

CSP Continuity — Security Posture Report

The product's current security posture, and the enterprise-grade checks it has passed · v0.4.0-preview.29

What this document is. Before offering CSP Continuity to enterprise and reseller channels, we put the shipping product through the same checks an enterprise security team runs on a vendor — full source review, secrets scanning across the entire repository history, dependency and static analysis, infrastructure-as-code scanning, and a read-only configuration audit of a real production marketplace installation. This report states the product's current security posture, and for every claim, how it was verified. The verification was publisher-conducted using industry-standard tooling and an adversarial methodology; it is not a third-party penetration test, and we do not represent it as one.

The trust model does most of the work: **CSP Continuity deploys entirely inside your Azure tenant and your cost data never leaves it** — there is no vendor-side service, storage, or access to evaluate.

Security posture — verified, as shipped

Every row below describes the product as it ships today, paired with how the claim was proven.

CONTROL	HOW IT WAS VERIFIED
✓ Your data stays in your tenant. The only recurring egress is the Azure Marketplace billing meter — a plan tier and a quantity. No cost figures, no identifiers, no telemetry to the publisher.	Every outbound call site in the source was read and catalogued; the full endpoint list is published on the security page.
✓ Sign-in is restricted to assigned users. Authentication is your Microsoft Entra tenant with "assignment required" enforced — only users and groups your admins assign can sign in. The installing administrator is assigned automatically; an in-app Access panel shows who has access.	Enforcement is Microsoft's, at the Entra authorize endpoint (unassigned users are rejected before reaching the app); configuration verified in the shipping onboarding automation.
✓ Every endpoint fails closed. The deployment template ships authentication in a reject-everything state, and application code independently re-verifies the authenticated principal on every route outside the four deliberately-public paths — defense in depth against platform misconfiguration.	Verified live: a fresh deployment returns 401 on every data endpoint before onboarding; a regression test enumerates the full route table so no future endpoint ships unguarded.
✓ No SQL injection surface. Database access is parameterized throughout; dynamic identifiers come only from fixed allowlists.	Adversarial grep + 468-rule static analysis + manual review of every query path: zero injectable paths found.
✓ Transport and at-rest encryption, all-green. HTTPS-only; TLS 1.2 minimum on app, SCM, SQL, and storage; SQL connections use <code>Encrypt=yes</code> with certificate validation; TDE on the database; SQL auditing on (90 days); FTPS and basic-auth publishing disabled.	Verified on a live production marketplace installation, not just in the template — with zero drift between the certified template and the live configuration.

✓	Supply-chain integrity is enforced, not advisory. The certified template pins the onboarding script's SHA-256; the script pins the application package's SHA-256 and refuses a mismatch; the running app re-verifies the same pin before any self-update. The verification command sits in the customer's copy-paste path on every onboarding surface, fail-stop on mismatch.	Chain verified end-to-end against the live published artifacts at release time, every release.
✓	Dependencies pinned and current. All bundled packages are version-pinned and carry zero known vulnerabilities; a CycloneDX SBOM is available on request.	pip-audit clean against the shipping pin set; the full 1,162-test suite passes against exactly those versions.
✓	Secretless sign-in; clean secret hygiene. Authentication uses a managed-identity federated credential — no stored client secret to leak, rotate, or expire. Every deployment's credentials are generated inside the customer tenant; no two installations share any secret; no secrets appear in logs or error responses.	Secret scanning is enforced on the source repository (two scanners, full-history coverage); no credentials in current sources; all logging call sites audited for leaks.
✓	Outbound destinations are allowlisted. The application's credentials are only ever presented to Microsoft endpoints; customer-configurable export destinations are validated against Azure Storage domains (public, Government, and China clouds) at every entry point and again at use.	Verified in source and exercised by dedicated tests, including hostname-spoofing cases.
✓	Hardened web surface. No CORS exposure, consistent output encoding on every rendering path, generic error responses with correlation IDs — diagnostic detail stays in the customer's own logs.	45+ rendering sites audited for XSS: none found. Error and logging paths reviewed for information leaks: none found.

The checks we ran

LAYER	WHAT WAS DONE	TOOLING / EVIDENCE
Architecture & permissions	Complete component map, data-flow diagram, identity-and-permission matrix with least-privilege analysis of every role, scope, and grant	Manual, evidence-cited review of ARM templates, application code, and onboarding automation
Secrets	Full git history scanned (356 commits), twice — with and without the repo's own allowlist — plus live verification of every candidate credential against Azure	gitleaks 8.30.1, trufflehog 3.97.9
Dependencies	Known-vulnerability audit of all pinned packages; CycloneDX SBOM generated	pip-audit 2.10.1, syft 1.54.0
Static analysis	468 rules across default + security-audit rulesets over all tracked source	semgrep 1.176.0
Infrastructure as code	Marketplace ARM templates scanned (25 resources); findings triaged against the managed-application deny-assignment model	checkov 3.3.22

Manual code review	Route-by-route authentication audit of every HTTP endpoint; SQL-injection audit of every query path; XSS audit of every HTML surface; SSRF and outbound-destination review; error-handling and logging leak review	Adversarial manual review, every finding verified against code before recording
Live deployment	Read-only configuration audit of a production marketplace installation (deny assignment active): TLS, auth config, firewall rules, RBAC, Defender posture, anonymous-endpoint probes	Azure CLI (read-only) against a real install

Hardening backlog — published

In priority order: automated offboarding (uninstall cleanup of the sign-in app registration and role assignments) with documentation; security gates (dependency audit, secret scan, template lint) as mandatory release blockers in the publisher build pipeline; a least-privilege runtime database login to replace the current admin-credential pattern; lifecycle-webhook rate limiting; narrowed Microsoft Graph consent; and additional HTTP security headers. None of these items exposes customer data across the tenant boundary; each is tracked with a concrete remediation.

Design decisions we want reviewers to see

Azure's managed-application **deny assignment** — the platform control that prevents anyone, including the publisher, from tampering with the deployed resources — also blocks Azure RBAC data-plane access inside the managed resource group. Several deliberate trade-offs follow, documented here rather than left for your scanners to find:

- **Key-based auth inside the tenant** (storage, telemetry ingestion) instead of RBAC, with keys held as platform-encrypted application settings behind the deny assignment. We chose an intact deny assignment over a weakened one. The keys never leave the tenant.
- **No Key Vault** — its data plane is blocked by the same deny assignment.
- **The SQL endpoint is network-reachable** (TLS-only, firewall-ruled, fully audited) because direct Power BI connectivity to the customer's own data is a product feature. The in-app firewall helper opens single IP addresses only.
- **A long-lived read-only access signature** keeps the application package mounted from the customer's own storage so operation never depends on publisher uptime — the signature is read-only and scoped to the already-public application package.
- Expect Microsoft Defender for Cloud to flag the SQL and storage items above on any install; each is a documented trade-off of this deployment model, not an oversight.