Assignment 02.02

Integrating OpenTelemetry & Security in eShop

1. Objective

1. **Implement OpenTelemetry tracing** on a single feature or use-case (end-to-end).
2. **Mask or exclude sensitive data** (e.g., email, payment details) from telemetry and logs.
3. **Set up a basic Grafana dashboard** to visualize the traces and metrics.
4. (Optional Extras) **Explore data encryption and compliance** in the database layer and introduce **column masking** for sensitive data.

2. Project Overview

eShop is a modern e-commerce system built using ASP.NET Core and microservices. The official repo can be found here:

- dotnet/eShop GitHub Repository

Key aspects of eShop include:

- **Microservices** split by bounded contexts (e.g., Ordering, Catalog, Basket).
- **Event-driven communication** using message brokers.
- **Containerization** with Docker.
- Multiple **frontend options** (Blazor, MVC, Angular, MAUI).
- Uses **IdentityServer** for authentication (though this project scope doesn't extend into deeper Auth improvements).

Students will **fork** this repository and focus on adding **OpenTelemetry** instrumentation for just **one** selected feature or user flow, rather than instrumenting the entire system.

3. Student Tasks

3.1 Integrate OpenTelemetry Tracing (Single Feature)

1. **Pick a Feature**

- Identify **one** user flow, such as “Place an Order” or “Add to Cart.”
- Ensure all calls (HTTP, database, messaging) within this feature are traced end-to-end.

2. **Instrument Services**

- Add OpenTelemetry libraries (e.g., OpenTelemetry.Instrumentation.AspNetCore and OpenTelemetry.Instrumentation.Http).
- Wrap key service calls with `Activity` or built-in instrumentation to collect spans.

3. **Data Scrubbing**

- **Mask or exclude sensitive data** like user emails and payment details from traces and logs.
- Validate your logs and traces to ensure no personal data is leaked.

4. **Exporters**

- Configure an exporter (e.g., **Jaeger** or **Prometheus**).
- This feeds your traces/metrics into tools that you can visualize in **Grafana**.

3.2 Observability Setup

1. **Metrics**

- Expose basic metrics (e.g., request duration, success/error rates).
- Use an OpenTelemetry collector or a built-in ASP.NET Core metrics endpoint.

2. **Grafana Dashboard**

- Install or configure **Grafana** to visualize traces or metrics.
- Demonstrate how to filter and analyze the single feature flow you instrumented.

3. **Simulate Load**

- Create a simple load test (using `curl`, **k6**, **Locust**, or **JMeter**) that triggers the selected feature repeatedly.
- Observe how traces and metrics appear in real time on Grafana.

3.3 (Optional) Data Security & Compliance

1. **Column Masking Challenge**

- **Partially or fully mask** columns containing sensitive data (e.g., credit cards, emails).
- Use either SQL Server Dynamic Data Masking (if using SQL Server) or custom masking logic in your data-access layer.
- Make sure your masked fields do not appear in logs/traces in plaintext.

4. Deliverables

1. **Source Code**

    - Forked eShop repository with instrumentation code for the single feature.
    - Updated configuration for OpenTelemetry (tracing + metrics) and any code for masking sensitive data.
    - (Optional) Data encryption or column masking logic.

2. **Documentation**

    - **README.md** with:
        - How to build and run the eShop environment with your changes.
        - Steps to configure and launch the OpenTelemetry collectors/exporters.
        - Instructions to set up or view the Grafana dashboard.
    - Basic **Architecture Diagram** or sequence diagram highlighting the traced feature.

3. **Grafana Dashboard**

    - A dashboard showing traces/metrics for the selected feature.
    - Screenshots or instructions for easy verification.

4. **Load Test Scripts**

    - Provide or link to a script (k6, JMeter, etc.) that generates activity on your selected feature.
    - This demonstrates how data flows end-to-end while being traced.

5. Evaluation Criteria

1. **Correctness & Completeness**

    - The chosen feature is properly traced end-to-end (frontend → microservice → database/message broker).

- Sensitive data (emails, payment info) is masked in logs/traces.

2. **Observability Setup**

   - Grafana dashboard effectively visualizes request flows and key metrics.
   - Students can demonstrate how to debug or analyze performance using these traces/metrics.

3. **Security & Compliance** (Optional)

   - If attempted: column masking or data encryption is implemented according to best practices.
   - No sensitive data leaks in logs, traces, or in the database output for non-privileged users.

4. **Documentation & Clarity**

   - README and architecture notes are clear and concise.
   - The single feature flow is explained, including why it was chosen and how it was instrumented.

6. Hints & Tips

- **OpenTelemetry Setup**

  - Start with a minimal approach: instrument HTTP client and server using built-in OTel packages.
  - Use `ActivitySource` for custom spans if you need to trace specific logic inside your code.

- **Data Scrubbing**

  - Check logs (Serilog, NLog, or `.NET ILogger`) for any personal data.
  - Implement your own log filter or OTel processor to remove PII from spans.

- **Grafana Dashboards**

  - If using Jaeger, you can embed Jaeger panels in Grafana, or use the Prometheus data source for metrics.
  - Make your panels user-friendly and label them to match your selected feature.

- **Load Testing**

    - Even a simple shell script that sends repeated HTTP requests can give you enough volume for demonstration.
    - Combine load generation with a break-it test approach (stop a service, observe logs/traces, etc.).

- **Column Masking**

    - If you choose to do this, experiment with partial masking: e.g., show only the first 3 letters of email.
    - Make sure you have a clear policy on who can see the real data (if implementing role-based unmasking).
    - SQLServer supports Dynamic Data Masking.
    - Postgres: https://www.bytebase.com/blog/postgres-data-masking/ or https://postgresql-anonymizer.readthedocs.io/en/latest/dynamic_masking/
    -
    - Question: can the DB Admins see the unmasked information?