Behavioral Authentication System: Final Project Report

Author: Kamalesh S
Company: Elevate Labs
Date: July 27, 2025

---

1. Executive Summary

This report details the design, implementation, and functionality of the Behavioral Authentication System, a project developed during the final phase of the cybersecurity internship at Elevate Labs. The system offers a novel approach to security by continuously analyzing user behavior—specifically keystroke and mouse dynamics—to detect anomalies in real-time. It leverages a Python and machine learning backend with a JavaScript frontend to create dynamic user profiles and assign risk scores, providing a layer of security beyond traditional static credentials. This document covers the project's architecture, features, recent security enhancements, and future potential.

---

2. Project Overview

The primary goal of this project was to develop a proof-of-concept for a real-time behavioral authentication system. Unlike conventional authentication that validates a user only at the point of entry, this system continuously monitors user interactions to ensure the person using the session is the legitimate owner. By capturing and analyzing subtle patterns in how a user types and moves their mouse, the system can flag suspicious activity that might indicate an account takeover.

Key Objectives:

- **Real-time Data Collection:** Capture keystroke and mouse dynamics from a web interface.
- **Machine Learning Analysis:** Build and train ML models to distinguish between normal and anomalous user behavior.
- **Dynamic Risk Scoring:** Calculate and display a real-time risk score based on the analysis.
- **Alerting:** Notify the system of high-risk activity, suggesting further action.
- **Modularity:** Create a decoupled frontend and backend for scalability and maintenance.

---

3. System Architecture

The system is composed of two main components: a frontend for data collection and a backend for data analysis and machine learning.

- **Frontend (JavaScript):** A web-based dashboard serves as the user interface. A data collection script (`behavioral-collector.js`) runs in the background, capturing all keystroke and mouse events. This data is periodically bundled and sent to the backend via a WebSocket connection.
- **Backend (Python):** A WebSocket server (`websocket_server.py`) listens for incoming data from the frontend. The `BehavioralAnalyzer` class then processes this data, using the `FeatureExtractor` to create a feature vector. This vector is fed into pre-trained machine learning models to calculate a risk score, which is sent back to the frontend for display.

Technologies Used:

- **Backend:** Python, Flask, websockets, Scikit-learn (IsolationForest, RandomForestClassifier), TensorFlow/Keras (LSTM), NumPy, Pandas
- **Frontend:** JavaScript, Node.js, Express.js, HTML5, CSS3
- **Development:** Git, Visual Studio Code

---

4. Core Features

- **Keystroke Dynamics Analysis:** Captures metrics like key press duration (dwell time) and time between key presses (flight time).
- **Mouse Dynamics Analysis:** Tracks mouse movement patterns, speed, acceleration, and click behavior.
- **User-Specific Anomaly Detection:** Employs an `IsolationForest` model for each user to detect deviations from their established behavioral baseline.
- **Global Threat-Pattern Recognition:** Utilizes a `RandomForestClassifier` and an `LSTM` model trained on data from multiple users to identify known patterns of malicious behavior.
- **Real-time Dashboard:** A simple web interface displays the live risk score and any security alerts generated by the system.

---

5. Security Enhancements & Code Refinements

During the final review, several critical improvements were made to enhance the security, robustness, and configurability of the system.

1. **Implemented Token-Based Authentication:** The WebSocket server was secured to prevent unauthorized connections. Clients must now present a valid, secret token upon connecting, significantly reducing the risk of malicious data injection or unauthorized access to the analysis engine.
2. **Removed Hardcoded User Credentials:** The frontend no longer uses a static, hardcoded `userId` . The application now prompts the user for a username on startup, allowing for more flexible testing and paving the way for a proper multi-user system.
3. **Made Server Configuration Dynamic:** The WebSocket server's host and port are no longer hardcoded. They are now configured via environment variables, which is a best practice for deploying applications, allowing for easy changes without modifying the source code.

4. **Corrected Data Flow:** Ensured the `userId` captured from the user prompt is correctly passed to the data collector and included in the payload sent to the backend, making the entire analysis pipeline user-aware.

---

6. Setup and Usage

To run the project, follow these steps:

1. Backend Setup:

```
# Navigate to the backend directory
cd backend

# (Recommended) Activate Python virtual environment
source ../venv-3.9/bin/activate

# Set environment variables for the server
export WEBSOCKET_HOST="localhost"
export WEBSOCKET_PORT="8765"
export AUTH_TOKEN="your-secret-auth-token"

# Start the server
python websocket_server.py
```

2. Frontend Setup:

```
# In a new terminal, navigate to the frontend directory
cd frontend

# Install dependencies
npm install

# Start the server
npm start
```

3. Accessing the Application: Open a web browser and navigate to `http://localhost:3000`. You will be prompted for a username. After entering one, you can interact with the page to see the risk score change in real-time.

---

7. Challenges and Future Work

Challenges:

- **Initial Environment Setup:** Overcame issues related to Python module availability and frontend server port conflicts.
- **WebSocket Stability:** Debugged connection state issues to ensure reliable data transmission.
- **ML Model Training:** Implemented an initial dummy data training routine to ensure the system is functional without pre-existing user data.

Future Enhancements:

- **Database Integration:** Integrate a robust database like PostgreSQL to persist user profiles and behavioral data.
- **Robust User Management:** Implement a full user registration and login system.
- **Browser Extension:** Develop a browser extension to collect data across all websites, not just the dashboard page.
- **Advanced ML Models:** Research and implement more sophisticated deep learning models for higher accuracy.

---

8. Conclusion

The Behavioral Authentication System successfully demonstrates a viable, modern approach to enhancing application security. By moving beyond static credentials and focusing on the dynamic, unique behaviors of a user, the system provides a powerful, continuous layer of protection. The recent security and code-quality improvements have made the proof-of-concept more robust and secure, laying a strong foundation for future development into a production-ready security solution.