PORTAINER.iO

Whitepaper

PORTAINER SECURITY ARCHITECTURE

Enterprise Security Reference

Audience: Chief Information Security Officers, Security Architects, Enterprise Risk Teams
Edition: Portainer Business Edition

Classification: Pre-Sales / Technical Due Diligence

23.03.26

EXECUTIVE SUMMARY

As organizations adopt containers and Kubernetes out of operational necessity rather than experimentation, security becomes a board-level concern rather than a purely technical one. Portainer is designed as an enterprise-grade container management platform that sits **between users and container platforms**—not as a simple UI overlay, but as a governed control plane that enforces access, policy, and visibility across the full container estate.

This whitepaper provides a deep technical examination of Portainer Business Edition's security architecture. It covers cryptographic controls, transport security, credential handling, identity integration, and operational threat mitigations. It is intended for CISOs, security architects, and technical reviewers performing vendor due diligence prior to enterprise deployment.

Each section identifies not only *what* Portainer does, but *how* it is implemented—referencing the underlying mechanisms so that security teams can independently verify claims and model residual risk.

DESIGN PRINCIPLES

1 LEAST PRIVILEGE BY DEFAULT – Users receive only the minimum access required to perform their role. No implicit elevation, no ambient cluster-admin.

2 CENTRALIZED GOVERNANCE – Security, access control, and auditing are enforced consistently across all clusters, environments, and teams from a single control plane.

3 SEPARATION OF DUTIES – Platform teams retain control of infrastructure while safely delegating self-service to application teams, with hard enforcement boundaries.

4 DEFENSE IN DEPTH – Portainer augments native Kubernetes and container runtime security rather than replacing it. Controls are additive, not substitutive.

5 ZERO TRUST BETWEEN COMPONENTS – All communication between the Portainer Server and remote agents is authenticated. For edge environments, mutual TLS provides cryptographic identity assurance at the transport layer.

6 DATA MINIMIZATION AT THE BOUNDARY – Sensitive data—credentials, secrets, tokens—is encrypted or redacted before it leaves the plane where it originated.

7 OPERATIONAL SIMPLICITY – Security controls must be understandable and operable by non-specialists to be effective at scale.

THREAT MODEL

Portainer is designed to mitigate common risks faced by organizations adopting containers at enterprise scale:

Threat	Portainer Mitigation
Excessive Kubernetes cluster-admin access	Fine-grained RBAC with least-privilege defaults
Uncontrolled `kubectl` and direct API access	Centralized access gateway; kubeconfig lifecycle control
Credential sprawl across teams and clusters	Encrypted credential store; scoped secret distribution
Insider threat — admin access to peer secrets	Sensitive data isolation per user/team; write-only credential model
Compromised agent communication channel	mTLS with mutual certificate authentication
Brute-force credential attacks on the auth endpoint	IP-based rate limiting with automated ban
Orphaned access after employee termination	Token invalidation via `TokenIssueAt;` user deletion invalidates all tokens and API keys
Misconfiguration-driven privilege escalation	Namespace scoping; guardrails; RBAC policy enforcement
Audit gap during incident response	Structured audit log stream with SIEM integration
Sensitive data leakage into log systems	Field-level redaction blacklist applied to all audit payloads

ARCHITECTURAL OVERVIEW

Portainer operates as a management layer deployed within or adjacent to container environments. It communicates securely with container runtimes and Kubernetes APIs using scoped credentials, and with remote environments via an authenticated agent channel.

CORE COMPONENTS

Portainer Server

The central control plane. Responsible for authentication, authorization, policy enforcement, data encryption, audit logging, and all user-facing API interactions. The Server never directly exposes container or Kubernetes APIs to end users — all requests are proxied and scoped through Portainer's authorization layer.

Portainer Edge Agent (Recommended)

The preferred method for connecting all environments to Portainer. The Edge Agent is deployed inside a managed Docker, Kubernetes, or Podman environment and initiates an outbound connection to the Portainer Server, reversing the connection direction. The Portainer Server never needs network-level access to the remote environment — no inbound firewall rules are required on the remote network. The Edge Agent channel supports mutual TLS, providing the strongest security posture available for remote environment communication. Edge Agent is the recommended deployment model for all new environments.

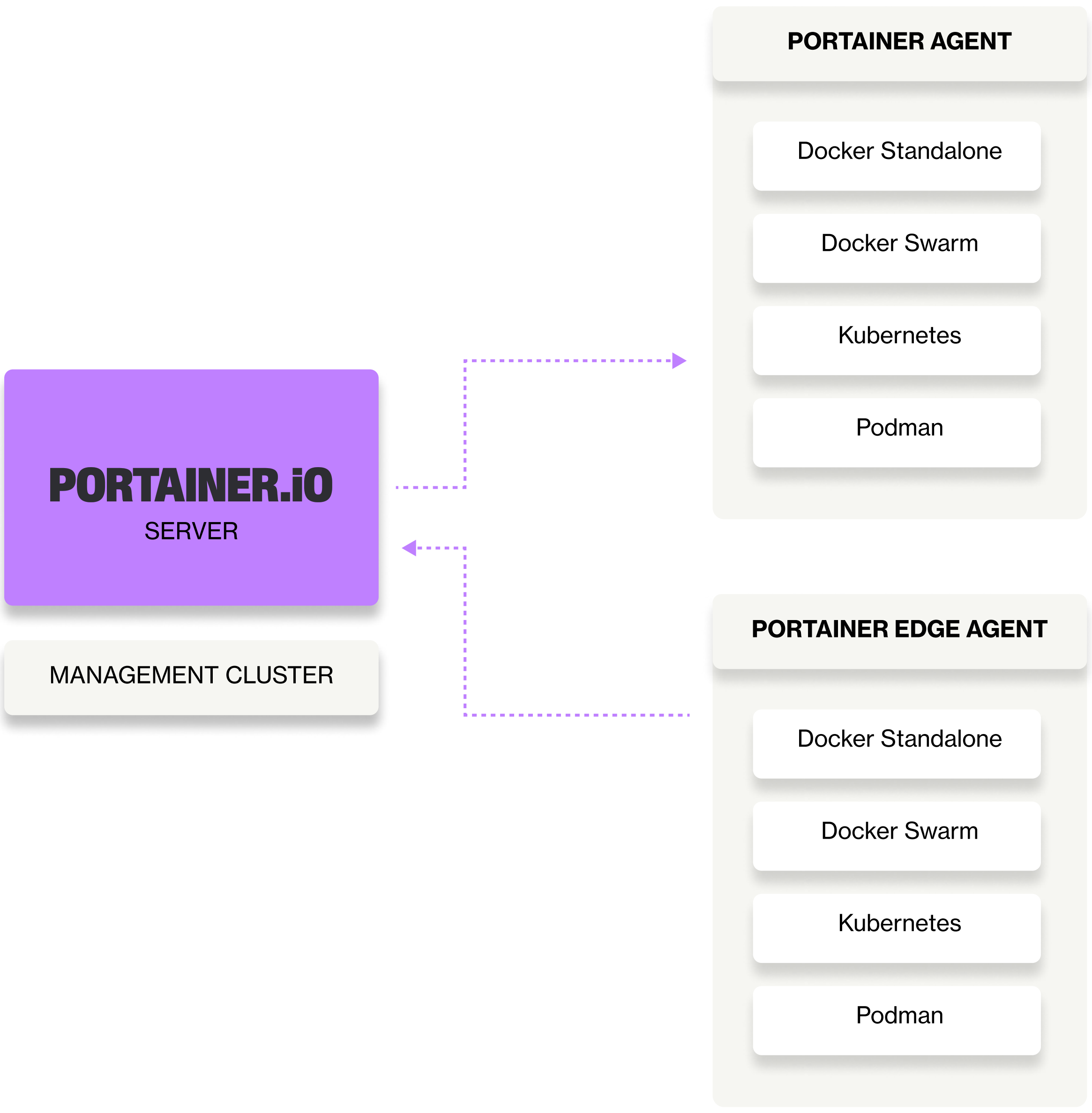
Portainer Edge Agent (Async Mode)

An extension of the Edge Agent model for environments with intermittent or highly constrained connectivity — industrial IoT networks, far-edge deployments, or offline-capable environments. Rather than maintaining a persistent tunnel, the Async Agent periodically polls a command queue, executes pending operations locally, and returns results on its next check-in. This model reduces data transfer volume compared to persistent tunnel mode and is suited to environments with limited bandwidth or intermittent connectivity.

All interactions between users and platforms are mediated by Portainer, enabling centralized enforcement of security controls regardless of where the target environment is physically located.

Portainer Agent (Legacy)

A legacy connection method retained for backwards compatibility. The Standard Agent is deployed inside a managed environment and exposes a TLS API that the Portainer Server connects to inbound. This requires the Portainer Server to have network access to the agent endpoint, and does not support mutual TLS. Organizations operating existing Standard Agent deployments are encouraged to migrate to the Edge Agent model.



SECTION 1: DATABASE ENCRYPTION

Overview

Portainer Server maintains a local database (BoltDB, a key-value store whose library is embedded within the Portainer binary, with the database file persisted on the host filesystem) that stores all application state: environment configurations, user accounts, RBAC assignments, credentials, and secrets. Portainer Business Edition supports full encryption of this database at rest.

Cryptographic Implementation

Portainer Business Edition uses **AES-256-GCM** (Advanced Encryption Standard, 256-bit key, Galois/Counter Mode) for database encryption. GCM mode provides both confidentiality and authenticated integrity (AEAD — Authenticated Encryption with Associated Data), meaning tampering with any encrypted record will be detected on decryption.

Encryption operates at the object serialization layer: when an encryption key is present, every record is encrypted before being written to BoltDB and decrypted on read. When encryption is enabled, **every record in the database is encrypted.**

Important: Database encryption is an opt-in feature that must be explicitly enabled by the operator via -`-secret-key-name`. When it is not enabled, **all database records are stored in plaintext**, including registry credentials, API keys, git credentials, and cloud provider credentials. Two exceptions apply regardless of database encryption status — internal user passwords are always stored as **bcrypt hashes** (never recoverable), and API key values are stored as **hashes** rather than plaintext. All other sensitive values require database encryption to be protected at rest. Enabling database encryption is strongly recommended for all production deployments.

Enabling Encryption

Database encryption is enabled by supplying `the` `--secret-key-name` CLI flag at Portainer startup. This flag specifies the name of a Docker Secret or Kubernetes Secret from which the 32-byte (256-bit) encryption key is read at runtime from `/run/secrets/<secret-key-name>`. The default secret name is `portainer`.

Docker Swarm deployment example:

```
docker secret create portainer /path/to/keyfile
# Reference via --secret-key-name portainer in the Portainer service definition
```

For Kubernetes deployments, the key is stored as a Kubernetes Secret and mounted into the Portainer pod. Operators using external secrets management (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) should use their platform's secrets injection mechanism to deliver the key material at runtime.

Migration from Unencrypted to Encrypted

Portainer detects at startup whether the database needs to be migrated:

- `portainer.db` exists and a key is supplied → migrates to encrypted `portainer.edb`
- `portainer.edb` exists and a key is supplied → normal encrypted operation
- `portainer.edb` exists but no key is supplied → Portainer refuses to start (fatal error)
- Both `portainer.db` and `portainer.edb` exist → Portainer refuses to start (ambiguous state)

Irreversibility

Full database encryption is a **one-way operation.** There is no supported path to decrypt an encrypted database back to plaintext. Loss of the key renders the database permanently unrecoverable. The key must be backed up securely and independently of the database file

Security Considerations for Operators

- **Database encryption is opt-in and disabled by default.** Without it, registry credentials, API keys, git credentials, and cloud provider credentials are stored in plaintext. Operators should treat enabling database encryption as a baseline production security requirement.
- The Portainer data volume should be mounted on an encrypted host volume in production, providing defense in depth.
- The encryption key must never appear in container startup logs, environment variable dumps, or `docker inspect` / `kubectl describe` output.
- The Kubernetes Secret holding the key should be protected with RBAC (restrict `get` on the secret resource) and etcd encryption at rest.

SECTION 2: SERVER-AGENT AUTHENTICATION AND AUTHORIZATION

Overview

Portainer manages environments using different agent types, each with a distinct security model. The **Edge Agent is the recommended connection method** for all new deployments. The Standard Agent remains available as a legacy option for existing deployments.

Edge Agent: Reversed-Channel with Chisel Tunnel (Recommended)

The Edge Agent is the preferred deployment model. Rather than the Server connecting to the Agent, the Edge Agent establishes an outbound connection to the Portainer Server's tunnel endpoint (default port 8000). This provides several security advantages:

- **No inbound firewall rules are required on the remote network.** The remote environment never exposes a listening port to the Portainer Server.
- **The remote environment's network perimeter remains closed.** This is critical for production environments, OT/ICS networks, and cloud VPCs where inbound access from management systems is undesirable.
- **mTLS is supported,** providing cryptographic identity assurance at the transport layer (Section 3).

The underlying tunnel technology is **Chisel** — an SSH-over-HTTP tunneling tool providing encryption and server identity verification via the tunnel server's SSH host key fingerprint.

Edge Key structure. The Edge Agent uses an Edge Key — a base64-encoded string containing: `<portainer_url>|<tunnel_addr>|<tunnel_fingerprint>|<endpoint_id>`. The `tunnel_fingerprint` is the SSH host key fingerprint of the Portainer tunnel server, which the Agent verifies on connection to prevent connecting to an arbitrary tunnel server.

The Edge Key is best understood as a configuration token; **mTLS (Section 3) is the mechanism that provides strong mutual authentication and should be enabled for all security-sensitive Edge Agent deployments.**

A special global Edge Key exists for auto-onboarding scenarios (`endpoint_id = 0`), which is shared across all environments using that feature rather than being unique per environment.

Async Edge Agent: Polling Model (Recommended for Constrained Environments)

An extension of the Edge Agent model for environments with intermittent or highly constrained connectivity — industrial IoT networks, far-edge deployments, or offline-capable environments. Rather than maintaining a persistent tunnel, the Async Agent periodically polls a command queue, executes pending operations locally, and returns results on the next check-in. This model reduces sustained data transfer volume and is suited to environments with limited bandwidth or intermittent connectivity.

Standard Agent: ECDSA Signature Authentication (Legacy)

The Standard Agent remains available for backwards compatibility but is not the recommended deployment model for new environments. It exposes a TLS API on port 9001, with the Portainer Server connecting inbound. Communication security is provided by:

ECDSA digital signature verification. The Portainer Server signs each request using an ECDSA private key and includes the signature and corresponding public key in request headers. On first contact, the Agent records the Portainer Server's public key and will only accept subsequent requests signed by that key — a one-time association that locks the Agent to a specific Portainer instance.

The Standard Agent does not support mutual TLS. An optional AGENT_SECRET environment variable can be configured on both the Agent and Server to add a shared secret to the signature process, tightening the association beyond the default fixed-message signing.

Security shutdown: If no Portainer instance associates with the Standard Agent within the configured timeout (default 72 hours), the Agent shuts down its API server, preventing orphaned agents from remaining permanently accessible.

Organizations with existing Standard Agent deployments are encouraged to migrate to the Edge Agent model to benefit from the stronger security posture mTLS provides.

Authorization Scoping

Regardless of agent type, each agent credential or key is associated with a specific environment record in the Portainer database. Agent connections are scoped to the environment they registered against and cannot access configuration or data belonging to other environments.

SECTION 3: MUTUAL TLS (mTLS) BETWEEN EDGE AGENT AND SERVER

Overview

Portainer Business Edition supports **mutual TLS (mTLS)** for the Edge Agent channel. mTLS provides cryptographic proof of identity at the transport layer for both parties — the Server verifies the Agent's identity and the Agent verifies the Server's identity. This is the primary mechanism for strong mutual authentication between Portainer and remote environments, and **should be considered mandatory for all production Edge Agent deployments.**

Why mTLS Matters for Edge Deployments

The Edge Key alone provides only tunnel fingerprint verification — it confirms the Agent is connecting to the correct tunnel server but does not provide per-agent identity assurance or prevent server impersonation by an attacker who controls network routing between the agent and server. mTLS closes this gap by requiring both parties to hold valid certificates signed by a trusted CA before the TLS handshake completes.

Certificate Infrastructure

Two certificate configurations are available:

Standard agent mTLS (port 9443)
Agent client certificates are validated against a CA supplied via the `--sslcacert` flag. This CA is loaded into the HTTPS server's `ClientCAs` pool with `tls.VerifyClientCertIfGiven`, enabling client certificate verification on the main HTTPS port without requiring all clients (browsers) to present a certificate.

Separate mTLS certificate configuration (Edge Agent) When a dedicated mTLS identity for the Portainer Server is required, three additional flags are used together:

- `--mtlscert` — path to the Portainer Server's dedicated mTLS certificate
- `--mtlskey` — path to the Portainer Server's dedicated mTLS private key
- `--mtlscacert` — path to the CA certificate used to validate Edge Agent certificates

All three flags must be supplied together to activate this mode (`UseSeparateCert`). When active, the Portainer Server presents the mTLS certificate to agents connecting using the corresponding SNI hostname, while continuing to serve the standard TLS certificate to browser clients on the same port.

What mTLS Protects Against

Attack Vector	Protection Mechanism
Rogue agent connecting to server	Server rejects: cert not signed by trusted CA
Server impersonation / MitM	Agent rejects: cert not signed by trusted CA
Network-level eavesdropping	All traffic encrypted by TLS with AEAD cipher suites

TLS Configuration

Portainer enforces **TLS 1.2 minimum** across all internal and external communications, with TLS 1.3 supported. HTTP/2 is explicitly disabled on the HTTPS server, preventing protocol downgrade vectors. The cipher suite preference list prioritizes forward-secrecy cipher suites (ECDHE key exchange).

SECTION 4: SECURE CREDENTIAL DISTRIBUTION TO REMOTE ENVIRONMENTS

Overview

A common challenge in distributed container management is getting registry credentials and other secrets to a remote environment for a deployment operation, without those credentials being stored persistently on the remote host or exposed in the management channel.

Portainer addresses this through a **just-in-time credential injection model**: credentials are transmitted to the Agent only at the moment they are needed, are never stored on the Agent, and are protected in transit by the authenticated, encrypted channel.

Registry Credential Flow

When a user triggers a deployment operation on a remote environment requiring a private registry pull:

1. **Credential retrieval:** The Portainer Server retrieves the registry credentials from its database (encrypted if database encryption is enabled). The credentials are decrypted in Server memory only.
2. **Scoped authorization check**: Portainer verifies the user has permission to deploy to the target environment and that the registry is permitted for use in that environment or namespace.
3. **Encrypted transmission:** The credentials are transmitted to the Agent over the authenticated, TLS-encrypted channel. They are not written to disk on the Agent.
4. **Runtime injection:** The Agent injects the credentials into the container pull operation (e.g., as a Docker registry auth config or Kubernetes `imagePullSecret`). The credentials exist in Agent memory only for the duration of the pull operation.
5. **No persistent storage on Agent:** The Agent does not cache or persist received credentials.

Kubernetes imagePullSecrets

For Kubernetes environments, Portainer creates or updates `imagePullSecret` resources in the target namespace at deployment time, scoped to that namespace only.

Security Considerations

- Registry credentials should be entered once by an authorized administrator and thereafter managed by Portainer.
- Portainer supports per-environment and per-namespace registry scoping.
- Credentials are not exposed in Portainer's audit log (see Section 6 for SIEM redaction).
- Without database encryption enabled, stored credentials are in plaintext in the database (Section 1).

SECTION 5: SENSITIVE DATA ISOLATION – ADMINS CANNOT SEE PEER SECRETS

Overview

Portainer Business Edition implements a **write-only credential model** ensuring that administrative access to the platform does not automatically confer visibility into credentials stored by other users or teams.

Write-Only Credential Model

When a user or administrator enters a sensitive credential into Portainer (a registry password, an API token, a cloud provider secret), the value is written to the BoltDB database as part of the full resource record. **Without database encryption enabled, these values are stored in plaintext.** When database encryption is enabled (Section 1), the entire record is encrypted using AES-256-GCM before being persisted to disk.

Portainer explicitly clears the `Password` field before serializing API responses for registry inspection (`GET /registries/{id}`), setting it to an empty string which the `omitempty` JSON tag then suppresses. The `Password` field also carries `omitempty` at the struct level, providing an additional layer of suppression. Empirical testing confirms that registry passwords are not returned in any API response, including list and inspect endpoints.

The unconditional protections — independent of database encryption — are:

- **Internal user passwords** are always stored as bcrypt hashes and are never recoverable.
- **API key values** are stored as hashes only; the plaintext value is never persisted.

Kubernetes Secret Visibility Control

- For Kubernetes environments, native Kubernetes Secrets are base64-encoded (not encrypted) and accessible to any principal with `get` permissions in a namespace. Portainer adds a governance layer:
  - Users are granted access to secrets through Portainer's RBAC, not directly via Kubernetes RBAC.
  - Portainer Business Edition supports restricting visibility of secret values to environment administrators only (`RestrictSecrets` configuration), while allowing application developers to reference secrets in deployments without reading their values.

Cross-Team Isolation

Resources created by one team are not visible to another unless explicitly shared by an administrator. This separation is enforced at the API layer — direct API calls with an admin token will not return plaintext credential values for resources the admin did not create.

SECTION 6: SIEM STREAM AND SENSITIVE DATA REDACTION

Overview

Portainer Business Edition generates a structured audit event stream suitable for ingestion by SIEM platforms (Splunk, Microsoft Sentinel, Elastic SIEM, and others). Portainer implements field-level redaction applied to all audit event payloads before they are written or forwarded, ensuring the audit stream cannot itself become a source of credential leakage.

Audit Log Architecture

Portainer captures two categories of audit events:

Authentication activity logs — emitted for every authentication event:

- `username` — the authenticating user
- `type` — `success`, `failure`, or `logout`
- `method` — the authentication method used (internal, LDAP, OAuth)
- `origin` — the source IP address

Authentication failures are emitted at syslog priority `LOG_AUTH | LOG_ALERT`. Successful authentications at `LOG_AUTH | LOG_INFO`.

User activity logs — emitted for all mutating API operations (POST, PUT, PATCH, DELETE) that return a 2xx response:

- `username` — the acting user
- `context` — the environment name (or "Portainer" for global operations)
- `action` — the HTTP method and URL path
- `payload` — the sanitized request body

DELETE operations are emitted at `LOG_SYSLOG | LOG_ALERT`. POST and PUT at `LOG_SYSLOG | LOG_NOTICE`. Read operations are not logged.

Sensitive Field Redaction

Portainer applies a redaction blacklist to all user activity log payloads before writing. Any key matching the blacklist (case-insensitively) has its value replaced with `[REDACTED]`. The redacted payload is re-serialized before being written to the log.

Blacklisted Field Key	Covers
`password`, `newpassword`	User passwords, registry passwords, LDAP bind passwords
`apikey`	API key values
`clientsecret`	OAuth client secrets
`secretaccesskey`	AWS/S3 secret access keys
`privatekey`	Private Key Material
`passphrase`	Passphrase values
`repositorypassword`	Git repository passwords
`azureauthenticationkey`	Azure credentials
`jsonkeybase64`	GKE/GCP JSON key files
`tlscacertfile`, `tlscertfile`, `tlskeyfile`	TLS certificate and key material
`kubeconfig`	kubeconfig file contents
`data`, `stringdata`, `binarydata`	Kubernetes Secret data fields

SIEM Integration

Portainer Business Edition supports syslog export with:

- **Protocol:** `udp`, `tcp`, or `tcp+tls`
- **Format:** `rfc3164` or `rfc5424` (default)
- **mTLS for syslog:** mutual TLS authentication to the syslog server via `-syslog-cert`, `-syslog-key`, and `-syslog-ca-cert`

SIEM Use Cases

- **Brute force detection:** Multiple `failure` authentication events from the same `origin` IP
- **Privilege escalation detection:** POST/PUT operations on `/api/users` or `/api/roles`
- **Anomalous deployment activity:** Stack or container operations from unexpected users or outside change windows
- **Credential modification monitoring:** PUT operations on `/api/registries` (payload redacted, event captured)
- **Offboarding verification:** User deletion events correlated against subsequent authentication attempts
- **Admin account use:** Any authentication event for the built-in administrator (user ID 1) should be treated as notable

SECTION 7: BRUTE FORCE PROTECTION AND ACCOUNT LOCKOUT

Overview

Portainer Business Edition protects the authentication endpoint against high-velocity credential attacks through **IP-based rate limiting**.

Implementation

The rate limiter is applied to the `/auth` and `/auth/oauth/validate` endpoints:

- **Threshold:** 10 requests per 1-second window per source IP
- **Ban duration:** 1 hour for IPs exceeding the threshold
- **Response:** Banned IPs receive `403 Forbidden`

The rate limiter is **IP-based only.** Portainer does not implement per-account failed attempt tracking or account lockout. Organizations requiring per-account lockout should enforce this at the upstream identity provider when using SSO (Section 8).

Administrative Account Protection

When external authentication is enabled in Portainer, **only the initial built-in administrator account (user ID 1) retains the ability to authenticate using internal credentials** — all other users must authenticate via the configured external provider. This is enforced at the authentication handler level. The internal admin account should be treated as a break-glass credential: strong unique password, stored in a privileged access vault, with all use generating audit events monitored via SIEM

Interaction with SSO

For organizations using SSO (Section 8), brute-force protection is largely delegated to the upstream identity provider, which typically provides more sophisticated controls including adaptive MFA, geo-based conditional access, per-account lockout, and real-time threat intelligence.

SECTION 8: SSO, UPSTREAM AUTHENTICATION, AND AUTHORIZATION MAPPING

Overview

Portainer Business Edition integrates with enterprise identity infrastructure through OIDC, SAML, and LDAP, allowing organizations to enforce consistent authentication controls without maintaining a separate credential store.

Supported Authentication Protocols

OAuth 2.0 / OpenID Connect (OIDC)
Supports Microsoft Entra ID, Okta, Keycloak, Auth0, Google Workspace, and any OIDC-compliant IdP. Uses the Authorization Code flow. Portainer validates the ID token signature against the IdP's JWKS endpoint and verifies claims (issuer, audience, expiry). The OAuth client secret is stored in Portainer's database (encrypted if database encryption is enabled).

SAML 2.0
Supports AD FS, Microsoft Entra ID, Okta, Ping Identity, and any SAML 2.0-compliant IdP. Portainer acts as the Service Provider, validating signed SAML Assertions against the IdP's configured X.509 signing certificate.

LDAP / Active Directory
Supports LDAPS or STARTTLS. Bind DN password stored in the database (encrypted if database encryption is enabled). Supports custom search filters, group membership queries, and automatic admin role assignment based on AD group membership (`AdminAutoPopulate`). User credentials are never stored by Portainer — only the bind DN service account credential is persisted.

MFA and Conditional Access

When using OIDC or SAML, MFA enforcement is the responsibility of the upstream IdP. The token Portainer receives is only issued after the IdP has completed its full authentication flow — including any MFA, device compliance checks, and conditional access policies. The organization's existing MFA policy is therefore automatically applied to Portainer access without additional configuration.

Session Management

Upon successful authentication, Portainer issues a **JWT signed using HMAC-SHA256**. Two distinct signing secrets are used:

- **Session JWT secret:** Generated randomly at startup, held in memory only. All active sessions are invalidated when Portainer restarts.
- **kubeconfig JWT secret:** Generated once and persisted to the database. Allows kubeconfig tokens to survive server restarts, which is necessary for `kubectl` workflows.

The default session timeout is 8 hours, configurable by administrators. JWTs are transmitted over HTTPS only.

SECTION 9: TEAM SYNCHRONIZATION FROM UPSTREAM GROUP MEMBERSHIP

Overview

Portainer Business Edition implements **automatic team synchronization** from upstream identity provider group memberships, enforced at every login. This prevents access drift — the accumulation of permissions that no longer reflect a user's current role.

How Team Sync Works

LDAP Group Synchronization (at every login)

1. User authenticates via LDAP.
2. Portainer queries configured group base DNs for the user's current memberships.
3. Portainer computes the delta against current Portainer team memberships.
4. User is **added** to teams corresponding to newly-present LDAP groups.
5. User is **removed** from teams corresponding to LDAP groups they have left.
6. The user's effective permissions reflect this state before their session token is issued.

If a user has been removed from an LDAP group, their Portainer team membership is updated at their next login with no administrative action required in Portainer. Portainer also supports automatic admin role assignment and removal based on AD admin group membership.

OIDC/SAML Group Claim Mapping

Group membership is extracted from identity token claims at login. Portainer maps claim values to Portainer teams via configurable regex mappings, supporting flexible matching between IdP group names and Portainer team identifiers.

Guaranteeing Synchronization

Team sync is **synchronous with authentication —** there is no lag between a group membership change in the IdP and its reflection in Portainer beyond the user's next login.

For immediate access revocation:

1. Remove the user from the upstream IdP group (prevents re-granting at next login)
2. Disable or delete the user's IdP account (prevents new tokens from being issued)
3. Disable or delete the user in Portainer (immediately invalidates all active tokens — Section 10)

Auto-Provisioning

When a user authenticates via SSO for the first time, a Portainer user account is automatically created and team memberships populated from IdP group claims. No pre-provisioning is required.

SECTION 10: USER TERMINATION – TOKEN INVALIDATION AND KUBECONFIG REVOCATION

Overview

Portainer addresses the persistent credential problem through a centrally-controlled token lifecycle that enables immediate, comprehensive revocation of all access upon user termination.

The TokenIssueAt Revocation Mechanism

Every JWT validation checks `user.TokenIssueAt` against the token's IssuedAt claim. Any token issued before `TokenIssueAt` is rejected regardless of its cryptographic validity or expiry time. `TokenIssueAt` is advanced when a user's password or role changes, immediately invalidating all previously issued tokens for that user.

What Happens on User Deletion

When a user account is deleted in Portainer:

1. **Kubernetes service account removal:** Portainer removes the user's Kubernetes RBAC bindings across all Kubernetes environments.
2. **Access policy cleanup:** Portainer-level access policies are removed.
3. **User record deletion:** Subsequent JWT validation attempts a database read for the user, which fails, causing all tokens to be rejected.
4. **Team membership cleanup:** All team memberships are deleted.
5. **API key deletion:** All persisted API keys are explicitly enumerated and deleted.
6. **Git credential deletion:** All persisted git credentials are deleted.

Portainer-Issued kubeconfig Architecture

Portainer Business Edition can issue `kubeconfig` files enabling `kubectl` access. These kubeconfigs authenticate via Portainer's proxy layer, not directly to the Kubernetes API server. The bearer token is a Portainer JWT — its validity is entirely controlled by Portainer. Disabling or deleting the user immediately revokes all kubeconfig files previously issued to them, across all environments.

Model	Revocation
Native Kubernetes kubeconfig	Requires ServiceAccount deletion — complex, often missed
Portainer-issued kubeconfig	Immediate via user deletion or `TokenIssueAt` advancement

The kubeconfig expiry is configurable via `KubeconfigExpiry`. Setting it to `0` issues non-expiring tokens relying entirely on the revocation mechanism above.

Access Token Lifecycle Summary

Token Type	Signing Secret	Signing Secret	Revocation
Session JWT	Random, in-memory	No	User deletion; `TokenIssueAt` advancement
kubeconfig JWT	Persistent, database-stored	Yes	User deletion; `TokenIssueAt` advancement
API access token	Hashed in database	Yes	Explicit deletion; user deletion removes all

Recommended Offboarding Procedure

1. **Disable the user in the upstream IdP —** prevents new SSO tokens from being issued.

1. **Delete the user in Portainer —** immediately invalidates all session JWTs, kubeconfig tokens, and API keys; removes all Kubernetes RBAC bindings.

2. **Verify via audit log —** confirm no active API calls under that user identity following step 2.

3. **Review any direct Kubernetes access** granted outside Portainer **—** outside Portainer's scope but part of the broader offboarding checklist.

AUTHORIZATION AND ROLE-BASED ACCESS CONTROL (RBAC)

Fine-Grained RBAC

Portainer implements a multi-level RBAC model:

- **Platform-Level Roles –** Control access to global configuration, user management, and environment administration.
- **Environment-Level Roles –** Scope a user's or team's access to specific clusters or Docker endpoints.
- **Resource-Level Roles –** Restrict actions (deploy, update, scale, delete) on specific namespaces, stacks, or services.

Predefined roles include Administrator, Environment Administrator, Operator, Helpdesk, Standard User, and Read-Only User. These built-in roles cover the majority of enterprise separation-of-duties requirements across platform, operations, and application teams.

Kubernetes RBAC Integration

Portainer maps its internal RBAC to Kubernetes-native ServiceAccounts and ClusterRoles/Roles. If a user bypassed Portainer and contacted the Kubernetes API directly, they would be limited to the permissions of the ServiceAccount Portainer created for them — not cluster-admin.

NETWORK SECURITY

Secure Communications

- TLS 1.2 minimum enforced for all UI, API, and agent communications; TLS 1.3 supported
- HTTP/2 explicitly disabled to prevent protocol-level downgrade vectors
- mTLS available for Edge Agent connections (Section 3) and for syslog SIEM forwarding (Section 6)

Reduced Attack Surface

- **Portainer Agent eliminates** the need to expose Docker socket or Kubernetes API endpoints on network-accessible ports (legacy connection method)
- **Edge Agent model —** the recommended approach — removes the need for any inbound connectivity to remote environments and supports mTLS

Port Reference

Component	Default Port	Protocol	Purpose
Portainer Server UI/API	9443	HTTPS	User and API access
Portainer Edge Tunnel	8000	HTTP/WS	Edge Agent tunnel endpoint
Portainer Agent	9001	HTTPS	Standard Agent API (legacy)

The Edge tunnel endpoint (port 8000) should be placed behind a TLS-terminating reverse proxy in production deployments.

AUDIT LOGGING AND COMPLIANCE

Portainer Business Edition provides a comprehensive audit trail across all user-driven mutating actions. All audit events include user identity, source IP, timestamp, and action detail, supporting compliance with:

- **ISO 27001** – A.12.4 (Logging and Monitoring), A.9 (Access Control)
- **SOC 2 Type II** – CC6 (Logical and Physical Access), CC7 (System Operations)
- **PCI-DSS** – Requirement 7 (Restrict Access), Requirement 10 (Log and Monitor)
- **NIST SP 800-190** (Application Container Security Guide)
- **CIS Kubernetes Benchmark** – access and audit logging controls

POLICY ENFORCEMENT AND GUARDRAILS

Portainer Business Edition enables platform teams to define security guardrails:

- **Registry restrictions** – Deployments restricted to approved registries; non-permitted registries blocked at the Portainer layer
- **Resource quotas** – Namespace-level CPU and memory limits enforceable through Portainer
- **Environment-specific rules** – Production environments configurable with stricter access controls than development
- **Kubernetes Secret access restriction** – `RestrictSecrets` prevents standard users from reading secret values while allowing them to reference secrets in workload configurations

PORTAINER VS. NATIVE KUBERNETES ACCESS

Capability	Native Kubernetes Access	Portainer-Governed Access
User Access	Direct `kubectl` / API	Centralized via Portainer
Privilege Model	Often cluster-admin	Least-privilege by default
RBAC Management	Manual, YAML-driven	Centralized, role-based
Auditability	Fragmented (API server audit log)	Unified, structured, SIEM-ready
Self-Service	High risk	Guardrail-based
kubeconfig Revocation	Complex, multi-step, often missed	Immediate via user deletion
Credential Distribution	Manual, error-prone	Encrypted, just-in-time, scoped
Team Access Sync	Not native	Login-time enforcement from IdP
Sensitive Data Isolation	No protection at platform layer	Write-only model, RBAC-scoped
Auth Endpoint Protection	Not native to API server	IP-based rate limiting with ban

SHARED RESPONSIBILITY MODEL

Layer	Responsible Party	Portainer's Role
Physical infrastructure	Cloud provider / data center operator	None
Host OS and kernel	Customer	None — Portainer is a workload
Container runtime	Customer	Guidance; not a runtime replacement
Network perimeter	Customer	Edge Agent removes inbound requirements
Portainer application security	**Portainer**	**Full ownership**
Access control and RBAC	**Portainer + Customer**	**Enforcement; customer configures policy**
Credential encryption	**Portainer**	**AES-256-GCM at rest (when enabled); TLS in transit**
Audit and logging	**Portainer + Customer SIEM**	**Event generation and forwarding; customer retention**
Underlying Kubernetes security	Customer	Portainer augments, does not replace

PORTAINER.iO

CONCLUSION

Portainer Business Edition's security architecture is designed to provide enterprise-grade controls for organizations operating container estates at scale, across heterogeneous and geographically distributed infrastructure.

Key security properties verified against the Portainer codebase:

- **All sensitive data at rest** is protected by AES-256-GCM when database encryption is enabled – an opt-in feature that must be explicitly configured. Without it, credentials are stored in plaintext. User passwords and API key values are hashed regardless of database encryption status.
- **All data in transit** is encrypted using TLS 1.2+. Edge Agent deployments support mutual TLS providing cryptographic identity assurance at the transport layer – this is the recommended configuration for all production deployments.
- **Credentials are not returned in plaintext** through the API after initial entry.
- **Team access is always in sync** with the upstream identity provider at login time, with no lag and no manual intervention required.
- **User termination immediately revokes** all associated tokens, session JWTs, kubeconfig tokens, and API keys – collapsing the access revocation window to the time of the administrative action.
- **Audit events are structured, redacted, and SIEM-ready** – the redaction blacklist ensures the audit stream cannot itself become a source of credential leakage.

For organizations seeking to operationalize container security without requiring every user to be a Kubernetes expert, Portainer Business Edition provides the governance infrastructure to do so with confidence.

For further technical detail, Portainer's documentation is available at docs.portainer.io. For security disclosures, contact security@portainer.io.

PORTAINER.iO