1.0

Web Application Security Assessment

Example SaaS Company

SaaS platform for document management and team collaboration.

ASSESSMENT PERIOD

2026-09-21 — 2026-09-25

REPORT DATE

2026-09-25

PREPARED BY

WappSec

ASSESSMENT TYPE

Web Application Security Assessment

01 · EXECUTIVE SUMMARY

Overview

The assessment evaluated the security of the Example SaaS Company's web application, with a focus on authentication, authorization, input handling, session management, OAuth account binding, and common web application security controls. Seven findings were identified, including weaknesses that could enable client-side code execution, unauthorized account association, and access to documents belonging to other users. Additional lower-severity issues related to redirects, session cookie configuration, error handling, and technology disclosure were also identified.

Overall Result



> The assessment identified two High-severity issues affecting the password reset and Google account binding flows. The findings demonstrate that an attacker may be able to execute JavaScript in a victim's application session and associate an attacker-controlled Google account with a victim's account. One Medium-severity issue additionally affects authorization of document access. Lower-severity findings concern redirect validation, session cookie configuration, error handling, and technology disclosure.

02 · SCOPE

In Scope

- https://example.com

Out of Scope

- Security of third-party services
- Infrastructure
- Denial-of-service testing

Test Accounts

NAME	ROLE	SUBSCRIPTION
Test User A	Regular User	Basic
Test User B	Regular User	Professional

03 · METHODOLOGY

The assessment covered the following areas:

- Authentication and Account Management
- Authorization and Access Control
- Application Workflows and Business Logic
- User Input and Data Handling
- Session and State Management
- API and Application Integrations
- Security Configuration and Error Handling

04 · FINDINGS OVERVIEW

#	TITLE	SEVERITY	STATUS
1	Reflected Cross-Site Scripting in Password Reset Flow	High	Open
2	Cross-Site Request Forgery in Google Account Binding	High	Open
3	Insecure Direct Object Reference in Document Download	Medium	Open
4	Open Redirect via Return URL Parameter	Low	Open
5	Missing SameSite Attribute on Session Cookie	Low	Open
6	Verbose Error Messages Disclose Internal Application Details	Low	Open
7	Technology Stack Information Disclosure	Informational	Informational

05 · DETAILED FINDINGS

1. Reflected Cross-Site Scripting in Password Reset Flow

High

STATUS
Open

SCOPE
https://example.com/password/reset?k=

Description

A user-controlled parameter in the password reset flow is reflected into client-side JavaScript without appropriate output encoding. A crafted value can break out of the intended string context and execute JavaScript in the application's origin. The application uses cookie-based authentication; exploitation does not depend on tokens being stored in browser storage.

Impact

A victim opening a malicious password-reset URL while authenticated may execute attacker-controlled JavaScript in the application's security context. Depending on the victim's privileges and application functionality, the script could perform actions available to the victim, access data exposed to JavaScript, or modify application state.

Steps to reproduce

1. Open the password reset page with a crafted value in the `k` parameter.
2. Observe that the value is reflected into a JavaScript context.
3. Supply a value that terminates the intended string and introduces JavaScript.
4. Confirm execution in the `example.com` origin using a harmless test message.

Evidence

REQUESTS

```
GET /password/reset?k=<crafted-value> HTTP/1.1
Host: example.com
```

RESPONSES

```
HTTP/1.1 200 OK
The response contains the supplied value inside a JavaScript context without appropriate encoding.
```

Recommendation

Apply context-appropriate output encoding, avoid concatenating untrusted input into executable JavaScript, and use safe serialization or DOM APIs where possible. Add XSS regression tests for the password reset flow.

2. Cross-Site Request Forgery in Google Account Binding

High

STATUS

Open

SCOPE

Google account binding functionality under the account settings / connected accounts section.

Description

The Google account binding flow does not sufficiently verify that the binding request was initiated by the currently authenticated user. It does not use an OAuth `state` value securely generated for the user's session and validated at the callback. An authorization response for an attacker-controlled Google account can consequently be associated with a victim's application session.

Impact

An attacker may be able to bind an attacker-controlled Google account to a victim's application account without the victim intentionally initiating the binding. If the application permits authentication through that linked account, it could provide another path to the victim's account. The exact impact depends on session and linked-authentication handling.

Steps to reproduce

1. Log in using a test victim account and navigate to Google account binding.
2. Start the Google authorization flow and identify the absence of a securely validated, session-bound `state` value.
3. Complete authorization using a Google account controlled by the tester.
4. Cause the authorization response to be processed in the victim's authenticated browser session.
5. Confirm that the Google account becomes associated with the victim's application account.

Evidence

REQUESTS

```
GET /account/connections/google
GET /oauth/google/authorize?...
GET /oauth/google/callback?code=<authorization-code>
```

RESPONSES

```
The OAuth callback is accepted without a session-bound `state` value being validated.
The Google account is added to the authenticated application account.
```

Recommendation

Generate a cryptographically random `state` value for each authorization attempt, bind it to the initiating user's session, and validate it server-side before processing the callback. Reject missing, expired, reused, or invalid values and review the account-linking flow for unintended account association.

3. Insecure Direct Object Reference in Document Download

Medium

STATUS
Open

SCOPE
https://example.com/documents/download?id=

Description

The document download endpoint accepts a client-supplied document identifier but does not consistently verify the authenticated user's access to that document. Changing the identifier can allow a regular user to retrieve a document belonging to another account.

Impact

An authenticated user may access documents belonging to other users or organizations, potentially disclosing customer or business data depending on the documents' contents.

Steps to reproduce

1. Log in as a regular test user and download a document belonging to that account.
2. Change the document identifier to one associated with another test account.
3. Submit the modified request and confirm that the server returns the other document rather than denying access.

Evidence

REQUESTS

```
GET /documents/download?id=1001 HTTP/1.1
GET /documents/download?id=1002 HTTP/1.1
```

RESPONSES

```
Request for the user's own document: HTTP/1.1 200 OK
Request for another user's document: HTTP/1.1 200 OK
The response contains the contents of a document belonging to another account.
```

Recommendation

Enforce server-side authorization for every document access and verify the authenticated user's or organization's entitlement before returning a document. Do not rely on unpredictable identifiers alone. Add cross-user and cross-organization authorization tests.

4. Open Redirect via Return URL Parameter

Low

STATUS
Open

SCOPE
https://example.com/?return_to=

Description

The application accepts an external destination in the user-controlled `return_to` parameter and redirects the browser to it without sufficiently restricting destinations.

Impact

An attacker could use a link on the trusted application domain to redirect users to an attacker-controlled website for phishing. No XSS or server-side request forgery was confirmed as part of this assessment.

Steps to reproduce

1. Open the application with a `return_to` value pointing to an external domain.
2. Confirm that the browser is redirected to that destination.

Evidence

REQUESTS

```
GET /?return_to=https://example-attacker.test/ HTTP/1.1
```

RESPONSES

```
HTTP/1.1 302 Found
The `Location` header points to the external destination.
```

Recommendation

Allow redirects only to approved internal paths or trusted destinations. Prefer relative paths and validate and normalize redirect targets server-side.

5. Missing SameSite Attribute on Session Cookie

Low

STATUS
Open

SCOPE
Application session cookie configuration.

Description

The session cookie has `Secure` and `HttpOnly` attributes but no explicit `SameSite` attribute. Depending on browser behavior and application request handling, the cookie may be sent in some cross-site request scenarios.

Impact

The missing explicit policy reduces a browser-level defense against cross-site requests. This finding does not by itself demonstrate a successful CSRF attack; practical impact depends on existing CSRF protections and the sensitivity of cross-site actions.

Steps to reproduce

1. Authenticate to the application.
2. Inspect the session cookie in browser developer tools or an intercepted HTTP response.
3. Confirm that `Secure` and `HttpOnly` are present but `SameSite` is not explicitly set.

Evidence

REQUESTS

```
GET /login HTTP/1.1
```

RESPONSES

```
HTTP/1.1 200 OK
Set-Cookie: session=<redacted>; Secure; HttpOnly
```

Recommendation

Set an explicit `SameSite` attribute, using `SameSite=Lax` or a stricter policy where compatible. Retain `Secure` and `HttpOnly`, and maintain server-side CSRF protections for state-changing requests.

6. Verbose Error Messages Disclose Internal Application Details

Low

STATUS

Open

SCOPE

Application error handling for malformed or unexpected requests.

Description

Certain invalid requests produce verbose error responses containing implementation details unnecessary for end users, such as exception names and internal application paths.

Impact

These details may help an attacker identify technologies or code paths for further testing. No direct compromise was demonstrated from the disclosure alone.

Steps to reproduce

1. Send malformed or unexpected input to an application endpoint.
2. Inspect the error response for internal implementation details.

Evidence

REQUESTS

```
GET /api/example?id=invalid HTTP/1.1
```

RESPONSES

```
HTTP/1.1 500 Internal Server Error
Response body contains an exception name and internal application path.
```

Recommendation

Return generic errors to external users, disable framework debug output in production, and log detailed exceptions server-side. Review error handling across other endpoints and HTTP methods.

7. Technology Stack Information Disclosure

Informational

STATUS

Informational

SCOPE

Public HTTP responses from the web application.

Description

Public HTTP response headers or other accessible response metadata disclose technology and framework information, potentially including component names or versions.

Impact

The information may assist fingerprinting and reconnaissance. No direct security impact was demonstrated from this disclosure alone.

Steps to reproduce

1. Send a normal HTTP request to the application.
2. Inspect the response headers and metadata for technology information exposed to unauthenticated users.

Evidence

REQUESTS

```
GET / HTTP/1.1
Host: example.com
```

RESPONSES

```
HTTP/1.1 200 OK
Response headers disclose framework or technology information.
```

Recommendation

Remove unnecessary technology-identifying headers, disable framework version disclosure where practical, and review public responses for unnecessary implementation details.

06 · LIMITATIONS

- Billing functionality was assessed at the application level, but live payment processing and payment-provider callback handling were not tested.

07 · CONCLUSION

The assessment identified security weaknesses across client-side input handling, account binding, authorization controls, and security configuration. The most significant issues affect the password reset and Google account binding flows, where exploitation may affect authenticated user sessions or account associations. Remediation should prioritize the two High-severity findings, followed by unauthorized document access. Lower-severity configuration and information disclosure issues should be addressed as part of ongoing security hardening. The results are subject to the stated scope and testing limitations.

08 · SEVERITY DEFINITIONS

SEVERITY	DESCRIPTION
Critical	Immediate remediation is required due to the potential for severe compromise of the application, sensitive data, or supporting infrastructure.
High	Significant security risk that could lead to unauthorized access, privilege escalation, or substantial impact under realistic conditions.
Medium	Moderate security weakness that presents meaningful risk but typically requires additional conditions or has more limited impact.
Low	Limited security issue with relatively low impact, though remediation is recommended as part of overall security improvement.
Informational	Security observation or best-practice recommendation that does not constitute a direct exploitable vulnerability.