ConSecure Interview Homework

Thank you for your interest in ConSecure.

You are expected to complete the following homework assignment and submit your work via a **Git repository**. Please ensure the repository is accessible and clearly structured.

Important Notes:

- Your submission should reflect your **own understanding**. Whether it's code, logic, design, or documentation—please ensure you are confident in what you've created.
- You may use reference materials or tools like **ChatGPT**, but it's essential that you **understand what you've implemented and why**. We're evaluating your **thinking and problem-solving**, not just the final output.

Next Steps:

The code and deliverables will be evaluated by our team. Based on the quality and clarity of your submission, selected candidates will be invited for the **next round of interview at our office/college**.

Deadline:

July 9, 11:30 PM IST

Please ensure you submit the assignment on time. Let us know if you have any questions.

SUBMIT the details here once done:
https://forms.gle/cPNKLMC3FE67c6uU9

Contact: arpithac@connectsecure.com

We look forward to reviewing your work!

Full Stack Engineering Take-Home Assignment: Threat Intelligence Dashboard

Date Issued: July 5, 2025

1. Introduction & Objective

Welcome! This take-home assignment is designed to assess your ability to design, build, and deploy a full stack web application. The goal is to create a "Threat Intelligence Dashboard" that allows for the browsing and analysis of cybersecurity threat data.

This project provides a baseline set of requirements that demonstrate core full stack competency. It also includes a series of advanced challenges that allow you to showcase the depth and breadth of your skills, from machine learning integration to modern DevOps practices. We expect this assignment to take a few hours, but there is no strict time limit. Please focus on creating a solution you are proud of.

2. The Scenario

You are a new engineer at a cutting-edge cybersecurity firm. Your first task is to prototype a new internal dashboard for the threat analysis team. The team currently works with a simple CSV file containing reports of various cyber threats. They need a web-based tool that will allow them to easily browse, search, and get quick statistics from this data. They would also love a feature that could provide a preliminary classification for new, unverified threat descriptions that come in from the field.

3. The Dataset

You will use the **"NLP Based Cyber Security Dataset"** available on Kaggle.

- **Link:** https://www.kaggle.com/datasets/hussainsheikh03/nlp-based-cyber-security-dataset

You will need to download the CSV file from this link to use as the data source for your application.

4. Technology Stack

You have the freedom to choose your preferred technologies. However, you must provide a brief justification for your chosen stack in your project's README.md.

- **Backend:** Node.js (Express/Koa), Python (Django/Flask), Go, Ruby on Rails, or Java (Spring Boot) are all excellent choices.
- **Frontend:** A modern SPA framework like React, Vue.js, or Angular is required.
- **Database:** PostgreSQL, MySQL, or MongoDB are recommended.

Part 1: The Core Application (Baseline Requirements)

A fully functional version of Part 1 is the minimum requirement for a successful submission. It demonstrates your competence in core full stack development.

A. Database & Data Ingestion

1. **Schema Design:** Design a database schema (e.g., SQL tables or a NoSQL collection structure) that appropriately represents the data in the provided CSV file.
2. **Ingestion Script:** Write a standalone script that parses the CSV file and populates your database with the data. This script should be runnable from the command line.

B. Backend API

Create a well-structured RESTful API with the following endpoints. Ensure your API responses use appropriate HTTP status codes and a consistent JSON format.

1. **GET /api/threats**
    - Fetches a paginated list of all threats from the database.
    - **Pagination:** Must support query parameters like page and limit (e.g., /api/threats?page=2&limit=20).
    - **Filtering:** Must support filtering by Threat Category via a query parameter (e.g., /api/threats?category=Phishing).
    - **Searching:** Must support a basic text search on the Cleaned Threat Description field (e.g., /api/threats?search=malicious%20payload).
2. **GET /api/threats/:id**
    - Fetches a single threat by its unique ID.
    - Should return a 404 Not Found if the ID does not exist.
3. **GET /api/threats/stats**
    - Returns key statistics about the dataset. At a minimum, this should include:
        - Total number of threats.
        - A count of threats for each Threat Category.
        - A count of threats for each Severity Score.

C. Frontend Application

Develop a clean, responsive, and user-friendly single-page application (SPA).

1. **Dashboard View:** Create a main view that displays the statistics from the /api/threats/stats endpoint in a visually appealing way (e.g., using cards or simple charts).
2. **Threats View:**
    - Display the list of threats fetched from /api/threats in a clear table or a list of cards.
    - Implement client-side controls for pagination that interact with your backend.
    - Include a search input and a filter dropdown (for Threat Category) that re-fetches data from the API based on user input.
    - Clicking on a specific threat should lead to a detail view (or expand to show more details) for that item.

Part 2: Advanced Challenges (High Skill Ceiling)

This section is your opportunity to impress. **You are not expected to complete all of these tasks.** Choose one or more that best align with your skills and interests.

A. Machine Learning Integration: Real-Time Threat Analysis

1. **Model Training:** Within your backend project, create a script to train a classification model.
    - Use the Cleaned Threat Description as the feature and Threat Category as the target.
    - A simple model like Logistic Regression with a TF-IDF vectorizer is sufficient.
    - Save the trained model and vectorizer as artifacts (e.g., pickle files, or using a dedicated format).
2. **Prediction Endpoint:** Create a new API endpoint: **POST /api/analyze**.
    - It should accept a JSON payload: { "description": "A new suspicious threat description..." }.
    - This endpoint should load your pre-trained model and return its prediction for the given description (e.g., { "predicted_category": "Ransomware" }).
3. **Frontend Integration:** Add an "Analysis" section to your frontend.
    - Include a text area where a user can paste a new threat description.
    - On submission, call your /api/analyze endpoint and display the returned prediction to the user.

B. Advanced Engineering & DevOps

1. **Containerization:** Provide a Dockerfile for your backend and frontend, and a docker-compose.yml file to orchestrate the entire application stack (backend,

frontend, database). The entire application should be launchable with a single docker-compose up command.

2. **Automated Testing:**
    - **Backend:** Write unit tests for your API endpoints and any critical business logic.
    - **Frontend:** Write component tests for a few key UI components.
3. **User Authentication:** Implement a simple and secure user authentication system (e.g., using JWTs). Protect one or more of your API endpoints (e.g., the /api/analyze endpoint) so they can only be accessed by logged-in users.
4. **Real-Time Updates with WebSockets:** When a user analyzes a new threat, use WebSockets to broadcast this event to all connected clients, perhaps updating a "Live Activity Feed" on the dashboard in real-time.

5. Evaluation Criteria

- **Core Competency (Meets Expectations):** A fully functional and bug-free implementation of all requirements in Part 1. The code is clean, and the README.md provides clear setup instructions.
- **Advanced Proficiency (Exceeds Expectations):** A solid Part 1 implementation plus one or two well-executed features from Part 2. The submission demonstrates strong knowledge of either ML integration or advanced engineering practices.
- **Outstanding (Strongly Exceeds Expectations):** A polished and robust solution that completes Part 1 and multiple advanced challenges from Part 2. The codebase is well-structured, thoroughly documented, tested, and easy to run (e.g., fully containerized). The system design is thoughtful and scalable.

6. Submission Guidelines

1. Please host your code in a private Git repository on GitHub or GitLab and invite us as collaborators.
2. Your submission must include a comprehensive **README.md** file at the root of your project. This file is critical and should contain:
    - A brief overview of your project and the features you implemented.
    - Your technology stack choice and a clear justification for it.
    - **Crucially, detailed, step-by-step instructions on how to set up, build, and run your application.** This includes database setup, data ingestion, and starting the servers. If you used Docker, provide the Docker-specific commands.
    - A description of how to run any tests you have written.

We are excited to see what you build. Good luck!