Black-Box Web Application Penetration Testing Report

Revel Electronics Website

Target: [REDACTED]

Assessment Type: Black-Box Web Application Penetration Test
Methodology: OWASP Web Security Testing Guide

Field	Value
Prepared by	Group 01
Compiled by	Nguyen Hoang Thanh Phong
Testing date	30-06-2026
Report date	05-07-2026
Classification	Portfolio / Academic Sample
Authorization status	Authorized testing, with no destructive exploitation

Version Control

Version	Date	Author	Description
1.0	05-07-2026	Nguyen Hoang Thanh Phong	Final Report

Confidentiality Statement

This report contains sensitive security information, including vulnerability details, proof-of-concept evidence, technical observations, and remediation guidance. The report should only be shared with authorized personnel for academic review, security improvement, and remediation purposes. Unauthorized distribution or public disclosure of this report may expose the target organization to additional security risk.

Table of Contents

1. Executive Summary

This report presents the results of a black-box web application penetration test conducted against the public-facing Revel Electronics website. The assessment was performed from the perspective of an unauthenticated external attacker and followed the OWASP Web Security Testing Guide methodology.

The testing approach was divided into reconnaissance, passive testing, and limited manual active testing. Burp Suite and OWASP ZAP were used primarily for passive proxying, sitemap generation, and request/response analysis. Manual active validation was performed using controlled, low-rate, non-destructive requests.

The most significant confirmed vulnerability was an error-based SQL Injection vulnerability in the `pid` parameter of `/en/product.php`. A controlled proof-of-concept request caused the application to disclose a MariaDB version string and part of the backend SQL query in the HTTP response. This indicates that user-controlled input is processed unsafely in a SQL query and that detailed database errors are exposed to unauthenticated users.

Additional findings include missing Content Security Policy, missing HTTP Strict Transport Security, incomplete browser security headers, incorrect content type handling for a JSON-like AJAX response, third-party resource loading without Subresource Integrity, public attachment exposure, and production comments in public HTML/JavaScript.

Overall Risk Rating: High

The overall risk is rated High because the confirmed SQL Injection vulnerability is reachable from a public unauthenticated endpoint. The issue should be remediated and retested as the highest priority.

2. Scope and Rules of Engagement

2.1 Target Scope

Item	Description
Target domain	[REDACTED]
Primary target URL	https://[REDACTED]/
Resolved public IP address	[REDACTED]
Testing type	Black-box web application penetration test
Access level	Unauthenticated external user
Environment	Public production website
Methodology	OWASP Web Security Testing Guide
Tools used	Burp Suite, OWASP ZAP, browser manual testing, curl, dig

2.2 In-Scope Areas

- Public homepage and language-specific landing pages.
- Product listing and product detail pages.
- News and application pages.
- Static content pages.
- AJAX product endpoint.
- Static JavaScript and CSS resources.
- Public images, PDF attachments, and related public paths.
- HTTP response headers and HTTPS redirection behavior.
- Publicly visible client-side source code.

2.3 Out-of-Scope Activities

- Denial-of-Service testing.
- High-volume brute forcing or credential stuffing.
- Phishing or social engineering.
- Destructive exploitation or data modification.
- Database dumping or extraction of sensitive data.
- Unauthorized access to backend or administrative systems.
- Testing of unrelated third-party infrastructure.
- Mass automated payload submission or aggressive scanning.

2.4 Testing Limitation

Automated active scanning was intentionally limited because the target website appeared to apply anti-automation, request filtering, or WAF-like controls. The tester observed that abnormal request patterns may trigger blocking of the tester IP address for approximately 24 hours.

As a result, automated active scanning was not used as the main validation method. Burp Suite and OWASP ZAP were used mainly for passive proxying, sitemap generation, and manual request analysis. Active testing was limited to low-rate, non-destructive manual validation. After confirming the SQL Injection behavior, no further database extraction was performed.

3. Methodology

The assessment followed the OWASP Web Security Testing Guide and was organized into the following phases:

1. Information Gathering and Reconnaissance.
2. Configuration and Deployment Management Testing.
3. Identity, Authentication, Authorization, and Session Management Review.
4. Input Validation Testing.
5. Error Handling Review.
6. Weak Cryptography and HTTPS Review.
7. Business Logic Review.
8. Client-Side Testing.
9. AJAX/API-like Endpoint Testing.
10. Reporting and Remediation Planning.

3.1 Reconnaissance

The reconnaissance phase focused on identifying public assets, application entry points, parameters, technologies, and external dependencies. DNS lookup confirmed that `[REDACTED]` resolved to `[REDACTED]` during testing.

Observed application areas included:

Endpoint / Path	Method	Parameters	Purpose / Notes
/	GET	None	Root endpoint
/en/	GET	None	English homepage
/en/index.php	GET	None	Homepage
/en/product.php	GET	id, pid	Product listing and product detail pages
/en/news.php	GET	id	News and application category pages
/en/news-single.php	GET	conid, id	News detail page
/en/webpage.php	GET	cid, fid	Static content pages

/ajax/getProduct.php	POST	cid, lang	AJAX product content retrieval
/attachment/	GET	N/A	Public attachment paths
/js/, /css/, /layer/	GET	N/A	Static client-side resources

3.2 Passive Testing

Passive testing was performed by reviewing normal application behavior and collected traffic without launching automated exploitation. The passive phase covered:

- HTTP response header review.
- HTML source review.
- JavaScript and CSS resource identification.
- Public file and attachment inventory review.
- Third-party resource review.
- Page metadata and comment review.
- Burp Suite and OWASP ZAP sitemap review.

3.3 Manual Active Testing

Manual active testing was conducted against selected dynamic PHP parameters using low-rate and non-destructive validation. The confirmed active testing result was an error-based SQL Injection vulnerability affecting the `pid` parameter of `/en/product.php`.

4. Attack Surface Overview

4.1 Public Website Functionality

The website primarily exposes public business and product information, including corporate content, product categories, news/application content, and contact information. No public login, user account workflow, shopping cart, or payment function was identified within the reviewed evidence.

4.2 Dynamic PHP Parameters

Parameter	Observed In	Security Relevance
id	product.php, news.php, news-single.php	Content lookup and category selection
pid	product.php	Product/category relationship lookup; confirmed SQL Injection point
cid	webpage.php, ajax/getProduct.php	Content/category identifier
fid	webpage.php	Page/content identifier
conid	news-single.php	News content identifier
lang	ajax/getProduct.php	Language selection

4.3 Static and Public Resources

The assessment identified public static and attachment paths, including `/css/`, `/js/`, `/layer/`, `/images/`, `/attachment/albumCateCover/`, `/attachment/newsPdf/`, and `/attachment/productPdf/`. These paths appear to support intended public website functionality, but they should be governed to prevent accidental exposure of internal files.

5. Positive Security Observations

ID	Observation	Evidence
POS-01	HTTP traffic redirects to HTTPS.	E-05
POS-02	HTTPS root path redirects to `/en/`.	E-06
POS-03	`X-Frame-Options: SAMEORIGIN` is present on sampled responses.	E-07, E-08
POS-04	`X-Content-Type-Options: nosniff` is present on sampled responses.	E-07, E-08
POS-05	No public authentication, payment, shopping cart, or user dashboard function was identified in the reviewed evidence.	E-02, E-03
POS-06	Anti-automation or WAF-like request filtering was observed and considered during testing.	E-16

6. Summary of Findings

ID	Finding Title	Severity	Status	Evidence
WEB-01	Error-Based SQL Injection in Product Page Parameter	High	Confirmed	E-09, E-10, E-11
WEB-02	Missing Content Security Policy Header	Medium	Confirmed	E-07, E-08
WEB-03	Missing HTTP Strict Transport Security Header	Low	Confirmed	E-05, E-06, E-07, E-08
WEB-04	Missing Referrer-Policy and Permissions-Policy Headers	Low	Confirmed	E-07, E-08
WEB-05	JSON-like AJAX Response Served with Incorrect Content Type	Low	Confirmed	E-12
WEB-06	Third-Party Resources Loaded Without Subresource Integrity	Low	Confirmed	E-13
WEB-07	Public Attachment Files and Paths Identified	Informational	Confirmed	E-04, E-14
WEB-08	Development Comments Present in Production HTML/JavaScript	Informational	Confirmed	E-15
WEB-09	Client-Side Dependency Review Required	Informational	Observed	E-04, E-13

7. Detailed Technical Findings

WEB-01: Error-Based SQL Injection in Product Page Parameter

Field	Value
Severity	High
Status	Confirmed
Affected Component(s)	/en/product.php parameter `pid`
OWASP WSTG Mapping	WSTG-INPV-05 - Testing for SQL Injection

Description

The product page was vulnerable to error-based SQL Injection. A controlled non-destructive payload submitted to the `pid` parameter caused the application to return a database error in the HTTP response. The error disclosed an XPath syntax error, the MariaDB version string, and part of the backend SQL query.

This confirms that user-controlled input is processed unsafely inside a backend SQL statement. The vulnerability is reachable from a public unauthenticated endpoint.

Evidence

The vulnerability was confirmed by comparing a normal baseline request with a modified request containing a controlled error-based SQL Injection payload.

- Appendix C, Evidence E-09: Baseline Product Page Response.
- Appendix C, Evidence E-10: SQL Injection Error Response.
- Appendix D, Evidence E-11: Raw HTTP Request and Response for SQL Injection validation.

```
Baseline request:
GET /en/product.php?id=152&pid=61 HTTP/2
Host: [REDACTED]

Modified request:
GET /en/product.php?id=152&pid=61+and+updatexml(1,concat(0x7e,version(),0x7e),1)--+- HTTP/2
Host: [REDACTED]

Observed error:
SQL Error:
XPATH syntax error: '~5.5.65-MariaDB~'

Observed query disclosure:
select * from album_cate where ac_lang='en' and ac_id=61 ...
```

Impact

An unauthenticated attacker may be able to manipulate backend SQL queries. Depending on database permissions and application logic, successful exploitation may allow database metadata enumeration, unauthorized data access, or data manipulation. The error response also discloses the backend DBMS version and query structure, which can assist further targeted attacks.

Root Cause

The application appears to concatenate user-controlled input into a SQL query without proper parameterization. Detailed database error messages are also displayed directly in production HTTP responses.

Recommendation

- Use parameterized queries or prepared statements for all database operations.
- Enforce strict server-side validation for numeric parameters such as `id`, `pid`, `cid`, `fid`, and `conid`.
- Reject unexpected input types before database interaction.
- Disable detailed SQL and PHP error output in production.
- Return generic error responses to users and log detailed errors only on the server side.
- Review all dynamic PHP endpoints for similar SQL Injection weaknesses.

Retest Guidance

Repeat the same controlled test after remediation. The application should not return SQL errors, database version information, or query fragments. Invalid input should be rejected safely or handled with a generic error response.

WEB-02: Missing Content Security Policy Header

Field	Value
Severity	Medium
Status	Confirmed
Affected Component(s)	Public HTML pages
OWASP WSTG Mapping	Configuration and Deployment Management Testing; Client-Side Testing

Description

The tested HTTP responses did not include a `Content-Security-Policy` header. CSP is a browser-side defense-in-depth mechanism that restricts where scripts, styles, images, frames, and other resources can be loaded from.

Evidence

Sampled responses contained `X-Frame-Options` and `X-Content-Type-Options`, but no `Content-Security-Policy` header was observed. See Appendix D, Evidence E-07 and E-08.

```
server: nginx
content-type: text/html; charset=UTF-8
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff

Not observed:
Content-Security-Policy
```

Impact

If a reflected, stored, or DOM-based XSS vulnerability exists, the absence of CSP may make exploitation easier. CSP does not replace secure coding, but it can reduce the impact of client-side injection vulnerabilities.

Recommendation

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://cdnjs.cloudflare.com; style-src 'self' 'unsafe-inline' https://fonts.googleapis.com https://cdnjs.cloudflare.com; font-src 'self' https://fonts.gstatic.com https://cdnjs.cloudflare.com; img-src 'self' data:; object-src 'none'; frame-ancestors 'self'; base-uri 'self';
```

The final policy should be tested in report-only mode before enforcement. Where possible, remove `unsafe-inline` and use nonces or hashes.

Retest Guidance

Collect response headers from major public pages and confirm that `Content-Security-Policy` is present and does not break legitimate functionality.

WEB-03: Missing HTTP Strict Transport Security Header

Field	Value
Severity	Low
Status	Confirmed
Affected Component(s)	HTTPS responses
OWASP WSTG Mapping	Testing for Weak Transport Layer Security; Configuration and Deployment Management Testing

Description

The website redirects HTTP traffic to HTTPS, which is positive. However, sampled HTTPS responses did not include the `Strict-Transport-Security` header. HSTS instructs browsers to use HTTPS for future requests to the site for a defined period.

Evidence

HTTP-to-HTTPS redirection was observed, but HSTS was not present on sampled HTTPS responses. See Appendix D, Evidence E-05, E-06, E-07, and E-08.

```
HTTP/1.1 301 Moved Permanently
Location: [REDACTED]

Not observed on HTTPS responses:
Strict-Transport-Security
```

Impact

Without HSTS, first-time visitors may be more exposed to downgrade or SSL stripping attacks in hostile network environments.

Recommendation

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

Only add the `preload` directive after confirming that all required subdomains support HTTPS and that preload requirements are acceptable operationally.

Retest Guidance

Verify that `Strict-Transport-Security` is present on HTTPS responses after remediation.

WEB-04: Missing Referrer-Policy and Permissions-Policy Headers

Field	Value
Severity	Low
Status	Confirmed
Affected Component(s)	Public HTTP responses
OWASP WSTG Mapping	Configuration and Deployment Management Testing

Description

The sampled responses did not include `Referrer-Policy` or `Permissions-Policy` headers. These headers are part of modern browser hardening and help control referrer leakage and access to browser features such as camera, microphone, and geolocation.

Evidence

See Appendix D, Evidence E-07 and E-08. The sampled responses include `X-Frame-Options` and `X-Content-Type-Options`, but `Referrer-Policy` and `Permissions-Policy` were not observed.

```
Not observed:
Referrer-Policy
Permissions-Policy
```

Impact

The direct risk is low for the tested website, but these headers provide defense-in-depth and reduce unnecessary browser data exposure.

Recommendation

```
Referrer-Policy: strict-origin-when-cross-origin
Permissions-Policy: camera=(), microphone=(), geolocation=()
```

Retest Guidance

Collect response headers and confirm that both headers are present on major public pages.

WEB-05: JSON-like AJAX Response Served with Incorrect Content Type

Field	Value
Severity	Low
Status	Confirmed
Affected Component(s)	/ajax/getProduct.php
OWASP WSTG Mapping	Configuration and Deployment Management Testing; Client-Side Testing; API Testing

Description

The AJAX endpoint `/ajax/getProduct.php` returns a JSON-like response body, but the HTTP response header declares the content as `text/html;charset=utf-8`.

Evidence

Burp Suite captured POST requests to `/ajax/getProduct.php` with `cid` and `lang` parameters. The response body contains JSON-style fields such as `html`, `name`, `cover`, and `url`, while the response header declares `text/html`. See Appendix D, Evidence E-12.

```
POST /ajax/getProduct.php HTTP/2
Host: [REDACTED]
Content-Type: application/x-www-form-urlencoded

cid=7&lang=en

Observed response header:
Content-Type: text/html;charset=utf-8

Observed response body pattern:
{"html":"<p>...</p>","name":"...","cover":"...","url":"..."}
```

Impact

Incorrect content typing may cause inconsistent browser, proxy, and security tool behavior. Because the response contains HTML that is later used by the client-side application, correct content type handling and safe rendering are important defense-in-depth controls.

Recommendation

```
Content-Type: application/json; charset=UTF-8
```

Generate JSON using safe encoding functions and sanitize any HTML content before storage or rendering.

Retest Guidance

Send a POST request to `/ajax/getProduct.php` and confirm that the response uses `application/json; charset=UTF-8`.

WEB-06: Third-Party Resources Loaded Without Subresource Integrity

Field	Value
Severity	Low
Status	Confirmed
Affected Component(s)	External CSS/font resources
OWASP WSTG Mapping	Client-Side Testing; Configuration and Deployment Management Testing

Description

The website loads third-party resources from external providers, including Google Fonts and cdnjs-hosted Font Awesome resources. The observed external stylesheet references did not include Subresource Integrity attributes.

Evidence

See Appendix D, Evidence E-13. The homepage HTML includes external resources from Google Fonts and cdnjs.

```
<link href="https://fonts.googleapis.com/..." rel="stylesheet">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.8.2/css/all.min.css">
```

Impact

If a third-party CDN resource is compromised or modified, the website may load altered content without browser integrity verification. The direct impact is lower for CSS than JavaScript, but external dependency integrity should still be controlled.

Recommendation

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.8.2/css/all.min.css"
      integrity="sha384-..."
      crossorigin="anonymous">
```

Alternatively, self-host third-party resources and manage updates through a controlled dependency process.

Retest Guidance

Review the HTML source and confirm that CDN-hosted resources include valid `integrity` and `crossorigin` attributes, or confirm that third-party resources are self-hosted.

WEB-07: Public Attachment Files and Paths Identified

Field	Value
Severity	Informational
Status	Confirmed
Affected Component(s)	/attachment/, /attachment/newsPdf/, /attachment/productPdf/
OWASP WSTG Mapping	Information Gathering; Configuration and Deployment Management Testing

Description

The public crawl identified multiple attachment paths, including product images, banner images, news PDFs, and product PDFs. These resources appear to support intended public website functionality.

Evidence

The ZAP URL inventory identified public attachment paths. See Appendix D, Evidence E-04 and E-14.

```
/attachment/albumCateCover/
/attachment/banner/
/attachment/newsPdf/
/attachment/productPdf/
```

Impact

This is not a vulnerability by itself if all files are intended to be public. However, public upload or attachment directories should be reviewed to ensure that no internal, draft, backup, or sensitive documents are exposed accidentally.

Recommendation

- Review all files under public attachment directories.
- Remove internal, backup, draft, or outdated documents.
- Disable directory listing if enabled.
- Avoid storing private documents under the web root.
- Strip unnecessary metadata from PDFs before publication.
- Apply access control or `X-Robots-Tag: noindex` where appropriate.

Retest Guidance

Verify that only intended public files are accessible and that directory listing is disabled.

WEB-08: Development Comments Present in Production HTML/JavaScript

Field	Value
Severity	Informational
Status	Confirmed
Affected Component(s)	Public HTML and JavaScript source
OWASP WSTG Mapping	Information Gathering; Client-Side Testing

Description

The reviewed HTML source contained commented-out menu structures and development-style comments. Comments are not directly exploitable in this case, but they may disclose implementation details, unused functionality, or development history.

Evidence

See Appendix D, Evidence E-15. The HTML source contains commented-out menu code.

```
<!--
<li class="has-submenu">
    ...
</li>
-->
```

Impact

Production comments may provide additional information to attackers during reconnaissance. In some cases, comments can reveal hidden functionality, internal paths, or disabled features.

Recommendation

- Remove unnecessary comments from production HTML and JavaScript.
- Keep comments in the source repository rather than in production output.
- Minify production JavaScript and CSS where appropriate.

- Avoid exposing developer names, dates, internal notes, or unused code blocks.

Retest Guidance
Review production HTML and JavaScript source and confirm that unnecessary comments have been removed.

WEB-09: Client-Side Dependency Review Required

Field	Value
Severity	Informational
Status	Observed
Affected Component(s)	JavaScript and frontend libraries
OWASP WSTG Mapping	Client-Side Testing

Description
The site loads several frontend resources, including local JavaScript files and third-party libraries. Passive testing identified `/js/jquery.js`, `/js/script.js`, `/js/slick.min.js`, and `/layer/layer.js` as part of the public client-side attack surface.

A full dependency inventory was not exported in the provided evidence. Therefore, this item is recorded as a dependency review recommendation rather than a confirmed vulnerable dependency finding.

Evidence
See Appendix D, Evidence E-04 and E-13 for observed static resources and JavaScript paths.

Impact
Outdated client-side libraries may expose users to browser-side vulnerabilities, especially when the application dynamically inserts HTML into the DOM.

Recommendation
- Identify exact versions of all JavaScript and CSS libraries.
- Upgrade unsupported or vulnerable libraries.
- Remove unused client-side libraries.
- Review custom JavaScript for unsafe DOM manipulation.
- Avoid rendering unsanitized HTML returned from AJAX responses.

Retest Guidance
After updating dependencies, review frontend functionality and re-run passive dependency checks.

8. Negative Testing Results

Area	Result
Public Authentication	No public login, registration, password reset, or user dashboard was identified in the reviewed evidence.
Session Management	No session cookie was observed in the submitted public-page header evidence.
Business Logic	No shopping cart, payment, booking, coupon, or transactional workflow was identified.
Automated Active Scan	Not performed due to WAF/anti-automation blocking risk.
Database Extraction	Not performed after SQL Injection confirmation to avoid unnecessary impact.
Destructive Testing	Not performed.

These negative results should not be interpreted as proof that the application is free of vulnerabilities. They reflect only the reviewed public scope and the limitations of this black-box assessment.

9. Risk Rating Rationale

Severity	Definition
Critical	May lead to full system compromise, large-scale sensitive data exposure, or remote code execution.
High	May allow unauthorized access to data, serious server-side exploitation, or significant compromise of application security.
Medium	Increases exploitability or weakens important security controls, often requiring additional conditions.
Low	Limited direct impact but should be remediated as part of security hardening.
Informational	Observation that improves security posture but does not represent direct exploitability.

The overall risk is High due to the confirmed unauthenticated error-based SQL Injection vulnerability. While several other findings are Low or Informational, SQL Injection on a public endpoint represents a serious server-side application security weakness.

10. Remediation Priority and Plan

Priority	Findings	Recommended Timeline
P1	WEB-01 Error-Based SQL Injection	Immediate / 0-7 days
P2	WEB-02 Missing CSP; WEB-05 Incorrect AJAX Content Type; WEB-09 Dependency Review	1-4 weeks
P3	WEB-03 Missing HSTS; WEB-04 Missing Browser Hardening Headers; WEB-06 Missing SRI	1-2 months
P4	WEB-07 Public Attachment Review; WEB-08 Production Comment Cleanup	1-3 months

10.1 Immediate Remediation

- Fix SQL Injection in `/en/product.php`.
- Review all dynamic PHP endpoints for similar SQL Injection flaws.
- Disable detailed SQL and PHP error output in production.
- Implement parameterized queries or prepared statements.
- Validate all numeric parameters server-side.

10.2 Short-Term Remediation

- Implement Content Security Policy.
- Correct the AJAX endpoint content type.
- Review client-side dependency versions.
- Review public attachment directories.
- Add missing browser hardening headers.

10.3 Long-Term Remediation

- Establish a secure coding standard for PHP database access.
- Add security testing to the deployment process.
- Conduct periodic manual penetration testing.
- Implement dependency management and patch review.
- Create a formal retesting process after remediation.

11. Retest Plan

Finding	Retest Activity	Expected Result
WEB-01 SQL Injection	Repeat controlled SQLi test against `pid`.	No SQL error, no DB version leakage, no query disclosure.
WEB-02 Missing CSP	Review HTTP headers.	CSP header is present and functional.
WEB-03 Missing HSTS	Review HTTPS headers.	HSTS header is present.
WEB-04 Missing Browser Headers	Review HTTP headers.	Referrer-Policy and Permissions-Policy are present.
WEB-05 AJAX Content Type	Test `/ajax/getProduct.php`.	Response uses `application/json`.
WEB-06 Missing SRI	Review HTML source.	CDN resources include SRI or are self-hosted.
WEB-07 Public Attachments	Review public file paths.	Only intended public files are accessible.
WEB-08 Production Comments	Review HTML/JS source.	Unnecessary comments are removed.
WEB-09 Dependency Review	Review JS/CSS versions.	Unsupported or vulnerable dependencies are updated.

12. Conclusion

The black-box web application penetration test identified one High-severity confirmed vulnerability and several configuration and client-side hardening weaknesses.

The most important issue is the confirmed error-based SQL Injection vulnerability in the `pid` parameter of `/en/product.php`. This vulnerability allows an unauthenticated user to trigger database errors and disclose the MariaDB version string and backend SQL query fragments. This issue should be remediated as the highest priority.

The website also lacks several modern HTTP security headers, including Content Security Policy and HTTP Strict Transport Security. Although these issues are not as severe as SQL Injection, they weaken the browser-side defense-in-depth posture of the application. The AJAX endpoint should also return the correct JSON content type, and third-party resource integrity should be improved.

Overall, the website should be considered High Risk until the SQL Injection vulnerability is remediated and retested. After remediation, a focused retest should confirm that the injection flaw is no longer exploitable and that detailed database errors are no longer exposed to users.

Appendix A: How Evidence IDs Relate to Findings

Evidence IDs are used to connect the detailed findings to the supporting screenshots, raw requests, and tool outputs. A finding ID such as `WEB-01` describes a security issue. An evidence ID such as `E-10` is the proof used to support that finding.

Item Type	Example	Meaning
Finding ID	WEB-01	A security issue or observation in the report.
Evidence ID	E-10	A screenshot, raw HTTP message, or tool output supporting a finding.
Appendix Reference	Appendix C, E-10	Where the supporting evidence is located.

Appendix B: Evidence Register and Mapping

Evidence ID	Related Finding / Section	Evidence Type	Description
E-01	Scope / Reconnaissance	DNS output	DNS lookup showing that `[REDACTED]` resolved to `[REDACTED]`.
E-02	Reconnaissance	Burp Suite screenshot	Burp sitemap showing discovered public endpoints and AJAX endpoint.
E-03	Reconnaissance	OWASP ZAP screenshot	ZAP sitemap showing public paths, static resources, and external resources.
E-04	Reconnaissance / WEB-07	ZAP URL list	URL inventory showing third-party resources, static files, and public attachment paths.
E-05	POS-01 / WEB-03	HTTP header output	HTTP to HTTPS redirection evidence.
E-06	POS-02 / WEB-03	HTTP header output	HTTPS root path redirecting to `/en/`.
E-07	WEB-02 / WEB-03 / WEB-04	HTTP header output	Homepage response headers showing existing and missing browser security headers.
E-08	WEB-02 / WEB-03 / WEB-04	HTTP header output	Product/news page response headers showing existing and missing browser security headers.
E-09	WEB-01	Screenshot	Baseline product page response.
E-10	WEB-01	Screenshot	SQL Injection error response showing MariaDB version disclosure.

E-11	WEB-01	Raw HTTP evidence	Raw request/response snippets for the SQL Injection validation.
E-12	WEB-05	Burp XML / response evidence	AJAX endpoint returning JSON-like body with `text/html` content type.
E-13	WEB-06 / WEB-09	HTML source	HTML source showing third-party resources and frontend files.
E-14	WEB-07	ZAP URL list	Public attachment paths such as `/attachment/newsPdf/` and `/attachment/productPdf/`.
E-15	WEB-08	HTML source	Production comments and commented-out HTML structures.
E-16	Testing Limitation	Tester observation	Anti-automation/WAF-like blocking behavior observed during testing.

Appendix C: Detailed Evidence Screenshots

Evidence E-02: Burp Suite Sitemap

Burp Suite was used as a passive proxy to capture normal browsing traffic and identify public application endpoints, including `/en/product.php`, `/en/news.php`, `/en/webpage.php`, and `/ajax/getProduct.php`.

[REDACTED SCREENSHOT]

Figure C-1. Burp Suite sitemap and captured public endpoints.

Evidence E-03: OWASP ZAP Sitemap

OWASP ZAP was used for passive proxying and sitemap generation. The screenshot shows public paths, third-party resources, and static resources discovered during normal browsing.

[REDACTED SCREENSHOT]

Figure C-2. OWASP ZAP sitemap and passive browsing evidence.

Evidence E-09: Baseline Product Page Response

A normal request to `/en/product.php?id=152&pid=61` returned the expected product page with HTTP 200 OK. This baseline was used to compare against the SQL Injection test response.

[REDACTED SCREENSHOT]

Figure C-3. Baseline request to the product page returning normal content.

Evidence E-10: SQL Injection Error Response

A controlled SQL Injection payload submitted to the `pid` parameter caused the application to return a database error message, disclose the MariaDB version, and reveal part of the backend SQL query.

[REDACTED SCREENSHOT]

Figure C-4. Raw Burp Repeater response showing SQL error and MariaDB version disclosure.

[REDACTED SCREENSHOT]

Figure C-5. Rendered response showing SQL error returned to the user interface.

Appendix D: Raw HTTP and Tool Outputs

Evidence E-01: DNS Lookup

```
dig [REDACTED] A +nocmd +noall +answer
[REDACTED].    21582    IN    A    [REDACTED]
```

Evidence E-05: HTTP to HTTPS Redirection

```
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Tue, 30 Jun 2026 00:53:56 GMT
Content-Type: text/html
Content-Length: 162
Connection: keep-alive
Location: [REDACTED]
```

Evidence E-06: HTTPS Root Redirection

```
HTTP/2 302
server: nginx
date: Tue, 30 Jun 2026 00:55:39 GMT
content-type: text/html; charset=UTF-8
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff
location: en/
```

Evidence E-07: Homepage Response Headers

```
HTTP/2 200
server: nginx
date: Tue, 30 Jun 2026 00:56:54 GMT
content-type: text/html; charset=UTF-8
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff
```

Evidence E-08: Product and News Page Response Headers

```
Product page headers:
HTTP/2 200
server: nginx
date: Tue, 30 Jun 2026 01:00:13 GMT
content-type: text/html; charset=UTF-8
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff

News page headers:
HTTP/2 200
server: nginx
date: Tue, 30 Jun 2026 01:02:38 GMT
content-type: text/html; charset=UTF-8
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff
```

Evidence E-11: SQL Injection Raw Request and Response Snippet

```
Modified request:
GET /en/product.php?id=152&pid=61+and+updatexml(1,concat(0x7e,version(),0x7e),1)--+- HTTP/2
Host: [REDACTED]

Observed response snippet:
SQL Error:
XPATH syntax error: '~5.5.65-MariaDB~'
```

```
Observed query snippet:
select * from album_cate where ac_lang='en' and ac_id=61 ...
```

Evidence E-12: AJAX Endpoint Content-Type Evidence

```
POST /ajax/getProduct.php HTTP/2
Host: [REDACTED]
Content-Type: application/x-www-form-urlencoded

cid=7&lang=en

Observed response header:
Content-Type: text/html;charset=utf-8

Observed body pattern:
{"html":"<p>...</p>","name":"...","cover":"...","url":"..."}
```

Evidence E-13: Third-Party Resource Evidence

```
<link
href="https://fonts.googleapis.com/css?family=Josefin+Sans:400,500,600,700|Lato|Roboto+Condensed:400,700&disp
lay=swap" rel="stylesheet">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.8.2/css/all.min.css">
<script src="../js/jquery.js"></script>
<script src="../layer/layer.js"></script>
<script src="../js/script.js"></script>
```

Evidence E-14: Public Attachment Paths

```
https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.8.2/css/all.min.css
https://[REDACTED]/attachment

. . .

https://[REDACTED]/css/style.css?v=1666845433
```

Evidence E-15: Production HTML Comment Evidence

```
<!--
<li class="has-submenu">
    <a href="#">About us</a>
    <ul class="submenu">
        ...
    </ul>
</li>
-->
```

Evidence E-16: Testing Limitation Observation

The tester observed that abnormal automated request patterns may trigger defensive blocking for approximately 24 hours. Therefore, automated active scanning was not performed, and testing was limited to passive proxying and manual low-rate validation.

Appendix E: References

- OWASP Web Security Testing Guide (WSTG): https://owasp.org/www-project-web-security-testing-guide/
- OWASP WSTG - Testing for SQL Injection: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection
- OWASP HTTP Headers Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
- OWASP HTTP Strict Transport Security Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html