Stealth Layer Security

SAMPLE DELIVERABLE

Sample Web Application & API Penetration Test Report

Fictional, client-safe example report for prospective clients

CLIENT	ExampleCorp Ltd. (fictional)
ASSESSMENT TYPE	Web Application and API Penetration Test
REPORT DATE	14 May 2026
CLASSIFICATION	Confidential - Sample Report
AUTHORIZATION	Written authorization confirmed before testing

This public sample contains synthetic data only. It does not describe a real client, system, credential, token, or vulnerability.

01

Confidentiality Notice

This sample report is provided by Stealth Layer Security for demonstration purposes. It is designed to show the structure, tone, and level of detail clients can expect from a professional security assessment deliverable.

All client names, domains, email addresses, tokens, invoice identifiers, screenshots, and vulnerability evidence in this report are fictional. The findings below do not describe a real organization or real system.

Stealth Layer Security performs testing only with written authorization, defined scope, agreed Rules of Engagement, and clear communication channels.

Document Name	SLS-Sample-Report.pdf
Prepared By	Stealth Layer Security
Version	1.0
Classification	Confidential - Sample Report
Distribution	Public website sample / prospective client review

02

Executive Summary

Stealth Layer Security performed an authorized penetration test of ExampleCorp's customer portal and supporting REST API. The objective was to identify weaknesses that could allow unauthorized access, sensitive data exposure, privilege escalation, account compromise, or service abuse.

Testing identified seven findings across the application and API. One issue was rated Critical, one High, two Medium, two Low, and one Informational.

The most significant risk was a broken object-level authorization weakness in the API that allowed one authenticated user to access invoice metadata associated with another customer by modifying an object identifier. This issue could expose customer billing records and business-sensitive information across tenants.

Overall Risk Posture

The application demonstrated several positive security practices, including HTTPS enforcement, authentication-protected portal access, and use of Secure and HttpOnly session cookie attributes. However, authorization enforcement, abuse protection, browser security headers, and error handling should be improved before broader production rollout.

Severity Distribution

Severity	Count	Description
Critical	1	Immediate threat requiring urgent remediation.
High	1	Significant risk to sensitive data or privileged workflows.
Medium	2	Meaningful weaknesses that increase attack feasibility.
Low	2	Hardening issues or limited-risk weaknesses.
Informational	1	Observation for security maintenance and maturity.

Recommended Immediate Actions

- Remediate SLS-001 by enforcing server-side object ownership checks on every tenant-owned API resource.
- Remediate SLS-002 by applying context-aware output encoding and safe rich-text sanitization.
- Add authentication rate limiting and alerting to reduce credential stuffing and password spraying risk.
- Schedule focused retesting after remediation to verify that controls are effective and regression-safe.

03

Scope

In-Scope Assets

Asset	Type	Environment
https://app.examplecorp.test	Web application	Staging
https://api.examplecorp.test/v1	REST API	Staging
Customer portal authentication flow	Login and session management	Staging
Invoice, profile, and support modules	Application features	Staging

Out-of-Scope Activities

Asset / Activity	Status
Production denial-of-service testing	Out of scope
Social engineering, phishing, or pretexting	Out of scope
Physical security testing	Out of scope
Third-party SaaS platforms	Out of scope
Persistent malware, destructive payloads, or data destruction	Prohibited
Testing outside written authorization	Prohibited

Test Accounts Provided

Role	Test Account
Standard Customer	customer.alpha@example.test
Standard Customer	customer.beta@example.test
Support User	support.user@example.test
Read-Only Admin	readonly.admin@example.test

04

Rules of Engagement

Testing was performed only after written authorization was received from ExampleCorp. Stealth Layer Security limited all activity to the approved scope and avoided destructive testing, service disruption, and modification of non-test data.

Rule	Description
Authorization	Testing began only after written authorization and scope confirmation.
Safety	No destructive payloads, persistence, malware, or production denial-of-service attempts were used.
Data Handling	Only synthetic test accounts and test records were used. Any sensitive data exposure was minimized and documented safely.
Communication	Testing status and potential service-impacting observations were escalated through the agreed contact channel.
Retesting	Remediation verification is performed only after the client confirms fixes are deployed to the agreed environment.

Methodology

1. Discovery and intake
2. Scope confirmation
3. Proposal and Statement of Work
4. Written authorization and Rules of Engagement
5. Asset and environment confirmation
6. Reconnaissance and information gathering
7. Vulnerability identification
8. Manual validation and exploitation where in scope
9. Risk rating and impact analysis
10. Reporting and debrief
11. Retesting and verification

05

Standards and References

The assessment was informed by established web, API, and application security standards. These references were used to structure testing, map findings, and communicate remediation priorities.

Standard / Framework	Usage
OWASP Web Security Testing Guide	Web application testing methodology and control validation.
OWASP API Security Top 10	API authorization, authentication, data exposure, and abuse review.
OWASP ASVS	Application security requirement mapping and assurance control review.
OWASP Top 10	Common web application risk classification.
NIST Cybersecurity Framework	Business risk and control context.
MITRE ATT&CK	Threat-informed impact analysis.
CIS Controls	Defensive hardening recommendations.
PTES	Penetration testing execution structure.

Risk Rating Approach

Severity was assigned using likelihood, impact, exposure, authentication requirement, data sensitivity, and business context. Ratings reflect both technical exploitability and potential operational or reputational consequence.

Factor	Description
Likelihood	How realistic exploitation is given attacker skill and preconditions.
Impact	Confidentiality, integrity, and availability consequences if exploited.
Exposure	Whether the issue is internet-facing or requires internal access.
Authentication	Whether authentication or a specific role is required.
Data Sensitivity	Type and regulatory weight of data placed at risk.
Business Context	Operational, contractual, financial, and reputational consequences.

06

Findings Summary

ID	Finding	Severity	Status
SLS-001	Broken Object-Level Authorization in Invoice API	Critical	Open
SLS-002	Stored Cross-Site Scripting in Support Message Thread	High	Open
SLS-003	Missing Rate Limiting on Authentication Endpoint	Medium	Open
SLS-004	Weak Session Cookie Configuration	Medium	Open
SLS-005	Incomplete Content Security Policy	Low	Open
SLS-006	Verbose Error Messages Expose Internal Details	Low	Open
SLS-007	Outdated JavaScript Dependency Detected	Informational	Open

Severity Definitions

Severity	Definition
Critical	Immediate threat. Exploitation likely enables full compromise or wide-impact data exposure.
High	Significant risk. Exploitation enables sensitive data access or major capability abuse.
Medium	Meaningful weakness. Often requires conditions or chained issues to fully exploit.
Low	Limited risk. Should be remediated but unlikely to cause material impact on its own.
Informational	Observation or hardening recommendation with no direct exploit path.

SLS-001 - Broken Object-Level Authorization in Invoice API

CRITICAL

Severity	Critical
Category	Broken Access Control
OWASP Mapping	OWASP API1:2023 Broken Object Level Authorization
Affected Asset	https://api.examplecorp.test/v1/invoices/{invoice_id}
Authentication Required	Yes - standard customer account
Status	Open

Description

The invoice API allowed an authenticated customer to retrieve invoice metadata belonging to another customer by changing the invoice_id value in the API request. The application verified that the user was authenticated but did not consistently verify that the requested invoice belonged to the authenticated user's tenant or account.

Business Impact

An attacker with any valid customer account could enumerate or access invoice records belonging to other customers. Depending on invoice contents, this could expose customer names, billing amounts, account references, subscription details, and commercial relationships.

Evidence

```
GET /v1/invoices/INV-2026-1048 HTTP/1.1
Host: api.examplecorp.test
Authorization: Bearer [customer_alpha_token]
Accept: application/json

HTTP/1.1 200 OK
{
  "invoice_id": "INV-2026-1048",
  "customer_name": "Example Beta LLC",
  "billing_email": "billing@example-beta.test",
  "amount_due": "2480.00",
  "currency": "USD",
  "status": "Due"
}
```

Root Cause

The API relied on direct object identifiers without enforcing object ownership checks in the service layer.

Recommendation

- Validate that invoice_id belongs to the authenticated user's tenant before returning data.

- Enforce authorization in the service layer, not only in the user interface.
- Use scoped identifiers where appropriate and avoid exposing predictable object references.
- Add authorization regression tests for cross-tenant access attempts.
- Log and alert on repeated access attempts to unauthorized object IDs.

Retest Procedure

1. Authenticate as Customer A.
2. Attempt to access Customer B invoice objects.
3. Confirm the API blocks access with a generic 403 or 404 response.
4. Confirm no invoice metadata is leaked in denied responses.
5. Confirm denied authorization attempts are logged.

SLS-002 - Stored Cross-Site Scripting in Support Message Thread

HIGH

Severity	High
Category	Cross-Site Scripting
OWASP Mapping	OWASP A03:2021 Injection
Affected Asset	https://app.examplecorp.test/support/tickets
Authentication Required	Yes - customer account to submit; support account to view
Status	Open

Description

The support ticket message field accepted HTML input that was later rendered in the support dashboard without sufficient output encoding. This allowed script execution in the browser context of a user viewing the ticket.

Business Impact

A malicious customer could execute script in the context of a support user or another portal user. This could enable unauthorized actions in the victim session, content manipulation, or exposure of portal data visible to the victim.

Evidence

```
Validation payload submitted to the support message field:

<img src=x onerror=alert('SLS-XSS-VALIDATION')>

Observed result:
JavaScript execution confirmed in support ticket detail view.
```

Root Cause

User-supplied content was rendered into HTML without context-aware output encoding or strict allowlist-based sanitization.

Recommendation

- Encode user input before inserting it into HTML, attributes, JavaScript, or URLs.
- Sanitize rich text input using a proven allowlist-based sanitizer.
- Disallow inline event handlers such as onerror, onclick, and onload.
- Deploy a restrictive Content Security Policy to reduce exploitability.
- Add automated regression tests for stored and reflected XSS cases.

Retest Procedure

1. Submit benign XSS validation payloads to support messages.
2. View the ticket from customer and support roles.
3. Confirm payloads render as inert text.
4. Confirm no JavaScript execution occurs.
5. Confirm CSP reduces script execution risk.

SLS-003 - Missing Rate Limiting on Authentication Endpoint

MEDIUM

Severity	Medium
Category	Authentication Security
OWASP Mapping	OWASP A07:2021 Identification and Authentication Failures
Affected Asset	https://api.examplecorp.test/v1/auth/login
Authentication Required	No
Status	Open

Description

The login endpoint did not enforce effective rate limiting or account-level throttling after repeated failed authentication attempts.

Business Impact

An attacker could perform credential stuffing or password spraying against known or guessed user accounts. Risk increases if users reuse passwords from unrelated breaches.

Evidence

```
Observed behavior after repeated failed login attempts against a test account:
- No progressive delay after repeated failures
- No temporary account lockout
- No IP-based throttling observed
- No user notification triggered
```

Root Cause

Authentication abuse controls were not applied consistently at the login endpoint.

Recommendation

- Rate limit by IP address, account, and device fingerprint where appropriate.
- Add progressive delays after repeated failures.
- Detect password spraying patterns across many accounts.
- Notify users of suspicious login attempts.
- Require MFA for administrative and privileged users.

Retest Procedure

1. Perform controlled failed login attempts against a test account.
2. Confirm throttling, delay, alerting, or lockout behavior.
3. Confirm legitimate users are not permanently denied service.
4. Review logs for suspicious authentication events.

SLS-004 - Weak Session Cookie Configuration

MEDIUM

Severity	Medium
Category	Session Management
OWASP Mapping	OWASP A05:2021 Security Misconfiguration
Affected Asset	https://app.examplecorp.test
Authentication Required	Yes
Status	Open

Description

The application's session cookie was not configured with all recommended browser security attributes. Specifically, the SameSite attribute was not explicitly set.

Business Impact

Missing or weak cookie attributes may increase exposure to session misuse under certain attack conditions, especially when combined with cross-site request forgery, cross-site scripting, or insecure browser contexts.

Evidence

```
Observed session cookie:
Set-Cookie: session_id=[redacted]; Path=/; Secure; HttpOnly

The Secure and HttpOnly attributes were present. SameSite was not explicitly defined.
```

Root Cause

Session cookie attributes were not configured according to a complete secure baseline.

Recommendation

- Set Secure, HttpOnly, and SameSite attributes consistently on session cookies.
- Use SameSite=Lax for normal portal workflows where compatible.
- Consider SameSite=Strict for highly sensitive workflows if business functionality allows.
- Document cookie security requirements in deployment tests.

Retest Procedure

1. Authenticate to the application.
2. Inspect all session cookies.

3 Confirm Secure, HttpOnly, and SameSite attributes are present.

4 Confirm the application remains functional after the setting is enforced.

SLS-005 - Incomplete Content Security Policy

LOW

Severity	Low
Category	Browser Security Hardening
OWASP Mapping	OWASP A05:2021 Security Misconfiguration
Affected Asset	https://app.examplecorp.test
Authentication Required	No
Status	Open

Description

The application did not define a complete Content Security Policy header. A strong CSP helps reduce the impact of cross-site scripting and content injection vulnerabilities.

Business Impact

CSP is a defense-in-depth control. Its absence does not directly create a vulnerability, but it can increase the impact of script injection issues.

Evidence

```
The following response header was not observed in application responses:
Content-Security-Policy
```

Root Cause

Browser hardening headers were not deployed as part of the application security baseline.

Recommendation

- Deploy CSP in report-only mode first, then enforce it after tuning.
- Avoid broad directives such as wildcard sources unless explicitly required.
- Set frame-ancestors to prevent clickjacking.
- Set base-uri and form-action to reduce injection impact.

Retest Procedure

1. Confirm Content-Security-Policy is present.
2. Confirm directives are scoped to expected origins.
3. Confirm unsafe-inline and unsafe-eval are not used unless documented.

4 Confirm the application still functions normally.

SLS-006 - Verbose Error Messages Expose Internal Details

LOW

Severity	Low
Category	Information Disclosure
OWASP Mapping	OWASP A05:2021 Security Misconfiguration
Affected Asset	https://api.examplecorp.test/v1/profile
Authentication Required	Yes
Status	Open

Description

Certain malformed API requests returned verbose error messages containing internal implementation details, including framework names, validation traces, and internal route references.

Business Impact

Verbose errors may help attackers understand application internals and accelerate vulnerability discovery. While this finding does not directly expose sensitive customer data, it reduces attacker effort.

Evidence

```
Example sanitized response:
{
  "error": "ValidationException",
  "framework": "ExampleFramework 4.2",
  "route": "ProfileController.updateBillingAddress",
  "debug_id": "trace-7fa29c"
}
```

Root Cause

Diagnostic details intended for developers were returned to authenticated users instead of being restricted to server-side logs.

Recommendation

- Return generic error messages to users.
- Keep detailed diagnostic information in server-side logs.
- Use request IDs for support correlation.
- Disable debug mode in client-accessible environments.

Retest Procedure

1. Submit malformed requests.
2. Confirm responses do not expose framework, stack trace, route, or database details.
3. Confirm server-side logs retain useful diagnostics for support teams.

SLS-007 - Outdated JavaScript Dependency Detected

INFORMATIONAL

Severity	Informational
Category	Dependency Management
OWASP Mapping	OWASP A06:2021 Vulnerable and Outdated Components
Affected Asset	https://app.examplecorp.test/assets/
Authentication Required	No
Status	Open

Description

A JavaScript dependency appeared to be outdated. No active exploit path was confirmed during testing; however, outdated dependencies should be tracked and updated through routine maintenance.

Business Impact

Outdated client-side dependencies can increase long-term exposure if known vulnerabilities exist or are later discovered.

Evidence

```
Observed dependency reference:
example-ui-helper.js version 2.1.0
Current vendor release: 2.4.x
```

Root Cause

Dependency versions were not fully aligned with the current approved baseline.

Recommendation

- Maintain a software bill of materials.
- Use automated dependency scanning.
- Pin approved versions.
- Remove unused libraries.
- Monitor vendor advisories.

Retest Procedure

1. Confirm the dependency has been updated.
2. Confirm a documented exception exists if the dependency cannot be upgraded.

3 Confirm no unused vulnerable libraries remain publicly served.

08

Positive Security Observations

The following controls were observed during testing. These observations do not eliminate the need to remediate the findings listed above, but they indicate useful security practices already present in the environment.

Control	Observation
HTTPS Enforcement	Application redirected HTTP to HTTPS.
Secure Cookie Flag	Session cookie used the Secure attribute.
HttpOnly Cookie Flag	Session cookie used the HttpOnly attribute.
Authentication Required	Customer portal required authentication before access.
Role-Based UI	Administrative navigation was hidden from standard users.
Test Environment	Assessment was performed in a dedicated staging environment.

Remediation Priority

Priority	Finding	Recommended Timeline
1	SLS-001 Broken Object-Level Authorization	Immediate
2	SLS-002 Stored Cross-Site Scripting	7 days
3	SLS-003 Missing Rate Limiting	14 days
4	SLS-004 Weak Session Cookie Configuration	14 days
5	SLS-005 Incomplete CSP	30 days
6	SLS-006 Verbose Error Messages	30 days
7	SLS-007 Outdated Dependency	Next maintenance cycle

09

Retesting Plan

After remediation, Stealth Layer Security recommends a focused retest covering the controls most directly related to the confirmed findings. Retesting should be performed in the same environment or in a comparable deployment with the same code and configuration changes.

Area	Retest Objective
Authorization	Confirm cross-tenant object access is blocked.
XSS	Confirm stored payloads are encoded or sanitized.
Authentication	Confirm rate limiting and abuse detection.
Session Security	Confirm cookie attributes are correctly set.
Security Headers	Confirm CSP and browser hardening headers.
Error Handling	Confirm internal details are not exposed.
Dependency Review	Confirm outdated library has been upgraded or risk-accepted.

Final Assessment

ExampleCorp's customer portal and API contain several security weaknesses that should be addressed before broader production rollout. The most important remediation item is the broken object-level authorization issue affecting invoice access, as it presents a realistic cross-tenant data exposure risk.

After remediation and retesting, Stealth Layer Security expects the overall risk level to decrease substantially. The recommended next step is to resolve Critical and High findings first, then validate fixes through a focused retest before relying on the application for sensitive customer workflows.

Client-Safe Disclaimer

This is a fictional sample report created for demonstration purposes. It does not describe a real client, real system, or real vulnerability. All domains, names, accounts, tokens, invoice IDs, and evidence values are synthetic. Stealth Layer Security performs testing only with written authorization, defined scope, and agreed Rules of Engagement.