pentested.com

Penetration Test Report

Example Ltd · Customer portal

November 2026

Document management

CLIENT
Example Ltd

PROJECT REF
SAMPLE-0001

TEST TYPE
Web application and API, authenticated (grey box)

TESTING WINDOW
2 to 6 November 2026, 5 tester days

SCORING
CVSS 3.1

TEST TEAM
Lead web application tester (CREST-certified), second tester for peer review

ENVIRONMENT
Production, tested 19:00 to 07:00 by agreement

VERSION
1.0, issued 9 November 2026 (initial report)

Scope

Asset	Detail
Web application	https://portal.example.com
API	https://api.example.com/v2 (41 endpoints from the supplied OpenAPI file)
Roles tested	Customer, Customer administrator, Support agent. Two accounts per role, in two separate tenants.
Out of scope	The payment provider's hosted checkout, denial-of-service testing, social engineering.

Test summary

We conducted a penetration test of the Example Ltd customer portal and its API. The purpose was to use active exploitation techniques to evaluate the security of the application against current industry standards. Access control between tenants and roles held up well under testing, and no injection weaknesses were found. One medium severity finding affects file handling, and the low severity findings are mostly missing browser security controls.

0

Critical findings

There are no critical severity findings.

0

High severity findings

There are no high severity findings.

1

Medium severity findings

Unrestricted file upload in support tickets.

7

Low severity findings

Missing security headers, legacy TLS versions, session lifetime and verbose errors.

Testing methodology

Application and API testing followed the OWASP Web Security Testing Guide. Supporting infrastructure was assessed against NIST SP 800-115. Automated tooling was used for coverage, and every finding below was confirmed by hand.

Information gathering	2 informational	**Configuration and deployment**	4 low
Identity management	No findings	**Authentication**	No findings
Authorisation	No findings	**Session management**	2 low
Input validation	No findings	**Error handling**	1 low
Cryptography	Counted under configuration	**Business logic**	1 medium
Client-side testing	No findings	**API testing**	No findings

Findings



Critical 0 High 0 Medium 1 Low 7 Info 2

ID	Finding	Severity	CVSS	Status
M-01	Unrestricted file upload in support tickets	Medium	5.4	Open
L-01	Content Security Policy not set	Low	3.1	Open
L-02	Session cookie set without SameSite	Low	3.1	Open
L-03	Verbose error messages disclose framework details	Low	3.7	Open
L-04	HTTP Strict Transport Security not set	Low	3.1	Open
L-05	TLS 1.0 and TLS 1.1 enabled	Low	3.7	Open
L-06	Software versions disclosed in response headers	Low	3.7	Open
L-07	Idle sessions remain valid for 12 hours	Low	3.1	Open
I-01	No security.txt file published	Info	0.0	Open
I-02	JavaScript library two major versions behind	Info	0.0	Open

Potential findings

P-01: The password reset endpoint returned responses around 180 ms slower for registered email addresses than for unregistered ones. The difference was not consistent enough to confirm account

enumeration within the testing window. We recommend reviewing the code path.

We recommend that critical, high and medium findings are remediated in that order. Potential findings are issues we could not validate and may need further review. Remediation support and re-testing are available for 90 days from the test date.

M-01 UNRESTRICTED FILE UPLOAD — MEDIUM

CVSS 3.1
5.4 (AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N)

CATEGORY
Business logic (WSTG-BUSL-08)

Summary

The application accepts files from users without checking what they really are. Files that should never be stored or served, such as HTML documents and scripts, can be uploaded and are then delivered to other users from the application's own domain.

Location

POST https://api.example.com/v2/tickets/{id}/attachments
GET https://portal.example.com/files/{file-id}

Discovery

The upload form limits attachments to images and PDFs, but the check is made only in the browser, against the file name. The API itself applies no restriction. Using a Customer account, the tester sent the upload request directly and stored an HTML file containing a harmless script that displays the page's domain.

The application then served the file inline, with the content type supplied at upload:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 412
Cache-Control: private
```

No Content-Disposition or X-Content-Type-Options header was returned. When a Support agent account in a second browser opened the attachment from the ticket, the script ran in the portal's origin with that agent's session. An attacker could use this to act as the support agent, who can view tickets from every tenant.

Remediation

- Validate uploads on the server against an allow list, using the file's content signature and not only its name or declared type.
- Serve attachments with `Content-Disposition: attachment` and `X-Content-Type-Options: nosniff`.
- Serve user files from a separate domain that holds no session cookies, so anything that does run cannot reach the portal.

- Rename files on upload, store them outside the web root and scan them for malware.

References

OWASP File Upload Cheat Sheet

L-01 CONTENT SECURITY POLICY — LOW

CVSS 3.1
3.1 (AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N)

CATEGORY
Configuration and deployment

Summary

A Content Security Policy (CSP) is an extra layer of defence that helps detect and reduce attacks such as cross-site scripting and content injection. It works by telling the browser which sources of scripts, styles and other content it may trust.

Location

Application wide (HTTP response headers) on portal.example.com.

Discovery

No `Content-Security-Policy` header was returned in any response. The headers below are from the dashboard page after login:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Cache-Control: no-store
Set-Cookie: portal_session=[redacted]; Path=/; Secure; HttpOnly
```

Without a policy, the browser will run any script that reaches the page. This raises the impact of finding M-01 and of any injection weakness introduced in future.

Remediation

- Configure the web server, application or load balancer to send the header on every response.
- Set `script-src` without `unsafe-inline` or `unsafe-eval`. Replace any library that requires them.
- Set `style-src` without `unsafe-inline` to prevent CSS injection.
- If inline scripts are unavoidable, use a random nonce generated for each page load.
- Deploy first as `Content-Security-Policy-Report-Only` to find breakages before enforcing.

References

OWASP Content Security Policy Cheat Sheet

L-02 SESSION COOKIE WITHOUT SAMESITE — LOW

CVSS 3.1
3.1 (AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:L/A:N)

CATEGORY
Session management (WSTG-SESS-02)

Summary

The SameSite cookie attribute controls whether a browser sends a cookie with requests that start on another website. Setting it limits cross-site request forgery, where a hostile page causes a logged-in user's browser to send a request to the application.

Location

The `portal_session` cookie, set by POST https://portal.example.com/login.

Discovery

The session cookie is set with Secure and HttpOnly, but with no SameSite attribute:

> Set-Cookie: portal_session=[redacted]; Path=/; Secure; HttpOnly

State-changing requests in the portal carry an anti-forgery token, and the tester confirmed that requests without a valid token were rejected. The finding is therefore rated low: the attribute is a missing second layer, not an exploitable gap on its own. Browser defaults differ, so the protection should be set explicitly.

Remediation

- Set `SameSite=Lax` on the session cookie, or `SameSite=Strict` if the portal is never entered from links on other sites.
- Keep the existing anti-forgery tokens. SameSite complements them and does not replace them.
- Consider the `__Host-` cookie name prefix, which stops the cookie being set by a subdomain.

References

OWASP Session Management Cheat Sheet

L-03 VERBOSE ERROR MESSAGES — LOW

CVSS 3.1
3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

CATEGORY
Error handling (WSTG-ERRH-01)

Summary

When an application fails, it should tell the user as little as possible about why. Detailed errors help developers, but they also tell an attacker which software is running and how the code is organised.

Location

GET https://api.example.com/v2/invoices/{id}, when the identifier is not a number.

Discovery

Supplying text where the API expects a numeric identifier returned an unhandled exception, including the framework name and version, internal class names and a file path on the server:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{"error":"TypeError","message":"Argument #1 ($id) must be of type int, string given",
 "file":"/srv/portal/app/Http/Controllers/InvoiceController.php","line":88,
 "framework":"[name and version redacted]"}
```

No sensitive customer data was exposed. The identified framework version had no known vulnerabilities at the time of testing, but the detail would help an attacker target the application if one is published.

Remediation

- Turn off debug mode in production.
- Return a generic error with a reference number, and write the detail to server logs only.
- Validate identifiers before they reach the controller, and return a 400 or 404 response for invalid input.

References

OWASP Error Handling Cheat Sheet