Contents

1 Technical and Organizational Measures (TOMs) for PII Guardian API

Effective Date: August 25, 2025

1.0.1 1. Introduction

This document outlines the Technical and Organizational Measures (TOMs) implemented within the PII Guardian API to ensure a level of security appropriate to the risk, in accordance with Article 32 of the General Data Protection Regulation (GDPR). The software is designed as a self-hosted solution, empowering the data controller to maintain full custody of their data while leveraging the application's built-in security controls.

1.0.2 2. Data Segregation & Access Control

Measures to ensure that data is accessible only by authorized personnel and that data from different tenants is strictly isolated.

1.0.2.1 2.1. Multi-Tenant Data Isolation Description

The application enforces strict data segregation between tenants at the database query level. An automated filtering mechanism ensures that all data retrieval operations are scoped to the authenticated user's or API key's tenant ID, making it architecturally impossible for one tenant to access another's data.

Implementation Details

This is implemented via a SQLAlchemy event listener (`_add_tenant_filter`) that injects a `WHERE tenant_id = :id` clause into all `SELECT` statements for tenant-scoped models.

Evidence: `app/db/session.py:56-91`

1.0.2.2 2.2. Role-Based Access Control (RBAC) Description

The system implements a two-role model (`admin`, `viewer`) to enforce the principle of least privilege. Administrative functions—such as creating or deleting users, managing API keys, and defining custom rules—are restricted to users with the `admin` role.

Implementation Details

API endpoints are protected by FastAPI dependencies that verify the authenticated user's role. The `get_current_admin_user` dependency explicitly checks for the `admin` role before allowing access to sensitive operations.

Evidence: `app/api/deps.py:86-95`

1.0.3 3. Authentication & Credential Security

Measures to ensure the secure handling and storage of user and system credentials.

1.0.3.1 3.1. Secure Credential Storage Description

User passwords and API keys are never stored in plaintext. The system uses the `bcrypt` algorithm, a strong, adaptive, and salted hashing function, to protect all stored credentials against offline attacks.

Implementation Details

The `passlib` library is configured to use `bcrypt` as the default hashing scheme for both passwords and API keys.

Evidence: `app/core/security.py:15-25`

1.0.3.2 3.2. Timing Attack Mitigation Description

API key authentication is designed to be resistant to timing attacks. In cases where an API key prefix is not found in the database, the system performs a verification against a dummy hash to ensure that failed lookups take a similar amount of time as successful ones, preventing key enumeration.

Implementation Details

The `authenticate_and_get_tenant` function includes a specific branch to perform a constant-time comparison even when a key is not found.

Evidence: `app/crud/crud_api_key.py:108-120`

1.0.4 4. Application & Network Security

Measures to protect the application from common web and network-based vulnerabilities.

1.0.4.1 4.1. Server-Side Request Forgery (SSRF) Prevention Description

The webhook notification service includes a security check to prevent SSRF attacks. Before sending an outbound request, the service resolves the webhook's hostname and verifies that it does not point to a private, loopback, or reserved IP address.

Implementation Details

The `_is_private_ip` helper function performs a DNS lookup and checks the resolved IP against known private address ranges (e.g., `10.0.0.0/8`, `192.168.0.0/16`).

Evidence: `app/services/webhook_service.py:26-40`

1.0.4.2 4.2. Regular Expression Denial of Service (ReDoS) Mitigation Description

To protect against malicious or inefficient regular expressions, all pattern-matching operations within the PII scanning engine are subject to a strict, short timeout. This prevents a single request from consuming excessive CPU resources and impacting service availability.

Implementation Details

The `regex` library is used, and its `finditer` method is called with a configurable `timeout` parameter.

Evidence: `app/core/config.py:117-119`, `app/services/pii_service.py:204-206`

1.0.5 5. Data Handling & Processing Security

Measures to ensure the secure handling of data during processing, such as file uploads.

1.0.5.1 5.1. Secure File Handling Description

Filenames provided during file uploads are sanitized to prevent path traversal attacks. The system removes directory separators and other potentially malicious characters to ensure that files can only be written to the intended storage location.

Implementation Details

The `secure_filename` utility, adapted from Werkzeug, is used to sanitize all filenames before they are passed to the object storage service.

Evidence: `app/core/utils.py`, `app/api/v1/endpoints/scan.py:214-215`

1.0.6 6. Logging & Auditing

Measures to ensure that logs provide necessary operational insight without exposing sensitive information.

1.0.6.1 6.1. Automated Redaction of Sensitive Data in Logs Description

The application's structured logging system includes an automatic redaction processor. This processor scans all log event data for keys commonly associated with sensitive information (e.g., `password`, `token`, `api_key`, `secret`) and redacts their corresponding values before the log is written.

Implementation Details

A custom `structlog` processor (`redact_processor`) uses a regular expression (`SENSITIVE_KEYS_PATTERN`) to identify and redact sensitive fields recursively within the log's event dictionary.

Evidence: `app/core/logging_config.py:16-41`