luca260601 / **fs26-dsy-threatwatch**

Code Issues Pull requests Agents Actions Projects Security Insights Sett

Watch 0

MIT license

0 stars **0** forks **0** watching Branches Activity
Tags

Private repository

1 Branch **0** Tags Go to file t Go to file Add file Code

luca260601 final version 8dc11ce · 2 minutes ago

backend	final version	2 minutes ago
demos	final version	2 minutes ago
docs	bug fix	last week
frontend	fix	3 days ago
loadtest	final version	2 minutes ago
nginx	bug fix	last week
.env.example	fix	3 days ago
.gitignore	Login und JWT Auth, Docker Setup, ...	last month
LICENSE	Initial commit	last month
README.md	bug fix	last week
docker-compose.yml	final version	2 minutes ago

README MIT license

ThreatWatch

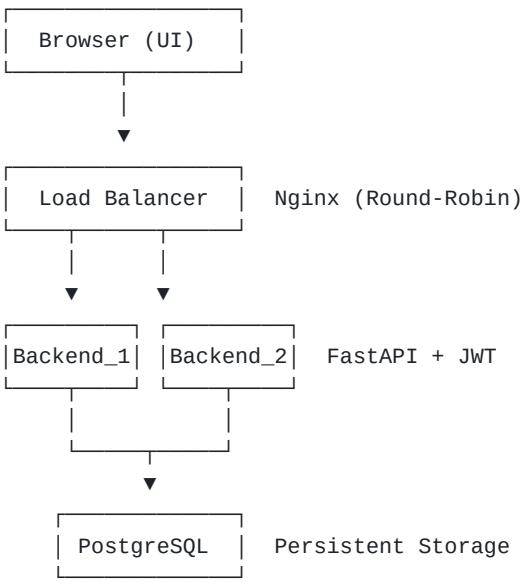
> **FS26 – Distributed Systems (DSy) Challenge Task**
> Mini-SIEM Security Dashboard mit Event Collection, Rule Engine, Alerting und IP-Blocking

Projektidee

ThreatWatch ist ein Security-Dashboard (Mini-SIEM), das Security-Events sammelt, mit Regeln auswertet und automatisch reagiert:

- **Event Collection** – Sammelt Security-Events (failed logins, port scans, SQL injection, DDoS, u.v.m.)
- **Rule Engine** – Wertet Events mit Window/Threshold Rules aus (z.B. "≥10 failed logins in 60s pro IP")
- **Alerting** – Erzeugt Alerts bei Regelverstössen, manuelles Resolven im Dashboard
- **Active Response** – Blockiert verdächtige IPs automatisch (TTL 30 Min), Events von blockierten IPs werden abgelehnt
- **Dashboard** – Web-UI zur Überwachung: Events, Alerts, Blocked IPs, Detection Rules

Architektur



Distributed System: 2 Backend-Instanzen hinter Nginx Load Balancer, gemeinsame PostgreSQL-DB. Fällt ein Backend aus, übernimmt das andere (Failover).

Quick Start

```
# 1. Repository klonen
git clone <repo-url>
cd fs26-dsy-threatwatch

# 2. Alles starten (ein Befehl)
docker compose up --build

# 3. Browser öffnen
open http://localhost
```

Login: `admin@threatwatch.local` / `Admin123!`

Stoppen

```
docker compose down       # Container stoppen
docker compose down -v    # + Datenbank löschen (Reset)
```

Requirements (Challenge Task)

Anforderung	Umsetzung
Load Balancing + Multiple Instances	Nginx Round-Robin → backend_1 / backend_2
Failover	Backend fällt aus → anderes übernimmt
Containerized	Docker + docker compose
Simple Frontend	HTML/JS Dashboard
JWT Authentication	Access Token (15 Min) + Refresh Token (7 Tage), OWASP-konform
Persistent Storage	PostgreSQL mit Docker Volume
One-command Setup	`docker compose up --build`
Load Test	k6 Scripts (loadtest/)

API Endpoints

Method	Endpoint	Auth	Beschreibung
POST	`/api/auth/register`	-	Benutzer registrieren
POST	`/api/auth/login`	-	Login → Access Token + Refresh Token
POST	`/api/auth/refresh`	-	Neues Token-Paar (Refresh Token Rotation)
POST	`/api/auth/logout`	-	Refresh Token serverseitig invalidieren
POST	`/api/events/`	JWT	Event erstellen (triggert Rule Engine)
GET	`/api/events/`	JWT	Events auflisten (Filter: type, severity, ip)
GET	`/api/alerts/`	JWT	Alerts auflisten (Filter: status)
POST	`/api/alerts/{id}/resolve`	JWT	Alert als resolved markieren
GET	`/api/blocked-ips/`	JWT	Blockierte IPs anzeigen
DELETE	`/api/blocked-ips/{ip}`	JWT	IP manuell freigeben
GET	`/api/rules/`	JWT	Detection Rules anzeigen
GET	`/api/health`	-	Health Check (DB-Verbindung)
GET	`/api/instance`	-	Welches Backend antwortet

Detection Rules

Regel	Event-Typ	Threshold	Zeitfenster	Aktion
Brute-Force Detection	`login_failed`	10	60s	Alert + IP Block
Port Scan Detection	`port_scan`	15	30s	Alert + IP Block
DDoS / Request Flood	`request_flood`	50	10s	Alert + IP Block
SQL Injection Attempt	`sql_injection`	3	60s	Alert + IP Block
404 Scanner Detection	`not_found_scan`	20	30s	Nur Alert
Unauthorized Access	`unauthorized_access`	5	60s	Alert + IP Block

Demo Scripts

Im `demos/` Ordner liegen einzelne Scripts für jede Attack-Simulation:

```
bash demos/demo_bruteforce.sh      # Brute-Force Attacke
bash demos/demo_portscan.sh        # Port Scan
bash demos/demo_ddos.sh            # DDoS / Request Flood
bash demos/demo_sqli.sh            # SQL Injection
bash demos/demo_404scan.sh         # 404 Scanner
bash demos/demo_unauthorized.sh    # Unauthorized Access
bash demos/simulate_all.sh         # Alle 6 auf einmal
```

Load Test (k6)

```
# Load Test (Ramp-up 10→20 User, 90s)
k6 run loadtest/load_test.js

# Failover Test
k6 run loadtest/failover_test.js
# Während der Test läuft, in einem zweiten Terminal:
docker compose stop backend_1
# Danach wieder starten:
docker compose start backend_1
```

Datenbank

Tabelle	Zweck
`users`	Login/Registrierung (email, password_hash, bcrypt)
`events`	Security-Events (ts, type, ip, severity, endpoint, message)

Tabelle	Zweck
`rules`	Detection-Regeln (event_type, window, threshold, group_by, action)
`alerts`	Ausgelöste Alerts (rule_id, key, status, event_count)
`blocked_ips`	Blockierte IPs mit TTL (ip, reason, blocked_until)
`refresh_tokens`	Refresh Tokens (token, user_id, expires_at, revoked)

DB inspizieren:

```
docker compose exec db psql -U threatwatch -c "SELECT * FROM users;"
docker compose exec db psql -U threatwatch -c "SELECT * FROM rules;"
docker compose exec db psql -U threatwatch -c "SELECT * FROM alerts;"
docker compose exec db psql -U threatwatch -c "SELECT * FROM blocked_ips;"
```

Tech Stack

Komponente	Technologie
Load Balancer	Nginx
Backend (2x)	FastAPI (Python)
Datenbank	PostgreSQL 16
ORM	SQLAlchemy + Alembic
Auth	JWT Access + Refresh Token (OWASP), python-jose, bcrypt
Frontend	HTML / CSS / JavaScript
Container	Docker + docker compose
Load Test	k6

Projektstruktur

```
fs26-dsy-threatwatch/
├── backend/
│   ├── app/
│   │   ├── api/            # FastAPI Router (auth, events, alerts, blocked_ips, rules,
system)
│   │   ├── core/           # Config, Security, Dependencies
│   │   ├── models/         # SQLAlchemy Models (User, Event, Rule, Alert, BlockedIP,
RefreshToken)
│   │   ├── schemas/        # Pydantic Schemas
│   │   ├── services/       # Rule Engine
│   │   ├── database.py     # DB Connection
│   │   └── main.py         # FastAPI App
│   ├── alembic/            # DB Migrations
│   ├── seed.py             # Admin User + Detection Rules
│   ├── entrypoint.sh       # Container Startup
│   ├── Dockerfile
│   └── requirements.txt
├── frontend/
│   └── index.html          # Dashboard SPA
├── nginx/
│   └── nginx.conf          # Load Balancer Config
├── loadtest/
│   ├── load_test.js        # k6 Load Test
│   └── failover_test.js    # k6 Failover Test
├── demos/                  # Attack Simulation Scripts
├── docs/
│   └── PLAN.md             # Projektplan
├── docker-compose.yml
└── README.md
```

Rele

No rel
Create

Packages

No packages published
Publish your first package

Author

Luca – FS26 DSy Challenge Task
OST – Ostschweizer Fachhochschule

Contributors 1

luca260601 Mythic

Languages

Python 50.9% **HTML** 36.2% **Shell** 6.6% **JavaScript** 5.1% **Other** 1.2%

Suggested workflows

Based on your tech stack

Python application

Create and test a Python application.

Configure

Jekyll using Docker image

Package a Jekyll site using the jekyll/builder Docker image.

Configure

Django

Build and Test a Django Project

Configure

More workflows

Dismiss suggestions