OrbixPanel Security Whitepaper

OrbixPanel Engineering

Version 1.1 — 2026-11-11

Contents

1. Executive Summary & Trust Model 1
2. Architecture & Trust Boundaries 2
3. Process & Application Isolation 3
4. Authentication & Secrets Management 4
5. The Agent Privilege Model 6
6. AI Copilot Privacy & Guardrails 6
7. Network, Traffic & TLS Security 7
8. Vulnerability Management & Incident Response 7
9. Operator Hardening Checklist 8
10. Document Control 8

This document describes how security is engineered into OrbixPanel across its control plane, server agent, and the applications it deploys. Claims in this paper correspond to the shipping implementation. Where a protection is on the roadmap rather than in production today, it is labeled **[Roadmap]** — we would rather tell you what is not yet done than imply a guarantee we do not deliver.

---

1. Executive Summary & Trust Model

OrbixPanel is a **self-hosted** server control panel. You install it on your own VPS or dedicated server; your data, credentials, and customer workloads never leave infrastructure you control. There is no OrbixPanel-operated middle tier that proxies your server traffic or holds your secrets.

The platform is built from three components, each with a distinct trust role:

Component	Technology	Runs as	Trust role
Control plane (API, auth, billing, reseller/tenant logic)	NestJS (Node.js)	Dedicated non-root service account	Issues sessions, stores encrypted secrets, authorizes every action
Server agent	Single Go binary	System service	Executes privileged host operations on instruction from the control plane
Dashboard UI	React (Vite)	Static assets behind nginx	Presentation only; holds no long-lived secrets

The security posture rests on four design decisions:

1. **Deny by default.** Every control-plane route requires a valid session unless it is explicitly marked public. Authorization is not opt-in per endpoint; it is enforced globally and waived only by exception.
2. **Secrets are encrypted at rest** with authenticated encryption (AES-256-GCM), never stored or logged in plaintext.

3. **The agent pulls, it does not listen.** The managed server does not expose an inbound management port to the control plane. The agent makes outbound, authenticated requests and executes only tasks addressed to it.
4. **Least privilege at every tier.** The control plane runs non-root; deployed applications run under an unprivileged service account; privileged host actions go through a narrow, auditable command surface — not a root shell.

The Shared Responsibility Model

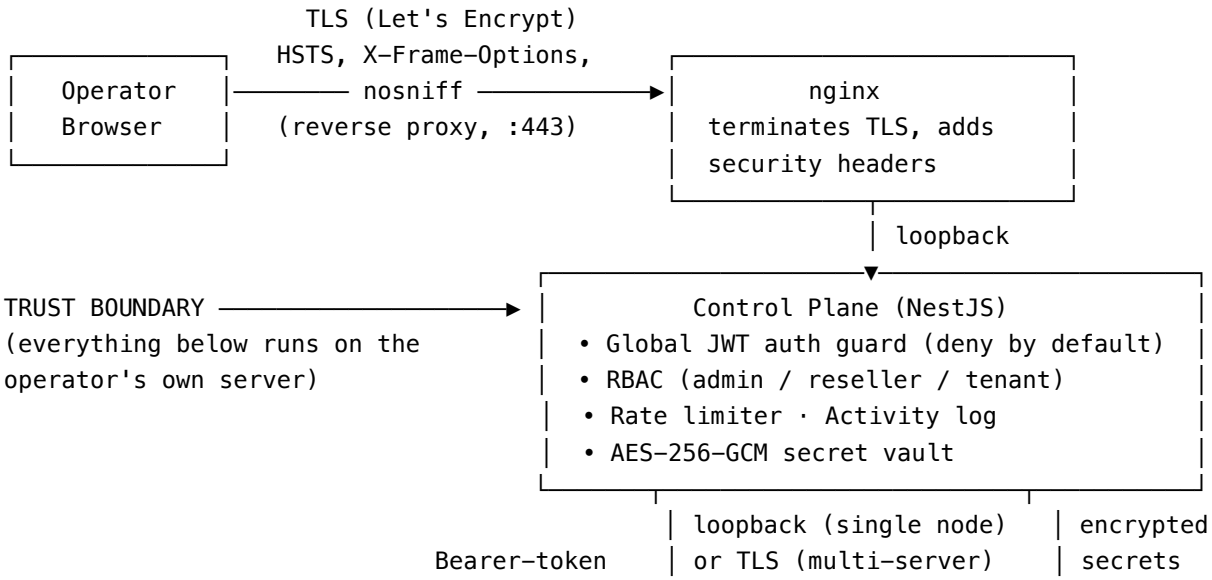
Because OrbixPanel is self-hosted, security is a partnership. The table below draws the line precisely.

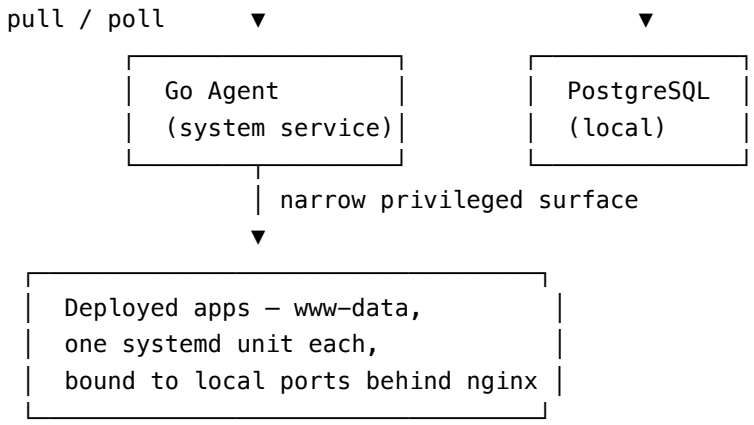
OrbixPanel (the software) secures	The operator (you) secures
Authentication & session integrity (JWT, 2FA, rate limiting)	Host OS and kernel patching
Role-based authorization on every action	SSH access, key management, and `sudo` policy outside OrbixPanel
Encryption of stored secrets (OAuth tokens, provider credentials)	Strength and secrecy of `ENCRYPTION_KEY` / `JWT_SECRET`
Agent authentication and task scoping	Host firewall / cloud security-group rules
Automated TLS issuance, renewal, and security headers	Database server credentials and network exposure
Third-party vulnerability tracking and one-click patching	Backups, disaster recovery, and cloud-account security

A single sentence captures the boundary: **OrbixPanel secures the control layer and the data it holds; you secure the host it runs on and the keys you give it.**

---

2. Architecture & Trust Boundaries





Control plane <-> agent

The agent **polls** the control plane (`GET /tasks/pending`) and reports status (`POST /agents/heartbeat`), authenticating every request with a **per-agent bearer token** issued at registration. Two consequences follow:

- **No inbound management port.** The control plane never opens a connection *into* the managed server. There is nothing for an attacker to reach on an inbound management channel because none is exposed.
- **Co-located by default.** In the standard single-node install, the agent and the control plane run on the same host and communicate over the **loopback interface** — that traffic never touches the network. In multi-server deployments the agent reaches the control plane over its TLS endpoint.

Control plane <-> browser

All browser traffic is served over TLS by nginx, which also terminates TLS for the panel and every deployed site. The dashboard is a static React bundle; it holds a short-lived session token in memory for API calls and no long-lived credentials.

---

3. Process & Application Isolation

What is shipped today:

- **The control plane runs non-root.** Its systemd unit runs as a dedicated service account with `PrivateTmp=true`, so a compromise of the Node process does not hand an attacker root and does not share a `/tmp` namespace with other services.
- **Each deployed application is its own systemd unit** (`orbixapp-<slug>.service`), running under the unprivileged **`www-data`** account — never root. Units carry restart limits (`Restart=on-failure`, start-burst caps) to contain crash-loop behavior, an always-on **`TasksMax` fork-bomb guard**, and stream to journald for tamper-evident logging.
- **Per-app resource ceilings.** Each app unit supports cgroup-enforced `MemoryHigh`/`MemoryMax` (throttle-then-OOM-kill the offending app, not the host) and `CPUQuota`, applied from the app's package limits. A memory-runaway app is contained to its own unit rather than tripping the host-wide OOM killer.

- **Every application unit is sandboxed** with a least-privilege systemd profile that confines a compromised app away from the host:
  - `NoNewPrivileges=true` and an **empty capability set** — setuid binaries and privilege-raising syscalls are dead ends; the process cannot gain new rights.
  - `ProtectSystem=full` + `ProtectHome=true` — `/usr`, `/boot`, `/etc` are read-only and all home trees are inaccessible; the app writes only within its own deployment tree.
  - `PrivateTmp=true` — a private `/tmp` per app, so apps on the same node cannot read or plant files in each other's temp space.
  - Kernel, cgroup, namespace and realtime restrictions (`ProtectKernelTunables/Modules`, `ProtectControlGroups`, `RestrictNamespaces`, `RestrictRealtime`, `LockPersonality`, `RestrictSUIDSGID`) plus a service-grade **syscall allowlist** (`SystemCallFilter=@system-service`) and a restricted set of socket address families close the common container-escape and kernel-attack avenues.
- **Deployed apps bind to local ports** and are reachable only through the nginx reverse proxy, which fronts them with TLS and the platform security headers. An app does not expose its own port to the public internet.

What we do *not* claim, and are building:

- **[Roadmap] Per-tenant OS-level isolation.** Today, applications deployed on a single node share the `www-data` service account. The systemd sandbox above gives each app a kernel-enforced boundary *against the host*, but apps sharing that account do **not** yet have a cryptographic or UID-enforced boundary *between tenants* on the same node. Operators running genuinely untrusted multi-tenant workloads should isolate tenants at the host level (separate nodes, or OS/container isolation) until the following land:
  - Dedicated per-tenant UID/GID with filesystem ownership separation (in progress)
  - cgroup-enforced CPU and memory ceilings per tenant. Per-*app* ceilings and an always-on fork-bomb guard ship today (above), and per-tenant **disk-quota enforcement** ships; per-tenant CPU/memory *slices* are being rolled out on top.

We state this plainly because a false isolation claim is more dangerous than a documented limitation.

---

4. Authentication & Secrets Management

4.1 Session authentication

- **Password storage:** user passwords are hashed with **bcrypt**; plaintext passwords are never stored and are never written to logs.
- **Password strength:** new and changed passwords must be at least **12 characters** and contain both letters and digits.
- **Sessions:** issued as signed JWTs (HS256) with configurable expiry (24 hours by default); expiration is always enforced (`ignoreExpiration: false`). Each token carries a unique identifier (`jti`).
- **Refresh & rotation:** login also issues an opaque **refresh token**, stored only as a SHA-256 hash. It **rotates on every use** — refreshing revokes the presented token and issues a new one — and presenting an already-used refresh token is treated as theft and **revokes every active session** for that user.
- **Revocation:** logout (or an administrative action) places the access token's `jti` on a **revoca-**

tion denylist that the global guard checks, so a token can be killed before its natural expiry — JWTs are no longer purely self-expiring.
- **Deny by default:** a single global guard protects every route. An endpoint is reachable without a session only if a developer explicitly marks it public, which makes the public surface small, deliberate, and reviewable.
- **Audience separation:** panel and tenant tokens are cryptographically the same family but are tagged and **validated apart** — a tenant-scoped token is rejected on admin-panel routes, so the two audiences cannot cross even though they share a signing secret.

4.2 Two-factor authentication

2FA uses a two-stage token exchange. After a correct password, the server issues a **pre-authentication token** that is purpose-scoped (`2fa-pending`) and valid for **five minutes** against the 2FA-verification route *only*. It cannot be used as a session token. A full session is issued exclusively after the second factor verifies.

4.3 Brute-force and abuse resistance

Rate limiting is applied globally (60 requests/minute) and tightened on sensitive routes — **login and registration are capped at 5 attempts/minute**. All write operations are recorded by an activity-log interceptor for after-the-fact audit.

4.4 Secret encryption at rest

Secrets that must be recovered later — OAuth client secrets, access tokens, and refresh tokens — are protected with **AES-256-GCM**, an authenticated cipher:

- A **unique random 96-bit IV** is generated per encryption operation.
- A **unique random 128-bit salt is generated per secret**, so two identical plaintexts never derive the same key and a database-wide attack against the key derivation is infeasible.
- The **GCM authentication tag** is stored alongside the ciphertext, so any tampering with stored secrets is detected on decryption rather than silently accepted.
- The 256-bit key is derived per secret with **scrypt** from the operator-controlled `ENCRYPTION_KEY` (falling back to `JWT_SECRET`), so the raw key never lives in the database and the operator owns the root of trust.
- **Defense in depth against database exfiltration:** when an operator sets a per-install `ENCRYPTION_SALT` (generated automatically by the installer), it is mixed into the key derivation and kept **only in the server environment, never in the database**. An attacker who steals the entire database — every per-secret salt included — still cannot derive any key without this out-of-band value.

Stored format is versioned — `v3:salt:iv:authTag:ciphertext` with the install salt, or `v2:salt:iv:authTag:ciphertext` without — so the scheme can evolve without re-encrypting the database, and no plaintext secret is ever persisted.

4.5 Git provider OAuth (GitHub / GitLab / Bitbucket)

OrbixPanel's one-click Git deployment is designed so that **tokens never transit a third-party OrbixPanel relay**:

- The OAuth **code-for-token exchange happens server-to-provider**, directly from your control plane to GitHub/GitLab/Bitbucket. OrbixPanel.com is not in the path.
- The resulting **access and refresh tokens are AES-256-GCM encrypted before they touch the database** (§4.4), and decrypted only in memory at the moment a clone or API call needs them.
- The OAuth **`state` parameter is itself an encrypted, timestamped, nonce-bearing blob**, giving CSRF protection and replay resistance on the callback — a forged or stale callback fails to decrypt and is rejected.
- Operators register the OAuth application once; individual tenants never handle provider API tokens or paste long-lived credentials.

4.6 Authorization (RBAC)

Beyond authentication, a role guard enforces **role-based access control** (admin / reseller / tenant). Privileged routes declare the roles they require, and requests lacking that role are refused regardless of a valid session. Additional global guards gate **licensed features** and **API-token scopes**, so an API token can be issued with a narrow capability set rather than full account power.

---

5. The Agent Privilege Model

The Go agent is the only component that performs privileged host operations, and it is deliberately constrained:

- **It executes, it does not decide.** The agent runs discrete, typed tasks dispatched by the control plane (write a file, reload a service, issue a certificate, deploy a release). It has no open-ended remote shell exposed to the network.
- **Authenticated per agent.** Every task poll and heartbeat carries the agent's unique bearer token; an unknown or missing token is rejected. The token is a 192-bit random value **stored only as a SHA-256 hash** — the plaintext is shown to the agent once at registration and is never recoverable from the database, so a database read does not yield working agent credentials. Tokens can be **rotated** without re-provisioning the host.
- **Narrow privileged surface.** Host actions that require elevation go through a constrained, auditable command set rather than a blanket root grant. Hardening this surface further (tighter argument allow-listing) is tracked as ongoing work.
- **Outbound-only in the standard topology.** Because the agent initiates every connection and listens for nothing, the managed server needs no inbound management port open to OrbixPanel.

---

6. AI Copilot Privacy & Guardrails

OrbixPanel's AI copilot is **optional, pluggable, and per-surface** — the entire platform functions with no AI configured at all.

- **100% local inference is supported.** With a local model served via **Ollama**, prompts, logs, and system statistics are processed **on your own server with zero external API calls**. No telemetry, no third-party model provider, no data egress.

- **Hosted models are opt-in.** If an operator chooses a hosted model (for small boxes that cannot run a local LLM), that is an explicit configuration choice, per surface, that the operator makes and can revoke.
- **Diagnostic-first by design.** The copilot's shipping role is to *recommend* — surface the likely cause of an incident from local logs and metrics — rather than silently execute privileged commands.
- **[Roadmap] Confirm-every-write actions.** Secure tool-calling, in which the copilot can *perform* an action, is being built behind an explicit **confirm-before-every-write** guardrail. Until it ships, the copilot does not take autonomous write actions on your host.

---

7. Network, Traffic & TLS Security

- **Automated TLS.** Certificates are issued and **auto-renewed via Let's Encrypt** using the webroot challenge (`/var/www/certbot`), which issues without reloading a possibly-broken nginx config and keeps renewal non-disruptive. The panel and every deployed site are served over HTTPS.
- **Reverse-proxy isolation.** nginx terminates TLS and proxies to applications bound on local ports; backends are not directly exposed.
- **Hardened default headers.** Every HTTPS server block ships with:
  - `Strict-Transport-Security: max-age=63072000; includeSubDomains` (2-year HSTS, on TLS)
  - `X-Frame-Options: SAMEORIGIN` (clickjacking defense)
  - `X-Content-Type-Options: nosniff` (MIME-sniffing defense)
  - `Referrer-Policy: strict-origin-when-cross-origin`
  - `Content-Security-Policy: frame-ancestors 'self'` — a clickjacking-focused CSP baseline applied to every deployed site (it constrains who may frame the site without restricting the resources the page itself loads).
- **[Configurable] Full Content-Security-Policy.** A *resource-restricting* CSP (script/style/connect sources) is not forced by default because a safe policy is application-specific and would break arbitrary apps; operators can add one per site. Documenting recommended per-runtime CSP baselines is on the roadmap.

---

8. Vulnerability Management & Incident Response

8.1 Third-party advisory tracking (shipped)

OrbixPanel ships a **Security Advisories** subsystem that tracks CVEs and updates for the third-party components it manages, **matches them against the versions actually installed** on your server, and surfaces **one-click upgrades** applied through the agent. This turns "is my stack exposed to the latest CVE?" from a manual audit into a dashboard answer with a remediation button.

8.2 Dependency & supply-chain hygiene

The codebase is built on maintained, audited ecosystems (NestJS/Node, Go, React). **Continuous integration runs static analysis and dependency scanning** on every change and on a

weekly schedule: **Semgrep** SAST plus **`npm audit`** on the Node services, and **`govulncheck`** (the Go vulnerability database) plus `go vet` on the agent — a build fails on a known high-severity advisory or a flagged code pattern. The single-binary Go agent also minimizes runtime dependency surface on the host.

8.3 Responsible disclosure

We welcome coordinated disclosure from security researchers.

- **Report to: security@orbixpanel.com**
- **Please include:** affected component and version, reproduction steps, and impact assessment. A proof-of-concept is appreciated.
- **Our commitment:** we will acknowledge receipt, work with you on remediation timelines, and credit reporters who wish to be named. Please do not publicly disclose before a fix is available, and do not access data or disrupt service beyond what is necessary to demonstrate a finding.
- **Policy & safe harbor:** the full coordinated-disclosure policy — scope, timelines, and a good-faith safe-harbor statement — is published as `SECURITY.md` in each OrbixPanel repository. A formal paid bug-bounty program is on the roadmap.

---

9. Operator Hardening Checklist

To hold up your side of the shared-responsibility model:

1. Set a **long, unique `ENCRYPTION_KEY`, `ENCRYPTION_SALT`, and `JWT_SECRET`** (distinct, high-entropy — the installer generates all three). These are the root of trust for stored secrets and sessions. Once set on a live install, **never rotate `ENCRYPTION_KEY`/`ENCRYPTION_SALT`** — doing so orphans already-encrypted secrets.
2. Keep the **host OS and kernel patched**; subscribe to the in-panel advisory feed.
3. Restrict the **host firewall / security group** to the ports you actually serve.
4. Enforce **key-based SSH** and disable password SSH at the OS level.
5. Enable **two-factor authentication** on administrator accounts.
6. Run **genuinely untrusted multi-tenant workloads on separate nodes** until per-tenant OS isolation (§3) ships.
7. Maintain **off-host backups** and test restores.
8. Optionally submit your domain to the **HSTS preload list** (OrbixPanel already serves a two-year `includeSubDomains` HSTS header; preload is a lasting commitment and is therefore left to the operator rather than forced).

---

10. Document Control

Field	Value
Title	OrbixPanel Security Whitepaper
Version	1.1
Date	2026-11-11
Owner	OrbixPanel Engineering

Field	Value
Security contact	security@orbixpanel.com
Review cadence	Revised on material architecture or control changes

*This whitepaper describes security controls as implemented at the version above. Items marked **[Roadmap]** are planned and not warranted as present. OrbixPanel is provided under its applicable license terms; nothing herein constitutes a warranty of fitness or a guarantee against compromise.*