Final Report

Biometric Enhanced Login Detection Pipeline (Rule + ML Hybrid)

Student Name: Justin Cruz

Course: Introduction to Enterprise Computing (CS 4365)

Instructor: Dr. Calton Pu

Semester: Summer 2025

Table of Contents

1 Introduction

Login authentication systems are frequent targets of brute force and evasive attacks. Standard rule-based defenses like rate limits or account lockouts are easy to configure but often fail to catch stealthy intrusions or flag legitimate users. Machine learning-based defenses offer pattern recognition advantages, but they tend to overflag, harder to tune in, and require careful data handling to work in practice.

Systems that rely solely on rules tend to miss advanced or slow attacks, while ML-only systems, while powerful, often trigger excessive false positives, disrupt user experience and erode trust. A balance between detection coverage and real-world usability is lacking in most existing systems.

This project addresses that gap by designing a hybrid login detection pipeline that layers rule-based heuristics with a machine learning model trained on keystroke biometrics. Logins are flagged as suspicious if either the rule logic or the ML classifier detects anomalies, an OR-based decision structure designed for maximum attacker recall.

The rule-based system, tested on simulated login metadata, achieved high recall with relatively few false positives. The ML classifier, trained on fixed-text samples from the KeyRecs dataset, consistently achieved perfect attacker recall but at the cost of significantly more false positives. The hybrid pipeline preserved this high recall while reducing false positives compared to the ML-only model, striking a more balanced tradeoff. These results demonstrate that layered detection can improve overall robustness without sacrificing practicality.

2 Related Work

This project leverages the KeyRecs dataset [Dias et al., 2023], which was developed to support research in keystroke-based authentication systems. The dataset includes fixed-text and free-text samples collected from 99 participants along with timing-based diagraph features (e.g., press-press, press-release intervals). Fixed-text samples were particularly useful for this project's ML component, as they offer repeated, structured input for anomaly detection.

Keystroke dynamics as a behavioral biometric has been studied in both academic and commercial settings, often using classifiers such as Random Forests, SVMs, or neural networks. Most of this research focuses on user-specific authentication or impersonation detection. In contrast, this project applies keystroke analysis in a user-agnostic way, as part of a broader layered login detection system.

While rule-based login defenses remain common in production systems due to their simplicity and clarity, few real-world implementations combine them effectively with ML-based detection. This project explicitly explores integration, showing how logic and biometric analysis can complement each other in a modular pipeline.

3 System Overview

This project implements a layered login anomaly detection system that integrates rule-based logic and a machine learning (ML) classifier into a unified pipeline. The goal is to combine the precision and interpretability of rule heuristics with the behavioral depth of a keystroke-based ML model.

3.1 Architecture and Pipeline Flow

The system consists of 5 stages:

1. Keystroke Preprocessing: Fixed-text samples from the KeyRecs dataset are cleaned, labeled, and transformed into numerical timing vectors.
2. Login Simulation: Synthetic login sessions are generated, including both normal users and two classes of attackers (brute-force and evasive), each enriched with corresponding keystroke data.
3. Rule-Based Detection: Sessions are evaluated for suspicious patterns using heuristics like high attempt volume, unrealistic typing speed, or low success rates across varied usernames.
4. ML-Based Detection: A Random Forest classifier is trained on the preprocessed KeyRecs vectors to identify typing patterns that deviate from legitimate users.
5. Pipeline Integration: Both detectors operate independently, A session is flagged if either the rule logic or the ML model marks it as anomalous. This is implemented using an OR-based decision structure.

3.2 Component Roles

- load_keyrecs.py: Extracts, filters, and processes the fixed-text portion of the KeyRecs dataset. Labels known participants as normal (0) and others as attackers (1). Outputs a vectorized .csv for ML training.
- generate_logins.py: Simulates login sessions for normal, brute-force, and evasive users. Each session includes metadata (timestamp, username, outcome) and an embedded keystroke vector sampled from the preprocessed KeyRecs data.
- rule-based_detector.py: Applies a sequence of domain-informed heuristics to each session:
    - Brute-force: rapid attempts within a short window
    - Evasive: varied usernames with low success rate over time
    - Unrealistic typing behavior: fast average delay or high variance

- train_ml_classifier.py: Trains a Random Forest classifier on labeled keystroke vectors. The model is stored as a .joblib object for reuse in the pipeline.
- pipeline_detector.py: Loads simulated login sessions and runs both the rule logic and the ML model independently on each session. It merges the results with OR logic to produce a final flag and reason for each session.
- evaluation.py: Computes classification metrics and confusion matrices for the rule-only, ML-only, and hybrid systems. Also generates comparison bar plot.

3.3 Design Goals

- High Recall: The pipeline prioritizes attacker detection over minimizing false positives. The OR logic ensures that if either subsystem detects suspicious activity, the session is flagged.
- Modularity: Rules and ML operate independently, which allows for tuning or replacing each component without affecting the rest of the pipeline.
- Interpretability: Rule-based detections are accompanied by labeled reasons (e.g., “brute_force_pattern”), and ML detections can be inspected using standard feature importance tools.
- Practicality: The system avoids assumptions like user registration or prior user-specific training, enabling deployment in environments where user identity may not be stable or known in advance.

4 Data Sources and Generation

This system combines real biometric data and simulated login sessions to evaluate hybrid detection under realistic threat profiles. All login metadata was synthetically generated. Keystroke data was drawn from real-world biometric dataset to improve behavioral fidelity.

4.1 Keystroke Data (KeyRecs Dataset)

The ML component uses fixed-text samples from the KeyRecs dataset [Dias et al., 2023], which includes timing-based keystroke patterns from 99 users. Only the fixed-text subset was used for consistency across samples. Each sample was converted into a 41-dimensional timing vector based on inter-key delays (e.g., press-press and press-release intervals).

Labeling:

- Normal class (0): Drawn from known participants with multiple valid samples
- Attack class (1): Drawn from shuffled or alternate users not in the known group

This separation supports user-agnostic classification: the model learns to differentiate legitimate versus impersonator timing patterns, rather than memorizing specific users. The preprocessed vectors are saved as .csv for training and then passed to train_ml_classifier.py.

4.2 Login Session Simulation

Login events were generated using generate_logins.py, simulating three distinct user types:

User Type	Attempts/Session	Success Rate	Typing Source	Notes
Normal	1-3	~90%	Known user vector	Modeled after real login success behavior
Brute-Force	10-50	~1%	Unknown user vector	High volume, reused usernames, fast timing
Evasive	5-20	~3%	Unknown user vector	Spaced attempts, varied usernames, low frequency

Each login entry contains:

- Username
- Timestamp
- Login result (success/fail)
- Typing vector (biometric input)
- User type label (normal/ brute/ evasive)

The simulator uses hardcoded parameters reflecting success/failure rates from real-world login security studies.

4.3 Labeling

Binary labels were applied programmatically:

- Is_attack = 0: Normal user sessions (legitimate)
- Is_attack = 1: Brute-force and evasive attacker sessions

Each detector uses a different view:

- Rule-based: Emulates login metadata
- ML classifier: Evaluates typing behavior only
- Hybrid pipeline: Applies OR logic between both detectors

This strict separation between keystroke-based training and synthetic login generation helps preserve test integrity and avoid leakage or overfitting.

5 Feature Engineering & Preprocessing

This system uses two distinct feature domains: biometric keystroke timings for ML-based detection, and login session meta data for rule-based detection. Each domain is processed independently and used in isolation by its respective detector.

5.1 Keystroke Vector Construction (ML Model Input)

Raw data from the KeyRecs fixed-text dataset is filtered into two groups:

- Known participants (p001-p010) labeled as normal (0)
- All other participants labeled as attacker (1)

Only the keystroke timing columns were used as features (columns 4 through n-1). Each sample is represented as a fixed-length 41-dimensional vector of inter-key delay values. No normalization, rescaling, or dimensionality reduction was applied. The output is saved to keystroke_vectors.csv.

The resulting dataset is stratified into train/test sets and used to train a RandomForestClassifer using class balancing. No session metadata (e.g., username, timestamp) is passed to the ML model, relies solely on typing behavior.

5.2 Session Metadata (Rule-Based Logic)

Synthetic login sessions generated by generate_logins.py include:

- Username
- Timestamps
- Login success/failure
- Typing vector (for downstream ML use)

Rule-based logic derives features from this metadata:

- Success rate per session
- Username reuse or churn
- Time between attempts
- Typing speed (average delay)
- Variance in timing

These derived statistics are used to trigger detection rules like LOW_SUCCESS_RATE, BRUTE_FORCE_TIME_THRESHOLD, and HIGH_VARIANCE_THRESHOLD. Thresholds are configurable. In tuning evaluation of rule-based only, thresholds are bundled into three profiles: strict, moderate, and relaxed.

5.3 Labeling and Split Policy

- Keystroke data was labeled as attacker or normal based on participant ID and stored independently of session metadata.
- Login sessions were labeled programmatically as attack (1) or normal (0) based on simulated user type (brute/evasive vs normal)
- No cross-contamination occurred between training and evaluation datasets.

6 Implementation Details

This system was implemented as three independent modules: a rule-based detector, a machine learning classifier trained in biometric features, and a hybrid pipeline that combines both for final decision-making. Each component was built as a standalone script to ensure modularity, reproducibility, and ease of testing.

6.1 Rule-Based Detection

The rule-based logic is implemented in rule_based_detector.py. It operates entirely on login metadata, such as username usage, login success rate, time between attempts, and basic typing statistics like average delay and variance.

A fixed set of hardcoded thresholds is used for detection (e.g., LOW_SUCCESS_RATE, BRUTE_FORCE_TIME_THRESHOLD, HIGH_VRIANCE_THRESHOLD). These applied uniformly to all sessions in the main pipeline. While current rules are static and general-purpose, the architecture is extensible for future per-use profile tuning or rule personalization, as explored in the tuning notebook.

A separate notebook (Tuning_Showcase_Thresholds.ipynb) was created to explore alternative threshold profiles (strict, moderate, and relaxed) and compare their effects on detection precision, recall, and false positives. However, this tuning was only applied to the rule-based detector in isolation. The ML model and pipeline were not integrated into the tuning notebook due to time constraints.

Triggered rules are returned alongside session flags, with justification recorded for interpretability.

6.2 Machine Learning Classifier

The ML model is trained using train_ml_classifier.py. It consumes only the 41-dimensional keystroke vectors extracted from the KeyRecs dataset. A RandomForestClassifier is trained using balanced class weights to account for uneven attacker-to-user ratios.

All typing samples are treated independently; no login metadata is used in training. The model is serialized with joblib and reloaded during pipeline execution for inference.

Predictions are binary, attacker (1) or normal (0), based purely on typing behavior.

6.3 Hybrid Detection Pipeline

The full pipeline is implemented in pipeline_detector.py. It accepts a simulated login dataset where each session included both metadata and a keystroke vector.

For each session:

- Rule-based logic is applied first using static thresholds.
- The ML Classifier is run separately on the typing vector.
- An OR logic decision rule determines the final flag:

```
If rule_flag or ml_flag:

        session['final_flag'] = 1

else:

        session['final_flag'] = 0
```

The final output includes:

- Binary final_flag
- Source of detection (ml_only, rule_only, or both)

The pipeline design ensures reproducibility and clear attribution of each detection.

7 Evaluation and Results

To evaluate the detection performance of each subsystem (Rule-Based, ML-Based, and Hybrid), metrics such as precision, recall, and F1-score were calculated using confusion matrices generated from simulated login sessions. All sessions were labeled based on the attacker type: *normal, brute-force, or evasive*.

Note: Results vary slightly across runs due to the randomized nature of the session simulator. The results presented here were taken from the final notebook run and reflect the expected behavior under typical conditions.

7.1 Notebook Evaluation Results (Final Test Run)

Detector	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall
Rule-Based	90	80	10	0.529	0.900
ML-Based	3555	232	10	0.939	1.000
Hybrid	100	95	0	0.513	1.000

- Rule-Based: Achieved strong recall (0.90), but precision was modest due to 80 false positives.
- ML-Based: Achieved perfect recall and high precision, but only with a high volume of false positives (232).
- Hybrid: Maintained perfect recall (1.0) and reduced false positives compared to ML-only, yielding better balance (F1: 0.678)

The hybrid detector preserved ML's strength in attacker detection while reducing the number of false alerts. This demonstrates the intended effect of combining broad logic (rules) with fine-grained biometrics (keystrokes).

7.2 Metric Comparison and Confusion Matrices

The following figures compare the confusion matrices and performance metrics for each detector:

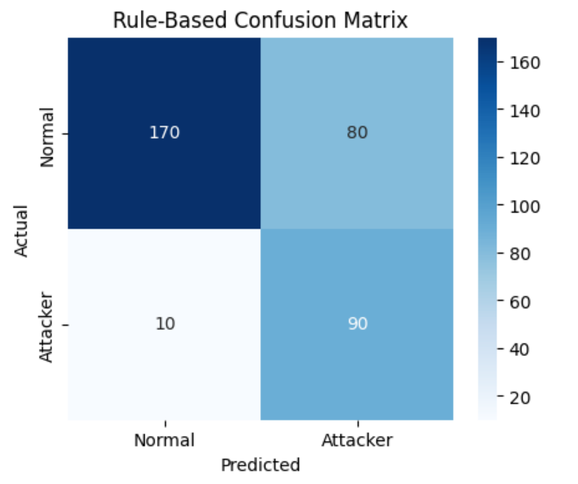

Figure 1: This matrix shows that the rule-based system correctly identified 90 of 100 attacker sessions but mistakenly flagged 80 normal sessions as suspicious. While recall is high (0.90), the precision is moderate due to the relatively high false positive count, reflecting the rule system's aggressive detection posture

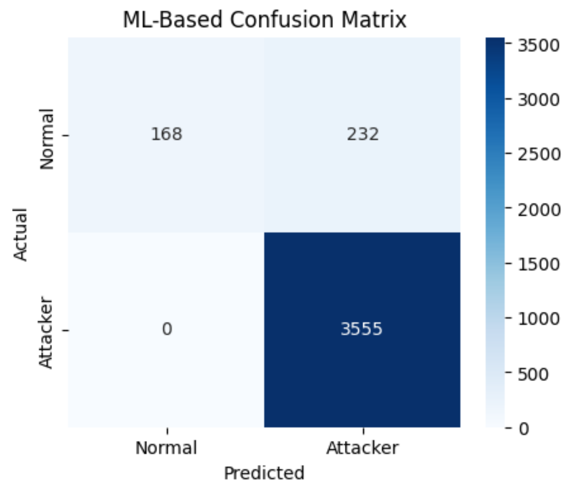

Figure 2: This matrix illustrates the ML classifier's perfect detection of all 3555 attacker sessions (recall = 1.0) but also reveals its downside: 232 false positives on normal sessions. This confirms that while biometric models trained on keystroke dynamics.

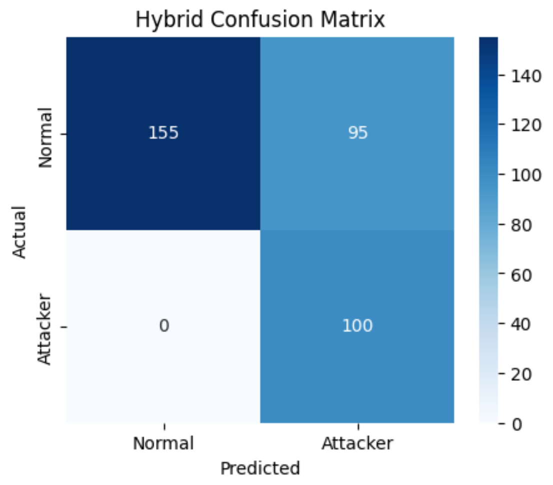

Figure 3: This matrix shows the hybrid system maintained perfect attacker recall (100/100), while slightly reducing false positives compared to the ML-only system. It struck a middle ground, better than rule-only in recall, and more conservative than ML-only in false alert rate. Aligning with its goal of balancing detection coverage and usability.

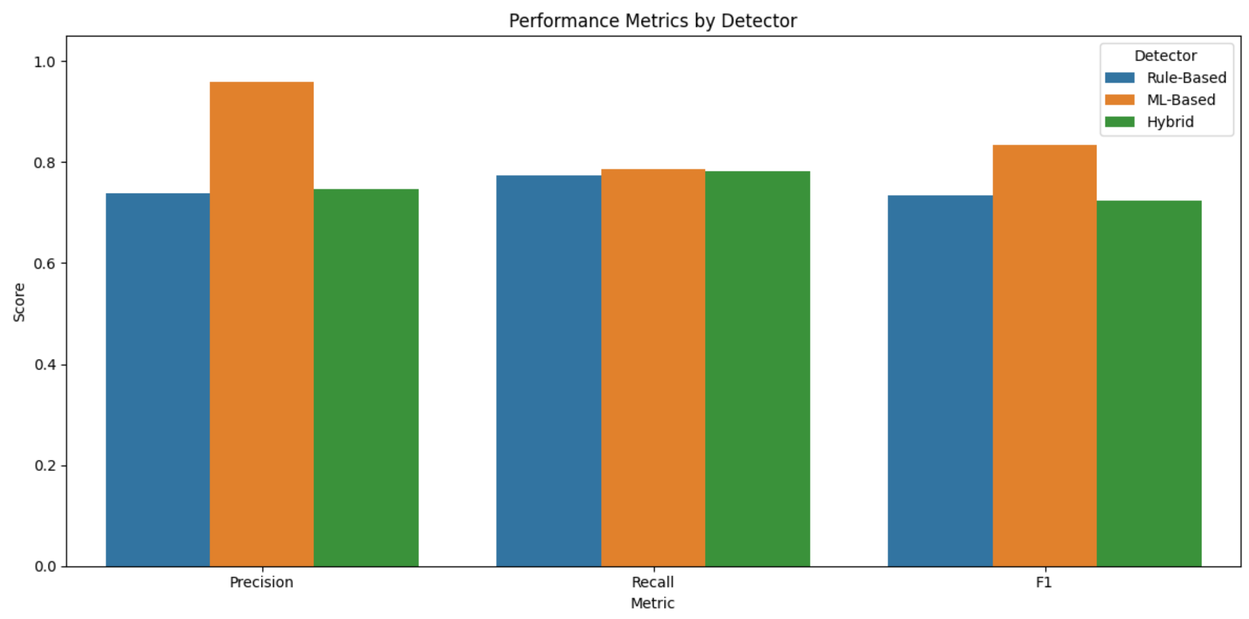

Figure 4: This bar chart compares key performance metrics across all detection methods. While ML-based detection led in precision and recall, its false positive cost was higher. The hybrid system achieved the best tradeoff in F1-score (0.678), validating the project's hypothesis that layered detection improves robustness without degrading practicality.

Disclaimers and Replicability Notes

- All results were generated using simulated login data and a fixed KeyRecs-trained ML model.
- Due to random session generation and sample variability, results may slightly shift across runs.
- These results are intended to show relative behavior and tradeoffs among detectors, not fixed performance guarantees.

8 Design Decisions

This section outlines the architectural and methodological choices made during development, along with reasoning and tradeoff explanations.

8.1 Layered Detection Architecture

The system was designed as a dual-layer detection pipeline, combining:

- Tactic-based detection (rules on login metadata), and
- Identity-based detection (keystroke biometric ML model).

This separation ensures broad behavioral coverage: login metadata captures volumetric and evasive attack patterns, while keystroke dynamics detect impersonation attempts that mimic valid usage but deviate in timing behavior.

This design reflects how enterprise-grade intrusion systems function in practice. By combining multiple signal sources, each specializes in a different threat category.

8.2 Rule-Base Detection Rationale

Rules were chosen based on industry heuristics (e.g., OWASP, Microsoft Identity Protection), including:

- Max failed attempts (BRUTE_FORCE_ATTEMPT_THRESHOLD)
- Min time between attempts (BRUTE_FORCE_TIME_THRESHOLD)
- Excessive username churn
- Abnormal typing speed or variance

The rule logic was implemented as a standalone module and evaluated independently. Profiles (strict, moderate, relaxed) were explored to show how the same system behaves under different security requirements. This tuning was kept modular and out-of-band, aligning with real-world usage where rule policies may vary by deployment context.

8.3 OR-Logic Pipeline Design

The final pipeline applies OR logic: a login is flagged as suspicious if either the rule-based logic or the ML classifier triggers.

This decision prioritizes recall (attacker coverage) over precision and reflects a real-world assumption: high-confidence rules or biometric deviations, even independently, are enough to warrant investigation.

Alternatives (e.g., AND logic or weighted voting) were not implemented due to time constraints, but this OR-based model reflects the security-first posture assumed in critical infrastructure systems.

8.4 Modular Implementation

Each component (rule detector, ML Classifier, pipeline logic, evaluation) was implemented as a standalone module with minimal coupling:

- rule_based_detector.py: purely threshold-based logic
- train_ml_classifer.py: standalone training on KeyRecs vectors
- pipeline_detector.py: unifies predictions with OR-logic
- evaluation.py: common evaluation utilities reused across experiments

This modularity allowed for testing and tuning of each part without breaking dependencies. It also makes the system extensible. New classifiers or alternate rulesets can be dropped into the pipeline with minimal changes.

9 Skills Learned

This project was not just a technical exercise, but a hands-on testbed for developing autonomous learning skills emphasized throughout the course. My learning is categorized below using the course's three-level skill framework:

L1: Basic Skills – Facts Over Fiction, Technical Grounding

This project required a precise evaluation of what the system actually does versus what it is expected or assumed to do. Early in development, I held the belief that training a model on both login metadata and keystroke dynamics would yield a stronger classifier. However, as implementation progressed, I recognized the risks of overfitting synthetic login patterns. Especially given that metadata was handcrafted and tied to simulation logic. I shifted focus to isolate the ML model to keystroke-only data from a real biometric (KeyRecs), reinforcing the value of grounding claims in verifiable behavior rather than theoretical assumptions. This helped me stay aligned with fact-based documentation and avoid overstating system capabilities.

L2: Meta-Skills – Applying Theory, Testing Assumptions

The evolution of the system's architecture, from a loosely defined rule/ML combo to a deliberately modular hybrid pipeline, reflected the application of course-taught theory. For example, I started with a vague idea that "rule and ML will cooperate" but refined it into a formal OR-logic pipeline where each subsystem (rule and ML) flags independently. The rule system blended login metadata with timing-derived keystroke summaries, while the ML model focused exclusively on biometric vectors.

Throughout, I tested assumptions about how each layer contributes to detection quality. I evaluated tuning profiles separately to validate rule logic under strict and relaxed policies. I also confronted the risk of simulation bias. Notably the danger of ordered attacker/user sequences leaking positional clues to the classifier. This led to shuffling strategies and a greater respect for how data design influences the outcome.

Training the ML model solely in real world keystroke data from KeyRecs helped mitigate bias introduced by synthetic simulation artifacts. It also raised new awareness that real data sets may still carry hidden human biases or noise. A reminder that data quality and collection contacts matter, even when using "real" sources.

L3: Advanced Skills – Trend Recognition and Pattern Cycles

One of the standout insights was recognizing the trend toward hybrid detection systems. The project's use of both deterministic rules and probabilistic classifiers echoes the broader industry move towards ensemble and defense-in-depth approaches in cybersecurity. I also noticed a generalization/specialization cycle firsthand: the rule-based detector started as a general approach but would have benefited from personalized user profiles. Time constraints limited full integration, but the architecture showed room for user specific specialization, another course taught trend pattern. Likewise, the early use of a monolithic script evolved into a more modular, decomposed pipeline. Showing the decentralization trend common in scalable system design.

10 Conclusion & Future Work

This project delivered a functioning hybrid login detection system that integrates rule-based heuristics and machine learning to flag suspicious authentication attempts. The final design emphasizes modularity, interpretability, and separation of concerns: rule logic operated on login metadata and timing summaries, while the ML classifier focused exclusively on keystroke-based biometric features derived from real human input (KeyRecs). Together, these components were integrated into a binary OR-based pipeline to ensure maximum attacker recall with reasonable false positive tradeoffs.

While the system achieved its core design goals, several areas for improvement and future expansion remain. Given more time, broader evaluation across multiple test runs would have helped validate the system's reliability under diverse conditions. Additionally, while rule tuning was explored in a separate notebook, it was not fully integrated into the full hybrid pipeline, a missed opportunity that could have improved adaptability across security contexts.

If continued, the next stage of this project would involve exploring more realistic or user-diverse datasets, implementing use-specific profiling in the rule engine, and experimenting with multiple ML classifiers trained on distinct feature subsets. The hybrid architecture could also evolve from a basic OR logic into more expressive logic trees (e.g., (A AND B) OR C) to better capture overlapping indicators of compromise.

11 Deliverables

The following components were produced and submitted as part of the final project:

- Login Simulator
  A python script (generate_login.py) that generates synthetic login attempts across three behavior types: norma, brute-force, and evasive. Each attempt is labeled with metadata including success/failure, session ID, inter-attempt delay, and keystroke vector.
- Rule-Based Detection System
  Implemented in rule_based_detector.py, this module uses hardcoded thresholds to flag suspicious logins based on metadata (e.g., attempt count, typing speed, username variety) and basic keystroke-derived states.
- Machine Learning Classifier
  Trained via train_ml_classifier.py on fixed-text biometric data from the KeyRecs dataset. Operates independently of metadata to avoid synthetic bias. Predicts normal vs. attacker keystroke profiles.
- Hybrid Detection Pipeline
  Combines the above two detectors using OR logic. Implemented in pipeline_detector.py, this system flags login if either the rule-based or ML model triggers, designed to maximize attacker recall.
- Evaluation Notebook
  A complete Jupyter notebook was created (Biometric Enhanced Login Detection Pipeline.ipynb) that demonstrates the entire flow: data loading, detection execution, result generation, and plotting. This notebook represents the final and most reproducible artifact.
- Evaluation Outputs
  Metrics and confusion matrices were generated for all three detection modes (rule-based, ML, hybrid). These include .txt files with precision/recall/F1 scores and .png visualizations such as confusion matrices and a comparative bar chart summarizing performance across detection layers.
- Threshold Tuning Showcase
  A standalone notebook (Tuning_Showcase_Thresholds.ipynb) was created to explore the effect of different threshold profiles (strict, moderate, relaxed) on rule-based detection performance. This served as an exploratory module rather than part of the main system.
- Final Report
  This document serves as the full write-up, explaining system design, methodology, evaluation, and skills learned.

- Video Demo
  A short video demonstrating system components and pipeline execution using the notebook interface that was created and submitted alongside the report.

12 References

1. OWASP Foundation. “Authentication Cheat Sheet.” https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

2. Microsoft. “Overview of Identity Protection in Azure AD.” https://learn.microsoft.com/en-us/azure/active-directory/identity-protection/overview-identity-protection

3. Dias, C., Carvalho, M. Coelho, D., & Correia, M.E. (2023). “KeyRecs: A Keystroke Dynamics Dataset for User Identification and Impersonation Detection.” *Data in Brief*, 48, 109296 https://doi.org/10.1016/j.dib.2023.109509

4. Auth0. “Credential Stuffing Attacks Explained.” https://auth0.com/blog/what-is-credential-stuffing/