Software Design Document

Secure Authentication & Abuse Detection Platform

1. Overview

1.1 Purpose

This project implements a **secure authentication system** that not only verifies user credentials but also **detects and responds to abusive login behavior** such as brute-force attacks, credential stuffing, and automated bot activity.

The goal is to demonstrate **security-aware backend system design** using Django and common production tools.

1.2 Problem Statement

Traditional authentication systems:

- Only check username and password
- Do not detect abnormal login behavior
- Do not log or alert on suspicious activity
- Are vulnerable to automated attacks

This results in:

- Account takeover risk
- Lack of audit trails
- Poor incident visibility

1.3 Solution Summary

The system adds **multiple security layers** around Django authentication:

- Rate limiting
- Risk evaluation
- Account lockouts

- Security event logging
- Asynchronous alerts

All decisions are **time-based**, **temporary**, and **auditable**.

---

2. System Goals & Non-Goals

2.1 Goals

- Detect brute-force and credential stuffing attacks
- Prevent abuse without permanently locking users
- Keep authentication fast and scalable
- Maintain detailed security logs
- Align with OWASP Top 10 principles

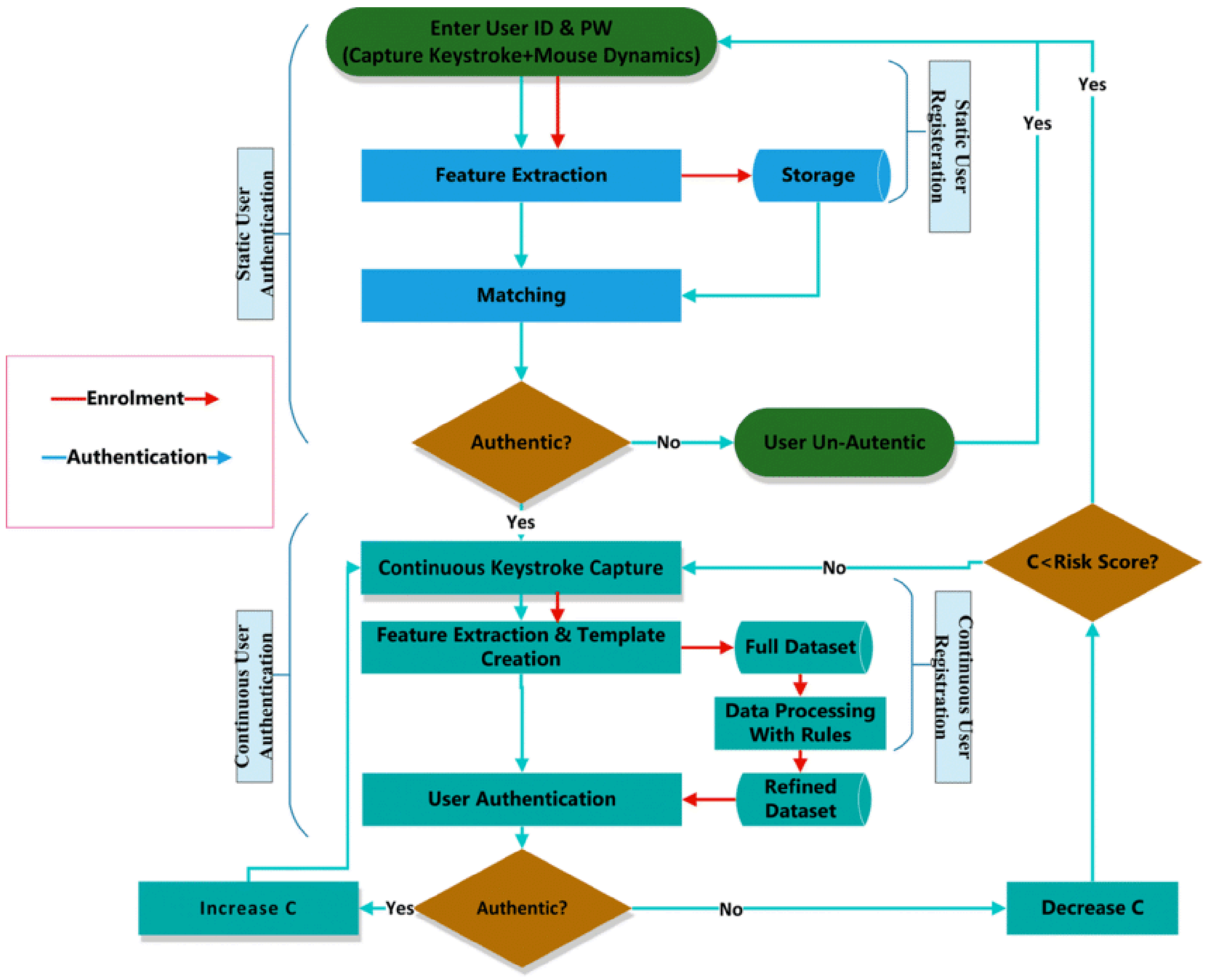
2.2 Non-Goals

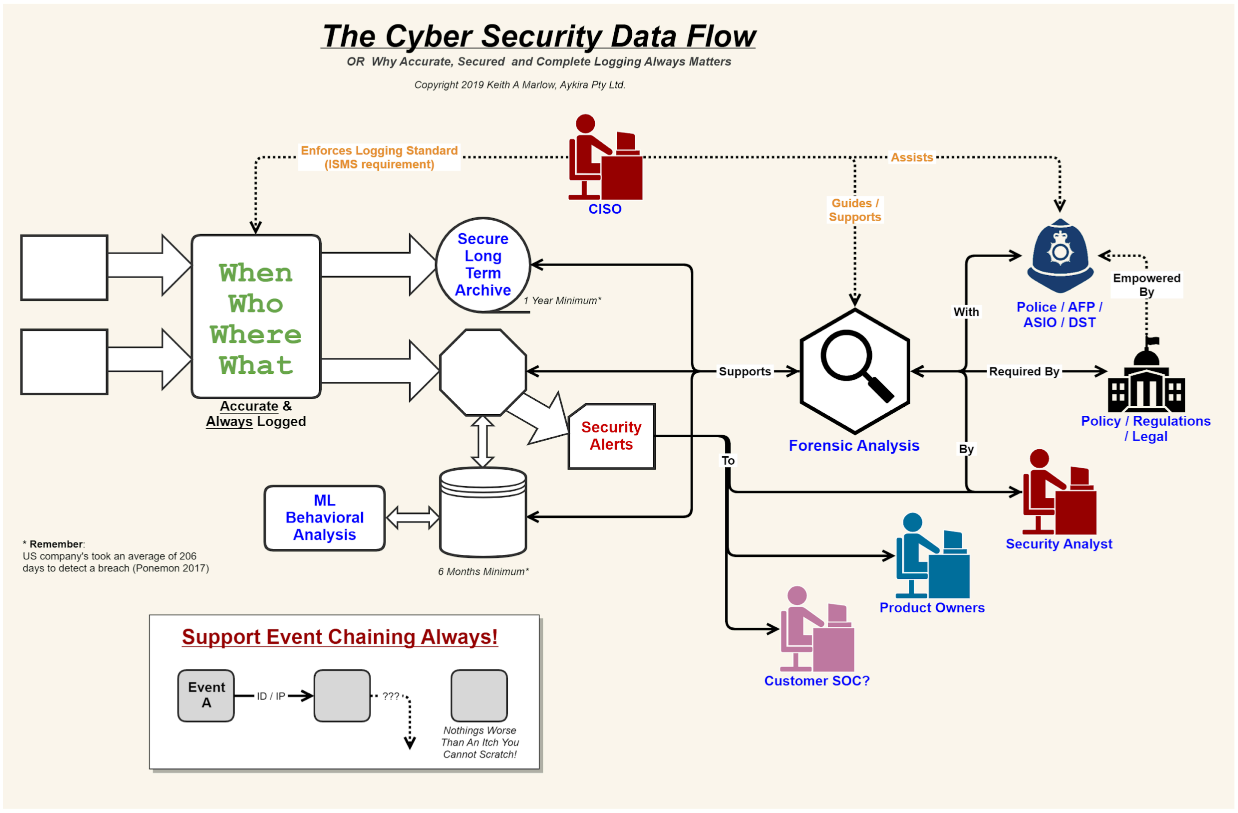
- No penetration testing tools
- No machine learning
- No biometric or MFA implementation
- No malware scanning

This is a **defensive backend system**, not a security product.

---

3. High-Level Architecture


Enter User ID & PW
(Capture Keystroke+Mouse Dynamics)
Feature Extraction
Storage
Static User Registeration
Matching
Static User Authentication
Enrolment
Authentication
Authentic?
No
User Un-Autentic
Yes
Yes
Yes
Continuous Keystroke Capture
No
C<Risk Score?
Feature Extraction & Template Creation
Full Dataset
Data Processing With Rules
Refined Dataset
Continuous User Registration
Continuous User Authentication
User Authentication
Increase C
Yes
Authentic?
No
Decrease C


The Cyber Security Data Flow
OR Why Accurate, Secured and Complete Logging Always Matters
Copyright 2019 Keith A Marlow, Aykira Pty Ltd.
Enforces Logging Standard (ISMS requirement)
CISO
Assists
Guides / Supports
When Who Where What
Accurate & Always Logged
Secure Long Term Archive
1 Year Minimum*
Security Alerts
Supports
Forensic Analysis
With
Police / AFP / ASIO / DST
Empowered By
Required By
Policy / Regulations / Legal
By
To
Security Analyst
Product Owners
Customer SOC?
ML Behavioral Analysis
6 Months Minimum*
* Remember: US company's took an average of 206 days to detect a breach (Ponemon 2017)
Support Event Chaining Always!
Event A
ID / IP
???
Nothings Worse Than An Itch You Cannot Scratch!

4

3.1 Components

Component	Responsibility
Django API	Authentication & orchestration
Redis	Rate limits, locks, temporary state
PostgreSQL	Persistent security logs
Celery	Background alerts & cleanup
Admin Dashboard	Security visibility

---

4. Authentication Flow (End-to-End)

4.1 Request Lifecycle

1. Client sends login request
2. Pre-authentication checks are performed

3. Django authenticates credentials
4. Post-authentication risk evaluation
5. Decision engine determines outcome
6. Security event is logged
7. Background actions triggered (if required)

4.2 Pre-Authentication Checks

Performed **before password verification**.

Checks:

- Is IP temporarily blocked?
- Is user account locked?
- Has IP exceeded rate limits?
- Has user exceeded failed attempts?
- Is IP attempting many users?

Result:

- If any check fails → request is rejected with HTTP 429
- Password is **not** checked

---

5. Rate Limiting Design

5.1 Time-Window Strategy

All limits are applied within **sliding time windows**, not lifetime.

5.2 IP-Based Rate Limit

Policy:

- Max 10 login attempts per IP per 10 minutes

Purpose:

- Stops rapid brute-force attacks

5.3 User-Based Rate Limit

Policy:

- Max 5 failed attempts per user per 15 minutes

Purpose:

- Stops credential stuffing, even across multiple IPs

5.4 IP → Many Users Detection

Policy:

- More than 10 unique users from same IP in 10 minutes

Purpose:

- Detects botnets and automation

---

6. Redis Design (Conceptual)

Redis is used to store **temporary counters and flags**.

6.1 Key Patterns

Key	Purpose	Expiry
`login:ip:<ip>`	IP attempt counter	10 min
`login:user:<username>`	User failed attempts	15 min
`login:ip_users:<ip>`	Unique users from IP	10 min
`blocked:ip:<ip>`	Temporary IP block	15 min
`locked:user:<username>`	Account lock	30 min

Redis auto-expires all keys.

---

7. Risk Evaluation Logic

7.1 Risk Factors

- Number of recent failures
- IP reputation
- New device or IP
- Login frequency
- Historical security events

7.2 Risk Scoring (Example)

Risk Score	Action
0–30	Allow login
31–60	Allow + log warning
>60	Lock account + alert

Risk scoring is rule-based and transparent.

---

8. Account Lock & Unlock Strategy

8.1 Lock Conditions

- User exceeds failed attempt threshold
- High-risk behavior detected

8.2 Unlock Mechanism

- Automatic unlock after timeout (Redis expiry)
- Optional admin-initiated unlock

- No permanent lockouts

This avoids denial-of-service against real users.

---

9. Security Event Logging

9.1 Events Logged

- Failed login
- Rate limit exceeded
- Account locked
- IP blocked
- Suspicious activity detected

9.2 Log Contents

- Event type
- User (if known)
- IP address
- Timestamp
- Severity
- Metadata (JSON)

Logs are stored in PostgreSQL for audit and investigation.

---

10. Asynchronous Processing (Celery)

10.1 Background Tasks

- Email alerts for high-severity events
- Daily security summaries
- Unlock scheduling
- IP reputation decay

10.2 Why Async?

- Keeps login fast
- Avoids blocking requests

- Improves scalability

---

11. Admin Security Dashboard

11.1 Visible Metrics

- Failed login count (24h)
- Locked accounts
- Blocked IPs
- High-severity events
- Top attacking IPs

Dashboard can be implemented using Django Admin or simple UI.

---

12. OWASP Mapping

Feature	OWASP Category
Rate limiting	A2: Broken Authentication
Generic error messages	A2
Account lockout	A2
Audit logging	A10: Logging & Monitoring
Token protection	A2

---

13. Deployment (Later Phase)

- Dockerized Django app
- Redis service
- PostgreSQL service
- Optional cloud deployment (Azure/AWS)

---

14. Summary

This system:

- Adds **defensive security layers** to authentication
- Detects abuse instead of just failing logins
- Uses proven industry patterns
- Demonstrates security-aware backend engineering