Dainvo Desktop
Enterprise Security Whitepaper

How Dainvo protects planning data on the desktop,
in the Enterprise portal, and across platform services.

APPLIES TO	Dainvo Desktop v0.7.5 (Beta) — Windows, macOS, Linux
	Dainvo Enterprise portal (users.dainvo.com) and platform services
REVISION	1.3 — July 2026
AUDIENCE	Enterprise security, IT, and compliance teams

Contents

About this document

This whitepaper describes the security architecture of the Dainvo desktop application, the Dainvo Enterprise portal, and the platform services behind them. Statements reflect the implementation as of July 2026. Where behavior varies by platform, is best effort, or is not yet implemented, the document says so plainly rather than glossing over it. Phrases such as "never contains" describe invariants that are enforced in application code and tests, not aspirations.

Release note: the enterprise identity suite (SSO, SCIM, organization audit, SIEM streaming) is delivered by the portal and platform. Managed desktop policy for the identity era and the ChatGPT/Claude connectors activate with the first Enterprise-capable desktop release (planned v0.7.6); v0.7.5 desktops remain personal clients.

1. Executive summary

Dainvo is a local-first desktop planner (calendar, tasks, time blocking, daily notes, email follow-up, online meetings) built on Electron for Windows, macOS, and Linux, with an Enterprise portal for organizations. Its security posture rests on six pillars.

Local data, encrypted for everyone. The local SQLite store is encrypted by default for every user — not as a paid or enterprise add-on — using database file encryption compatible with SQLCipher. The key is generated per app profile and wrapped by the operating system's secure storage.

A hard process boundary. Provider SDKs, OAuth tokens, database access, sync, and filesystem access all live in the Electron main process. The renderer (the UI) runs with context isolation, no Node integration, and the Chromium sandbox enabled. It receives only typed, sanitized data objects over a narrow preload API, and it cannot open new windows or navigate away from the app.

Management you can verify. Enterprise policy is signed with Ed25519 and verified on the device against a trust root **pinned inside the application binary**. A compromised distribution channel or portal database cannot inject policy. Policy that is invalid, downgraded, or missing a required signature is rejected, and the app returns to its unmanaged secure defaults.

Enterprise identity, standards first. Organizations bring their own identity provider through SAML 2.0 SSO with DNS domain ownership verification, tested providers, and optional strict enforcement; SCIM 2.0 provisioning makes the IdP directory the source of truth, and deprovisioning takes effect on the next request because every online enterprise call validates the live session.

Failure means stop, not degrade. When a security dependency is unavailable — secure key storage, an encryption migration, a signature check — Dainvo fails closed rather than continuing in a weaker state, and managed devices are held to the strictest behavior.

Evidence without exposure. Managed desktops produce append-only audit logs and sanitized compliance reports; organizations get an audit trail protected by a hash chain, exports with signed manifests, and delivery of security events to their own SIEM — all redacted by design: no tokens, no key material, no filesystem paths, no user content.

Dainvo AI, the app's natural language assistant, follows the same philosophy: inference runs on the device by default, AI data access is off until a user grants it for each account and scope, every write the AI drafts requires user confirmation, and AI activity has its own audit category. External ChatGPT and Claude connections are off by default, relay through encrypted envelopes to the user's own desktop, and require an explicit policy opt-in in organizations.

2. Security posture at a glance

Control area	Implementation
Data at rest	Encrypted SQLite (SQLCipher compatible) for **all** users; random key per profile, wrapped by the OS credential store (Keychain / DPAPI / Secret Service)
Secrets	TokenVault in the main process; blobs encrypted with `safeStorage`; fails closed if encryption is unavailable
Renderer isolation	`contextIsolation: true`, `nodeIntegration: false`, `sandbox: true`; typed preload APIs only
Navigation control	All popups denied; navigation away from the app blocked; vetted external links open in the OS browser
Enterprise identity	SAML 2.0 SSO per organization with DNS domain verification, tested providers, and optional strict enforcement; SCIM 2.0 provisioning with instant deprovisioning; TOTP MFA, with sensitive admin actions gated behind a strong session (AAL2)
Policy integrity	Detached Ed25519 signature over a normalized SHA-256 policy hash, verified against a public key pinned in the binary; rollback rejection
Policy failure mode	Fails closed to unmanaged secure defaults; unknown keys invalidate the policy
Update control	Policy can disable updates, force the stable channel, block beta, and turn off automatic download and install on quit
Endpoint audit	Append-only local `audit_events` table (8 categories, including `ai`); rotating sanitized JSONL export for collection
Organization audit and SIEM	Audit trail protected by a hash chain per organization; portal and email alerts; NDJSON/CSV exports with manifests signed with Ed25519; security event delivery to customer SIEM webhooks signed with HMAC
Compliance	Local JSON compliance reports; optional sanitized evidence upload; portal device inventory with drift flags
AI	Inference on the device; model downloads verified against pinned SHA-256 hashes; access off by default with 10 granular scopes; every write requires confirmation; policy kill switches
External AI connectors	ChatGPT and Claude via OAuth 2.1 with PKCE; encrypted request envelopes relayed to the user's single active desktop; off by default; organizations opt in per connector through signed policy
Telemetry	Sentry crash reporting with PII off and sanitization of every event; policy off switch
Payments	Handoff to Stripe Checkout or Apple StoreKit; no card data in the app or backend code
Server side	Supabase with Row Level Security on user data; no signing keys in the portal database; no service role credentials on devices

3. Threat model

The design addresses the following threats. Each maps to controls detailed in later sections.

T1 — Lost or stolen endpoint. Calendar and task data sits inside an encrypted SQLite file whose key is wrapped by the OS credential store, and tokens are encrypted separately in the TokenVault (§5, §6). Daily note Markdown files and exports live outside the database by design, which is why policy can require and attest full disk encryption (§9). Residual risk: an attacker who can unlock the OS user session has what that user has — disk encryption and OS session controls remain your compensating layer (§16).

T2 — Malicious or compromised web content in the UI. The renderer is sandboxed, isolated, and has no Node access. Even if compromised, it cannot reach tokens, the database, the filesystem, or provider clients — those exist only behind validated, typed IPC in the main process. Window opens are denied, navigation is pinned to the app, and external links open in the system browser (§4).

T3 — A compromised portal or distribution channel injecting policy. Policy is accepted only with a valid detached Ed25519 signature verified against a trust root **pinned in the application binary**; the signer list delivered with the response is ignored. The portal database holds no signing private key, so compromising it does not confer signing ability (§8.2, §15).

T4 — Policy rollback or replay. A bundle for the same organization whose version `createdAt` precedes the applied one is refused (§8.2).

T5 — Local privilege abuse against MDM policy. Packaged builds ignore policy path environment overrides, so a user without administrator rights cannot redirect the trust root. On Windows, the provided installer creates `%ProgramData%\Dainvo` with write access restricted to SYSTEM and Administrators before a standard user can claim it (§8.3).

T6 — Network attackers and malicious sync endpoints. Production CalDAV sync requires HTTPS with limited, revalidated redirects; localhost, private, link-local, multicast, and cloud metadata addresses are blocked, closing off SSRF paths. Provider traffic uses the providers' TLS endpoints, and token exchange for providers that require a confidential secret is brokered on the server (§6.2, §14).

T7 — Credential theft from the app's own storage. Tokens exist only as blobs encrypted by the operating system; there is no plaintext fallback in production. On an unmanaged Linux machine without a keyring the app degrades explicitly — visible notice, posture reporting, automatic rewrap once a keyring appears — while managed devices fail closed instead (§5.3, §6.1).

T8 — Prompt injection and AI overreach. The model, local or remote, is treated as untrusted. It can only call a typed tool broker that enforces policy in application code: reads are filtered by the scopes each account grants (default: none), writes are held as pending actions until the user confirms them in the app, policy is checked again at confirmation time, and tool errors are sanitized so internal details cannot leak through chat output. Enterprise policy can remove remote providers and model downloads entirely (§11).

T9 — Sensitive data leaking through logs, errors, telemetry, or evidence. Fields resembling tokens are redacted from logs and errors; Sentry runs with PII off and a sanitizer on every event and breadcrumb; audit metadata, compliance reports, and uploaded evidence are restricted to safe status codes and labels, excluding tokens, paths, key material, and user content. The evidence collector can fail its run if sensitive patterns survive sanitization (§10, §12).

T10 — Abuse of the external AI connector path. Only OAuth clients whose registered name and every callback URL match the official ChatGPT or Claude client are accepted; consent issues identity scopes only, and tokens are isolated to the MCP audience by a custom access token hook. Requests and results exist only as AES-256-GCM ciphertext in envelopes with a deadline of 30 seconds; the cloud endpoint holds no database access; each connector is off by default, bound to one active desktop, and enterprise use requires an explicit opt-in in signed policy. Disabling a connector fails calls immediately and disconnecting revokes the OAuth grant (§11.3).

T11 — Identity and provisioning abuse. SAML domains route sign-in only after DNS TXT ownership verification, and verification is rechecked daily. A new SAML provider cannot enforce anything until an admin completes a real test sign-in through it, and enforcement can be enabled only from that linked, tested session at AAL2. SCIM bearer tokens are shown once, stored only as a keyed hash, rotate atomically with a retiring overlap of 24 hours, and can be revoked independently. Deprovisioning takes effect on the next request because every online enterprise call validates the live session (§7.2, §7.3).

Out of scope for the app's controls: a fully compromised operating system or user session, malicious software running at the OS level, and the internal security of the third-party services a user connects.

4. Application architecture and hardening

4.1 Process model

Dainvo uses Electron with a strict separation between main, preload, and renderer code:

- The **main process** owns SQLite, provider SDKs, OAuth and token storage, sync engines, enterprise policy, audit logging, and all filesystem access.
- The **preload layer** exposes narrow, typed APIs (for example `calendarApi`, `taskApi`, `settingsApi`, `notesApi`) — never raw `ipcRenderer`, Node primitives, filesystem handles, database handles, or provider clients.
- The **renderer** owns UI state only. It calls main process capabilities exclusively through typed IPC and receives sanitized data objects.

4.2 Renderer hardening

Application windows are created with:

- `contextIsolation: true`
- `nodeIntegration: false`
- `sandbox: true`

Navigation is locked down in the main process. The window open handler **denies every popup request**; URLs that pass the app's vetting are handed to the operating system browser instead. Navigation events away from the app are blocked the same way, and OAuth consent screens run in dedicated authentication windows with their own navigation interception.

All renderer inputs are validated in the main process. Errors shown to users are safe and concise; stack traces are not shown in production, and fields resembling tokens are redacted from logs and error surfaces.

4.3 Local-first data flow

The local SQLite database is the source of truth for UI rendering. Provider sync reconciles remote calendar and task state with the local store in the background, from the main process and a trusted sync worker only. The app remains functional offline, and a cached license permits offline startup for signed-in users.

5. Data inventory and data at rest

5.1 Where data lives

Data	Location	Protection	Leaves the device?
Calendar events, tasks, buckets, sync metadata, preferences	Local SQLite under the app profile	Encrypted database file (all users); key wrapped by OS secure storage	Only to the provider that owns each item, through sync the user configured
OAuth tokens, CalDAV/iCloud credentials, provider API keys	`account_tokens` table via TokenVault	Blobs encrypted with `safeStorage`; retrieval in the main process only	Never (upstream revocation calls excepted)
Database encryption key	App profile key record	Blob encrypted by the OS; the raw key is never persisted	Never
Daily notes	A folder the user chooses; one Markdown file per day	Plain files owned by the user (by design); covered by OS disk encryption	Never (the user's own files)
Bucket file links	Local path references in the database	Reference only — files are never copied, uploaded, read, or moved	Never
Local AI models and inference	Model storage managed by the app; runtime on the device	Artifacts verified against pinned SHA-256 hashes; prompts and results stay on the device	Model download only (inbound)
External connector requests	Encrypted envelopes in the relay queue (server side)	AES-256-GCM ciphertext only; deadline of 30 seconds; wiped on expiry, deleted within about an hour of completion	Relayed between the AI client and the user's desktop; never stored in plaintext
Audit events	`audit_events` table plus optional JSONL export	Inside the encrypted database; JSONL is sanitized by construction	Only through admin collection or evidence summaries enabled by policy
Compliance reports	`<userData>/enterprise/compliance/`	Safe status content only	Optional sanitized summary upload, enabled by policy

Data	Location	Protection	Leaves the device?
Dainvo account, license, device, and organization records	Supabase backend	Row Level Security; see §15	Server data by nature

5.2 Database encryption (all users)

The local SQLite cache is encrypted for **all users** with database file encryption compatible with SQLCipher. Dainvo generates one random database key per app profile and stores only a key blob encrypted by Electron `safeStorage` under the app's user data directory. Raw key material is never written to disk and never crosses the preload or renderer boundary. The database is opened only from the main process or its trusted sync worker, an Electron utility process that receives open metadata over process messaging — never through APIs the renderer can reach.

Migration. On first launch after the encryption rollout, an existing plaintext database is WAL checkpointed, plaintext `-wal`/`-shm` sidecars are removed, and the database is copied into an encrypted backup before the live database is rekeyed into encrypted storage. If key storage, backup encryption, migration, or validation of the encrypted reopen fails, **startup fails closed** — Dainvo does not silently continue on plaintext.

Key rotation. An internal admin command in the main process rekeys the live database (`hexrekey`), gated by the exact confirmation phrase `ROTATE_DATABASE_ENCRYPTION_KEY`. It disposes the sync worker first, updates the encrypted key record only after the rekey succeeds, and audits success, failure, or denial with a correlation ID. Enterprise policy can disable manual rotation, and the denial is itself audited.

Recovery. Startup failures surface safe reason codes (`missing_key`, `key_record_invalid`, `key_decrypt_failed`, `wrong_key_or_corrupt_database`, `key_storage_unavailable`, `backup_failed`, `migration_failed`) with guidance codes — never raw crypto errors or paths. Recovery uses the timestamped encrypted backup taken before migration, together with the same profile key record.

Known limits, stated plainly. SQLite databases held in memory and SQLite temporary tables sit outside database file encryption; encrypted connections use `temp_store = MEMORY` where practical to avoid temp table spill. Encryption status visible to the renderer is safe metadata only.

5.3 Key storage backends per OS

Database keys and token vault entries are wrapped by Electron `safeStorage`, which uses the platform credential store: macOS Keychain, Windows DPAPI, and the Linux Secret Service (GNOME Keyring or KWallet).

Linux deserves specific attention:

- With no reachable keyring on an **unmanaged** system, Dainvo enables Chromium's reduced security `basic_text` backend so the key stays wrapped in a degraded payload envelope. The app shows a reduced security notice on the sign-in screen, reports `electron-safe-storage-basic-text` in posture and compliance reports, and **automatically rewraps the key with the real keyring once one becomes available**.

- **Managed devices fail closed** in that state (recovery reason `key_storage_unavailable`, guidance `repair_key_storage`) — deploy GNOME Keyring or KWallet before the policy. The strictly confined Snap declares the `password-manager-service` interface; connect it (`sudo snap connect dainvo:password-manager-service`) and restart to restore keyring storage.
- No plaintext development fallback is configured on any platform.

6. Credentials and provider access

6.1 Token vault

OAuth access and refresh tokens, along with stored provider secrets, are encrypted through the **TokenVault** in the main process using `safeStorage`, and persisted only as encrypted blobs in the local `account_tokens` table. Token retrieval happens in the main process only; renderer APIs may expose connection status (such as `hasTokens`) but never token payloads or encrypted blobs. If encryption is unavailable in production, the vault **fails closed** rather than storing plaintext, with the Linux behavior described in §5.3. Provider errors are sanitized so tokens, authorization headers, auth codes, and raw credential payloads cannot leak into logs or the UI.

6.2 OAuth model

- Installed desktop apps are public OAuth clients. Public desktop client IDs and tenant IDs are bundled so users can sign in without developer configuration. Packaged provider app credentials are build and runtime configuration, never treated as user secrets, and never exposed to the renderer, logs, or errors.
- Provider authorization flows use **PKCE** (S256 code challenges) where the provider supports it.
- For providers whose token exchange requires a confidential client secret (Todoist, TickTick), the exchange is brokered by an **authenticated Supabase Edge Function** (`provider-oauth-token`). The desktop must present a valid Dainvo session; the function performs the exchange, returns tokens to the device, and **stores nothing**. All task data syncing stays direct from client to provider, and the secret never ships in the binary.
- Provider scopes follow least privilege. Provider SDK and network calls run only in the main process.
- Generic CalDAV and Apple iCloud credentials are supplied by the user (app passwords are recommended and supported). They are stored only through the encrypted TokenVault and are never returned to the renderer after submission.
- **CalDAV URL hardening:** production sync requires HTTPS; redirects are limited and revalidated; localhost, private, link-local, multicast, and cloud metadata addresses are blocked. Raw ICS bodies are stored only in the encrypted local database and are not logged or exposed to renderer APIs.

6.3 Disconnect and revocation

Dainvo provides two administrative disconnect operations.

`revokeAndDisconnect` attempts upstream OAuth revocation where the provider supports it for desktop flows, then deletes local tokens. Verified behavior: Google, Todoist (with client credentials), and Zoom revoke upstream; Microsoft, Webex, and TickTick record the revocation as unsupported or delegated to admins for this flow and still delete local tokens; CalDAV and iCloud have no OAuth grant, so local credentials are deleted. **A failed revocation never leaves local tokens behind** — local disconnect always completes, and the outcome (`revocation_failed_local_disconnect_completed`) is recorded.

`removeLocalData` additionally purges the provider's local cache (synced task rows, online meeting records) before deleting the account row.

7. Identity, licensing, and billing

7.1 Accounts and authentication

- Dainvo accounts use **Supabase Auth** with email confirmation. Sign-in options are email and password plus Google, Microsoft, Apple, and GitHub. On the Mac App Store build, Sign in with Apple completes natively in the app, and the other identity providers use Apple's secure in-app authentication session.
- **Multi-factor authentication** with authenticator apps (TOTP) is available to every account, with support for multiple factors. Supabase Auth does not provide recovery codes, so Dainvo supports enrolling several TOTP factors and offers an audited staff reset that also revokes sessions.
- New passwords are checked against known breach corpuses (Have I Been Pwned) at the platform level.
- Sensitive administrative actions — such as enabling SSO enforcement or staff recovery operations — require a strong, recently verified session (AAL2).

7.2 Enterprise SSO (SAML 2.0)

Organizations can require their own identity provider for company access.

- **Per-organization SAML 2.0 providers** are configured in the Enterprise portal and registered with Supabase Auth through the `enterprise-sso` Edge Function, which uses a narrowly scoped Management API token. Okta, Microsoft Entra, Google Workspace, and any standard SAML 2.0 IdP are supported through a metadata URL or XML.
- Dainvo stores provider IDs, IdP entity IDs, domains, metadata XML hashes, and certificate fingerprints with validity dates. Raw metadata XML is submitted to Supabase Auth but not stored by Dainvo.
- **Domain ownership is proven before routing.** A work domain is registered only after DNS TXT verification; the raw challenge is displayed once and only its SHA-256 hash is stored. Challenges expire after seven days. Verified records are rechecked daily; three consecutive misses start a grace period of 30 days before routing is disabled. Certificate metadata is also checked daily, with alerts 60, 30, 14, 7, and 1 days before the final usable signing certificate expires.
- **Test before trust.** New providers stay in `testing` until an owner or admin completes the explicit SAML test and link flow. Enforcement can be enabled only from that linked, tested SAML session at AAL2.

- An organization may run **multiple tested providers**; every mapped provider is accepted for enforced access, while exactly one provider is the primary domain route for SP-initiated login from the Dainvo sign-in page.
- Users who authenticate through a configured provider and domain are provisioned just in time as `member`. Owner and admin roles remain managed inside Dainvo; mapping IdP groups to roles is deliberately not enabled.
- **Enforcement semantics.** An enforced organization rejects password and social sessions, and SAML sessions from unlinked providers, for enterprise access — personal Dainvo features remain available to the user. Every online enterprise request validates the JWT `session_id` against live Supabase sessions, so a SCIM deprovision or staff session reset takes effect immediately.
- **No password bypass exists for customers.** A staff administrator at AAL2 may suspend enforcement for at most 30 minutes with a support reason; the action alerts organization owners and admins and re-enforces automatically when it expires.

7.3 SCIM 2.0 provisioning

When an organization connects SCIM, the identity provider's directory becomes the source of truth for membership.

- The `enterprise-scim` Edge Function implements SCIM 2.0: discovery (`ServiceProviderConfig`, `ResourceTypes`, `Schemas`), Users and Groups with GET/POST/PUT/PATCH/DELETE, and standard filters (`userName`, `externalId`, `id`, and `displayName` for groups).
- **Bearer token hygiene.** The per-organization SCIM token is generated in the portal, displayed once, and stored only as an HMAC hash computed with a server-side pepper. Token IDs and hashes are unreadable from the browser through column grants. Rotation is atomic: a new active token is created and the previous one moves to `retiring` for 24 hours, in one database transaction; either token can be revoked independently, and a failed rotation leaves the prior state intact.
- **Lifecycle.** SCIM-created users become `member` and can be provisioned before their first SAML sign-in. Setting `active=false` or deleting a user disables enterprise membership and attempts session revocation for the linked account; the underlying Supabase Auth user is not deleted. When SCIM is active, SAML just-in-time provisioning requires an active matching SCIM user — domain matching alone cannot re-add someone enterprise IT removed.
- Email changes keep the existing membership link, so a previous address cannot retain stale access. Group sync is stored for visibility and audit only; groups do not assign Dainvo roles in this version.
- Not supported in this version: ETags, SCIM bulk operations, mapping groups to roles, and OAuth client credentials for SCIM.

7.4 Licensing, billing, and devices

- **Payments never touch Dainvo.** Direct builds hand off to Stripe Checkout in the browser; Mac App Store builds use Apple StoreKit, verified on the server through `verify-app-store-purchase` and App Store server notifications. Dainvo holds no card data.

- Plan access and subscriptions are managed at `users.dainvo.com`; the desktop reads license state (with an offline cache) and enforces plan features locally.
- **Device management:** signed-in devices are registered per account with device limits per plan. Users can review active devices and remotely sign out a device from Settings or the portal. This personal revoke flow exists on every plan and is separate from enterprise inventory (§10.3).
- **Account deletion** removes the Dainvo login, profile, app access, and registered devices through a dedicated backend function.
- The desktop app never holds Supabase service role credentials, and the portal and Edge Functions never hold policy signing private keys (§8.2, §15).

8. Enterprise managed policy

Dainvo ships **one application build**. Management activates only for devices that receive policy through one of two channels; everyone else keeps the standard experience. Policy delivery is additionally gated by platform client requirements: organizations set a minimum Enterprise-capable app version, and desktops below it receive `not_configured` and stay unmanaged.

8.1 Account-delivered policy (primary)

1. An org admin publishes policy in the `users.dainvo.com` portal.
2. On sign-in, and periodically afterward, the Electron main process calls the authenticated `enterprise-account-policy` Supabase Edge Function. The function returns the latest signed policy **only** for an active organization membership with active Enterprise entitlement, and **only** to a device the account has registered — an unregistered device receives `device_unregistered` and stays unmanaged.
3. The response contains only the policy JSON, a detached signature, public signer configuration, and safe metadata (org ID and name, revision, hash).
4. The desktop verifies the signature locally **before** applying anything.

8.2 Signature verification and trust root

- Signatures are **detached Ed25519** over a deterministic `sha256:` hash of the parsed, normalized policy (keys sorted by UTF-16 code unit, so JSON whitespace, key order, and machine locale cannot change the fingerprint).
- Policy delivered through the account channel is verified **only against a public key pinned in the application binary** (key ID `dainvo-enterprise-root-2026`). The signer list in the server response is ignored. Signatures stay meaningful even if the portal database is compromised, because the portal holds no signing private key to steal.
- **The app fails closed when:** a required signature is missing, a signature is malformed, the signature covers a different policy hash, the signature is cryptographically invalid, the key ID is untrusted, or the signer configuration is malformed. Unknown policy keys or malformed values invalidate the entire policy file, and Dainvo falls back to unmanaged secure defaults.
- **Rollback protection:** a bundle for the same organization whose version `createdAt` precedes the currently applied one is refused.
- The policy cache is stored under the app profile with file permissions restricted to the owner where the OS honors them, so cached policy applies at the next startup before the network refresh completes. The cache is cleared on sign-out or when an admin removes device management.
- Reports and audit events expose only signature status, the requirement flag, signer key ID, algorithm, policy hash, and verification timestamp — never key PEM, signature bytes, file paths, or crypto errors.

8.3 Local MDM policy (advanced fallback)

For MDM or IT deployment without account delivery, Dainvo reads a local managed policy at startup:

- macOS: `/Library/Application Support/Dainvo/enterprise-policy.json`
- Windows: `%ProgramData%\Dainvo\enterprise-policy.json`
- Linux: `/etc/dainvo/enterprise-policy.json`

Signed bundles (`enterprise-policy.json` plus `enterprise-policy.sig` and `enterprise-policy-signers.json`) use the same verification semantics. Starter installer scripts are provided for all three platforms, and CLI tooling (`enterprise:sign-policy`, `enterprise:verify-policy`) covers signing and validation before deployment.

Hardening notes for IT:

- Environment variable overrides for the policy path are honored **only in unpackaged development builds**. Packaged builds ignore them, so a user without administrator rights cannot redirect the policy trust root.
- On Windows, deploy with the provided installer script (or equivalent MDM configuration) so `%ProgramData%\Dainvo` is created with write access restricted to SYSTEM and Administrators before a standard user can claim the directory. A registry (HKLM) policy location is planned.
- Audit metadata for policy loads records the `policySource` (platform default, override, or account cache), so fleet attestation can distinguish channels.
- The renderer never receives the raw policy path, raw policy JSON, or parse errors — only effective preference and status metadata, such as which update fields are locked.

8.4 Precedence

1. Development environment override (unpackaged builds only)
2. Local platform managed policy (MDM/IT)
3. Cached policy from the signed-in account
4. Unmanaged defaults

Policy overrides effective runtime behavior but does not overwrite the user's stored preferences; removing management restores the user's own settings.

8.5 Management removal

Admins can remove management for an org device from the portal's Enterprise Devices page; the portal records a safe `remove_management` admin action. The desktop includes its account device ID and its organization device key during policy refresh; when the server reports removal, the desktop clears the policy cache and stops applying management — without signing the user out or touching the personal device record. Removal is checked against **both** identifiers, so it remains effective even after the local cache is already cleared.

9. Policy controls reference

```
{
  "version": 1,
  "policySignature": { "required": true },
  "telemetry": { "sentryEnabled": false },
  "ai": {
    "remoteProvidersEnabled": false,
    "localModelDownloadsEnabled": false,
    "externalMcpClients": {
      "chatgpt": false,
      "claude": false
    }
  },
  "providers": {
    "google": true, "outlook": true, "caldav": true, "icloud": true,
    "todoist": true, "ticktick": true, "zoom": true, "webex": true
  },
  "updates": {
    "enabled": true,
    "forceChannel": "stable",
    "allowBetaChannel": false,
    "autoDownload": false,
    "installOnQuit": false
  },
  "audit": {
    "enabled": true,
    "jsonlExportEnabled": true,
    "maxFileBytes": 5242880,
    "maxFiles": 5
  },
  "databaseEncryption": {
    "required": true,
    "allowStartupOnFailure": false,
    "manualRotationEnabled": true
  },
  "fullDiskEncryption": { "required": true },
  "complianceReport": {
    "localExportEnabled": true,
    "exportOnStartup": true,
    "maxFiles": 5,
    "remoteEvidenceUploadEnabled": true,
    "remoteEvidenceUploadIntervalMinutes": 360
  }
}
```

Control	Effect
`providers.*`	Disable any calendar, task, or meeting provider across the fleet
`updates.*`	Disable updates (status reports `enterprise-policy`), force the stable channel, block beta, and turn off automatic download and install on quit; locked controls appear in the app as managed by the organization
`ai.remoteProvidersEnabled`	Disable remote AI providers categorically

Control	Effect
`ai.localModelDownloadsEnabled`	Block local AI model downloads
`ai.externalMcpClients.chatgpt` / `.claude`	Independent opt-ins for the ChatGPT and Claude connections. Missing values mean `false`; each connector requires an explicit opt-in in signed policy even for users with paid access. Disabling a value blocks new OAuth consent and live relayed calls after policy refresh
`telemetry.sentryEnabled`	Prevent crash reporting from initializing
`audit.*`	Enable the durable audit log and rotating JSONL export, bounded in file size and count
`databaseEncryption.*`	**Tightening only:** `required: false` and `allowStartupOnFailure: true` are *invalid values* that void the policy — posture can be preserved or tightened, never weakened. `manualRotationEnabled: false` blocks rotation attempts and audits the denial
`fullDiskEncryption.required`	Requires disk encryption in posture reports — FileVault on macOS, BitLocker status for the system drive on Windows; Linux currently reports unknown. Attestation, not enforcement; it does not replace database encryption
`complianceReport.*`	Local JSON snapshot export, export at startup, retention (1–100 files), and sanitized evidence upload on an interval between 15 and 1440 minutes

Rollout caveat for `ai.externalMcpClients`: desktop versions released before external MCP support do not recognize this key. They treat the whole policy file as invalid and fall back to unmanaged defaults — disabling every locally enforced control on those desktops, not just the connectors (connector eligibility still fails closed on the server). Publish a policy containing this key only after every managed desktop runs a release that includes ChatGPT/Claude connection support.

Optional identity labels (`orgId`, `orgName`, `policyId`, `policyName`, `policyRevision`) are constrained to safe identifier characters and are reported together with the deterministic `sha256:` policy hash for fleet attestation.

10. Audit, compliance reporting, and device evidence

10.1 Local audit log and export

With `audit.enabled`, Dainvo writes append-only audit records to the local SQLite `audit_events` table across eight categories: `auth`, `provider`, `token`, `policy`, `settings`, `updates`, `ai`, and `database`. Outcomes are typed (`success`, `failure`, `denied`, `unsupported`, `revocation_failed_local_disconnect_completed`).

With `jsonlExportEnabled`, rotating JSONL files are exported under the app profile (`<userData>/audit/audit-events.jsonl`), bounded in size and count by policy, for collection by your tooling. Each line is one sanitized event: category, action, outcome, actor type, optional account and provider identifiers, timestamp, and redacted metadata. **Audit metadata never contains tokens, auth codes, raw provider payloads, stack traces, database paths, or the body content of events and tasks.**

Covered lifecycles include database encryption events (encrypted open, backup creation, migration from plaintext to encrypted storage, key rotation, recovery state) and policy and compliance events (policy load with source and identity hash, signature verification, report generation and export, evidence upload).

If SQLite itself cannot open at startup on a managed device, sanitized fallback JSONL is written and replayed into `audit_events` after the next successful startup — the audit trail survives database outages.

10.2 Compliance reports

`getEnterpriseSecurityReport()` and `getEnterpriseComplianceReport()` expose safe posture metadata only: database encryption state, disk encryption posture codes, recovery reason and guidance codes, audit and export state, managed policy identity, hash, and signature state, update management state, plus app version, build, platform, and distribution channel. They never include database paths, key IDs, key blobs, command output, disk labels, usernames, or raw crypto errors.

When enabled by policy, devices write a local JSON snapshot (`<userData>/enterprise/compliance/…`, with bounded retention, optionally at every startup). The preload and renderer receive only a stable relative location label, never the raw user data path.

10.3 Remote device evidence and portal inventory

With a **verified signed** org policy that enables it, the desktop uploads a sanitized compliance evidence summary to the `enterprise-device-evidence` Edge Function at startup and on the configured interval (15 to 1440 minutes). The upload is skipped when the policy is unsigned or unverified, missing its org identity or hash, disabled by policy, or the user is signed out.

Device identity is scoped to the organization by construction: the desktop computes `deviceKey = sha256(orgId + ":" + localAppDeviceId)` and never uploads the raw local device ID.

The backend stores only normalized inventory fields — org, uploading user, device key, optional account device ID, app, platform, and version, policy hash, encryption, disk encryption, and audit state, signature status, timestamps, compliance status, and warning codes. It stores no raw reports, tokens, paths, stack traces, provider payloads, or full machine identifiers. Evidence history is retained as the latest state plus up to 180 days.

The portal's **Enterprise Devices** page combines evidence with active signed-in desktop devices for active org members and flags `noncompliant`, `stale` (more than 48 hours without fresh evidence), `unmanaged` (an active device with no evidence), `wrong_policy_hash`, and `missing_audit_export`.

10.4 Endpoint evidence collector

An admin collector gathers compliance JSON and audit JSONL from an endpoint into a timestamped directory with a manifest of sources, outputs, redaction counts, and warnings (optionally zipped). It copies **only** sanitized compliance and audit artifacts — never the SQLite database, token vault records, key records, model files, or provider caches — and `--fail-on-sensitive` exits nonzero if known sensitive patterns remain after sanitization.

10.5 Portal administration and audit

Organization owners and admins manage members, roles, and status; view policy history; download signed policy bundles; and review a portal audit trail that records bundle creation with org identity, policy revision, and policy hash, plus device admin actions such as `remove_management`. Enterprise portal access is gated by entitlement: accounts without it receive `403` from the enterprise functions and are redirected in the UI — including accounts that were previously org admins but whose entitlement lapsed.

10.6 Organization security activity, signed exports, and SIEM delivery

Beyond endpoint logs, each organization has a security activity trail in the portal (Advanced security › Security activity), served by the `enterprise-audit` Edge Function.

- **Integrity protection.** Organization audit events form a chain of hashes per organization, so removed or altered records are detectable rather than silent.
- **Alerts.** Open security alerts (for example certificate expiry warnings, DNS verification failures, or staff enforcement suspensions) surface in the portal and can be emailed to admins. Email alerting is optional; portal alerts work without it.
- **Signed exports.** Admins export the trail as NDJSON or CSV. Each export ships with a manifest signed with a dedicated Ed25519 audit key: the manifest is canonicalized as RFC 8785 JSON and signed, and it carries a `contentSha256` for the data file plus the `signerKeyId`. Recipients verify the signature with the published public key and compare the data file hash before trusting the export. The audit signing key is separate from the policy signing key.
- **SIEM delivery.** Organizations register webhook destinations and Dainvo delivers security events signed with a shared HMAC secret. New destinations stay in `testing` until a signed test delivery receives a `2xx` response, and an edited endpoint keeps its existing URL until the candidate URL passes. Secret rotation signs with both the active and retiring secret for 24 hours; immediate revocation of the previous secret exists for suspected compromise. Delivery history is visible per attempt, failed deliveries land in a dead letter queue with manual retry, and test payloads contain no customer audit event data.
- **Scheduled operations.** Recurring checks (daily DNS reverification, certificate expiry alerts, delivery dispatch, retention maintenance) run through a scheduler that authenticates to the `enterprise-security-jobs` function with a dedicated job token — not a user session and not the service role key.

11. Dainvo AI: security and privacy model

Dainvo AI (Alpha) provides natural language commands, chat, and automation rules. Its design assumes the model is a **tool caller, not a trusted principal** — the same guardrails apply to local and remote providers because they live in application code, not in prompts.

11.1 Inference on the device

- The local provider runs Google's openly released **Gemma 4** models (E2B and E4B) on the **LiteRT-LM** runtime, entirely on the device. Local models interpret natural commands on the user's machine; prompts and results do not leave it.
- Models are downloaded on demand from Hugging Face (`litert-community`) and verified against **SHA-256 hashes pinned in the app's model catalog** before use; a failed digest check invalidates the artifact. An installed model file is not treated as ready until the runtime loads and warms it successfully.
- Platform gating: Local AI is not available on Intel Macs or on 32-bit and ARM Windows in this release.
- Enterprise policy can block local model downloads (`ai.localModelDownloadsEnabled: false`).

11.2 The Dainvo MCP tool surface

- Every provider — the local model, and remote adapters behind the same contract — interacts with Dainvo only through a small, typed MCP tool broker in the main process that enforces policy on every call.
- **AI data access is off by default.** Users grant access per connected account across ten scopes: calendar read and write, task read and write, bucket read and write, email read, document search, and automation rule read and write. Reads are filtered by policy; denied scopes return explicit `access_denied` results rather than fabricated empty answers.
- **Every AI write in the app requires confirmation.** Drafted changes are held as pending actions and execute only after the user confirms them in the app. Policy is **checked again at confirmation time**; pending actions expire and can be cancelled on the service side; confirmations are bound to their chat session, so one chat cannot confirm another chat's pending action.
- **Prompt injection containment:** because the model can only emit typed tool calls against this broker, injected instructions cannot reach tokens, raw files, the database, or the network. The worst an injected prompt can achieve is drafting an action that still needs approval, within scopes the user granted.
- **Error sanitization at the broker boundary:** raw validation, native, or path error text never reaches chat output; replies contain only copy the app authored. A second sanitization layer guards the remote provider reply paths as well.
- **AI actions are audited** under the dedicated `ai` audit category.
- In the current release, chat runs on the local provider and the remote provider connection surface in the app is paused. Enterprise policy can disable remote AI providers categorically (`ai.remoteProvidersEnabled: false`).

11.3 External AI connectors: ChatGPT and Claude (next desktop release)

Users can connect ChatGPT and Claude to Dainvo through a remote MCP bridge. The design goal is that the cloud carries messages, never data authority.

Authorization.

- ChatGPT and Claude are separate connections, and both start **off**. Personal access requires an active paid entitlement; enterprise access additionally requires the explicit policy opt-in described in §9.
- Clients authorize through the Supabase OAuth 2.1 server with PKCE and dynamic client registration, but registration is restricted: **only OAuth clients whose registered name and every callback URL match the official ChatGPT or Claude client are accepted.**
- Consent at `users.dainvo.com/oauth/consent` grants identity scopes only (`openid`, `email`, `profile`). Tool and data access is enforced separately by Dainvo policy — the OAuth grant conveys who the user is, not what the client may touch.
- Access tokens are isolated to the MCP audience by a custom access token hook and validated against asymmetric JWT signing keys (RS256 or ES256) via JWKS. The MCP endpoint's database role cannot log in and has **no table access**.

Request path.

- The `dainvo-mcp` function accepts a tool call, encrypts it into an envelope with AES-256-GCM (keys derived per user, client, and device from a server secret that is never exposed to clients or the renderer), and queues it. The user's desktop polls the `dainvo-mcp-relay` function (every 2 seconds while active, backing off to 10 seconds when idle; polling doubles as the liveness heartbeat), decrypts the request, executes it through the **same local MCP broker and account data selections described in §11.2**, and returns the encrypted result.
- The remote endpoint never opens the Dainvo database. Arguments and results exist server side only as ciphertext; envelopes carry a deadline of 30 seconds, expired envelopes are wiped immediately, finished envelopes are deleted within about an hour, and a repeated JSON-RPC request ID never replays a stored result.
- The relay tables have Row Level Security enabled with **no client policies** — only the service role and narrowly granted relay RPCs can touch them.

Control and revocation.

- One desktop is active per user and connector; moving the route to another desktop requires an explicit takeover from the new desktop. The desktop must be open and signed in for calls to succeed.
- Writes are approved in the host client: ChatGPT or Claude shows its own approval before calling a Dainvo action, and Dainvo enforces the account data selections on every call rather than adding a second prompt.
- Disabling a connector makes calls fail immediately; disconnecting revokes the OAuth grant. For organizations, setting the policy value to `false` blocks new consent and live relayed calls after policy refresh, and server-side eligibility fails closed regardless of desktop version.
- Connector audit rows intentionally exclude arguments, results, tokens, and error text.

12. Telemetry and crash reporting

Dainvo uses Sentry for crash and error reporting, initialized with `sendDefaultPii: false` and a sanitizer applied to **every event and breadcrumb** before send; anything resembling a token or a path is redacted at the source. Tags are limited to safe operational dimensions (process, distribution channel, key storage class). Enterprise policy (`telemetry.sentryEnabled: false`) prevents Sentry from initializing across the fleet.

Calendar weather, when a user selects a weather location, sends that location's latitude, longitude, time zone, and requested date range to Open-Meteo to retrieve forecasts. Coordinates are not logged beyond the typed weather API.

There is no content analytics: calendar bodies, tasks, notes, and email content are not telemetry data.

13. Updates and supply chain

- **Direct builds** (Windows installer, macOS DMG, Linux packages) update through the app's own updater against a dedicated public GitHub releases repository. Managed update policy is enforced in the main process: updates can be disabled (status reports `enterprise-policy`), pinned to stable, and stripped of beta, automatic download, and install on quit behavior.
- **Store builds** (Mac App Store, Microsoft Store MSI for x64, x86, and ARM64, Snap Store) are updated by their stores and pass each store's review and signing pipeline. Store builds surface their distribution identity in Settings and hide direct updater controls, with release guardrails that keep store packages from re-enabling the direct updater.
- macOS builds are code signed with the **hardened runtime**; Mac App Store builds are sandboxed with Apple entitlements and provisioning.
- Release automation validates artifacts before publication: configuration checks fail the build when required sign-in settings are missing from the final bundle; OAuth release configuration is pinned to CI variables so stale secrets or local files cannot override it; App Store Connect validation runs before Mac App Store artifacts publish; and native addon checks prevent packaging the AI runtime without its required LiteRT libraries.
- Renderer, main, and preload code is TypeScript with typed IPC contracts. Secrets are never committed; OAuth app credentials enter builds only through controlled release configuration (a local `.env`, Infisical export, or GitHub Actions secrets) — never through source. The confidential Todoist and TickTick exchange secrets stay on the server entirely (§6.2).

14. Network surface

Dainvo connects only to services the user (or policy) enables:

Destination	Purpose	Notes
Google APIs	Calendar, Tasks, and Meet sync	OAuth (PKCE where supported); scopes follow least privilege
Microsoft Graph	Outlook Calendar and Email, To Do, Teams	OAuth
Apple iCloud / CalDAV servers	Calendar sync	HTTPS enforced; URL validation blocks SSRF targets
Todoist, TickTick	Task sync	OAuth; token exchange brokered on the server, sync direct to provider
Zoom, Webex	Meeting create, update, cancel	OAuth; join links only — host start URLs never reach the renderer
Supabase (Dainvo backend)	Auth, licensing, org policy, device evidence, MCP relay	Authenticated; Row Level Security; no service role credentials on the device
Stripe / Apple StoreKit	Billing	Browser or StoreKit handoff; no card data in the app
Open-Meteo	Weather (optional, per user)	Selected location coordinates only
Hugging Face	Local AI model download (optional)	Artifacts pinned by SHA-256; policy can block downloads
Sentry	Crash reporting	Sanitized, PII off; policy can disable it
GitHub releases feed / app stores	Updates	Controlled by policy

Organization-side deliveries (SIEM webhooks, alert email) originate from the platform, not from user desktops, and go only to destinations the organization registered and tested.

What never leaves the device: the SQLite database and its keys, token vault contents, daily note files, linked local files, raw local device IDs, local AI prompts and outputs, and raw audit rows. Provider content flows only to the provider that owns it.

15. Server-side posture

- The `users.dainvo.com` portal is a static client against Supabase; browser code uses only publishable keys. Supabase service role keys, Stripe secrets, and webhook secrets are confined to Edge Function and backend configuration and are never present in the portal bundle or the desktop app.
- Tables holding user data are protected with **Postgres Row Level Security**; authenticated users reach their own rows through scoped RPCs (for example `get_my_license`, `get_my_devices`, `revoke_my_device`). Identity configuration tables are scoped per organization with writes restricted to service paths, and SCIM token IDs and hashes are unreadable from browser roles through column grants.
- **Policy signing private keys are held on the server, in the portal signing path only**, and are never distributed to clients. Desktop trust derives from the pinned public key (§8.2), so compromising portal data does not confer signing ability. The organization audit export signing key is a separate Ed25519 key with its own ID.
- Enterprise Edge Functions (`enterprise-account-policy`, `enterprise-device-evidence`, `enterprise-organizations`, `create-enterprise-policy-bundle`, `enterprise-sso`, `enterprise-scim`, `enterprise-audit`) authorize on active org membership and entitlement; access without entitlement returns `403`. `enterprise-sso` uses a narrowly scoped Supabase Management API token to register SAML providers; `enterprise-scim` authenticates SCIM routes with organization bearer tokens; `enterprise-security-jobs` accepts only the scheduler's job token.
- A separate staff surface (`admin-enterprise-security`, used from the internal back office) handles the audited recovery actions described in §7.2 — the 30 minute enforcement suspension and MFA reset — and requires staff sessions at AAL2.
- Billing functions (`create-checkout-session`, `create-billing-portal-session`, `stripe-webhook`, `verify-app-store-purchase`, `app-store-notifications`) keep payment processing with Stripe and Apple. `provider-oauth-token` performs stateless, authenticated OAuth exchange (§6.2). `delete-account` supports full account removal.
- The MCP bridge (`dainvo-mcp`, `dainvo-mcp-relay`) runs with a dedicated database role that cannot log in and has no table access; its queue tables have Row Level Security enabled with no client policies, and their contents are ciphertext with short retention (§11.3).
- The backend stores account and profile data, license and billing linkage, registered devices, org membership and identity configuration, policy bundles and history, portal and organization audit records, SIEM destination configuration, and normalized device evidence (§10.3). It does not store desktop database contents, the device token vault, raw compliance reports, or plaintext connector traffic.

16. Shared responsibility

Dainvo provides: local storage encrypted by default, key protection integrated with the operating system, behavior that fails closed, signed policy with a pinned trust root, enterprise identity built on open standards (SAML 2.0, SCIM 2.0, TOTP MFA), sanitized audit and compliance evidence with signed exports and SIEM delivery, and policy control over providers, updates, AI, external connectors, and telemetry.

Your organization should:

- Enforce operating system **full disk encryption** (and use `fullDiskEncryption.required: true` to attest it). Disk encryption covers daily notes, exports, swap, crash dumps, and backups that sit outside the encrypted database.
- On Linux fleets, deploy **GNOME Keyring or KWallet** (and connect the Snap keyring interface) before applying policy — managed devices fail closed without OS key storage.
- Deploy Windows MDM policy files with the provided script or equivalent, so `%ProgramData%\Dainvo` allows writes by Administrators only.
- Keep portal admin accounts under your privileged access controls: enroll at least two TOTP factors per owner/admin, verify work domains promptly, and complete the SAML test flow before relying on a provider.
- Keep the DNS TXT verification records in place — routing is disabled if verification keeps failing past the grace period.
- Store the SCIM bearer token in your IdP's secret store, rotate it through the portal's overlapping rotation, and revoke immediately on suspicion.
- Validate policy bundles with `enterprise:verify-policy` before deployment and confirm the reported policy hash matches after rollout. Upgrade every managed desktop before publishing a policy that includes `ai.externalMcpClients` (§9).
- Verify signed audit exports: check the manifest signature against the published audit public key and compare the data file hash with `contentSha256` before trusting an export.
- Collect and retain endpoint JSONL audit exports and compliance snapshots per your SIEM and GRC processes, and point organization SIEM delivery at a receiver that validates the HMAC signature.
- Govern the third-party services users connect (Google, Microsoft, and so on) — and the ChatGPT/Claude workspaces you opt in — under your existing SaaS controls. Dainvo enforces which providers and connectors are allowed, not those services' internal security.

17. Current limitations

Stated for accuracy — these are the honest edges of the current release:

- Dainvo Desktop is in **Beta** (v0.7.5); Dainvo AI is in **Alpha**.
- **Desktop release gating:** v0.7.5 desktops are personal clients and receive `not_configured` from the enterprise policy endpoint. Managed policy delivery, enforcement of the identity policies, and the ChatGPT/Claude connectors activate with the first Enterprise-capable desktop release (planned v0.7.6).
- Endpoint audit export is local JSONL; organization security events stream to SIEM from the platform side. There is no syslog transport, and desktops do not push their local audit logs directly to SIEM destinations.
- Detection of full disk encryption is best effort (FileVault and BitLocker checks; Linux reports unknown). It is an attestation signal, not an enforcement mechanism.
- SCIM in this version has no ETags, no bulk operations, no OAuth client credentials, and no mapping of IdP groups to Dainvo roles; groups sync for visibility and audit only.
- Supabase Auth provides no MFA recovery codes; Dainvo compensates with multiple TOTP factors and an audited staff reset that revokes sessions.
- SSO enforcement governs enterprise access; personal Dainvo features remain available to the signed-in user, and offline desktop data cannot be remotely erased — the desktop clears its enterprise context at its next authoritative online denial.
- A Windows registry (HKLM) policy location is planned; today the local MDM path relies on correctly permissioned `%ProgramData%` deployment.
- SQLite structures held in memory and temporary tables sit outside database file encryption (mitigated with `temp_store = MEMORY` where practical).
- Local AI is unavailable on Intel Macs and on 32-bit and ARM Windows in this release; the in-app remote AI provider surface is paused.
- Upstream OAuth revocation varies by provider (§6.3); Microsoft, Webex, and TickTick desktop flows record revocation as unsupported and rely on local token deletion plus revocation on the admin side.
- No formal third-party certifications (such as SOC 2 or ISO 27001) are claimed in this document.

Revision history

Revision	Date	Changes
1.0	July 2026	Initial document
1.1	July 2026	Added threat model, posture summary, and data inventory; verified navigation hardening, PKCE, server-side token exchange, RLS, audit categories, and update feed against the implementation
1.2	July 2026	Editorial pass for plain language; added sign-in provider and SSO status, account deletion, and revision history
1.3	July 2026	Added the enterprise identity suite (SAML 2.0 SSO, SCIM 2.0, MFA), organization security activity with signed exports and SIEM delivery, the ChatGPT/Claude external MCP connectors, new threat model entries, and release gating notes

This document describes security behavior of Dainvo Desktop v0.7.5 (Beta), the Dainvo Enterprise portal, and Dainvo platform services as of July 2026. Behavior may change in later releases; consult the current documentation for updates.