



# Advanced Password Authentication System

Developed by: Harshith Gowda R — Technologies: Python · Tkinter · SQLite · Bcrypt

A concise technical overview for developers and security stakeholders evaluating an on-prem authentication prototype designed to detect and mitigate brute-force attacks.

# Why this matters

## Risk Landscape

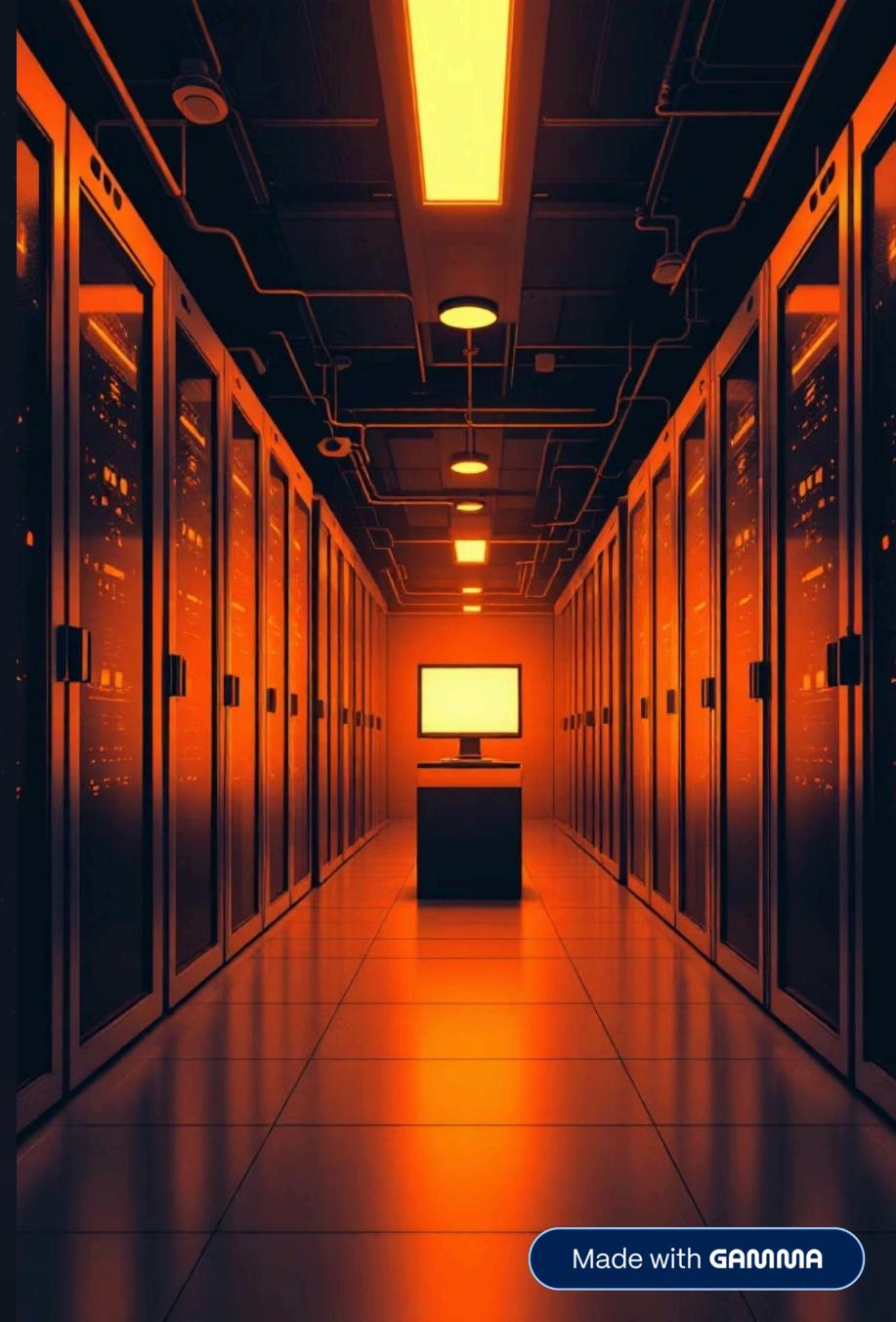
Weak passwords and automated attacks remain primary vectors for data breaches.

## Operational Impact

Compromised accounts lead to data loss, downtime, and reputational damage.

## Design Goal

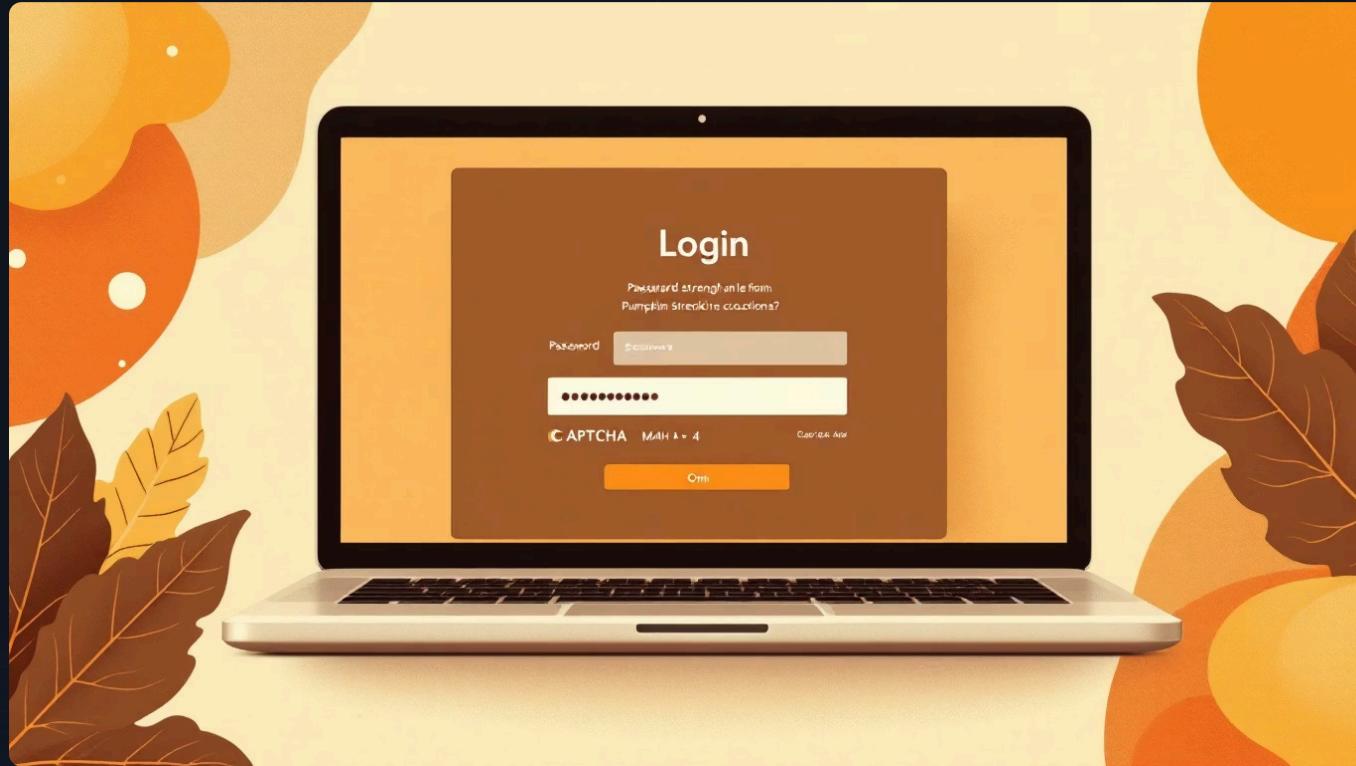
Provide robust, low-friction defenses against credential stuffing and brute force.





## Problem Statement

Users create weak passwords; attackers submit repeated login attempts; no reliable detection or account protection exists.  
Result: compromised accounts and data exfiltration risk. This system addresses detection, throttling, and secure password storage.



## System Objectives

- Secure authentication using industry-grade hashing
- Detect and block brute force attempts
- Temporary account locks after failed attempts
- Generate detailed security logs for auditing
- Enforce password strength at registration

# Password Security Mechanics

Passwords are never stored in plaintext. The system uses Bcrypt for salted hashing with adjustable work factor. Hashing plus a password strength policy reduces attack surface by increasing attacker cost and enforcing entropy at creation.

## Key Properties

Salted hashes, adjustable rounds, no reversible storage.

## Policy Enforcement

Length, mixed character classes, and common-password rejection.

```
...  
.bcrypt hashed bypes  
Password haseed;  
password password:  
Password passsword;  
Password password
```



## Lock & CAPTCHA

After 3 fails, 30s lock + CAPTCHA

## On Failure

Increment fail counter

## Verify Hash

Compare stored password hash

## Submit Credentials

User sends login data

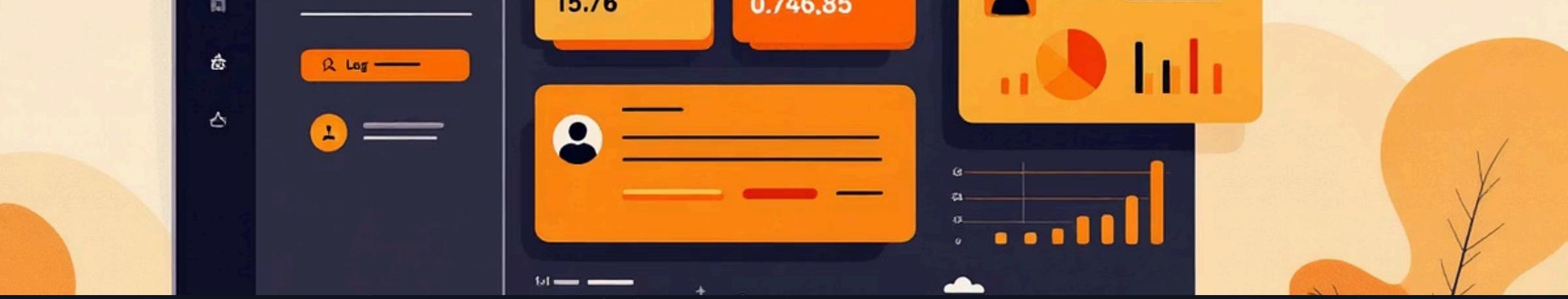
The diagram illustrates precise request handling: failed counter increments per account, CAPTCHA triggers to block bots, and deterministic 30-second lockout to throttle repeated attempts while preserving usability.



## Brute Force & CAPTCHA Strategy

Limit: 3 incorrect attempts per account. On threshold breach the account is locked for 30 seconds and a lightweight math CAPTCHA appears to distinguish human users from bots. This reduces automated velocity while allowing users a quick recovery path.

- Per-account counters, reset on successful login
- CAPTCHA only after threshold to minimize friction



# Admin Panel & Logging

Administrators can view registered users, failed logins, lock status, and system events. Logs capture timestamps, usernames, source IP (if available), and event type. All logs are appended to a local file for audit and incident response.



## Audit Trail

Persistent event records for post-incident analysis.



## Real-time Alerts

Failed attempt spikes flagged for investigation (future enhancement).

# Advantages & Roadmap



## Current Benefits

Improved security posture,  
low development complexity,  
fast local deployment.



## Planned Enhancements

OTP, email alerts, face  
recognition, and optional  
cloud backend.



## Scalability Path

Web version + cloud DB to  
support centralized logging  
and multi-node rate limiting.

# Conclusion & Recommendations

- Use Bcrypt with conservative cost parameter and enforce strong password rules.
- Maintain per-account counters, 30s lockouts, and CAPTCHA to throttle attacks.
- Implement log aggregation and alerts for suspicious patterns before scaling to cloud.
- Next steps: add OTP, email alerts, and optional cloud-based central logging for incident response.

Consider threat modeling and red-team testing before production rollout. Balance lockout duration and user experience to avoid denial-of-service risk against legitimate users.

