessible endpoints (open forms/APIs, search, the login/registration flow itself). The IDOR / Business Logic / Race-Mass-Assignment checks run **only on the pre-login accessible surface** (e.g. public APIs, publicly accessible id-based endpoints); therefore, in these categories, **limited or "Not assessable"** results are normal and expected on some targets (due to a target-specific small surface, not a lack of engine). Deep in-session authorization/business-logic vulnerabilities **after** login are out of scope and are addressed in the **Full-Scope Pentest**. The strongest result of this scan was detected in the **Injection (SQLi/XSS) Verification** area.
- **API attack surface: 1** API endpoint was discovered but **requires authentication** (401/403) — as authenticated deep testing is outside the scope of this package, it was not probed (Full-Scope Pentest recommended).
- **Recommended first step:** Remediate the findings from the checks that were run; ready-made steps are in the "AI Solution Recommendations" section.

Improvement Priorities

⚡ **Most Urgent:** SQL Injection indicator; Authentication bypass indicator — ' OR 1=1-- (POST /rest/user/login).

🗓 **Medium-term / Process:** /api/products/1; Reflected XSS indicator.

2. Findings

2.1 Vulnerability Distribution

A total of **4** finding indicators were detected; the severity distribution is below.

2.2 Master Findings Table

ID	Title	State	Severity
CT-1	SQL Injection indicator	Open	High
CT-2	Authentication bypass indicator — ' OR 1=1-- — POST /rest/user/login	Open	High
CT-3	/api/products/1	Open	Medium
CT-4	Reflected XSS indicator · confidence: low (indirect indicator)	Open	Low

8 verification categories were run — 2 clean, 3 produced a finding, 3 not assessable

✓ ATTEMPTED — NO EVIDENCE OF A VULNERABILITY FOUND

- **SSRF Verification** Clean (2 input points tried, no evidence found)
- **RCE / Command Injection Verification** Clean (4 input points tried, no evidence found)

⚠ FINDING PRESENT — DETAILED IN THE SECTIONS BELOW

- **Injection (SQLi/XSS) Verification** Finding present (2 — vulnerability indicator; above)
- **Broken Access Control (IDOR) Verification** Finding present (1 — limited/indirect indicator; above)
- **Login Bypass (SQLi Indicator)** Finding present (1 — vulnerability indicator; above)

⚠ NOT ASSESSABLE — DOES NOT MEAN "SECURE"

- **File Upload Verification** Not assessable (no testable input point found — NOT "clean")
- **Business Logic Verification** Not assessable (no testable input point found — NOT "clean")
- **Race / Mass-Assignment Verification** Not assessable (no testable input point found — NOT "clean")

2.3 Detailed Findings

[HIGH] CT-1 · SQL Injection indicator

State: Open

Description: An input reaches a DB query (injection indicator).

How it was detected: A database error signature was observed in the response (with the "')" payload): "SQLITE_ERROR"

Business Impact: Unauthorized access/modification of customer/account records is possible; data breach, breach of UK GDPR / Data Protection Act 2018 and loss of trust.

RECOMMENDED FIX

Use parameterized queries/prepared statements; hide DB error messages.

Reference: CWE-89 · OWASP A03:2021 Injection

[HIGH] CT-2 · Authentication bypass indicator — ' OR 1=1--

State: Open

Affected point: `POST /rest/user/login`

Description: Auth-bypass indicator via a classic SQLi payload on the login form.

How it was detected: Control (invalid credentials) → HTTP 401 (failed). SQLi payload ' OR 1=1-- → HTTP 200 + CLEAR success signal: "{"authentication":{"token":"","bid":1,"umail":"admin@juice-sh.op"}}". Authentication is BEING BYPASSED via SQL injection (a session/authorization token was returned — strong evidence). No session takeover/exploitation was performed; the token value is not shown in the report (redacted).

Business Impact: Authentication bypass may allow unauthorized access; account and data security directly at risk.

RECOMMENDED FIX

Use parameterized queries in auth; validate input; return uniform error messages.

Reference: CWE-287 · OWASP A07:2021 Identification & Authentication Failures

[LOW] CT-4 · Reflected XSS indicator

State: Open

Description: User input reflects unencoded in HTML (reflected XSS indicator).

How it was detected: The marker string was reflected but PARTIALLY/ENCODED (special characters < > " were escaped) — a context-dependent low-confidence indicator; manual verification is recommended.

Business Impact: Running script in the victim's browser enables session theft/impersonation and account takeover.

RECOMMENDED FIX

Context-encode output (HTML entity encoding); restrict inline script via CSP.

Reference: CWE-79 · OWASP A03:2021 Injection

Positive Assurance

The following controls were executed with no significant vulnerability indicator: SSRF Verification, RCE / Command Injection Verification, File Upload Verification, Business Logic Verification, Race / Mass-Assignment Verification.

3. Controls & Methodology

Assessment Summary

All 7 active security control categories were assessed. 8 unique pages/endpoints were scanned, **19** input points tested, **149** requests sent in total. In **2** checks a high/critical-level vulnerability indicator was found (detailed below).

*Discovery method: Because a JavaScript-rendered (SPA) target was detected, this scan was performed by **rendering the pages with a headless browser**.*

CONTROLS SUMMARY

Control	Result	Confidence
Injection (SQLi/XSS) Verification	⚠ Vulnerability indicator	High
Broken Access Control (IDOR) Verification	⚠ Limited indicator	Medium

Control	Result	Confidence
SSRF Verification	✓ No vulnerability evidence	Medium
File Upload Verification	No input point (out of scope)	Out of scope
Business Logic Verification	No input point (out of scope)	Out of scope
Race / Mass-Assignment Verification	No input point (out of scope)	Out of scope
RCE / Command Injection Verification	✓ No vulnerability evidence	Medium
Login Bypass (SQLi Indicator)	⚠ Vulnerability indicator	High

Confidence is shown only for checks that actually ran (an input point was found); checks with no input point are **out of scope**. Among those tested: SSRF/RCE indirect (time-based, no OOB) → Medium; observational (File Upload/Business Logic/Race) → Low; Injection error/reflection-based → High.

POSITIVE ASSURANCE — ACTIVE VERIFICATION METHODS ATTEMPTED

Including the checks that produced no finding, each of the 8 active control categories was actually run on the discovered surface (a total of **149** requests, **8** unique pages). The table below also shows the "no finding" results transparently — how many input points were tried and in how many no evidence was found:

Control	Input points tried	Requests sent	Result
Injection (SQLi/XSS) Verification	10	121	**⚠ Finding present (2 — vulnerability indicator; above)**
Broken Access Control (IDOR) Verification	2	6	**⚠ Finding present (1 — limited/indirect indicator; above)**
SSRF Verification	2	3	**✓ Clean (2 input points tried, no evidence found)**
File Upload Verification	0	1	**⚠ Not assessable (no testable input point found — NOT "clean")**
Business Logic Verification	0	1	**⚠ Not assessable (no testable input point found — NOT "clean")**
Race / Mass-Assignment Verification	0	1	**⚠ Not assessable (no testable input point found — NOT "clean")**
RCE / Command Injection Verification	4	13	**✓ Clean (4 input points tried, no evidence found)**
Login Bypass (SQLi Indicator)	1	3	**⚠ Finding present (1 — vulnerability indicator; above)**

Three-state distinction (honesty): ✓ *Clean* = the check ran, no evidence found · ⚠ *Finding present* = detailed above · ⚠ *Not assessable* = no testable input point found (does NOT mean secure).

What this package DOES and does NOT assess

DOES (under the "prove — do not exploit" principle; harmless, non-data-altering probes): Indicators of SQLi/XSS injection, broken access control (IDOR), SSRF, file upload, business logic, race/mass-assignment and RCE/command

injection — on the surface that **does not require** authentication, across the 8 discovered pages.

DOES NOT: Data-altering/deleting exploitation, payment completion or real RCE execution are **not** performed (only indicators/evidence are collected). The IDOR / Business Logic / Race-Mass-Assignment checks run **only on the pre-login accessible surface**; deep in-session authorization/privilege-escalation/business-logic vulnerabilities **after** login are **outside** this package — they belong to the **Full-Scope Pentest** (authenticated, scope-contracted). Therefore, in these three categories, **limited or "Not assessable"** results are normal and expected on some targets (due to a target-specific small surface, not a lack of engine). "No finding" in a check **DOES NOT PROVE** it is secure, because active exploitation is deliberately kept limited/passive-safe.

Injection (SQLi/XSS) Verification

WHAT WAS CHECKED

On the input points discovered from the scanned pages (home page + internal links + well-known paths) — URL query parameters + form fields — harmless verification probes per input point:

- **SQLi (error-based):** A single quote (`'`) was injected and a database error signature (MySQL/PostgreSQL/Oracle/MSSQL/SQLite) was searched for in the response.
- **SQLi (time-based):** At points with no error, a single harmless delay probe (SLEEP) measured the response time against the baseline (blind-SQLi indicator).
- **SQLi (boolean-based):** At numeric/ID-like points, two requests with TRUE (`1=1`) and FALSE (`1=2`) conditions were sent and their responses (status + content length) compared; if the TRUE response is consistent on repeat and PERSISTENTLY different from FALSE, it is a boolean-based SQLi indicator (a stability check is performed against false positives).
- **XSS (reflected):** A unique, harmless marker string was injected and it was checked whether it is reflected **unencoded** in the response HTML (no JS was run; stored XSS was not attempted).

FINDINGS

Input Point	Type	Technique	Evidence	Confidence (rationale)	Severity
GET /rest/products/search?q	SQLi	error-based	A database error signature was observed in the response (with the "')" payload): "SQLITE_ERROR"	High — database error signature in the response (direct evidence)	High
GET /redirect?to	XSS	reflection	The marker string was reflected but PARTIALLY/ENCODED (special characters `< > "` were escaped) — a context-dependent low-confidence indicator; manual verification is recommended.	Low — reflected but encoded/escaped; a weak context-dependent indicator	Low

Scope and method: This package operates under the "prove — do not exploit" principle. The backend sent a limited number of **harmless** verification probes to the target; no data was retrieved, modified or deleted. Inter-request delays and a target-health circuit breaker (consecutive 5xx / excessive slowdown / WAF) are applied. Areas that require authentication and internal logic are outside the scope of this package.

Broken Access Control (IDOR) Verification

WHAT WAS CHECKED

On endpoints discovered from the scanned pages that contain a predictable/numeric ID (e.g. `?id=123` , `/user/45`):

- The ID value was changed **to a neighbouring value** (N-1 / N+1) and a **GET** request was sent without authentication.
- Only the response **status and size** were compared; **the returned content was not stored/quoted**.
- Returning a different, valid-looking resource was counted as an enumerable-access indicator.

FINDINGS

Endpoint	ID	Observation	Severity
/api/products/1	path-id	Without authentication, the neighbouring ID (2) returned a 200 response with the SAME STRUCTURE (JSON skeleton) but DIFFERENT CONTENT — most likely another record's data (the returned data is not shown in the report). An indicator of enumerable resource access (possible IDOR).	Medium

SCOPE (IMPORTANT)

This package operates **without authentication**. Therefore it can only detect unauthorized access to **publicly accessible, enumerable resources**. Classic IDOR (one user accessing another logged-in user's data) requires **two different accounts/sessions** and is outside the scope of this package. The absence of findings in this section **does not prove** there is no IDOR in authenticated flows — that requires a separate authenticated test (**Review required / Out of scope**).

> In addition, sequential numeric IDs (`/{1..3}`) were derived from **1** collection-like endpoint (e.g. `/rest/products`) and tested via GET using the content-difference method.

SSRF Verification

WHAT WAS CHECKED

- Parameters that could trigger a server-side fetch (url/webhook/image/redirect etc.) were identified.
- These parameters were given a delayed echo URL **under our control**; the target's response time was compared with the baseline (if the server fetches this URL, the response is delayed).
- Internal network / cloud metadata / localhost (169.254.169.254, RFC1918, 127.0.0.1 etc.) were **never** targeted (hard-guard embedded in the code).

FINDINGS

No clear vulnerability indicator was found against the harmless probes sent.

> Since no OOB verification infrastructure was used, this detection is **time-based, indirect and of medium reliability**; an additional/manual test is recommended for definitive verification.

File Upload Verification

WHAT WAS CHECKED

- A file-upload form (input type=file) was identified.
- A one-time, **harmless and non-executable (inert)**, double-extension (.php.txt) test file was sent; only the accept/reject status was observed.
- The uploaded file was **not recalled/executed** (rule embedded in the code).

FINDINGS

No clear vulnerability indicator was found against the harmless probes sent.

> No file-upload form (input type=file) and no file-upload endpoint in the network traffic was found across the 8 unique pages scanned. **Note:** The target is a JavaScript-rendered (SPA) application and this scan was performed by **rendering the pages with a headless browser**; that no testable input point was found even so shows that there really is no input point on the page after rendering (not a raw-HTML limitation — a stronger "clean" indicator; authenticated flows are nonetheless out of scope).

Business Logic Verification

WHAT WAS CHECKED

- On the home page/forms, **client-side modifiable** price/quantity fields (hidden input) were observed (observation only — no request sent).
- It was checked whether a "success/confirmation" step page is reachable **via GET only** without a precondition (step-skipping indicator).

⚠ This check sends **no state-changing request (POST/PUT/...)** — a cart/payment is **never** created/completed (rule embedded in the code).

FINDINGS

No clear vulnerability indicator was found against the harmless probes sent.

> Business-logic vulnerabilities are context-specific; this check is at the surface/indicator level. Definitive verification requires an authenticated manual test.

> No observable client-side price/quantity field or directly reachable "confirmation" step was found across the 8 unique pages scanned. **Note:** The target is a JavaScript-rendered (SPA) application and this scan was performed by **rendering the pages with a headless browser**; that no testable input point was found even so shows that there really is no input point on the page after rendering (not a raw-HTML limitation — a stronger "clean" indicator; authenticated flows are nonetheless out of scope).

Race / Mass-Assignment Verification

WHAT WAS CHECKED

- A registration/profile-like POST form was identified (payment/checkout endpoints **excluded** — blocklist embedded in the code).
- A **single** request with extra `isAdmin/role` fields added to the form was sent; only accept/reject was observed (a privilege change was **not confirmed**; **no** retry).
- The race-condition (concurrency) test was **not run automatically**, as it carries the risk of actually altering a consumable resource (note below).

FINDINGS

No clear vulnerability indicator was found against the harmless probes sent.

> The mass-assignment indicator was derived only from the first response (low confidence). For race conditions, manual verification with a safe/testable endpoint is recommended.

The race-condition (concurrency) test was **not run** in this automatic scan, as it carries the risk of actually altering a consumable resource (coupon/stock); manual verification with a safe/testable endpoint is recommended. No registration/profile form suitable for mass-assignment (non-checkout/payment) was found across the 8 unique pages scanned. **Note:** The target is a JavaScript-rendered (SPA) application and this scan was performed by **rendering the pages with a headless browser**; that no testable input point was found even so shows that there really is no input point on the page after rendering (not a raw-HTML limitation — a stronger "clean" indicator; authenticated flows are nonetheless out of scope).

RCE / Command Injection Verification

WHAT WAS CHECKED

- Input parameters that could reach a command were identified.
- Only **harmless, time-based** delay payloads (sleep) were sent; the response time was compared with the baseline (blind evidence).
- Real command execution (file read/write, network connection, reverse shell) was **never** attempted (hard-guard embedded in the code: only a fixed sleep-payload list).

FINDINGS

No clear vulnerability indicator was found against the harmless probes sent.

Since no OOB/canary infrastructure was used, this detection is **time-based, indirect and of medium reliability** (network latency can mislead); a manual test is recommended for definitive verification.

Login Bypass (SQLi Indicator)

WHAT WAS CHECKED

- The login endpoint was first sent **invalid credentials** (control); then classic SQLi payloads (`' OR '1'='1` etc.) were tried and it was observed whether — contrary to the control — a session/success (token/2xx) was returned.
- The login POST is already a permitted flow; this is a SINGLE, harmless observation.

⚠ NO session takeover/exploitation — only the observation of "whether an authentication-bypass indicator is present".

FINDINGS

Input/Endpoint	Technique	Evidence	Confidence	Side-effect risk	Severity
POST /rest/user/login	login bypass (SQLi: `' OR 1=1--`)	Control (invalid credentials) → HTTP 401 (failed). SQLi payload `' OR 1=1--` → HTTP 200 + CLEAR success signal: "{"authentication": {"token":"***","bid":1,"umail":"admin@juice-sh.op"}}". Authentication is BEING BYPASSED via SQL injection (a session/authorization token was returned — strong evidence). No session takeover/exploitation was performed; the token value is not shown in the report (redacted).	High	none	High

The indicator is based on comparison with the control attempt; definitive verification requires a manual test.

Methodology

Stage	Description
Discovery	The external surface, pages and (for SPAs) real endpoints/params from the JS bundle are extracted.
Automated detection	Deterministic, safe (read-only) indicators are sought on the discovered surface.
Manual-deterministic verification	Findings are verified with code-based rules; “prove, don't exploit”.
Authz & session	In the authenticated package, cookie/session/authorization and post-login surface are examined.
Configuration	Header, TLS, email/DNS and exposure configurations are observed.

Risk Rating Criteria

Severity	Definition
Critical	Directly/easily exploitable indicator with serious data/access impact.
High	Prominent, priority-to-fix strong indicator.
Medium	Requires attention; context-dependent verification recommended.
Low	Hardening/maturity opportunity; low priority.

Severity is a band label only (no numeric CVSS score); all findings are framed as “indicator, verification required”.

---

4. AI Fix Suggestions

This section contains remediation recommendations for the findings detected in the active verification checks that were run.

Injection (SQLi/XSS) Verification

Injection (SQLi/XSS) — remediation

- **SQLi:** Write all database queries with **parameterised queries / prepared statements**; never concatenate user input into the query as a string. If you use an ORM, avoid raw SQL concatenation. Do not show database error messages to the end user.
- **XSS:** Apply **context-appropriate output encoding** (HTML entity encoding) when printing user input to HTML; where possible use a template engine that auto-escapes. Restrict inline scripts with a `Content-Security-Policy` header.
- After remediation, re-test the same input points.

Broken Access Control (IDOR) Verification

Broken Access Control (IDOR) — remediation

- **Object-level authorization:** On every resource access, verify server-side whether the requesting user has the right to access that object (a valid ID alone is not enough).
- **Unpredictable identifiers:** Use UUIDs/random identifiers instead of sequential numeric IDs; make enumeration harder.
- Place resources that should not be public behind authentication.

SSRF Verification

SSRF — proactive hardening

- Apply a server-side **allowlist** + internal-network blocking on all fields that accept a URL from the user (proactive).
- When an external fetch is required, add scheme/host validation + timeout + size limit.

File Upload Verification

File Upload — proactive hardening

- Apply server-side type/MIME validation + allowlist + storage outside the web root at upload endpoints (proactive).

Business Logic Verification

Business Logic — proactive hardening

- Validate critical values (price/quantity) server-side; apply step-order control in multi-step flows (proactive).

Race / Mass-Assignment Verification

Race / Mass-Assignment — proactive hardening

- Apply a field allowlist (mass-assignment protection) in model binding; use an atomic/idempotent design in critical operations (proactive).

RCE / Command Injection Verification

RCE / Command Injection — proactive hardening

- Review code paths that call system commands; validate input with an allowlist, avoid shell string concatenation (proactive).
- Apply least privilege + outbound network restriction.

Login Bypass (SQLi Indicator)

Login Bypass / SQL Injection — remediation

- Use **parameterised queries / prepared statements** in authentication queries; never place user input directly into SQL.
- Input validation + safe ORM APIs; return a uniform error message on failed login.

5. Appendix — Glossary

SQL Injection	An injection flaw where user input reaches a database query.
XSS	Cross-Site Scripting — injection of malicious scripts into pages.
IDOR	Insecure Direct Object Reference — accessing others' records via ID manipulation.
SSRF	Server-Side Request Forgery.