CONTEXT PACK v1

Project: Brand Protection Monitor (PoC)

Date: 2026-02-01

Purpose: Normalize context into a unified, deduplicated specification, ready to generate a complete PRD without guesswork.

Sources

• Tech Challenge.pdf — Full Stack Engineering Challenge: Brand Protection Monitor (PoC).

• EjemploEndpointsExternos.json — Examples of CT endpoints: get-sth and get-entries.

0) One-liner & Elevator Pitch (Normalized)

One-liner: A full-stack web application for security/engineering teams that monitors a public Certificate Transparency (CT) Log and detects potential phishing domains or brand abuse by matching configurable keywords against certificate Common Name (CN) and Subject Alternative Names (SAN).

Why now / why us / key differentiator

• **Why now:** Domain abuse and certificate issuance can be detected early via public CT Logs.

• **Why us:** PoC focused on the minimum viable loop: ingest → process → highlight → export, with a mandatory stack (Go / React / PostgreSQL).

• **Key differentiator:** Direct CN/SAN matching against a persisted keyword list, with operational visibility into the monitoring cycle.

1) Objectives & Success Metrics (Draft)

Business objectives

• Detect potential brand abuse (phishing / suspicious domains) from certificates observed in CT.

• Provide a web interface to manage keywords and review matches.

• Demonstrate technical feasibility of the monitoring loop within a PoC scope (not processing the entire log in real time).

KPIs (no targets; normalized definitions)

KPI-ID	Name	Definition	Formula	Owner	Frequency	Data source
KPI-01	Certificates processed (per cycle)	Number of certificate entries evaluated in the most recent monitoring cycle.	count(entries_in_cycle)	Backend	Per cycle	CT Log / monitor run
KPI-02	Certificates with match (per cycle)	Number of certificates whose CN or SAN contains at least one keyword.	count(matches_in_cycle)	Backend	Per cycle	DB (matched_certificates)
KPI-03	Active keywords	Number of monitoring keywords currently configured.	count(keywords)	User	On-demand	DB (keywords)

KPI-ID	Name	Definition	Formula	Owner	Frequency	Data source
KPI-04	Monitor status	Indicates whether the monitor is active and processing.	state ∈ {idle, running, error}	Backend	Real-time	Runtime / DB state

2) Scope & Phasing

In-Scope (MVP)

• Monitor a specific CT Log: https://oak.ct.letsencrypt.org/2026h2.

• Go backend retrieves a fixed recent batch periodically (e.g., last 100 entries every minute).

• Parse certificates and identify whether CN or any SAN contains a monitored keyword.

• Persist keywords and matched certificates in PostgreSQL.

• React UI to: manage keywords; view monitoring status; list matches with highlighting; display last-cycle metrics (e.g., total processed).

• Export: backend endpoint + frontend button to download stored matches as CSV.

Phase 2 / Later

• Any expansion beyond a fixed recent batch (e.g., full incremental processing by tree_size).

• Support multiple CT logs or dynamically configurable logs.

• Proactive alerts (email/webhook) and additional classification.

Explicit Non-Goals

• Process the full CT log volume in real time (explicitly out of scope for the PoC).

• Store all certificates from the CT log (store only matches).

• Automated response/remediation (takedowns, blocks, etc.).

3) Users, Roles & Stakeholders

Role: Operator (UI user)

• Responsibilities: create/delete/list keywords; review matches; export CSV.

• Primary workflows: (1) manage keywords → (2) observe status/metrics → (3) review match list → (4) export.

• Permission sensitivity: Medium (modifies detection signals and accesses results).

Role: Backend Service (Go)

• Responsibilities: CT consumption, certificate parsing, matching, persistence, REST APIs.

• Permission sensitivity: High (DB access + core logic).

4) Problem Statements & Use Cases

Top problems (ranked)

• P1: Late detection of suspicious domains issued with TLS certificates.

• P2: Manual CT log inspection is not operationally viable for an end user.

• P3: Lack of visibility into processing state (is the monitor active, and how much did it process?).

Use cases (grouped)

• Configuration: manage keywords.

• Monitoring: run CT analysis cycles.

• Analysis: identify CN/SAN matches.

• Reporting: export results.

5) Module Inventory (Normalized)

CT Log Consumer

• Purpose: Connect to a CT Log and retrieve recent entries via CT endpoints (get-sth and get-entries).

• Core capabilities: fetch STH; compute start/end range; retrieve batch; run periodic polling.

• Inputs/Outputs: input = CT log URL + batch size + schedule; output = raw entries for processing.

• Dependencies: public CT log (Let’s Encrypt Oak 2026h2).

• Documents: RFC 6962 (get-entries) (hinted).

Keyword Management

• Purpose: CRUD of monitored keywords from the UI.

• Core capabilities: add/remove/view keywords; persist in PostgreSQL.

• Dependencies: PostgreSQL; backend REST API.

Certificate Parser & Matcher

• Purpose: Parse certificate data to extract CN and SAN, and evaluate matches against keywords.

• Core capabilities: parse CN; parse SAN; substring matching; record which keyword matched.

• Dependencies: CT Log Consumer; DB; Go X.509 libraries (implicit).

Monitoring Dashboard (Frontend)

• Purpose: Provide configuration and visualization interface.

• Core capabilities: manage keywords; display matches with highlighting (domain, issuer, date, matched keyword); show monitor state and per-cycle metrics.

• Dependencies: backend REST API; React + TypeScript; Tailwind CSS.

Export (CSV)

• Purpose: Export stored matches.

• Core capabilities: backend endpoint to download CSV; UI button.

• Dependencies: backend API; DB.

6) Key User Flows (High-Level)

Flow F-01: Monitor cycle (polling)

• Trigger: scheduler/timer in backend (e.g., every minute).

• Steps: (1) GET /ct/v1/get-sth → (2) derive recent range (start/end) → (3) GET /ct/v1/get-entries?start=...&end;=... → (4) parse CN/SAN → (5) compare with keywords → (6) persist matches and cycle metrics.

• Outcome: matches stored and visible in dashboard.

Happy path & failure modes

• Happy path: CT reachable; parsing OK; DB OK; UI shows new matches.

• Failure mode 1: CT log unavailable → mark error in status/metrics; retry next cycle.

• Failure mode 2: certificate parse error → skip entry and record error metric (exact behavior not specified).

• Failure mode 3: DB unavailable → cycle fails; no persistence; UI shows error/unhealthy monitor.

Flow F-02: Manage keywords

• Trigger: UI user action (add/remove).

• Steps: UI → REST API → DB → UI refreshes list.

• Outcome: keywords persisted and used by the monitor.

Flow F-03: Export matches to CSV

• Trigger: Export button in UI.

• Steps: UI calls backend endpoint → backend queries matches → generates CSV → download response.

• Outcome: CSV containing "all currently stored matched certificates".

7) Business Rules Catalog (BR-xxx)

BR-ID	Rule	Applies to	Inputs	Decision / output	Edge cases / gaps
BR-001	Monitored keywords must be persisted in PostgreSQL.	Keyword Mgmt	keyword value	persisted keyword list	N/A
BR-002	Matching evaluates the Common Name (CN) and all Subject Alternative Names (SAN).	Matcher	CN + SANs + keywords	match/no match + matched keyword	Cardinality not specified
BR-003	Backend retrieves a fixed recent batch periodically and does not process the full CT log in real time.	CT Consumer	STH + batch params	entries processed per cycle	Range derived from tree_size
BR-004	It is not required to store all certificates; only those that match at least one keyword.	Persistence	matching result	insert matched_certificates	Deduplication not specified
BR-005	UI must show that the monitor is active/processing and include metrics such as total processed in the last cycle.	Dashboard	monitor run	state + metrics	Exact metrics not specified
BR-006	Provide a backend endpoint and frontend button to export all stored matches to CSV.	Export	matched_certificates	downloadable CSV	CSV format not specified

8) States & Statuses (State Machines)

Entity: Monitor

• States: idle, running, error.

• Transitions: idle → running; running → idle; running → error; error → running (implicit).

• Audit/log: UI shows state and per-cycle metrics; logging details are not specified.

9) Data & Entities (High-Level)

Keyword

• Identifier: keyword_id (not specified).

• Key fields: value; created_at (implicit).

• Source of truth: PostgreSQL.

• Quality: duplicates/case rules not specified.

MatchedCertificate

• Identifier: match_id (not specified).

• Key fields (UI): domain, issuer, date, matched_keyword.

• Source of truth: PostgreSQL.

MonitorRun / MonitorState

• Key fields: processed_count_last_cycle; timestamp; state.

• Source of truth: backend runtime or DB (not specified).

10) Integrations & External Systems

CT Log (Let’s Encrypt Oak 2026h2)

• Purpose: certificate source.

• Data in: get-sth; get-entries (leaf_input, extra_data).

• Auth: none.

• Failure handling: not detailed; UI must reflect status.

```
GET https://oak.ct.letsencrypt.org/2026h2/ct/v1/get-sth
GET https://oak.ct.letsencrypt.org/2026h2/ct/v1/get-entries?start=...&end;=...
```

11) Non-Functional Requirements (NFRs)

• Security: no authentication (PoC) (not specified further).

• Performance: fixed recent batch (e.g., 100) periodically.

• Observability: state + total processed per cycle (minimum).

12) Risks, Constraints & Assumptions

Risks

• CT log volume/rate can vary.

• Certificate parsing can fail.

• Duplication between cycles if state is not tracked.

Constraints

• Mandatory stack: Go, PostgreSQL, React + TypeScript, Tailwind, REST.

• PoC: no full-log processing; only matches persisted.

Assumptions

• A-01 (Medium): case-insensitive matching.

• A-02 (High): single user (no auth).

• A-03 (Medium): possible multi-match per certificate (not specified).

13) Open Questions / Ambiguities to Resolve

ID	Question	Why it matters	Proposed options
OQ-01	Is matching case-sensitive or case-insensitive?	Impacts false negatives/positives and UX.	a) case-insensitive; b) configurable
OQ-02	How should matches be deduplicated across cycles?	Reduces noise in dashboard/CSV; defines the unique key.	a) fingerprint/hash; b) (domain, issuer, date, keyword); c) allow duplicates
OQ-03	What is the exact CSV format?	Downstream compatibility.	Minimum: domain, issuer, date, matched_keyword
OQ-04	How should monitor state be persisted?	Avoids reprocessing and improves metrics.	a) monitor_state table; b) runtime memory

PRD Generation Hand-off

Recommended PRD outline (module-based)

• 1. Overview & Context

• 2. Goals, Non-Goals, Success Metrics (KPIs)

• 3. Users & Permissions

• 4. System Architecture (Frontend / Backend / DB)

• 5. Data Model & Storage (PostgreSQL)

• 6. Backend: CT consumption, parsing, matching, persistence

• 7. Frontend: keyword management, dashboard, export

• 8. REST API Specification

• 9. Business Rules (BR-xxx) & State Machines

• 10. NFRs, Observability, Error Handling

• 11. Risks, Limitations, Rollout & Future Phases

Suggested story ID naming convention

• HU-xxx, BR-xxx, KPI-xxx

Minimal missing inputs to unlock a 60+ page PRD

• Matching decision (case sensitivity; substring vs other).

• Match cardinality and persistence rules.

- Deduplication strategy and unique key.
- Exact CSV format.
- Monitor state model (DB vs memory) and additional metrics.