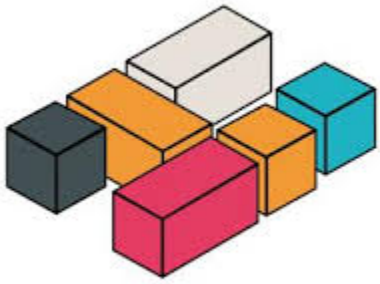
CONVOCATION
RESEARCH + DESIGN

Horizontal x Convocation Research+Design:

Shira 2.0.0 Security Audit

May 2026

Prepared for: Horizontal
Prepared by: Convocation Research + Design
276 Fifth Avenue, Suite 704-33
New York, NY 10001, US
general@convocation.design
https://convocation.design/



Prepared for: Horizontal
Prepared by: Convocation Research+Design
Client Project: Shira 2.0.0
Client Website: https://wearehorizontal.org/index
Github: https://github.com/Horizontal-org/shira
Contract Reference: CORD-SHIRA-2.0-2026
Assessment Period: February 1st, 2026 to March 14th, 2026 + May 2026 Fix Verification
Security Researchers: Sam Smith, Chelsea Manning, Caroline Sinders, and Cooper Quintin

Table of Contents

Executive Summary

Overview

This report presents the integrated findings of Convocation Research+Design's (CoRD) security engagement with Horizontal regarding Shira 2.0.0, a multilingual anti-phishing training platform serving human rights defenders, internet freedom activists, and civil society organizations. The engagement comprised two phases: an initial security audit conducted from February 1st, 2026 through March 14th, 2026, and a fix verification review conducted in May 2026 following Horizontal's remediation work. This document consolidates the outcomes of both phases into a single deliverable.

The original audit identified thirteen vulnerabilities (five High, seven Medium, and one Low severity, with no Critical findings), along with four observation-class items addressing security headers, logging of personally identifiable information, registration passphrase storage, and configuration of the root .gitignore, and a comprehensive review of dependency-level vulnerabilities. Following Horizontal's remediation cycle and CoRD's verification of the implemented fixes, seven findings are now fully Resolved, three are Partially Resolved with documented residual recommendations, and three have been designated as Risk Accepted by Horizontal with the residual risk documented for future revisitation. No findings remain in their original Open state.

Engagement Outcomes

The cross-tenant authorization vulnerabilities that drove the highest-priority recommendations in the original audit, namely CVE-003 and CVE-004 (cross-tenant question read and delete), are fully resolved. The CVE-008 stored XSS finding is also fully resolved. The cross-tenant enumeration vector in CVE-005 (anonymous GET of any space's quiz runs) has been eliminated by deletion of that endpoint, although Horizontal has accepted residual risk on the related unauthenticated POST and PATCH paths because these are required to support the platform's anonymous-learner product model. The XSS-to-token-theft chain identified in the original audit is now broken in two places:

explanation content is sanitized at write time, and the production frontend no longer holds the JWT in script-readable storage.

The three Risk Accepted findings (CVE-005, CVE-006, CVE-007) all share a common pattern: Horizontal has implemented the audit's primary recommendation in each case (deletion of the cross-tenant GET endpoint for CVE-005 and a global rate-limiting throttler for CVE-006 and CVE-007), but has elected not to pursue the audit's optional secondary recommendations (per-request integrity controls on quiz runs, and CAPTCHA or stricter per-route limits on invitation and survey submission). CoRD documents the residual risk in the corresponding finding sections of this report and in a dedicated Risk Acceptance Summary so that Horizontal can revisit these decisions in future release cycles if the platform's user base or threat profile changes materially.

The three Partially Resolved findings (CVE-002, CVE-011, CVE-012) each share a different common pattern: the structural intent of the fix is correct, but a parameter, threshold, or code path leaves residual risk that CoRD believes can be closed with low-effort follow-up work. The most material residual items are the JWT_SECRET length floor of six characters and the related async/no-await implementation issue under CVE-002, the absence of stricter per-route rate limits on authentication endpoints under CVE-011, and the dev-mode localStorage fallback that remains in the spaces frontend under CVE-012.

Overall Security Posture

CoRD's overall assessment is that Shira 2.0.0 is in a substantially stronger security posture than at the time of the original audit, particularly with respect to the multi-tenant authorization model that is core to the new spaces feature set. Horizontal's engineering team has delivered substantive, well-targeted fixes for the thirteen findings, and CoRD does not consider any of the remaining residual items to be blocking for the current beta release. The clustering of fixes around input validation, authorization scoping, and rate limiting addresses the systemic gaps identified in the original audit's executive summary, and the platform's intentional security design (NestJS with JWT-based authentication, role-based access control via decorators, and clear separation between public and protected routes) has been preserved and strengthened through the remediation cycle.

Statistics at a Glance

Original security audit findings: Critical: 0; High: 5; Medium: 7; Low: 1; observation-class items: 4 (security headers, PII in logs, registration passphrase storage, root .gitignore configuration); Total: 13 vulnerabilities, 4 observations, plus a separate dependency review

of 80 npm advisories.
Final remediation findings: Resolved: 7 (CVE-001, CVE-003, CVE-004, CVE-008, CVE-009, CVE-010, CVE-013); Partially Resolved: 3 (CVE-002, CVE-011, CVE-012); Risk Accepted: 3 (CVE-005, CVE-006, CVE-007); Open: 0.

Introduction

Project Background

Horizontal's Shira 2.0.0 is a multilingual anti-phishing training platform designed specifically for frontline human rights defenders and civil society actors operating in repressive environments. Building on the foundation of v1.0 (which has seen ~3,000 quiz completions across hundreds of users globally), this major release introduces substantial new functionality including private organizational spaces with granular access controls, SMS/messaging phishing simulations beyond traditional email vectors, custom quiz creation capabilities, and comprehensive analytics for security trainers. The platform adopts a learner-centered, non-punitive approach to security education, moving beyond traditional phishing simulations to provide interactive quizzes that build confidence and understanding without creating anxiety or mistrust. These features were developed in direct response to feedback from organizations requiring more sophisticated tools to address the increasingly complex phishing threats targeting civil society.

New Features for Shira v2.0.0beta5+

- Private "spaces" where admins can manage their organization (including quizzes, learners, and results)
- Creating, editing, and deleting custom quizzes, including naming and renaming the quiz, and creating, editing, deleting, and reordering quiz questions
- Managing the visibility of quizzes including the following statuses: unpublished, published as public quiz (anyone with the quiz URL can take the quiz), published as private quiz (only learners invited to the space can take the quiz)
- Inviting learners to a space: the invitation is generated and sent by email, and the invited learner has to click on the link to join the space; the learner does *not* create credentials as they never have to log into their space. A learner can be created manually or learners can be created in bulk by importing a CSV
- Assigning a quiz to a space's learners: the invitation to take a quiz is sent by email, and the invited learner has to click on the link to take the quiz; they will then receive a confirmation email once the quiz has been successfully submitted)
- Results aggregating the participation rate and success rate for quizzes

Security Audit Objectives

- Identify security vulnerabilities in the Shira 2.0.0 codebase (TypeScript/React frontend, Node.js backend) and infrastructure.
- Identify security vulnerabilities in the Shira 2.0.0 features include: private spaces, custom quiz creation, role-based access, bulk learner management, and analytics.
- Assess the platform's resilience against threats specific to human rights defenders in hostile environments.
- Evaluate the implementation of Security by Design principles appropriate for high-risk users.
- Provide actionable, prioritized remediation recommendations aligned with the April 2026 release timeline.
- Ensure privacy and data protection measures meet the needs of users under surveillance.
- Authentication, authorization, and session management systems.
- Ensure secure data storage, handling, and localization infrastructure.
- Ensure secure deployment architecture and CI/CD practices.

Security Audit Scope

The following table records the specific tasks performed during the audit phase, the outcome of each, and any associated notes. Tasks are grouped by the security domain they address. The table reflects the audit phase only; the verification phase is captured separately in the per-finding remediation narratives in the Detailed Vulnerability Findings section. This coverage record is reproduced from the audit deliverable to document the breadth of work performed and to allow the reader to confirm that areas of inquiry were systematically addressed.

Repo Locations:

- Frontend applications: shira-ui (React/TypeScript) across all four components:
    - Public website (https://shira.app)
    - Super admin panel (https://super.shira.app)
    - Space admin panel (https://space.shira.app)
    - Public quiz interface (https://quiz.shira.app)
    - Public help beta (https://beta.shira.app/help)
    - Public get started beta (https://beta.space.shira.app/get-started)
    - Public user beta (https://shira.app/beta-user/)

- Public Shira 2.0.0-beta6 release: https://github.com/Horizontal-org/shira/releases/tag/2.0.0-beta6
- New Shira 2.0.0 Dependancies:
    - Nth-check: https://github.com/Horizontal-org/shira/security/dependabot/13
    - Glob: https://github.com/Horizontal-org/shira/security/dependabot/114
    - Postcss: https://github.com/Horizontal-org/shira/security/dependabot/14
    - Fast-xml-parser: https://github.com/Horizontal-org/shira/security/dependabot/165
    - Webpack-dev-server: https://github.com/Horizontal-org/shira/security/dependabot/53
    - Jsonpath: https://github.com/Horizontal-org/shira/security/dependabot/166
    - Elliptic: https://github.com/Horizontal-org/shira/security/dependabot/136
    - MJML: https://github.com/Horizontal-org/shira/security/dependabot/127
    - kangax: https://github.com/Horizontal-org/shira/security/dependabot/17
- Building upon Shira 1.0's security audit: https://shira.app/assets/shira-final.pdf

Domain	Task	Done	Notes
Auth and session	Review auth controllers and services for DTOs and ValidationPipe; no credentials in URLs or logs	Yes	Auth uses DTOs and ValidationPipe; no credentials in URLs.
Auth and session	Cookie options: secure: true when HTTPS	No	Not implemented at audit time; tracked as CVE-001.
Auth and session	JWT_SECRET required and validated at bootstrap	No	No startup check at audit time; tracked as CVE-002.
Auth and session	Password reset invalidates previous tokens	Yes	request-password-reset.auth.service.ts updates usedAt for existing resets.
Auth and session	Search for hardcoded secrets	Yes	None found; JWT uses process.env.JWT_SECRET.
Authorization / IDOR	Verify resource IDs scoped to space or organization	Partial	Quiz endpoints use ValidateSpaceQuizService or SpaceId(); question by id and delete question did not at audit time (tracked as CVE-003 and CVE-004).
Authorization / IDOR	Document unauthenticated routes	Yes	See Authentication Architecture and Route Coverage section.
Authorization / IDOR	Flag invitation and survey as abuse risks	Yes	Documented as CVE-006 and CVE-007.

Domain	Task	Done	Notes
Authorization / IDOR	Endpoints trusting client spaceId without re-validation	Yes	Quiz-run and list question endpoints flagged as CVE-005 and CVE-003.
Unauthenticated endpoints	List all routes without AuthController or UseGuards	Yes	See Authentication Architecture and Route Coverage section.
Unauthenticated endpoints	Recommend rate limiting or CAPTCHA for invitation and survey	Yes	Captured in remediation narratives for CVE-006 and CVE-007.
Unauthenticated endpoints	Error responses do not reveal resource existence	Partial	Reset password uses generic 'Invalid or expired reset link'; confirm registration used NotFoundException 404 (tracked as CVE-013).
Input validation	Raw SQL parameterized in application code	Yes	Only migrations use raw SQL; no user input is concatenated.
Input validation	All HTML persistence paths sanitized	Partial	Question content sanitized at audit time; explanation content not sanitized (tracked as CVE-008).
Input validation	Frontend dangerouslySetInnerHTML sources reviewed	Yes	Documented; some content from API or import.
Input validation	File upload: type, size, path	No	No MIME or size validation at audit time (tracked as CVE-009).
Sensitive data	.env in .gitignore; no secrets logged or sent to client	Partial	App-level .gitignore files include .env; root does not (observation finding).
Sensitive data	Review log statements for PII	Yes	Several auth and learner logs include email and userId (observation finding).
Sensitive data	Registration and passphrase exposure in emails	Yes	Magic links and passphrase codes sent; necessity assessed.
Security headers / CORS	Document CORS and recommend restrictions	Yes	enableCors() with no options at audit time (tracked as CVE-010).
Security headers / CORS	Recommend Helmet and CSP	Yes	Captured as observation finding.
Rate limiting	List high-value targets	Yes	Login, register, reset-password, invitation, survey, confirm, quiz-run.
Rate limiting	Recommend rate limits	Yes	Tracked as CVE-006, CVE-007, and CVE-011.

Domain	Task	Done	Notes
Dependencies	npm audit (root and apps)	Yes	80 vulnerabilities (4 low, 17 moderate, 58 high, 1 critical) identified.
Dependencies	npm outdated and CVE check	Yes	See Dependency Findings section.
Frontend	Auth token only in the httpOnly cookie	No	Spaces app also stored token in localStorage at audit time (tracked as CVE-012).
Frontend	No secrets in client bundles	Yes	Only REACT_APP_* used.
Frontend	dangerouslySetInnerHTML sources	Yes	Content from API or import; explanation content unsanitized at audit time.

Audit Methodology

Convocation Research+Design's methodology for this assessment encompasses both technical vulnerability analysis and contextual threat modeling specific to human rights defenders. The audit spans from February 1st 2026 through March 15, 2026, including a remediation period, and employs automated scanning, manual code review, dynamic testing, and architecture analysis tailored to the unique security requirements of activists operating under surveillance and repression. Our methodology combines automated scanning, manual code review, and threat modeling specifically tailored for high-risk human rights contexts. This assessment builds upon findings from the Subgraph v1.0.0 audit (July 2023: https://shira.app/assets/shira-final.pdf), with particular attention to previously identified vulnerabilities.

1. Static Code Analysis

- Manual security review of authentication, authorization, and data handling logic
- Automated vulnerability scanning using industry-standard tools
- Dependency vulnerability assessment for supply chain risks
- TypeScript/React-specific security pattern analysis

2. Dynamic Analysis

- Authentication bypass testing
- Session management security
- Input validation and injection testing (SQL, XSS, command injection)
- Access control verification across all user roles
- API endpoint security testing

3. Architecture Review

- Multi-component architecture security assessment
- Data flow analysis between components

- Infrastructure hardening evaluation
- Localization and content injection vector analysis

4. Threat Modeling

- STRIDE-based threat identification
- Context-specific threat actor profiling for human rights defenders
- Risk scenario development based on real-world attacks
- Privacy and anonymity preservation assessment

Tools and Techniques

- Static analysis: Turborepo, npm audit, NestJS API, React public/spaces apps
- Dynamic testing: Wireshark, Burp Suite Pro, OWASP ZAP
- Dependency scanning: Snyk, OWASP Dependency Check
- Custom scripts for context-specific testing

Testing Environment

- Version: Shira 2.0.0-beta5
- Audit Type: Static & dynamic analysis and codebase review
- Security Analyst: Chelsea E. Manning
- Repositories: https://github.com/Horizontal-org/shira
- Environment: Staging environment with admin access
- Testing Period: February 15th, 2026 to March 14th, 2026

Verification Methodology

CoRD's verification followed three steps for each finding. First, the diff supplied by Horizontal (a pull request or commit hash) was retrieved as a patch and reviewed line by line against the original audit description. Second, the current state of each affected file was retrieved from the development branch on GitHub to confirm that the fix had not been reverted or superseded by a subsequent change. Third, where the change was observable from outside the application, behavior was confirmed against the deployed beta at https://beta.space.shira.app/backend. CORS, throttler, and cookie-flag changes were validated this way; finer-grained authorization checks (the IDOR fixes in particular) were verified by code review only, since exercising them live would have required valid SpaceAdmin credentials in two distinct spaces.

For the three Risk Accepted findings, CoRD reviewed the implemented mitigations against the original audit recommendation, identified the gap between what was implemented and what would constitute full remediation, and documented the residual risk. The Risk Accepted designation reflects Horizontal's decision and does not constitute CoRD's recommendation that the risk be left in place; rather, it is a record of the current posture so that the residual risk is explicit and traceable in the engagement record.

Researchers Note: CoRD did not perform a fresh full-stack security re-audit as part of the verification phase. The verification scope was limited to the thirteen findings enumerated in the March 2026 report. Dependency findings and observation-class items were out of scope for verification.

Key Findings:

The table below summarizes all thirteen vulnerabilities identified in the March 2026 audit, with both their original status (as recorded at the close of the audit phase) and their final status following CoRD's May 2026 verification of Horizontal's remediation work. Detailed narratives for each finding follow in the next section.

CVE ID	Severity	Issue	Original Status	Final Status
SHIRA2-001	Medium	Cookie access_token has no secure: true	Open	Resolved
SHIRA2-002	Medium	JWT_SECRET not validated at startup	Open	Partially Resolved
SHIRA2-003	High	IDOR: getQuestion by id not scoped to space	Open	Resolved
SHIRA2-004	High	IDOR: delete question by id not scoped to space	Open	Resolved
SHIRA2-005	High	Unauthenticated quiz-run API	Open	Risk Accepted
SHIRA2-006	Medium	POST /invitation unauthenticated and unrate-limited	Open	Risk Accepted
SHIRA2-007	Medium	POST /survey unauthenticated and unrate-limited	Open	Risk Accepted
SHIRA2-008	High	Explanation content not sanitized (XSS)	Open	Resolved
SHIRA2-009	Medium	File upload: no MIME type or size validation	Open	Resolved
SHIRA2-010	Medium	CORS allows any origin (non-credentialed)	Open	Resolved
SHIRA2-011	Medium	No rate limiting	Open	Partially Resolved
SHIRA2-012	High	Spaces app stores JWT in localStorage	Open	Partially Resolved

CVE ID	Severity	Issue	Original Status	Final Status
SHIRA2-013	Low	Confirm registration returns 404 for invalid hash	Open	Resolved

Status Definitions

Resolved: CoRD has reviewed the remediation and confirms that the original vulnerability is no longer present. The fix matches the audit's recommendation and CoRD has no significant residual concerns specific to the original finding. Minor follow-up suggestions, where given, are noted in the corresponding finding section but do not affect the resolved classification.

Partially Resolved: CoRD has reviewed the remediation and finds that the structural intent of the fix is correct, but a parameter, threshold, or code path leaves residual risk that CoRD believes can be closed with low-effort follow-up. Specific recommendations to fully close each Partially Resolved finding are documented in the corresponding finding section and are repeated in the Outstanding Recommendations section of this report.

Risk Accepted: Horizontal has implemented the audit's primary recommendation in part and has communicated to CoRD that no further remediation is planned. The residual risk is documented in the corresponding finding section and in the Risk Acceptance Summary so that the current posture is explicit and traceable. The Risk Accepted designation reflects Horizontal's decision and does not constitute CoRD's recommendation that the residual risk be left in place.

Detailed Vulnerability Findings

The following sections present each of the thirteen vulnerabilities identified during the March 2026 audit, integrated with CoRD's May 2026 verification of Horizontal's remediation work. Each section is structured into two parts: an Original Finding subsection that preserves the audit's description, impact analysis, affected components, and proof of concept, followed by a Remediation and Verification subsection that documents the specific pull request or commit that delivered the fix, the change made, and CoRD's assessment of the resulting security posture, including any residual recommendations.

Summary Table of Vulnerabilities Findings

ID	Issue	Location	Severity	Recommendation
CVE-2026-SHIRA2-001	Cookie access_token has no secure: true	apps/api/src/modules/auth/controllers/login.auth.controller.ts	Medium	Set secure: true when NODE_ENV === 'production' or when using HTTPS.
CVE-2026-SHIRA2-002	JWT_SECRET not validated at startup	apps/api/src/modules/auth/strategy/jwt.auth.strategy.ts, auth.module.ts	Medium	Require and validate JWT_SECRET at bootstrap (e.g., length ≥ 32); fail fast if missing.
CVE-2026-SHIRA2-003	IDOR: Get question by id not scoped to space	apps/api/src/modules/question/controllers/list.question.controller.ts (getQuestion)	High	Filter question(s) by user's space (e.g., via quiz relation) or restrict to demo; do not return any question by id without space check.
CVE-2026-SHIRA2-004	IDOR: Delete question by id not scoped to space	apps/api/src/modules/question/controllers/delete.question.controller.ts	High	Verify that question belongs to the user's space (e.g. via quiz → space) before deletion.
CVE-2026-SHIRA2-005	Unauthenticated quiz-run API	apps/api/src/modules/quiz_result/controller/quiz-run.controller.ts	High	Protect with AuthController or equivalent; validate spaceId against the authenticated user for GET requests.
CVE-2026-SHIRA2-006	POST /invitation unauthenticated and unrate-limited	apps/api/src/modules/auth/controllers/invite.auth.controller.ts	Medium	Add rate limiting and, optionally, CAPTCHA; consider auth if the invitation is admin-only.

ID	Issue	Location	Severity	Recommendation
CVE-2026-SHIRA2-007	POST /survey unauthenticated and unrate-limited	apps/api/src/modules/survey/controllers/create.survey.controller.ts	Medium	Add rate limiting; consider CAPTCHA or auth for survey submission.
CVE-2026-SHIRA2-008	Explanation content not sanitized (XSS)	apps/api/src/modules/quiz/services/create-question.quiz.service.ts, edit-question.quiz.service.ts	High	Apply QuestionSanitizer (or equivalent) to explanation.text before persisting.
CVE-2026-SHIRA2-009	File upload: no MIME type or size validation	apps/api/src/modules/question_image/controllers/create.question_image.controller.ts, create.question_image.service.ts	Medium	Validate MIME type (e.g. image/*), max size, and optionally magic bytes; reject invalid uploads.
CVE-2026-SHIRA2-010	CORS allows any origin	apps/api/src/main.ts	Medium	Restrict origin to known frontend URLs in production.
CVE-2026-SHIRA2-011	No rate limiting	Entire API	Medium	Add ThrottlerGuard or similar for login, register, reset-password, invitation, survey, confirm, and quiz-run.
CVE-2026-SHIRA2-012	Spaces app stores JWT in localStorage	apps/spaces/src/fetch/auth.ts	High	Use only httpOnly cookie set by API; remove localStorage token and send credentials via cookie.

ID	Issue	Location	Severity	Recommendation
CVE-2026-SHIRA2-013	Confirm registration returns 404 for invalid hash	apps/api/src/modules/auth/services/confirm-registration.auth.service.ts	Low	Prefer same response for invalid vs expired (e.g., generic "Invalid or expired link") to avoid enumeration.

Detailed Vulnerabilities Findings:

CVE-2026-SHIRA2-001: Cookie access_token has no secure: true

Severity: Medium | **Category:** Auth & Session

CVSS Score: 4.3 (AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N)

CWE ID: CWE-614 (Sensitive Cookie in HTTPS Session Without 'Secure' Attribute)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #299, commit 0b2f8fc

Original Finding

Description: The login controller sets the JWT in an access_token cookie with httpOnly: true and an expires value but does not set secure: true. On production deployments served over HTTPS, cookies without the Secure attribute may still be transmitted over HTTP under specific edge conditions such as mixed-content scenarios or misconfiguration, increasing the risk of token capture by network adversaries.

Impact: In deployments where HTTPS is used but the Secure flag is omitted, the session token could be transmitted over unencrypted channels in edge cases, leading to session hijacking if an attacker can intercept traffic.

Affected Components: apps/api/src/modules/auth/controllers/login.auth.controller.ts (lines 28-34)

Proof of Concept:

```
response
  .cookie('access_token', authToken.access_token, {
    httpOnly: true,
    expires: new Date(Date.now() + 1000 * 60 * 60),
    domain: process.env.COOKIE_DOMAIN,
```

```
 })
 .send({
```

Remediation and Verification

PR #299 commit 0b2f8fc adds secure: true to the cookie options block in apps/api/src/modules/auth/controllers/login.auth.controller.ts. The current development branch confirms the change is in place.

```
response.cookie('access_token', authToken.access_token, {
  httpOnly: true,
  expires: new Date(Date.now() + 1000 * 60 * 60),
  domain: process.env.COOKIE_DOMAIN,
  secure: true,
});
```

The flag is set unconditionally rather than being gated on NODE_ENV. This is the safer choice for production but will prevent local HTTP development against this branch unless TLS is also used in development. CoRD considers this finding fully resolved. As a small follow-up, CoRD recommends adding a SameSite attribute (Lax for typical session use, or Strict where the application does not need cross-site navigation to authenticated pages) to harden against CSRF in concert with the existing CORS allowlist.

CVE-2026-SHIRA2-002: JWT_SECRET not validated at startup

Severity: Medium | **Category:** Auth & Session

CVSS Score: 7.0 (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N)

CWE ID: CWE-755 (Improper Handling of Exceptional Conditions)

Original Status: Open | Final Status: **Partially Resolved** | Fix Reference: PR #299, commit b1ea50f

Original Finding

Description: The API reads process.env.JWT_SECRET in the JWT strategy and in the auth module's JwtModule.register() without checking that it is present or meets a minimum strength. If the variable is missing or weak, the application still starts and JWT verification may fail or use a weak or default secret.

Impact: Missing or weak JWT_SECRET can lead to trivial token forgery and full authentication bypass. Failing fast at bootstrap would prevent running in an insecure configuration.

Affected Components: apps/api/src/modules/auth/strategy/jwt.auth.strategy.ts (line 24); apps/api/src/modules/auth/auth.module.ts (line 32)

Proof of Concept:

```
// jwt.auth.strategy.ts
super({
  jwtFromRequest: (req: Request) => {
    if (req?.cookies?.['access_token']) return req.cookies['access_token'];
    return ExtractJwt.fromAuthHeaderAsBearerToken()(req);
  },
  ignoreExpiration: false,
  secretOrKey: process.env.JWT_SECRET,  // no validation
});

// auth.module.ts
JwtModule.register({
  secret: process.env.JWT_SECRET,  // no validation
  signOptions: { expiresIn: '1d' },
}),
```

Remediation and Verification

PR #299 commit b1ea50f introduces a validateJwt() function that runs before NestFactory.create() in apps/api/src/main.ts. The function reads process.env.JWT_SECRET and throws if it is missing or shorter than six characters.

```
async function validateJwt() {
  const JWT_SECRET = process.env.JWT_SECRET
  if (!JWT_SECRET || JWT_SECRET.length < 6) {
    throw new Error('JWT_SECRET not valid');
  }
}

async function bootstrap() {
  validateJwt()
  const app = await NestFactory.create<NestExpressApplication>(IndexModule);
  ...
```

The bootstrap validation is now present, which addresses the structural finding. However, the length floor of six characters is too low to be meaningful as an integrity control. The audit recommended a minimum of 32 characters, consistent with industry guidance for HMAC-SHA256 secrets used by NestJS JwtModule. A six-character secret is well within reach of offline brute-forcing for an attacker who can capture a single JWT, which is the precise risk the bootstrap check is meant to prevent.

There is a second issue with the implementation that warrants attention. The validateJwt function is declared as an async function and is invoked without await. A throw inside an async function is converted to a rejected Promise rather than a synchronous exception; because the caller does not await, the rejection becomes an unhandled rejection. CoRD verified experimentally that, under Node.js 22 with default settings, the unhandled rejection does cause the process to exit before NestFactory.create() completes its asynchronous

work, so the fail-fast behavior is achieved in practice. However, this depends entirely on Node's --unhandled-rejections=throw default. Any operator who runs the API under --unhandled-rejections=warn (or older Node behavior) would see the API start successfully with an invalid JWT_SECRET, which is the exact outcome the check is intended to prevent. The cleaner pattern is either to declare validateJwt as a synchronous function or to add await at the call site in bootstrap.

CoRD classifies this finding as Partially Resolved. The recommended follow-up is to raise the threshold to at least 32 characters, reject a small set of obvious placeholder values such as 'changeme', 'secret', or 'jwt', and either remove the async qualifier from validateJwt or add await to its invocation so that the check does not depend on Node's unhandled-rejection handling.

CVE-2026-SHIRA2-003: IDOR: getQuestion by id not scoped to space

Severity: High | **Category:** Authorization / IDOR

CVSS Score: 6.5 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N)

CWE ID: CWE-639 (Authorization Bypass Through User-Controlled Key)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #270

Original Finding

Description: The getQuestion endpoint is protected by @AuthController and @Roles(Role.SpaceAdmin) but fetches a question only by primary key id. It does not verify that the question belongs to the authenticated user's space (for example via a quiz-to-space relation). A SpaceAdmin of space A can therefore request any question id and retrieve content from another space.

Impact: Cross-tenant data exposure: space admins can read quiz questions belonging to other spaces by enumerating or guessing question IDs.

Affected Components:
apps/api/src/modules/question/controllers/list.question.controller.ts (lines 61-99, especially 68-94)

Proof of Concept:

```
@Get(':id')
@Roles(Role.SpaceAdmin)
async getQuestion(@Param('id') id: string, @Query('lang') lang: string) {
 // ...
 const query = this.questionRepository
  .createQueryBuilder('question')
```

```
  // ... joins ...
  .where('question.id = :id', { id })   // no space scoping
  .andWhere('questionTranslations.languageId = :languageId', {
    languageId,
  });
 const res = (await query.getMany()).shift();
 // ...
}
```

Remediation and Verification

PR #270 refactors apps/api/src/modules/question/controllers/list.question.controller.ts to delegate to a new ListQuestionService and adds @SpaceId() injection to the controller method.

```
@Get(':id')
@Roles(Role.SpaceAdmin)
async getQuestionById(
  @Param('id') id: string,
  @Query('lang') lang: string,
  @SpaceId() spaceId: number,
) {
  return this.listQuestionService.getQuestion(spaceId, id, lang);
}
```

The service then enforces space scoping in the underlying TypeORM query through joins on quizQuestions and quiz, with an explicit andWhere clause that constrains results to the authenticated user's space:

```
.leftJoin('question.quizQuestions', 'quizQuestions')
.leftJoin('quizQuestions.quiz', 'quiz')
...
.andWhere('quiz.space_id = :spaceId', { spaceId });
```

CoRD reviewed an earlier iteration of this fix that contained a stray closing parenthesis in the SQL fragment. The current development branch shows that issue corrected. CoRD considers this finding fully resolved and recommends one further hardening step: add an integration test that asserts a SpaceAdmin in space A receives null (or 404) when requesting a question id known to belong to space B, so that future refactors cannot silently re-introduce the IDOR.

CVE-2026-SHIRA2-004: IDOR: delete question by id not scoped to space

Severity: High | **Category:** Authorization / IDOR

CVSS Score: 8.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H)

CWE ID: CWE-639 (Authorization Bypass Through User-Controlled Key)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #270 and PR #304

Original Finding

Description: The delete-question controller is protected by authentication and the SpaceAdmin role, but it deletes by question id only. It does not check that the question belongs to the user's space (for example via a quiz-to-space relation). A SpaceAdmin of one space can therefore delete questions in another space.

Impact: Cross-tenant data loss and denial of service: space admins can delete quiz questions belonging to other spaces.

Affected Components:
apps/api/src/modules/question/controllers/delete.question.controller.ts (lines 20-27)

Proof of Concept:

```
@Delete(':id')
@Roles(Role.SpaceAdmin)
async handler(
  @Param('id') id: number
) {
  console.log('Deleting question with id:', id);
  await this.questionTranslationRepository.delete({ questionId: id });
  await this.questionRepository.delete(id);  // no space check
  return true;
}
```

Remediation and Verification

The remediation is split across two pull requests and produces a two-path architecture for question deletion. PR #270 patch 03 marks DeleteQuestionController as DEPRECATED and the current development branch restricts it to SuperAdmin only.

```
@Delete(':id')
@Roles(Role.SuperAdmin)
async handler(@Param('id') id: number) {
  await this.questionTranslationRepository.delete({ questionId: id });
  await this.questionRepository.delete(id);
  return true;
}
```

PR #304 then re-adds DeleteQuestionQuizController under apps/api/src/modules/quiz/controller/, scoped to SpaceAdmin, which validates ownership before delegating to the deletion service:

```
@Post('question/delete')
@Roles(Role.SpaceAdmin)
async handler(@Body() deleteDto: DeleteQuestionQuizDto, @SpaceId() spaceId: number) {
  await this.validateSpaceQuizService.execute(spaceId, deleteDto.quizId)
  await this.deleteQuestionQuizService.execute(deleteDto);
}
```

This is a clean architectural fix. SpaceAdmins no longer have any path that can delete by raw question id, and the path they do have (POST /quiz/question/delete) calls validateSpaceQuizService.execute(spaceId, quizId) before performing the deletion. CoRD considers this finding fully resolved. The validateSpaceQuizService logic should be covered by a regression test that confirms a SpaceAdmin in space A receives an authorization error when attempting to delete a question belonging to a quiz in space B.

CVE-2026-SHIRA2-005: Unauthenticated quiz-run API

Severity: High | **Category:** Authentication / IDOR

CVSS Score: 8.2 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:N)

CWE ID: CWE-306 (Missing Authentication for Critical Function)

Original Status: Open | Final Status: **Risk Accepted** | Fix Reference: PR #270 (partial mitigation)

Original Finding

Description: The quiz-run controller has no @AuthController or UseGuards(AuthGuard('jwt')). Anyone can POST to start a run, PATCH to finish a run, and GET runs by spaceId. The spaceId in GET is taken from the URL and not validated against an authenticated user.

Impact: Unauthenticated users can create and finish quiz runs and list runs for any space, enabling data manipulation and information disclosure.

Affected Components: apps/api/src/modules/quiz_result/controller/quiz-run.controller.ts (lines 10-41)

Proof of Concept:

```
@Controller('quiz-run')
export class QuizRunController {
 // No AuthController or UseGuards
 @Post()
 async start(@Body() dto: StartQuizRunDto) {
  return await this.startRun.execute(dto);
 }
 @Patch(':runId/finish')
 async finish(
  @Param('runId', ParseIntPipe) runId: number,
  @Body() dto: FinishQuizRunDto
 ) {
  return await this.finishRun.execute(runId, dto);
 }
 @Get(':spaceId')
```

```
  async getAllRuns(
    @Param('spaceId', ParseIntPipe) spaceId: number,
  ) {
    return await this.quizRuns.getLatestBySpaceId(spaceId);  // spaceId from URL
  }
}
```

Remediation and Verification

PR #270 patch 10 removes the GET /:spaceId handler from QuizRunController entirely, along with the associated GetQuizRunsByQuizService and its interface. This eliminates the most severe exploit path in the original finding: anonymous enumeration of any space's quiz runs by URL parameter.

The current development branch state of apps/api/src/modules/quiz_result/controller/quiz-run.controller.ts is shown below. The decorator is @Controller, not @AuthController, so POST and PATCH remain unauthenticated.

```
@Controller('quiz-run')
export class QuizRunController {
  @Post() async start(@Body() dto: StartQuizRunDto) { … }
  @Patch(':runId/finish') async finish(…) { … }
}
```

CoRD's reading is that the unauthenticated POST and PATCH paths are consistent with Shira's product model: published-as-public quizzes are explicitly designed for unauthenticated learners, so the start and finish endpoints must accept anonymous traffic. The remaining residual risk is that any party who can guess or iterate runIds can call PATCH /quiz-run/:runId/finish on a run created by a different learner. This affects analytics integrity rather than confidentiality and is meaningfully reduced by the global throttler now in place.

Per Horizontal's April 2026 remediation summary, this finding has been designated as one that will not be remediated further. CoRD has updated the status to Risk Accepted to reflect Horizontal's decision. The mitigations described above (deletion of the GET endpoint, application of the global throttler) are in place; Horizontal has chosen not to pursue authentication or per-request integrity controls on POST and PATCH. CoRD documents the residual risk for future revisitation: an attacker who can guess or iterate sequential runIds can mark arbitrary runs as finished, polluting completion analytics for any tenant. Should Horizontal wish to revisit this decision, the lowest-effort hardening would be to return an opaque server-issued run token from POST /quiz-run and require it as a header on PATCH /quiz-run/:runId/finish.

CVE-2026-SHIRA2-006: POST /invitation unauthenticated and unrate-limited

Severity: Medium **|** **Category:** Rate Limiting / Abuse

CVSS Score: 7.2 (AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:L/A:L)

CWE ID: CWE-307 (Improper Restriction of Excessive Authentication Attempts) and CWE-770 (Allocation of Resources Without Limits or Throttling)

Original Status: Open | Final Status: **Risk Accepted** | Fix Reference: PR #299 (partial mitigation)

Original Finding

Description: The invitation controller accepts POST requests without authentication and without rate limiting. An attacker can send a high volume of invitation requests to arbitrary email addresses, leading to invitation bombing and potential abuse of email delivery or downstream services.

Impact: Invitation bombing, email abuse, and increased load on the application and email infrastructure.

Affected Components: apps/api/src/modules/auth/controllers/invite.auth.controller.ts (lines 5-16)

Proof of Concept:

```
@Controller('invitation')
export class InviteAuthController {
  @Post()
  async sendInvitation(@Body() invitationDto: SendInvitationDto) {
    await this.sendInvitationService.execute(invitationDto)
    return { success: true, message: 'Invitation sent successfully' }
  }
}
```

Remediation and Verification

PR #299 commit 747cf23 introduces ThrottlerModule.forRoot at the application level and registers ThrottlerGuard as APP_GUARD in apps/api/src/index.module.ts. The throttler configuration is 30 requests per 60 seconds:

```
ThrottlerModule.forRoot({
  throttlers: [{ ttl: 60000, limit: 30 }],
}),
...
{ provide: APP_GUARD, useClass: ThrottlerGuard }
```

The same patch adds app.set('trust proxy', 1) in main.ts so that the throttler keys on the real client IP behind nginx rather than the proxy. CoRD verified the live API at https://beta.space.shira.app/backend returns x-ratelimit-limit: 30, x-ratelimit-remaining, and x-ratelimit-reset headers, confirming the throttler is active in production.

Per Horizontal's April 2026 remediation summary, this finding has been designated as one that will not be remediated further. The audit's primary recommendation (rate limiting) is implemented via the global throttler. The audit's optional recommendations (CAPTCHA, and consideration of authentication for an admin-only invitation flow) are not being pursued. CoRD has updated the status to Risk Accepted to reflect Horizontal's decision and documents the residual risk: at the current rate, an attacker rotating IPs can still send a meaningful volume of invitations to attacker-controlled or harassment-target email addresses, and a single IP can sustain roughly 1,800 invitation requests per hour. Should Horizontal wish to revisit this decision, layering a CAPTCHA on the invitation form or applying a stricter per-route throttler (for example, 5 per minute per IP) would close this gap with minimal user-experience impact.

CVE-2026-SHIRA2-007: POST /survey unauthenticated and unrate-limited

Severity: Medium **|** **Category:** Rate Limiting / Abuse

CVSS Score: 6.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:L)

CWE ID: CWE-307 / CWE-770

Original Status: Open | Final Status: **Risk Accepted** | Fix Reference: PR #299 (partial mitigation)

Original Finding

Description: The survey controller accepts POST requests without authentication and without rate limiting. Attackers can submit a large number of survey responses, polluting data or causing resource exhaustion.

Impact: Survey spam, data pollution, and potential denial of service via excessive writes.

Affected Components: apps/api/src/modules/survey/controllers/create.survey.controller.ts (lines 5-15)

Proof of Concept:

```
@Controller('survey')
export class CreateSurveyController {
  @Post('')
  async handler(@Body() newSurvey: CreateSurveyDto) {
    this.createSurveyService.create(newSurvey)
  }
}
```

Remediation and Verification

The same global throttler configuration described under CVE-006 covers the survey controller. Survey submission is now subject to 30 requests per 60 seconds per IP.

Per Horizontal's April 2026 remediation summary, this finding has been designated as one that will not be remediated further. CoRD has updated the status to Risk Accepted to reflect Horizontal's decision. The global throttler is the sole mitigation; CAPTCHA and authentication are not being pursued. The residual risk is data pollution of survey results at up to 30 submissions per minute per IP, which is meaningful for a low-volume survey product. Should Horizontal wish to revisit this decision, the same per-route throttler tightening recommended under CVE-006 applies here.

CVE-2026-SHIRA2-008: Explanation content not sanitized (XSS)

Severity: High | **Category:** Input Validation

CVSS Score: 6.1 (AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N)

CWE ID: CWE-79 (Cross-site Scripting)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #289

Original Finding

Description: Question body content is sanitized with QuestionSanitizer.sanitizeQuestionContent() before persistence, but explanation text is stored as-is. Both the create and edit question services persist explanation.text (or content) without sanitization. When explanation content is rendered (for example in the frontend), stored XSS is possible.

Impact: Authenticated users such as a SpaceAdmin can inject a script into explanation content; when that content is rendered, other users may execute the script in their browser.

Affected Components:
apps/api/src/modules/quiz/services/create-question.quiz.service.ts (lines 98-104); apps/api/src/modules/quiz/services/edit-question.quiz.service.ts (lines 106-113)

Proof of Concept:

```
// create-question.quiz.service.ts - question content sanitized, explanation not
const sanitizedContent = QuestionSanitizer.sanitizeQuestionContent(originalContent);
// ...
const newExplanationTranslation =
  this.explanationTranslationRepo.create({
    explanation: savedExplanation,
    content: explanation.text,  // unsanitized
```

```
  languageId: language.id,
 })

// edit-question.quiz.service.ts - same pattern
content: explanation.text,  // unsanitized
```

Remediation and Verification

PR #289 applies QuestionSanitizer.sanitizeQuestionContent() to explanation.text in both create-question.quiz.service.ts and edit-question.quiz.service.ts:

```
const newExplanationTranslation = this.explanationTranslationRepo.create({
  explanation: savedExplanation,
  content: QuestionSanitizer.sanitizeQuestionContent(explanation.text),
  languageId: language.id,
});
```

This matches the audit's recommendation. The sanitizer is the same one already applied to question body content, so the input-validation layer is now consistent across the two fields. CoRD considers this finding fully resolved. As a defense-in-depth follow-up, CoRD recommends adding a Content-Security-Policy on the rendering frontend (which appears separately as an observation in this report) so that even an as-yet-unknown bypass in the sanitizer does not lead directly to script execution. Together with the localStorage migration in CVE-012, the original XSS-to-token-theft chain identified in the audit is broken in two places.

CVE-2026-SHIRA2-009: File upload: no MIME type or size validation

Severity: Medium | **Category:** Input Validation

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

CWE ID: CWE-434 (Unrestricted Upload of File with Dangerous Type)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #302

Original Finding

Description: The question-image upload endpoint accepts a file via FileInterceptor('file') and passes it to the service. There is no validation of MIME type, file size, or magic bytes. The path is built from quizId and a sanitized filename, but the stored file type is not restricted.

Impact: Malicious or oversized uploads (for example, executable content or large files) can be stored and served, leading to storage abuse, potential execution if served with the wrong Content-Type, or denial of service.

Affected Components:

apps/api/src/modules/question_image/controllers/create.question_image.controller.ts (lines 17-34);
apps/api/src/modules/question_image/services/create.question_image.service.ts (lines 19-56)

Proof of Concept:

```
// create.question_image.controller.ts
@Post('upload')
@UseInterceptors(FileInterceptor('file'))
async uploadFile(
  @UploadedFile('file') file: Express.Multer.File,
  @Query('quizId') quizId: number,
  @Query('questionId') questionId: number | null
) {
  if (!quizId || !file ) {
    throw new UnprocessableEntityException()
  }
  return this.createQuestionImageService.execute({ file, quizId, questionId });  // no MIME/size check
}
```

Remediation and Verification

PR #302 adds two complementary controls in apps/api/src/modules/question_image/. First, a 5 MB size cap at the multer interceptor level:

```
@UseInterceptors(FileInterceptor('file', {
  limits: { fileSize: 5 * 1024 * 1024 },
}))
```

Second, magic-byte content-type detection using the file-type package, run at the start of the service execute():

```
private async validateFile(file: Express.Multer.File) {
  const type = await fileTypeFromBuffer(file.buffer);
  if (!type || ![‘image/jpeg', 'image/png', 'image/webp'].includes(type.mime)) {
    throw new FileInvalidException()
  }
}
```

Magic-byte detection is the right approach because it cannot be spoofed by simply renaming a file or setting an attacker-controlled Content-Type header. The new FileInvalidException returns 400 with a stable error code, which is appropriate. CoRD considers this finding fully resolved. One small follow-up that is outside the original scope of CVE-009: the controller still accepts quizId from a query string parameter without space validation, so a malicious authenticated SpaceAdmin could potentially attempt to upload images attached to quizzes outside their space. CoRD recommends extending

validateSpaceQuizService (already used for question deletion under CVE-004) to the upload path.

CVE-2026-SHIRA2-010: CORS allows any origin (non-credentialed)

Severity: Medium | **Category:** Security Headers / CORS

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

CWE ID: CWE-942 (Overly Permissive Cross-domain White List)

Original Status: Open | Final Status: **Resolved** | Fix Reference: Commit 3536a20

Original Finding

Description: The API calls app.enableCors() with no options, which relies on framework defaults. In the current configuration, Access-Control-Allow-Origin: * allows any origin to access non-credentialed API endpoints in a browser. Although cookies are not sent in this configuration, it still expands the attack surface: unauthenticated or weakly protected endpoints are accessible from any website.

Impact: Any origin can interact with the API on behalf of users who have cookies; combined with other issues, this can facilitate cross-site attacks.

Affected Components: apps/api/src/main.ts (line 13)

Proof of Concept:

```
// apps/api/src/main.ts
async function bootstrap() {
  const app = await NestFactory.create(IndexModule);
  // No CORS options specified - relies on framework defaults
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

Remediation and Verification

Commit 3536a20 finalizes the CORS configuration in apps/api/src/main.ts to an explicit allowlist:

```
app.enableCors({
  origin: [process.env.SPACE_URL, process.env.PUBLIC_URL, process.env.SUPERADMIN_URL],
  credentials: true,
});
```

CoRD verified the live behavior at https://beta.space.shira.app/backend. A preflight from Origin: https://evil.example.com receives no Access-Control-Allow-Origin header, while a preflight from Origin: https://beta.space.shira.app receives access-control-allow-origin:

https://beta.space.shira.app and access-control-allow-credentials: true. The Vary: Origin header is correctly emitted in both cases. This finding is fully resolved. Going forward, when additional frontend hosts are added (for instance for staging environments), they should be added to this allowlist via environment variables; wildcards or patterns should not be reintroduced.

CVE-2026-SHIRA2-011: No rate limiting

Severity: Medium | **Category:** Rate Limiting / Abuse

CVSS Score: 6.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L)

CWE ID: CWE-307 (Improper Restriction of Excessive Authentication Attempts)

Original Status: Open | Final Status: **Partially Resolved** | Fix Reference: PR #299

Original Finding

Description: The API does not implement application-level rate limiting (for example NestJS ThrottlerGuard). High-value endpoints such as login, register, reset-password, invitation, survey, confirm, and quiz-run can be called without throttling, enabling brute-force attempts, abuse, and denial of service.

Impact: Brute-force attacks on authentication, invitation/survey abuse, and resource exhaustion from unbounded request rates.

Affected Components: Entire API; bootstrap in apps/api/src/main.ts (no ThrottlerModule/ThrottlerGuard)

Proof of Concept:

```
// main.ts - no ThrottlerGuard or rate-limiting middleware
async function bootstrap() {
  const app = await NestFactory.create(IndexModule);
  app.enableCors();
  app.useGlobalInterceptors(...);
  app.useGlobalPipes(...);
  await app.listen(3000);
}
```

Remediation and Verification

As described under CVE-006, a global @nestjs/throttler is now configured. The current setting of 30 requests per 60 seconds per IP, applied uniformly across all routes, is a meaningful baseline and resolves the categorical finding. However, the audit specifically called out that high-value endpoints, including login, register, reset-password, and confirm,

warrant tighter limits because the consequences if brute-force succeeds are direct (credential compromise) and the legitimate request volume on these endpoints is low.

CoRD classifies this finding as Partially Resolved. The recommended follow-up is to layer per-route @Throttle() decorators on /login, /reset-password, and /space-registration with stricter limits in the range of 5 to 10 requests per minute per IP, and to consider returning 429 with progressive backoff on repeated failures. Note that /invitation and /survey are no longer included in this recommendation, as Horizontal has accepted the residual risk on those endpoints under CVE-006 and CVE-007 respectively. The deletion of the legacy /register and /confirm endpoints in PR #302 substantially reduces the relevant attack surface; the remaining priorities are the three routes listed above.

CVE-2026-SHIRA2-012: Spaces app stores JWT in localStorage

Severity: High | **Category:** Frontend / Token Storage

CVSS Score: 6.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

CWE ID: CWE-522 (Insufficiently Protected Credentials)

Original Status: Open | Final Status: **Partially Resolved** | Fix Reference: PR #291

Original Finding

Description: The Spaces frontend stores the access token and X-Space id in localStorage after login and reads them in checkAuth. Tokens in localStorage are accessible to any script on the same origin (including XSS) and are not protected by HttpOnly as they would be with the existing cookie policy. The API already sets an httpOnly cookie; the client duplicates the token in a less secure store.

Impact: If an XSS flaw exists (for example via unsanitized content), an attacker can steal the token from localStorage and impersonate the user. Storing tokens only in httpOnly cookies would limit this risk.

Affected Components: apps/spaces/src/fetch/auth.ts (lines 27, 30, 39-40)

Proof of Concept:

```
export const login = async(email, pass) => {
 try {
  const res = await axios.post(`${process.env.REACT_APP_API_URL}/login`, {
   email: email,
   password: pass
  })
  window.localStorage.setItem('shira_access_token', res.data.access_token)
  // ...
  window.localStorage.setItem('shira_x_space', res.data.user.spaces[0].id)
```

```
  // ...
 }
}

export const checkAuth = async() => {
 const token = window.localStorage.getItem('shira_access_token')
 const spaceId = window.localStorage.getItem('shira_x_space')
 // ...
}
```

Remediation and Verification

PR #291 introduces an isProduction() check in apps/spaces/src/fetch/auth.ts that gates the localStorage path. In production, the client sets axios.defaults.withCredentials = true and relies on the API's httpOnly access_token cookie. A new POST /logout endpoint at apps/api/src/modules/auth/controllers/logout.auth.controller.ts clears that cookie, and the spaces app's logout slice now calls the API rather than just clearing localStorage.

In non-production builds, however, the client still writes shira_access_token to window.localStorage and reads it back as a Bearer token:

```
if (isProduction()) {
 axios.defaults.withCredentials = true;
} else {
 const token = res.data.access_token;
 axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
 window.localStorage.setItem('shira_access_token', res.data.access_token);
}
```

Independently, shira_x_space is written to localStorage unconditionally in both modes. The space identifier is not itself a credential, but its presence in localStorage means any successful XSS still has signal about which tenant the victim is logged into.

CoRD classifies this finding as Partially Resolved. The intended production posture is correct and matches the audit recommendation. Two follow-ups would close the finding fully. First, isolate the dev-mode Bearer token path behind a build-time feature flag (or remove it altogether and require a working cookie path in development) so that production-build artifacts cannot fall through to the localStorage branch under any runtime condition. Second, store the space identifier in React state or in a non-localStorage store so that no part of the authenticated session is reachable from script.

CVE-2026-SHIRA2-013: Confirm registration returns 404 for invalid hash

Severity: Low **|** **Category:** Sensitive Data / Enumeration

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

CWE ID: CWE-203 (Observable Discrepancy in Information)

Original Status: Open | Final Status: **Resolved** | Fix Reference: PR #302

Original Finding

Description: When the confirm-registration endpoint is called with an invalid or non-existent invitation hash, the service throws NotFoundException(), which typically results in an HTTP 404. Other error conditions such as expired hashes may return different status codes or messages. This discrepancy allows an attacker to distinguish 'hash does not exist' from 'hash expired or already used'.

Impact: User enumeration: attackers can probe hashes to discover valid versus invalid registration links, weakening the secrecy of invitation links.

Affected Components:
apps/api/src/modules/auth/services/confirm-registration.auth.service.ts (lines 35-41)

Proof of Concept:

```
const registration = await this.regRepo.findOne({ where: {
   invitationHash: inviteHash
}})

if (!registration) {
  throw new NotFoundException()  // 404 reveals hash does not exist
}
```

Remediation and Verification

PR #302 patch 2 removes the entire legacy registration flow: apps/api/src/modules/auth/controllers/confirm.auth.controller.ts, registration.auth.controller.ts, the corresponding services, interfaces, and provider entries are all deleted. The remaining production registration path is space-registration, which authenticates against a passphrase rather than an opaque invitation hash and therefore does not expose the same enumeration discrepancy.

CoRD considers this finding fully resolved with a small note for the auth service team: the surviving validate-registration.auth.service.ts returns distinct exceptions for 'Passphrase invalid' (UnauthorizedException) and 'user already exists' (AlreadyExistUserException). This is not the same vulnerability as the original CVE-013, but it is the same general class of issue. If product needs allow it, normalizing these to a single generic message during the public-facing space-registration flow would be a worthwhile hardening step.

Dependencies Findings:

This section outlines the findings of CoRD's security audit of Horizontal's code base review; the following items were identified during the audit but classified as observations rather than CVE-level vulnerabilities due to their nature (hardening, compliance, low-impact data handling). CoRD conducted a dependency audit of the repository using `nix-shell -p nodejs_24 --run "npm audit"` at the project root. The scan identified 80 known vulnerabilities across the project's dependency tree (4 low, 17 moderate, 58 high, and 1 critical). Each finding was triaged between February 24 and March 14, 2026, and assigned one of four remediation paths: no fix available (accepted risk), complex or deferred (requires upstream updates or architectural changes such as migrating off Create React App), review and apply (non-breaking fixes available via `npm audit fix`), or verify (possible false positive). Dependency findings were out of scope for the May 2026 verification phase; CoRD recommends that Horizontal schedule a follow-up dependency review to determine whether previously deferred or accepted-risk dependencies have since become fixable. The following table summarizes the dependency findings as recorded at the close of the audit phase.

Severity	Package	Versions	Advisory / Issue	Fix	Status
Critical	fast-xml-parser	≤4.5.3	Entity encoding bypass, DoS, stack overflow	npm audit fix	Deferred (Minio upstream)
High	axios	≤0.30.2	CSRF, SSRF/credential leakage, DoS - apps/spaces	audit fix --force (breaking)	Review
High	elliptic	*	Weak crypto (browserify-sign, create-ecdh)	--force (breaking)	No fix available
High	jsonpath	*	Code injection (bfj)	npm audit fix	No fix available
High	html-minifier (kangax)	*	REDoS via mjml-core	--force → mailer@1.8.1 (breaking)	No fix available
High	glob	10.2.0-10.4.5	CLI command injection (@nestjs-modules/mailer)	--force (breaking)	Complex
High	MJML	≤5.0.0-alpha	Via html-minifier REDoS	--force (breaking)	No fix available
High	minimatch	multiple	ReDoS - many transitive uses	npm audit fix	Review
High	multer	≤2.0.2	DoS - @nestjs/platform-express	npm audit fix	Review

Severity	Package	Versions	Advisory / Issue	Fix	Status
High	nth-check	<2.0.1	ReDoS (svgo → @svgr/webpack → react-scripts)	--force (breaking)	Complex
High	object-path	≤0.11.7	Prototype pollution (sort-by, spaces app)	--force (breaking)	Review
High	public	*	XSS/path traversal - possible name collision	n/a	Verify (false positive)
High	rollup	<4.59.0, <2.80.0	Path traversal / file write	npm audit fix	Review
High	serialize-javascript	≤7.0.2	RCE - workbox-build, terser-webpack-plugin	--force (breaking)	Review
High	storybook	8.1.0-8.6.16	WebSocket hijacking (dev-only)	npm audit fix	Review
High	underscore	≤1.13.7	DoS via jsonpath	npm audit fix	Review
High	webpack-dev-server	≤5.2.0	Source code exposure (react-scripts)	Requires CRA migration	Complex
Moderate	ajv	<6.14.0; 7.x <8.18.0	ReDoS with $data - schematics/devkit	--force (breaking)	Complex
Moderate	bn.js	various	Infinite loop (elliptic chain)	npm audit fix	Review
Moderate	postcss	<8.4.31	Parsing - resolve-url-loader	--force (breaking)	Complex

Dependency Triage Summary

No fix available (accepted risk): jsonpath, elliptic, MJML, kangax (html-minifier).

Complex or deferred: webpack-dev-server (requires CRA migration), fast-xml-parser (waiting for Minio upstream), ajv, glob, postcss, nth-check.

Review and apply non-breaking fixes: axios, minimatch, multer, object-path, rollup, serialize-javascript, storybook, underscore, bn.js. The package named 'public' should be verified as a possible false positive.

Observation Findings

The audit identified four observation-class items that did not meet the threshold for CVE-style classification but warrant attention as part of standard security hygiene. None of

these are exploitable in the same direct sense as the CVE findings, but each represents a hardening or compliance gap that CoRD recommends Horizontal address in the next development cycle. Observation findings were out of scope for the May 2026 verification phase and remain as documented at the close of the audit.

Observation: No security headers (Helmet/CSP)

Severity: Low | **Category:** Security Headers / CORS | **CWE:** CWE-693 (Protection Mechanism Failure)

The NestJS application does not use Helmet or set security headers such as X-Content-Type-Options, X-Frame-Options, or Content-Security-Policy. The API and any HTML it serves are therefore missing standard browser-side protections against client-side attacks. The impact is reduced defense-in-depth against clickjacking, MIME sniffing, and related issues when the API or its frontends are used in a browser context.

The recommended remediation is to install Helmet and apply it as global middleware in apps/api/src/main.ts, and to configure a Content-Security-Policy for any HTML-serving routes. Equivalent headers should be applied to the React frontends (super.shira.app, space.shira.app, quiz.shira.app).

Observation: Logging of PII (email, userId)

Severity: Low | **Category:** Sensitive Data | **CWE:** CWE-532 (Insertion of Sensitive Information into Log File)

Several auth and learner services log email addresses and user IDs in plaintext. Examples include log lines such as 'Processing password reset request for email: ...', 'Reset password token is valid for user: ...', and 'Inviting learner with email: ...'. Log files may be retained or accessible to more parties than intended, increasing the exposure surface for PII. The compliance impact depends on whether Horizontal's deployments are subject to GDPR or CCPA.

Affected components include: apps/api/src/modules/auth/services/request-password-reset.auth.service.ts, validate-reset-password-token.auth.service.ts, confirm-reset-password.auth.service.ts, and apps/api/src/modules/learner/services/invite.learner.service.ts. The recommended remediation is to redact or omit email and userId in production log output, or to use structured logs with minimal identifiers (truncated hashes or correlation IDs).

Observation: Registration entity stores passphrase in plaintext

Severity: Low | **Category:** Sensitive Data / Storage

The registration entity at apps/api/src/modules/auth/domain/registration.entity.ts stores the registration passphrase in plaintext. The impact is limited because the passphrase is one-time use and has a TTL, but storing it as ciphertext or as a hash would reduce exposure in the event of a database compromise. CoRD recommends assessing whether the passphrase can be hashed (with a verifier check on use) or whether the TTL can be shortened further.

Observation: Root .gitignore does not list .env

Severity: Low | **Category:** Configuration / Secrets

The .gitignore files in apps/api/, apps/public/, and apps/spaces/ all include .env, but the root .gitignore does not. This creates a small risk that environment files placed at the repository root could be committed inadvertently. The recommended remediation is to add .env and .env.* (excluding .env.example) to the root .gitignore, and to audit git history for any previously committed secrets, rotating any that are found.

Authentication Architecture and Route Coverage

This section documents Shira 2.0's authentication and route protection architecture as observed during the audit, with annotations recording how each unauthenticated route was treated through the verification phase. The architecture establishes context for the IDOR and missing-authentication findings (particularly CVE-003, CVE-004, and CVE-005) and for the rate-limiting findings against unauthenticated public endpoints (CVE-006 and CVE-007).

At a high level, the NestJS API enforces authentication and authorization through three layered controls. Protected routes are decorated with AuthController (or UseGuards(AuthGuard('jwt'))), which validates the JWT presented as a cookie or Bearer token. Authenticated requests then pass through RolesGuard, which checks that the user holds the required role (SuperAdmin, SpaceAdmin, or Learner) for the endpoint. Finally, controllers that operate on tenant-scoped resources read the X-Space header and call validateUserSpaceAccess (or use the @SpaceId() injector introduced through the verification phase) to confirm that the authenticated user is acting within a space they

belong to. Routes that do not need authentication are exposed without these guards; the public routes are listed in the table below.

Browser clients reach the API either with a cookie (httpOnly access_token, set by the API on successful login) or, in the case of older client builds, with a Bearer token. The verification phase work on CVE-012 has substantially eliminated the Bearer-token path in production builds of the spaces application, leaving the cookie as the primary authentication transport. The API enforces a CORS allowlist (resolved under CVE-010) to ensure that only known frontend origins can perform credentialed requests.

```
None
flowchart LR
  subgraph client [Client]
    Browser
  end
  subgraph api [NestJS API]
    Public[Public routes]
    Protected[Protected routes]
    Public --> Login[login, register, invitation, survey, reset-password, confirm, quiz-run, learner-quiz,
quiz/hash, demo, field_of_work, app, language]
    Protected --> AuthGuard[AuthGuard JWT + RolesGuard]
    AuthGuard --> XSpace[X-Space header + validateUserSpaceAccess]
    XSpace --> Controllers[quiz, question, space, organization, learners, etc.]
  end
  Browser -->|Cookie or Bearer| AuthGuard
  Browser -->|No auth| Public
```

Unauthenticated Routes

The following table lists every route in the API that does not use AuthController or UseGuards(AuthGuard('jwt')). Each row records the route, the methods it supports, whether public access is intentional, and CoRD's risk assessment as updated to reflect the verification phase outcomes (verification-phase content shown in yellow within each cell).

Route prefix or path	Methods	Intended public?	Risk note
/login	POST	Yes	Rate limit recommended; partially addressed by global throttler under CVE-011 (Partially Resolved).
/register	POST	Yes (legacy)	Endpoint deleted in PR #302; legacy registration flow superseded by /space-registration.

Route prefix or path	Methods	Intended public?	Risk note
/confirm	GET	Yes (token-based, legacy)	Endpoint deleted in PR #302; CVE-013 enumeration vector eliminated.
/reset-password	POST, GET validate/:token, POST confirm/:token	Yes	Rate limit recommended on POST; generic errors. Per-route limit recommended (CVE-011).
/invitation	POST	Unclear	Abuse risk: invitation bombing. Global throttler applied (CVE-006); Horizontal accepted residual risk.
/space-registration	POST	Yes	Rate limit recommended; per-route limit pending (CVE-011).
/survey	POST	Yes	Abuse risk: spam and DoS. Global throttler applied (CVE-007); Horizontal accepted residual risk.
/learner-quiz/:hash	GET	Yes (hash is secret)	Acceptable if hash remains unguessable; avoid leaking existence in error responses.
/quiz/hash/:hash	GET	Yes (hash is secret)	Acceptable; same considerations as /learner-quiz/:hash.
/quiz-run	POST, PATCH /:runId/finish	Per product model, yes	GET /:spaceId deleted in PR #270 (cross-tenant enumeration eliminated). POST and PATCH remain unauthenticated; Horizontal accepted residual risk on these (CVE-005).
/passphrase/:code/check-expired	GET	Yes	Token-based; acceptable.
/question/demo	GET	Yes	Read-only demo data.
/field_of_work	GET	Yes	Read-only reference data.
/app	GET	Yes	Read-only reference data.
/language	GET	Yes	Read-only reference data.
/i/url/:name	GET	Yes (presigned URL)	Validate name and path.
/ (index)	GET	Yes	Health and information endpoint.

Security by Design Assessment and Recommendations

Shira 2.0.0 Privacy and Security by Design

This section covers our analysis related to Shira 2.0, and our recommendation as to how to make the product more secure following Privacy by Design and Security by Design Principles. We pull from the United Kingdom's Government Security Office's Secure by Design Principles. These principles help define how the organization, or company, behind a product or piece of software can create their own systems to respond to any privacy and security concerns that appear related to the project at hand. These two methodologies combine best practices directly related to building products/software (across design and technology), and best practices for teams to implement who are in charge of maintaining, building and sustaining that software and product.

Security by Design has 10 principles: create responsibility for cyber risk; source secure technology products; adopt a risk driven approach; design usable security controls; build in detect and respond security; design flexible architectures; minimize the attack surface; defend in depth; embed continuous assurance; and make changes securely. These principles can also be adapted to the project at hand; not all principles need to be engaged with if it's not directly relevant or they can be reinterpreted for the software/project that is being analyzed.

While Shira 2.0.0 has many surfaces related to it, including different areas in which privacy concerns and security vulnerabilities can manifest, this Security by Design overview is presented to be a reflexive experiment for the Shira 2.0.0 team to consider, where as the Privacy by Design analysis below will contain much more specific suggestions for privacy improvements to the Shira 2.0.0 product.

Shira 2.0.0 Security by Design Analysis

- Create responsibility for cyber risk
    - This section refers to assigning specific team members responsibility for directly addressing any security or privacy risks and vulnerabilities. It's not enough to assume the general engineering team this responsibility; by ensuring that specific team members hold ownership over this role, and that the ownership is spread across teams, this ensures that the risks are addressed adequately and to minimize user harm.
    - The Horizontal team should consider what team members will be assigned to this responsibility and which team lead or executive staff member will oversee this specific response and responsibility as well.
- Source secure technology products
    - This section refers to "managing third party security risks" and "discovering vulnerabilities"; please refer to our above findings for this section.

    - Discovering vulnerabilities and addressing those vulnerabilities are important to ensure that the software/project is 'secure' and to be able to fully embody a Security by Design and a Privacy by Design methodology. By responding to and fixing vulnerabilities, this ensures the project is using or sourcing 'secure technology products.'
- Adopt a risk driven approach
    - This is usually defined by: documenting service assets, understanding the importance of service assets, sourcing threat assessments, performing threat modeling, performing security risk assets, responding to and mitigating security risks, etc
    - From CoRD Labs' analysis, given the background of the Horizontal team, and why Shira 2.0.0 was made, alongside reaching out to CoRD to conduct an in-depth security audit, the team seems to be engaging with this section of 'adopting a risk driven approach.' Adopting a risk driven approach can also be applied to actively threat modeling while building a product, including thinking about some of the suggestions below, such as analytics privacy, space management security and other considerations.
    - Once the Horizontal team has addressed our findings in this report, they will be able to check the box on this particular principle has been fulfilled.
- Design usable security controls
    - This refers to ensuring any security tools are following usability standards and that that secure tool is minimizing user friction. Given that Shira 2.0.0 is also an educational game, and not necessarily a tool for increasing security or privacy, but rather, teaching about those topics, this principle is not immediately relevant.
    - However, Shira 2.0.0 does seem to be designed with usability in mind, and centering all different types of users, their backgrounds and their needs; in essence, why the quiz exists and how the quiz was made does center usability.
- Build in detect and respond security
    - This refers to designing for the inevitability of security vulnerabilities and incidents, including thinking through the capability to detect, respond to and recover from incidents and designing fewer weak points where compromises could occur or go undetected.
    - Our above sections provide indepth suggestions on how to center 'build in detect and respond security' that fits Shira 2.0's design and relevant threat models. It's important to consider what kind of information Shira 2.0.0 is storing, how Shira 2.0.0 is build/architected and what kind of information

could be gleaned about its infrastructure, storage, and data users give to Shira 2.0.0 from Shira 2.0.0 the product.
    - For example, to embody this principle, directly addressing the findings in the above section of Core Security Architecture for Shira's Four-Component System is paramount, alongside addressing the other risks and vulnerabilities we surfaced. This principle also posits is it possible to have a system in place that programmatically checks for vulnerabilities or to make that checking a part of the maintenance of the Shira 2.0.0 project.
- Design flexible architectures
    - This principle suggests building software in a way that allows for legacy code to be updated without causing risk and vulnerabilities to the overall project, particularly to allow easier integration for new security changes to address risks as they are found.
    - Because Shira 2.0.0 asks for such minimal user data, there is a lowered risk but it is something for the Horizontal team to consider: has Shira 2.0.0 been built in a way that will allow for different types of updates without breaking other parts of the Shira 2.0.0 product or functionality?
- Minimize the attack surface
    - Given the specific, discreet and contained nature of Shira 2.0, it appears that much of the potential attack surface has already been minimized from how the quiz and its tooling have currently been designed.
- Defend in depth
    - This principle refers to “creating layered controls across a service so its harder for attackers to fully compromise a system” and given how Shira 2.0.0 is designed, and what it aims to do as a project, our analysis has determined that this principle has already been engaged with.
- Embed continuous assurance
    - Given the specific nature of what Shira 2.0.0 does as a product, this principle is not entirely relevant for this specific Security by Design analysis.
- Make changes securely
    - Given that the purpose of this security audit is to ensure that Shira 2.0.0 is surfacing risks and vulnerabilities and seeking expertise in how to address those findings as securely and safely as possible, this principle is actively being engaged with at the current moment.
    - For future or next steps, we encourage the Horizontal team to create or maintain their systems that ensure security of their software is centered alongside design and product changes, and that security not be compromised for a product change and vice versa. Security must be equally balanced and focused alongside product, design, and infrastructural changes.

- We recommend that the Horizontal team regularly ensure that the Shira 2.0.0 software and related libraries are consistently updated, and appropriately threat modeled over time, e.g. ensuring continued and consistent maintenance of the Shira project for user safety.

Threat Modeling for Shira 2.0.0

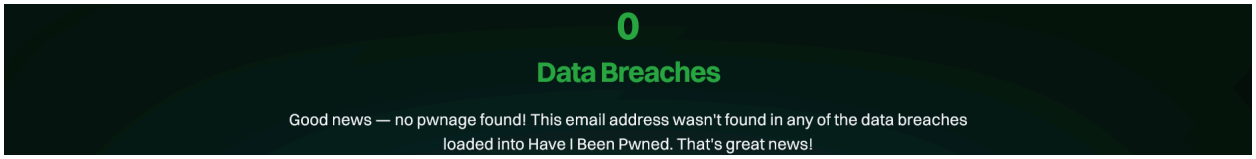
As an anti-phishing training platform serving civil society organizations, Shira 2.0.0 faces realistic but manageable security threats. While the platform is unlikely to be a primary target for sophisticated state actors or commercial spyware vendors, it could face opportunistic attacks, supply chain vulnerabilities, and become collateral damage in broader campaigns against civil society. The most probable threats include automated vulnerability scanning, dependency chain attacks common to JavaScript applications, and potential interest from regional actors if the platform gains significant adoption among sensitive organizations.

Contextual Risk Factors

Given Shira's deployment context, the following factors elevate certain risks:

- Device seizure scenarios
- Network surveillance capabilities
- Legal compulsion for data disclosure
- Targeted attacks against human rights defenders

Researchers Note: The quiz goes back to "contact@wearehorizontal.org" which has zero breaches according to HavelbeenPwned.com but should be monitored for future breaches, as well as implement hardware key 2FA.



Probable Threat Scenarios

Supply Chain and Dependency Risks

The most immediate and realistic threat to Shira 2.0.0 stems from its JavaScript ecosystem dependencies. Recent incidents demonstrate the vulnerability of npm-packages to

compromise. The 3CX incident affected over 600,000 organizations through compromised build infrastructure[1], while the Codecov breach impacted hundreds of technology companies through a single compromised script[2]. For a React/Node.js application like Shira, with potentially hundreds of dependencies, this represents the highest probability attack vector from bad actors.

The platform's previous audit by Subgraph (July 2023) identified multiple vulnerable JavaScript dependencies across all three components, including critical and high severity issues that remained unresolved[3]. This pattern of unpatched dependencies is common in civil society technology projects with limited resources for continuous security maintenance.

Opportunistic and Automated Attacks

Shira 2.0.0 will certainly face automated attacks common to all web applications. These include credential stuffing against administrator accounts, exploitation of known CVEs in unpatched dependencies, and SQL injection attempts against any database interfaces. The relatively small team and limited resources typical of human rights technology projects make timely patching challenging.

Regional Blocking and Interference

Given Shira's multilingual support for Mandarin, Persian, Russian, and Arabic, the platform may face access restrictions in countries with extensive internet controls. China's Great Firewall routinely blocks foreign educational platforms[4], Iran's National Information Network restricts access to international sites[5], and Russia's Sovereign Internet law enables similar controls[6]. However, these would likely be reactive blocks if the platform gains popularity, not proactive targeting.

Targeted Phishing Against Administrators

A realistic threat involves social engineering attacks against Shira administrators and space admins. Recorded Future documented campaigns specifically targeting NGO administrators

---

[1] CrowdStrike, "CrowdStrike Falcon Platform Detects and Prevents Active Intrusion Campaign Targeting 3CX Customers," March 29, 2023, https://www.crowdstrike.com/blog/crowdstrike-detects-and-prevents-active-intrusion-campaign-targeting-3cx-customers/.
[2] Reuters, "Codecov Hackers Breached Hundreds of Customer Sites," April 15, 2021, https://www.reuters.com/technology/codecov-hackers-breached-hundreds-customer-sites-sources-2021-04-15/.
[3] Subgraph Technologies, "Shira Security Audit Findings Report," July 31, 2023, provided document. https://drive.google.com/file/d/1TNf33bkFOZpUhl5adXVzaIB7arRe1vUB/view.
[4] Freedom House, "Freedom on the Net 2024: China," 2024, https://freedomhouse.org/country/china/freedom-net/2024.
[5] Freedom House, "Freedom on the Net 2024: Iran," 2024, https://freedomhouse.org/country/iran/freedom-net/2024.
[6] Human Rights Watch, "Russia: Growing Internet Isolation, Control, Censorship," June 18, 2020, https://www.hrw.org/news/2020/06/18/russia-growing-internet-isolation-control-censorship.

with credential harvesting pages mimicking common platforms[7]. Given that Shira administrators would potentially have access to multiple organizations' data, they represent attractive targets for adversaries seeking to map civil society networks.

Data Value for Reconnaissance

While Shira itself is unlikely to be targeted by APT groups, the aggregate data it collects about organizational security awareness levels could have intelligence value. Organizations consistently failing phishing simulations might be marked for future targeting. However, this would likely require the platform to achieve significant scale first.

Unlikely but Mentioned Threats

Commercial Spyware Targeting

Companies like NSO Group, Candiru, and Intellexa target specific individuals, not training platforms. These tools cost millions of dollars and are deployed against high-value individuals like journalists, activists, and politicians[8]. Shira as a platform would not warrant such expensive targeting, though individual users might separately face such threats.

Nation-State APT Direct Targeting

While reports document APT groups like Iran's APT42 targeting activists[9] and China's various groups targeting pro-democracy organizations[10], these actors focus on stealing data, not compromising training platforms. Shira might see scanning or reconnaissance, but dedicated APT operations against the platform itself are improbable unless it becomes critical infrastructure for major civil society organizations.

Realistic Security Recommendations

Immediate Priorities

1. Implement automated dependency scanning and updates
2. Deploy rate limiting and basic DDoS protection

---

[7] Recorded Future, "RedAlpha Credential Harvesting Campaign Targeting Humanitarian Organizations," December 2023, https://www.recordedfuture.com/research/redalpha-credential-harvesting-campaign.
[8] Citizen Lab, "Pegasus vs. Predator: Dissident's Doubly-Infected iPhone Reveals Cytrox Mercenary Spyware," December 16, 2021, https://citizenlab.ca/2021/12/pegasus-vs-predator-dissidents-doubly-infected-iphone-reveals-cytrox-mercenary-spyware/.
[9] Proofpoint, "Welcome to New York: Exploring TA453's Foray into LNKs and Mac Malware," September 2023, https://www.proofpoint.com/us/blog/threat-insight/welcome-new-york-exploring-ta453s-foray-lnks-and-mac-malware.
[10] Mandiant, "APT41: A Dual Espionage and Cyber Crime Operation," 2024, https://www.mandiant.com/resources/apt41-dual-espionage-and-cyber-crime-operation.

3. Enforce strong authentication (2FA) for all administrative accounts
4. Regular security updates for all non-automatable components

Medium-term Improvements

1. Content Security Policy to prevent XSS
2. Subresource Integrity for any CDN resources
3. Database encryption at rest
4. Audit logging for administrative actions

User Protection Measures

1. Optional email anonymization for learners
2. Automatic data deletion after training completion
3. Clear privacy policy about data handling
4. HTTPS enforcement with HTTP Strict Transport Security

Shira 2.0.0 faces real but manageable security challenges typical of civil society technology platforms. The greatest risks come from common web application vulnerabilities and supply chain attacks rather than sophisticated targeted operations. By focusing on fundamental security practices, particularly around dependency management and access control, the platform can provide adequate protection for its users without requiring nation-state level defenses.

Risk Matrix

Broad Contextual Risk Assessment for Human Rights Defenders

Risk Level	Technical Impact	Human Rights Context Impact
Critical	Complete system compromise, admin access breach	User identification leading to arrest/harassment, mass surveillance enablement, organizational infiltration
High	Data breach, authentication bypass, service manipulation	Targeted phishing campaign enablement, security training undermined, organizational data exposed
Medium	Limited data exposure, partial service disruption	Individual user tracking, reduced platform trust, temporary training unavailability

Low	Minor information disclosure, cosmetic issues	Minimal direct impact on user safety

Likelihood Factors

- **High Likelihood:** Unpatched dependencies, weak authentication, predictable tokens
- **Medium Likelihood:** Complex injection attacks, sophisticated social engineering
- **Low Likelihood:** Zero-day exploits (very low) & physical access attacks (also low)

Remediations and Fix Verifications

This section documents Convocation Research+Design's verification of Horizontal's remediation work for the thirteen vulnerabilities identified in CoRD's March 2026 security audit of Shira 2.0.0. Verification was conducted against the development branch of the github.com/Horizontal-org/shira repository and against the live deployment at https://beta.space.shira.app/backend. Each finding was traced to the specific pull request or commit identified by Horizontal's engineering team in theirremediation summary, the diffs were reviewed in context, and where feasible the resulting behavior was confirmed against the deployed beta environment.

Of the thirteen findings, seven are now Resolved, three are Partially Resolved, and three have been designated by Horizontal as Risk Accepted. None remain fully Open. The Partially Resolved findings each share a common pattern: the structural intent of the fix is correct, but a parameter, threshold, or code path leaves residual risk that CoRD believes can be closed with low effort. The three Risk Accepted findings (CVE-005, CVE-006, CVE-007) are items where Horizontal has implemented the audit's primary recommendation (deletion of the cross-tenant GET endpoint for CVE-005, global rate limiting for CVE-006 and CVE-007) but has decided not to pursue additional hardening such as per-request authentication on quiz runs, CAPTCHA on invitation, or stricter per-route limits on survey submission. CoRD documents the residual risk in each section so that Horizontal can revisit these decisions in future release cycles.

The most material residual items among the Partially Resolved set are the JWT_SECRET length floor of six characters under CVE-002, an additional implementation issue with the same fix where validateJwt is async but invoked without await, the absence of stricter per-route rate limits on authentication endpoints under CVE-011, and the dev-mode localStorage fallback that remains in the spaces frontend under CVE-012.

The cross-tenant authorization findings that drove the highest-priority recommendations in the original audit, namely CVE-003 and CVE-004 (cross-tenant question read and delete), are fully resolved, as is the CVE-008 stored XSS finding. The cross-tenant enumeration vector in CVE-005 (anonymous GET of any space's quiz runs) is also eliminated by deletion of that endpoint, even though Horizontal has accepted residual risk on the related

unauthenticated POST and PATCH paths. The XSS-to-token-theft chain identified in the original audit is broken in two places: explanation content is now sanitized at write time, and the production frontend no longer holds the JWT in script-readable storage.

Remediation Repo Locations:

- 009, 013: https://github.com/Horizontal-org/shira/pull/302
- 001, 002, 006, 007, 011: https://github.com/Horizontal-org/shira/pull/299
- 003, 004, 005: https://github.com/Horizontal-org/shira/pull/270
- 004: https://github.com/Horizontal-org/shira/pull/304
- 012: https://github.com/Horizontal-org/shira/pull/291
- 008: https://github.com/Horizontal-org/shira/pull/289
- 010:https://github.com/Horizontal-org/shira/commit/3536a20d743f2b0526e892be01f63cbf0701071c

Remediation Status Summary

The table below maps each original finding to its updated status, the pull request or commit that contains the fix, and a one-line summary. Detailed verification narratives follow in the next section.

CVE-ID	Severity	PR / Commit	Updated Status	Notes
SHIRA2-001	Medium	#299	Resolved	secure: true on access_token cookie.
SHIRA2-002	Medium	#299	Partially Resolved	Bootstrap validation added but length floor of 6 is far below recommended 32; fix is also called without await on an async function.
SHIRA2-003	High	#270	Resolved	getQuestion now requires SpaceId and filters by quiz.space_id.
SHIRA2-004	High	#270, #304	Resolved	Legacy DELETE locked to SuperAdmin; SpaceAdmin path validates space-quiz ownership.
SHIRA2-005	High	#270	Risk Accepted	GET endpoint removed entirely (eliminates cross-tenant exposure). POST/PATCH remain unauthenticated; Horizontal has accepted the residual risk and will not pursue further remediation.
SHIRA2-006	Medium	#299	Risk Accepted	Global ThrottlerGuard (30 req/60s) applies to /invitation. Horizontal will not add CAPTCHA or authentication beyond the throttler; residual risk accepted.

SHIRA2-007	Medium	#299	Risk Accepted	Same global throttler covers /survey. Horizontal will not add CAPTCHA or authentication beyond the throttler; residual risk accepted.
SHIRA2-008	High	#289	Resolved	QuestionSanitizer applied to explanation content in both create and edit services.
SHIRA2-009	Medium	#302	Resolved	5MB size limit at multer; magic-byte MIME validation via file-type library; restricted to jpeg/png/webp.
SHIRA2-010	Medium	3536a20	Resolved	CORS restricted to SPACE_URL, PUBLIC_URL, SUPERADMIN_URL with credentials:true. Verified live: unknown origins receive no Access-Control-Allow-Origin header.
SHIRA2-011	Medium	#299	Partially Resolved	Global throttler at 30/min/IP is a baseline. Authentication endpoints should have stricter per-route limits (5-10/min).
SHIRA2-012	High	#291	Partially Resolved	Production path now uses httpOnly cookie + withCredentials. Dev-mode code path still writes shira_access_token to localStorage; shira_x_space is written unconditionally.
SHIRA2-013	Low	#302	Resolved	Legacy /confirm and /register endpoints removed entirely; enumeration vector eliminated.

Risk Acceptance Summary

Horizontal has formally accepted the residual risk associated with three findings from the March 2026 audit: CVE-2026-SHIRA2-005 (unauthenticated quiz-run POST and PATCH), CVE-2026-SHIRA2-006 (no CAPTCHA or authentication on invitation despite throttling), and CVE-2026-SHIRA2-007 (no CAPTCHA or authentication on survey despite throttling). For each of these findings, mitigation has been applied (deletion of the cross-tenant quiz-run GET endpoint, and a global 30-requests-per-60-second throttler covering all endpoints), and the gap between the applied mitigation and full remediation has been documented in the corresponding finding section above.

CoRD does not view this acceptance as creating direct, high-severity exposure given the mitigations in place. The principal residual risks are that an attacker iterating runIds may pollute analytics by marking unrelated runs as finished, and that an attacker rotating IPs

may sustain meaningful invitation or survey volume despite the global throttler. These are integrity and abuse risks rather than confidentiality risks.

CoRD recommends that Horizontal revisit these decisions if the platform's user base or threat profile changes materially, and in particular if the platform begins to be used by organizations whose adversaries are known to operate IP-rotation infrastructure or coordinated abuse campaigns. The lowest-effort hardening for each Risk Accepted finding is documented in the corresponding finding section, ranging from an opaque server-issued run token for the quiz-run controller (CVE-005) to a stricter per-route throttler or CAPTCHA on invitation and survey endpoints (CVE-006 and CVE-007).

The Risk Accepted designation reflects Horizontal's decision and does not constitute CoRD's recommendation that the risk be left in place. It is a record of the current posture so that the residual risk is explicit and traceable in the engagement record.

Conclusion

Shira 2.0.0 demonstrates a sound architectural foundation: NestJS with JWT-based authentication, role-based access control via decorators, and clear separation between public and protected routes. The codebase shows evidence of intentional security thinking from the outset: question content sanitization was implemented before this audit began, httpOnly cookies were used for token transport on the API side, and authentication guards were applied consistently across most sensitive controllers. These design choices provided meaningful starting points for a tool serving at-risk populations.

The original audit identified five High-severity vulnerabilities that, taken together, represented significant risk to multi-tenant data integrity and user safety. The IDOR findings (CVE-2026-SHIRA2-003 and -004) allowed cross-space data access and deletion by any authenticated SpaceAdmin, the unauthenticated quiz-run API (CVE-2026-SHIRA2-005) exposed run data and manipulation to anonymous attackers, the stored XSS vector in explanation content (CVE-2026-SHIRA2-008) enabled script injection in a multi-tenant context, and the localStorage token storage (CVE-2026-SHIRA2-012) compounded those risks by enabling token theft via injected scripts. These findings were not exotic; they followed well-documented vulnerability patterns (OWASP A01:2021 Broken Access Control and A03:2021 Injection) and were addressable with targeted, low-to-medium effort code changes. The Medium-severity findings represented standard hardening items typical at this stage of development that did not individually pose acute exploitation risk, but collectively widened the attack surface.

Horizontal's engineering team has delivered substantive, well-targeted fixes for the thirteen findings in CoRD's March 2026 audit. Seven of the findings are fully resolved, three are partially resolved with documented residual recommendations, and three have been designated by Horizontal as Risk Accepted with the residual risk documented for future revisitation. The fully resolved set includes the cross-tenant authorization fixes (CVE-003 and CVE-004) and the stored XSS fix (CVE-008), which together address the most exploitable findings in the original audit. The Partially Resolved findings are each tractable with low-effort follow-up work, and CoRD does not consider any of them blocking for the current Shira release.

CoRD's overall view is that Shira 2.0.0 is in a substantially stronger security posture than at the time of the original audit, particularly with respect to the multi-tenant authorization model that is core to the new spaces feature set. The clustering of fixes around input validation, authorization scoping, and rate limiting addresses the systemic gaps identified in the audit's executive summary.

CoRD thanks the Horizontal team for their constructive engagement throughout both the audit and verification phases of this work. The platform's security posture and the quality of the remediation work reflect an organization that takes seriously its responsibility to the human rights defenders, journalists, and civil society organizations who use Shira to build phishing resilience in genuinely hostile environments.

Appendices

Appendix A: Glossary of Terms

API (Application Programming Interface): A way for different software systems to communicate with each other. In Shira, the API connects the frontend (what users see) to the backend (where data is processed).

APT (Advanced Persistent Threat): A prolonged, targeted cyberattack where an intruder gains access to a network and remains undetected. APT groups are typically state-sponsored (e.g., APT42, linked to Iran).

Authentication: The process of verifying someone is who they claim to be (example: logging in with a password and a second factor like a code from an app).

Authorization: Determining what an authenticated user is allowed to do. A space admin can manage quizzes but cannot access super admin functions.

Bidirectional Text (BiDi): Text that includes both left-to-right (English) and right-to-left (Arabic, Persian) scripts. Improper handling can create security vulnerabilities.

Certificate Pinning: A security technique where an application only trusts specific cryptographic certificates, making it harder for attackers to intercept encrypted traffic.

CI/CD (Continuous Integration/Continuous Deployment): Automated processes that build, test, and deploy code changes. A compromised CI/CD pipeline can inject malicious code into otherwise trusted software.

Content Security Policy (CSP): A browser security mechanism that restricts what resources (scripts, images, etc.) a webpage can load, reducing the risk of code injection attacks.

Credential Stuffing: An automated attack that tries username/password combinations leaked from other breaches against a different service, hoping users reused passwords.

CVE (Common Vulnerabilities and Exposures): A standardized identifier for a known security vulnerability (e.g., CVE-2024-XXXXX). Used to track and reference specific flaws in software.

CVSS (Common Vulnerability Scoring System): A numerical score (0.0-1.0.0) rating the severity of a security vulnerability. Higher scores indicate greater risk.

CWE (Common Weakness Enumeration): A catalog of software and hardware weakness types. Used in findings to classify the category of vulnerability (e.g., CWE-79 for cross-site scripting).

DDoS (Distributed Denial of Service): An attack that floods a server with traffic from many sources to make it unavailable to legitimate users.

Dependency: Third-party code libraries a project relies on. Shira uses hundreds of npm packages; a vulnerability in any one can affect the entire application.

Differential Privacy: A mathematical technique for sharing aggregate statistics about a group while protecting individual data points from being identified.

DKIM/SPF/DMARC: Email authentication standards that verify messages actually come from the claimed sender domain, preventing email spoofing.

Domain Fronting: A technique that disguises the true destination of internet traffic by routing it through a trusted domain (e.g., a cloud provider), used to circumvent censorship.

Encrypted ClientHello (ECH): A protocol extension that encrypts the initial part of a secure connection, preventing network observers from seeing which website a user is visiting.

Homograph Attack: A trick using visually identical characters from different alphabets (e.g., Latin "a" vs. Cyrillic "a") to create deceptive URLs or email addresses.

HSTS (HTTP Strict Transport Security): A security header that forces browsers to only connect to a site over encrypted HTTPS, preventing downgrade attacks.

Iframe: An embedded frame within a webpage that can display separate content. "Sandboxed" iframes restrict what the embedded content can do.

k-Anonymity: A privacy property where any individual's data is indistinguishable from at least k-1 other individuals in the dataset. Shira uses a threshold of 5.

Lateral Movement: After gaining access to one system, an attacker's ability to move to other connected systems. Component isolation prevents this.

MITM (Man-in-the-Middle): An attack where someone secretly intercepts and potentially alters communications between two parties who believe they are communicating directly.

npm (Node Package Manager): The package manager for JavaScript/Node.js. "npm audit" scans installed packages for known vulnerabilities.

OWASP (Open Web Application Security Project): A nonprofit producing widely used security standards, tools, and guidelines for web application security.

Penetration Testing (Pen Testing): Authorized simulated attacks against a system to identify exploitable vulnerabilities before real attackers do.

PII (Personally Identifiable Information): Any data that could identify a specific individual: names, email addresses, IP addresses, quiz results tied to a person.

PostgreSQL RLS (Row-Level Security): A database feature that restricts which rows a query can access based on the user's role, enforcing data isolation between organizations.

Rate Limiting: Restricting how many requests a user or IP address can make in a given time period to prevent abuse, brute-force attacks, or spam.

Role-Based Access Control (RBAC): A system where permissions are assigned to roles (super admin, space admin, learner) rather than individual users.

SRI (Subresource Integrity): A browser feature that verifies externally loaded files (scripts, stylesheets) haven't been tampered with by checking a cryptographic hash.

STRIDE: A threat modeling framework categorizing threats as: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

Supply Chain Attack: Compromising a target indirectly by attacking a trusted third-party component they depend on (e.g., a compromised npm package).

TOTP (Time-Based One-Time Password): A temporary code generated by an authenticator app (e.g., Google Authenticator) used as a second factor during login.

Tor: A free software enabling anonymous internet communication by routing traffic through multiple encrypted relays worldwide.

Two-Factor Authentication (2FA): Requiring two separate forms of verification to log in, typically a password plus a code from a device or app.

VPC (Virtual Private Cloud): An isolated section of cloud infrastructure where resources run in a logically separated network, limiting exposure.

WAF (Web Application Firewall): A security layer that filters and monitors HTTP traffic to a web application, blocking common attack patterns like SQL injection and XSS.

XSS (Cross-Site Scripting): An attack where malicious scripts are injected into a trusted website, executing in other users' browsers to steal data or hijack sessions.

Zero-Day Exploit: An attack exploiting a vulnerability unknown to the software vendor, meaning no patch exists yet.

Appendix B: Expanded References

Security Standards and Frameworks for Shira 2.0's Security Audit

1. **OWASP 2025** - Standard awareness document for critical web application security risks
   https://owasp.org/Top10/2025/0x00_2025-Introduction/
2. **OWASP Top 10 Proactive Controls** - Security techniques for developers
   https://top10proactive.owasp.org/
3. **OWASP Application Security Verification Standard (ASVS)** - Comprehensive security requirements
   https://owasp.org/www-project-application-security-verification-standard/
4. **NIST Cybersecurity Framework 2.0(2026)** - Risk management guidelines with new Govern function
   https://www.nist.gov/cyberframework
5. **NIST Privacy Framework 1.1** - Privacy risk management aligned with CSF 2.0
   https://www.nist.gov/privacy-framework
6. **ISO/IEC 27001:2022** - Information security management systems standard
   https://www.iso.org/standard/27001

Human Rights Defender Security Resources

7. **Security in-a-Box** - Digital security guide for activists and HRDs
   https://securityinabox.org/
8. **Digital First Aid Kit** - Support for facing common digital threats
   https://www.digitaldefenders.org/digitalfirstaid/
9. **Surveillance Self-Defense (EFF)** - Guide to protecting against surveillance
   https://ssd.eff.org/
10. **Amnesty Security Lab Digital Security Hub** - Resources for civil society
    https://securitylab.amnesty.org/digital-resources/
11. **Front Line Defenders Digital Protection Resources**
    https://www.frontlinedefenders.org/en/digital-security-resources
12. **ProtectDefenders.eu Resources** - Compilation of HRD security materials
    https://protectdefenders.eu/news-and-resources/

Technical Security References

13. **OWASP Dependency Check** - Software supply chain security tool
    https://owasp.org/www-project-dependency-check/
14. **OWASP SAMM** - Software Assurance Maturity Model
    https://owaspsamm.org/
15. **CIS Controls** - Prioritized cybersecurity actions
    https://www.cisecurity.org/controls

Research Papers and Reports

16. **Digital Security for Human Rights Defenders** - Human Rights Research Organization
    https://www.humanrightsresearch.org/post/digital-security-for-human-rights-defenders
17. **Access Now Digital Security Helpline Report** - Analysis of 10,000+ cases
    https://www.accessnow.org/helpline
18. **Pegasus Project Investigation** - Surveillance of activists and journalists
    https://forbiddenstories.org/case/the-pegasus-project/

Anti-Phishing Specific Resources

19. **APWG Phishing Activity Trends Reports** - Industry phishing statistics
    https://apwg.org/trendsreports/
20. **Google Safe Browsing** - Phishing and malware detection
    https://safebrowsing.google.com/

Disclaimer: This report represents a point-in-time assessment of Shira 2.0.0's security posture. Security is an ongoing process, and new vulnerabilities may emerge after this assessment.

Prepared by: *Convocation Research+Design*
Prepared For: *Horizontal*

- End of Security Audit Report -