026-Koeder-Token-Dienst

IT-Sicherheitsaufgabe
Anforderungsniveau: Schwer

1 Ziel der Aufgabe

Ein Köder-Token ist ein Lockmittel: eine Bildadresse, die niemand im Normalbetrieb anfasst. Wird sie doch abgerufen, ist das ein Alarm. Sie bauen den Dienst dahinter – er erzeugt solche Token, nimmt Treffer entgegen und meldet sie weiter. Der anspruchsvolle Teil ist nicht die Zufallskennung, sondern die Zuverlässigkeit der Auslösekette und die Frage, wie Sie verhindern, dass der Dienst selbst zum Werkzeug eines Angreifers wird.

Veranschlagte Bearbeitungszeit: 6–8 Stunden

Geprüft werden:

- Backend in Rust (z. B. Axum oder Actix Web) oder Node mit TypeScript, PostgreSQL als Datenhaltung
- Unratbare Kennungen und ein Auslösepfad, der nichts verrät
- Zustellung im Hintergrund mit Wiederholung und signiertem Webhook
- Abwehr von Missbrauch: SSRF, Aufzählen von Token, Meldungsflut
- Tests, die die Abwehr belegen, nicht nur den guten Pfad

Bewusst *nicht* verlangt sind Mandantenfähigkeit, eine Benutzerverwaltung und eine Oberfläche. Der Umfang ist auf den Kern zugeschnitten: Token erzeugen, Zugriff registrieren, alarmieren.

2 Aufgabenbeschreibung

1. **Datenmodell**

   Legen Sie ein Schema mit versionierten Migrationen an (z. B. `sqlx migrate` oder `node-pg-migrate`). Drei Tabellen genügen:

   ```
   token(id, kennung, bezeichnung, webhook_ziel, geheimnis, aktiv, angelegt_am)
   ausloesung(id, token_id, zeit, quell_ip, user_agent, referrer)
   zustellung(id, token_id, zustand, versuche, naechster_versuch_um,
              letzter_fehler, idempotenzschluessel)
   ```

   Auf `token.kennung` liegt eine Eindeutigkeitsbedingung, die Fremdschlüssel sind gesetzt.

2. **Verwaltungs-API**

   Drei Endpunkte reichen:

   ```
   POST   /v1/token        anlegen -> Kennung + fertige Bildadresse
   GET    /v1/token        Liste (ohne Seitenaufteilung)
   DELETE /v1/token/{id}   löschen
   ```

Die Verwaltungs-API ist durch einen einzigen Schlüssel geschützt, der aus einer Umgebungsvariablen kommt und in `Authorization: Bearer <schluessel>` erwartet wird; der Vergleich läuft in konstanter Zeit. Eingaben werden validiert (`validator` bzw. `zod`); Fehler kommen einheitlich zurück – mit Fehlerschlüssel, Meldung und Vorgangskennung, aber ohne interne Details.

Die Kennung enthält mindestens 128 Bit aus einem kryptographisch sicheren Zufallsgenerator und ist nicht ableitbar aus Zeit oder laufender Nummer.

3. **Auslösepfad**

   `GET /t/{kennung}` liefert ein 1x1-Pixel als PNG, zur Einbettung in ein Dokument oder eine Seite. Dieser Pfad ist öffentlich erreichbar und dabei stumm:

   - Er antwortet für bekannte und unbekannte Kennungen identisch – gleicher Statuscode, gleicher Rumpf, gleiche Kopfzeilen – und taugt nicht als Orakel zum Aufzählen. Gelöschte Token verhalten sich wie unbekannte.
   - Er spiegelt nichts aus der Anfrage in die Antwort zurück und verarbeitet den Treffer nicht im Anfragepfad zu Ende: speichern, Zustellauftrag erzeugen, antworten – der Versand läuft im Hintergrund.

   Quell-IP, User-Agent, Referrer und Zeitpunkt werden festgehalten, auf 512 Zeichen gekappt und nirgends als HTML ausgegeben. Eine Adresse aus `X-Forwarded-For` gilt nur bei Anfragen von einem konfigurierten Proxy.

4. **Entprellen und Melden**

   Ein eingebetteter Pixel wird beim Öffnen eines Dokuments leicht zwanzigmal geladen. Fassen Sie Treffer desselben Tokens aus derselben Quelle innerhalb eines Fensters (Vorgabe: 15 Minuten) zu einer Meldung zusammen: gespeichert wird alles, gemeldet einer.

   Jede Meldung erzeugt einen Zustellauftrag auf das beim Token hinterlegte Webhook-Ziel. Ein Arbeiter holt fällige Aufträge, stellt zu und schreibt den Ausgang zurück. Fehlversuche werden mit wachsender Wartezeit wiederholt (1 min, 5, 25; danach `aufgegeben`) und tragen einen Idempotenzschlüssel. Zwei Arbeiter dürfen denselben Auftrag nicht doppelt zustellen (`FOR UPDATE SKIP LOCKED` oder Vergleichbares).

   Zugestellt wird ein `POST` mit JSON-Rumpf und zwei Kopfzeilen:

   ```
   X-Koeder-Zeit:      <unix-sekunden>
   X-Koeder-Signatur:  sha256=hex(hmac(geheimnis, zeit + "." + rumpf))
   ```

   Der Zeitstempel in der Signatur erlaubt es dem Empfänger, alte Meldungen abzulehnen. Das Geheimnis wird beim Anlegen des Tokens erzeugt und genau einmal ausgegeben.

5. **Den Dienst nicht missbrauchbar machen**

   Ein Webhook-Ziel ist eine vom Nutzer bestimmte Adresse, die Ihr Server aufruft – ohne Prüfung ein Zugang in Ihr internes Netz. Setzen Sie durch:

   - Nur `https`, nur Standardports, keine Zugangsdaten in der Adresse.
   - Der Name wird aufgelöst, *alle* erhaltenen Adressen werden geprüft: abgewiesen werden Loopback, private Bereiche, link-local einschließlich `169.254.169.254`, `0.0.0.0/8`, Multicast, die IPv6-Entsprechungen und IPv4-in-IPv6-Abbildungen.
   - Verbunden wird mit der geprüften Adresse, nicht erneut über den Namen – sonst kann zwischen Prüfung und Verbindung eine andere Antwort kommen.

- Weiterleitungen werden nicht verfolgt, Zeitüberschreitung nach wenigen Sekunden, Antwortgröße begrenzt. Die Prüfung greift bei jedem Zustellversuch, nicht nur beim Anlegen.

Dazu eine Grenze gegen Meldungsfluten: Ratenbegrenzung am Auslösepfad je Kennung (z. B. `governor` bzw. `rate-limiter-flexible`). Über der Grenze wird weiter gezählt, aber nicht mehr gemeldet – der Alarm darf nicht zum Lastmittel gegen den eigenen Kunden werden.

3 Anforderungen

- Rust mit Axum oder Actix Web und `sqlx` oder Diesel, oder Node mit TypeScript im `strict`-Modus; kein `unwrap` auf Nutzereingaben, kein `any`.
- Ein `docker compose` startet Datenbank und Dienst; die Konfiguration kommt aus Umgebungsvariablen, eine `.env.beispiel` liegt bei. Protokolliert wird strukturiert (`tracing` bzw. `pino`) mit Vorgangskennung, nie der Verwaltungsschlüssel und nie ein Token-Geheimnis.
- Bei Rust: **die Version in `Cargo.toml` wird nach SemVer mitgepflegt** – bei jeder funktionalen Änderung.
- **Kein Formatierer ohne Projektkonfiguration**: Liegt `rustfmt` oder Prettier bei, dann mit eingechecktem `rustfmt.toml` bzw. `.prettierrc` – keine Läufe, die unbeteiligte Dateien umschreiben.
- Keine echten Zugangsdaten, Kundennamen oder Schlüssel im Repository, auch nicht in Tests – erfundene Werte nutzen (`example.invalid`, `hunter2`, `AAAAAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEE`) und vor der Abgabe die Historie prüfen.
- Tests mit `cargo test` bzw. Vitest; Datenbank aus `testcontainers` oder Compose, ausgehende Aufrufe mit `wiremock` bzw. `nock` nachgebildet. Mindestens:
  - **Aufzählschutz**: unbekannte und bekannte Kennung liefern dieselbe Antwort – Statuscode, Rumpf und Kopfzeilen Byte für Byte gleich, ausgenommen die von Natur aus wechselnden (`Date`, Vorgangskennung).
  - **SSRF**: Tabellentest über abzuweisende Ziele – `127.0.0.1`, `::1`, `::ffff:127.0.0.1`, `10.0.0.5` und `169.254.169.254`, dazu ein Name, der auf eine private Adresse zeigt.
  - **Entprellung und Zustellung**: 20 Treffer in einer Minute ergeben eine Meldung, aber 20 gespeicherte Auslösungen; Wiederholung nach Fehler, Aufgabe nach der letzten Stufe, korrekte Signatur, keine Doppelzustellung bei zwei Arbeitern.

4 Abgabe

- Ein Git-Repository mit nachvollziehbarer Historie. `docker compose up` startet Datenbank und Dienst, die Migrationen laufen beim Start oder über einen dokumentierten Befehl.
- README mit Startanleitung, Datenmodell, dem Weg eines Treffers vom Aufruf bis zur Meldung, den Entscheidungen (Speicherung der Quell-IP, Entprellung, Wiederholung) und den bekannten Grenzen.
- Ein durchgehendes Beispiel, das ein Prüfer nachvollziehen kann:

```
curl -X POST localhost:8080/v1/token -H 'content-type: application/json' \
  -H 'authorization: Bearer hunter2' \
  -d '{"bezeichnung":"Vertragsordner Muster",
       "webhook_ziel":"https://alarm.example.invalid/koeder"}'
curl -s localhost:8080/t/A1B2C3D4E5F60718293A4B5C6D7E8F90 -o /dev/null
# danach: ein signierter POST auf dem Webhook-Ziel, sichtbar im Protokoll
#         des Arbeiters oder in einem lokalen Empfänger
```

- Eine kurze Liste der bedachten Angriffe und der Stellen im Code, an denen die Abwehr sitzt.

5 Bewertungskriterien

- **Zuverlässigkeit der Auslösekette** – Ein Treffer geht auch bei nicht erreichbarem Empfänger nicht verloren; keine Doppelmeldung, keine Meldungsflut. Hauptkriterium.
- **Nicht missbrauchbar** – Der Dienst taugt nicht als Sprungbrett ins interne Netz; die Zielprüfung greift zur Zustellzeit, nicht nur beim Anlegen. Der Auslösepfad verrät nicht, welche Kennungen es gibt.
- **Sorgfalt bei Geheimnissen** – Kennungen sind unratbar, Geheimnisse stehen nicht in Protokollen, das Repository enthält keine echt aussehenden Zugangsdaten.
- **Tests, Struktur, Dokumentation** – Geprüft wird, was im Betrieb schiefgeht: die tote Gegenstelle, die zweite Instanz des Arbeiters, die unbekannte Kennung. Kennungserzeugung, Zielprüfung, Entprellung und Zustellung sind ohne laufenden Server prüfbar, und das README erklärt Entscheidungen, nicht nur Befehle.