**Chapter 4 Design and Implementation**

This chapter delves into the design and implementation of the credit card fraud detection system using keystroke dynamics. It provides a comprehensive overview of the system architecture, detailing the client-side HTML pages, data processing functions, transaction processing, similarity calculation, and the weighted score. The chapter is divided into two main sections: the design and system architecture, followed by the implementation of the design.

**4.1 Design and System Architecture**

**4.1.1 System Overview**

The proposed system is designed as a client-server architecture, comprising three main components: the client-side JavaScript application, the server-side Flask application and the database. This architecture was chosen due to its scalability, flexibility, and support for efficient communication between the client and server components. Figure 4.1 illustrates the system's architecture.

[Figure 4.1 - System Architecture Diagram]

**4.1.2 Client-Side JavaScript Application**

The client-side application employs HTML, CSS, and JavaScript to create an intuitive user interface that consists of three main HTML pages: 'enroll.html', 'transaction.html', and 'verify.2fa.html'. Each page serves a specific function:

1. enroll.html: This page allows users to enter their personal information, credit card details, and create a typing profile during the enrollment phase. It captures typing data, such as key press and key release times, dwell time, digraph latencies, typing speed, and typing errors, which are then stored in a secure database to form a reference profile for each user.
2. transaction.html: This page is used for users to input credit card information and transaction amounts. Additionally, users complete a typing test, and the system captures their typing patterns similar to the enrollment phase. The collected data is used to assess the authenticity of the transaction.
3. verify.2fa.html: This page is displayed when the system detects a potentially fraudulent transaction based on the typing patterns. It prompts the user to perform additional verification steps, such as entering a one-time password or answering security questions.

The client-side application is designed with a modular architecture, consisting of several distinct modules that work together to capture and transmit user typing data:

• Data Capturing Module: This module employs event listeners to monitor user typing during enrollment and transaction phases.

• Data Serialization Module: This module converts the captured typing data into a JSON format using JavaScript's JSON.stringify() method to facilitate efficient data transmission.

• Data Transmission Module: This module securely sends serialized JSON data to the server-side Flask application via an HTTP POST request using XMLHttpRequest

**4.1.3 Server-Side Flask Application**

The server-side Flask application, developed with Python and the Flask web framework, receives incoming HTTP POST requests from the client-side application and processes the typing data. The server-side application is designed with a modular architecture, consisting of several distinct modules that work together to preprocess, extract features, calculate similarity, and process transactions:

• Data Processing Module: This module preprocesses the typing data by cleaning, normalizing, and transforming it to ensure data quality and compatibility with subsequent steps.

• Feature Extraction Module: This module extracts features from the preprocessed typing pattern data, such as dwell time, digraph latency, typing speed, and typing errors. These features form a feature vector used for similarity calculation and classification algorithms.

• Similarity Calculation Module: This module calculates the similarity between enrollment and transaction typing patterns by determining a weighted score. The score is then compared to a predetermined threshold to assess transaction authenticity.

• Transaction Processing Module: This module processes transactions based on the weighted score and threshold. If the 'is\_typing\_pattern\_different' function returns True, the transaction is flagged as potentially fraudulent, and the system may prompt the user for additional verification.

**4.2 Software and Hardware Requirements**

The following software and hardware components are required for the implementation of the system:

**Software Requirements:**

1. Python: The server-side Flask application is developed using the Python programming language. Flask is a lightweight web framework that enables the development of server-side applications.
2. JavaScript: The client-side scripting language is used to capture the user's typing data during the enrollment and transaction phases. JavaScript also enables seamless communication between the client-side and server-side components of the system.
3. HTML and CSS: These technologies are used to design the user interface for enrollment and transaction processes, ensuring an intuitive and user-friendly experience.
4. Microsoft SQL Server: This enterprise relational database management system is used to store and manage user profiles and historical data. Microsoft SQL Server is chosen for its scalability, security features, and comprehensive data management capabilities.

**Hardware Requirements:**

1. Web Server: A web server is required to host the Flask application and handle incoming requests from users. The server should have sufficient processing power, memory, and storage capacity to handle the expected workload and ensure smooth operation of the system.
2. Database Server: A separate server may be required to host the Microsoft SQL Server database, depending on the system's scalability requirements. This server should also have adequate processing power, memory, and storage capacity to handle the growing number of users and transactions.
3. Client Devices: Users will interact with the system using their personal computers or mobile devices. These devices should have compatible web browsers to support JavaScript-based data capture and communication with the server-side application.

**4.3 Data Collection**

Data collection occurs during enrollment and transaction phases. During enrollment, users complete a typing test while the system records their typing patterns. The collected data creates a reference profile for each user. In the transaction phase, users input credit card information and transaction amounts and complete another typing test, allowing the system to record their typing patterns.

Data collection is implemented using event listeners in the client-side JavaScript application, which monitors user typing during the typing test. These event listeners capture key press and key release events and store relevant data in arrays or objects for further processing.

**4.4 Data Processing**

The server-side Flask application processes the received typing data to ensure data quality and compatibility with subsequent steps. The data processing stage involves several subtasks, including data cleaning and normalization.

**4.5 Feature Extraction**

The feature extraction module in the server-side Flask application extracts relevant features from the preprocessed typing pattern data. These extracted features are combined to create a feature vector representing each user's typing pattern, which serves as input for similarity calculation and classification algorithms.

**4.6 Similarity Calculation**

The similarity calculation module computes the similarity between enrollment and transaction typing patterns by calculating Euclidean distances, computing the mean and standard deviation, and determining a weighted score. The weighted score is then compared to a predetermined threshold to assess the authenticity of a transaction.

1. Euclidean Distance: The module calculates the Euclidean distance for dwell time and digraph latencies to obtain their respective differences.
2. Mean and Standard Deviation: The module computes the mean and standard deviation to normalize the differences between typing pattern features, accounting for variability and minimizing the effect of outliers.
3. Weighted Score Calculation: The weighted score is calculated by multiplying the normalized differences by their respective weights and summing the results. Weights are assigned based on the importance of each feature in distinguishing genuine users from fraudulent ones.
4. Threshold Comparison: The weighted score is compared to a predetermined threshold to determine the authenticity of a transaction. If the score is above the threshold, the transaction is flagged as potentially fraudulent and may require additional verification steps. If the score is below the threshold, the transaction is considered successful and genuine.

**4.7 Transaction Processing**

The transaction processing module decides on the authenticity of a transaction based on the weighted score and threshold. If the **'is\_typing\_pattern\_different'** function returns True, the transaction is flagged as potentially fraudulent, and the system may prompt the user for additional verification, such as entering a one-time password or answering security questions. If the function returns False, the transaction is considered successful, and the user receives a confirmation message.

**4.8 Implementation**

The implementation of the keystroke dynamics-based authentication system follows the design and system architecture, focusing on creating efficient and reliable functions for data collection, processing, feature extraction, similarity calculation, and transaction processing.

**4.8.1 Client-Side Data Collection**

The JavaScript code embedded in the HTML document handles the client-side data collection. The script listens for **focus**, **keydown**, and **keyup** events on the typing test input field. The focus event marks the start time of typing, while the keydown and keyup events record the timestamp and the key code of each key press and release, respectively. The script also calculates digraph latencies, the time between consecutive key releases. The collected data is stored in hidden fields within the form and sent to the server when the form is submitted.

**4.8.2 Server-Side Data Processing**

Once the data is collected from the user's end, the server-side Python script initiates data processing. The data received from the client-side includes key press times, key release times, and digraph latencies. These are processed by three distinct functions: **‘process\_key\_press\_data’**, **‘process\_key\_release\_data’**, and **‘process\_digraph\_latencies’**.

The **‘process\_key\_press\_data’** function handles the key press times. It parses the stringified data received from the client-side into a usable format. As the key press times are already in a suitable form (i.e., a list of timestamped key presses), this function simply returns the data without applying any transformations.

In contrast, the **‘process\_key\_release\_data’** function transforms the stringified list of key release times into a more efficient data structure. It converts the data into a dictionary that maps each key code to its corresponding release time. This change in data structure is strategic as it allows for efficient lookup operations in the subsequent stages. Instead of iterating through a list to find the release time for a specific key, the system can directly access it using the key code in constant time.

Lastly, the ‘**process\_digraph\_latencies’** function is used to process the digraph latencies data, which represents the time taken between consecutive key releases. This function leverages Python's json library to decode the stringified JSON data received from the client-side, transforming it into a Python dictionary. As a result, the system can efficiently access the latency between any two keys, contributing to a more accurate and detailed user typing profile.

**4.8.3 Feature Extraction**

Feature Extraction is a fundamental step in our system and is where the keystroke dynamics, the unique aspects of a user's typing behavior, are identified and quantified.

The process begins with the ‘**calculate\_dwell\_time’** function. Dwell time is the duration between when a key is pressed and when it is released. In other words, it measures how long a key is held down. This is a notable feature as everyone tends to have a specific rhythm when typing, and the dwell time is an integral part of this rhythm. The **‘calculate\_dwell\_time’** function iterates through the processed key press and key release data, computing the dwell time for each key and storing this information.

Next, the **‘calculate\_typing\_speed’** function is used to evaluate the user's typing speed. This function considers the total duration of the typing test and the number of words typed during that period. By dividing the total number of words by the time taken (converted to minutes), it computes the typing speed in words per minute (WPM). Typing speed is a significant feature as individuals tend to maintain a relatively consistent speed, making it a useful indicator for user identification.

The final function involved in feature extraction is **‘calculate\_typing\_errors’**. This function focuses on the accuracy of the user's typing by comparing the typed text against the original text. The comparison uses the Levenshtein distance, a string metric for measuring the difference between two sequences. Essentially, it quantifies the number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other. This function provides an estimation of the number of typing errors, which can be a distinguishing factor in a user's typing pattern.

Together, these functions extract and quantify the unique elements of a user's typing behavior, forming a feature vector that represents the user's typing pattern. This feature vector is then used in subsequent steps for similarity calculation and decision-making on the authenticity of the transactions.

**4.8.4 Similarity Calculation**

This stage of the implementation focuses on establishing a similarity score between the current typing pattern and the user's registered typing pattern. The calculation of this similarity score involves multiple steps and functions, each contributing to the final score that will be used to assess the potential fraudulence of a transaction.

**4.8.4.1 Euclidean Distance Calculation**

At the core of the similarity calculation is **‘euclidean\_distance’** function, which calculates the Euclidean distance between two sets of data. This function takes two vectors as inputs, each vector representing a set of features of a typing pattern, and it calculates the 'distance' or dissimilarity between them.

In the context of our system, the Euclidean distance is calculated for each key feature, namely the dwell time and digraph latencies. By doing so, we obtain the differences in these features between the reference typing pattern (from the enrollment phase) and the current typing pattern (from the transaction phase).

**4.8.4.2 Normalization Using Mean and Standard Deviation**

After obtaining the raw differences for each feature, the next step is to normalize these differences. Normalization is crucial as it helps to put different features on a similar scale, which ensures that no single feature dominates the final similarity score due to its scale.

The **‘is\_typing\_pattern\_different’** function calculates the mean and standard deviation of the differences for each feature. It then subtracts the mean from each difference and divides the result by the standard deviation, effectively normalizing the differences. This process ensures that the influence of each feature on the final similarity score is proportional to its variability, minimizing the effect of outliers.

**4.8.4.3 Weighted Score Calculation**

Weighted Score Calculation is a critical step in the similarity calculation process. The purpose of this step is to provide a combined score that captures the dissimilarity between the current typing pattern and the registered pattern. In essence, the weighted score calculation process is a means of transforming the differences in typing pattern features into a single, comparable metric.

To achieve this, each normalized feature difference is multiplied by a pre-determined weight factor. These weight factors are not chosen arbitrarily; they are established based on the significance of each feature in distinguishing between genuine and fraudulent typing patterns. For instance, if dwell time is considered a more defining feature of a person's typing pattern than typing errors, it would be given a higher weight. These are the weights factors assigned for each feature:

1. Dwell time weight = 0.4
2. Digraph latencies weight = 0.4
3. Typing speed weight = 0.1
4. Typing error weight = 0.1

The **‘is\_typing\_pattern\_different’** function carries out the weighted score calculation. The function takes as input the normalized differences for each feature along with their corresponding weights. It then performs a multiplication operation, where each normalized difference is multiplied by its associated weight. The output of this operation is a set of weighted differences.

The final step in the weighted score calculation is the summation of these weighted differences. **‘The is\_typing\_pattern\_different’** function completes this by summing the weighted differences across all features. This sum is the weighted score - a composite measure representing the overall dissimilarity between the current typing pattern and the registered typing pattern.

**4.8.5 Transaction Processing**

The final stage of the implementation is transaction processing. The ‘**process\_transaction’** function calculates the typing speed, typing errors, dwell time, and digraph latencies for the current transaction. It then calls the ‘**is\_typing\_pattern\_different’** function to calculate the weighted score. If this score exceeds a predefined threshold, the function flags the transaction as potential fraud and returns an appropriate message. If the score is below the threshold, the function approves the transaction and returns a success message.

In conclusion, the implementation of the keystroke dynamics-based authentication system effectively brings the design and system architecture to life. The careful choice of technologies, the thoughtful design of functions, and the meticulous attention to detail throughout the implementation process culminate in a robust, efficient, and reliable system for user authentication. The system stands as a testament to the potential of keystroke dynamics as a reliable method for user authentication, providing a solid foundation for future research and development in this field.

**4.8.6** **User Interaction** **and validation**

The system interacts with the user via a web application interface, enabled by Flask, a micro web framework written in Python. The application is structured with three main routes: **/**, **/enroll**, and **/transaction**. Each route corresponds to a different part of the user journey.

**4.8.6.1 Home Page**

Upon accessing the system, users are automatically redirected to the enrollment page. This is facilitated by the root route (**/**), which serves as the default entry point to the application.

**4.8.6.2 Enrollment Page**

Here, the user is presented with an enrollment form, where they're required to input their personal information and complete a typing test. The form collects various details such as the user's name, email, credit card number, and phone number. Additionally, it captures crucial typing data that consists of key press and release times.

Once the user submits the form, the system validates the provided data. Next, it extracts the typing data and calculates various typing characteristics, including dwell time, typing speed, and typing errors. If a user with the same email already exists, an error message is displayed. Otherwise, the system creates a new user record with the provided details and calculated typing features, which is then added to the database. Upon successful enrollment, the user is automatically redirected to the transaction page.

**4.8.6.3 Transaction Page**

On this page, the user can perform a transaction. They're required to fill in a form with their credit card number, transaction amount, expiry date, and CVC.

When the user submits the transaction form, the system validates the entered data and checks if a user with the provided credit card number exists in the database. If the user doesn't exist, an error message is shown. If the user does exist, the system then processes the transaction. During the processing, it checks if the current typing pattern of the user matches their registered typing pattern. If the patterns match, the transaction is completed. If they don't match, the system triggers a two-factor authentication (2FA) protocol, where a one-time password (OTP) is sent to the user's registered phone number.

**4.8.6.4 2FA Verification Page**

If 2FA is triggered, the user is redirected to the 2FA verification page. Here, they're asked to input the OTP that was sent to their phone. If the entered OTP matches the one sent by the system, the transaction is confirmed and completed. If it doesn't, an error message is displayed, prompting the user to try again.