**Secure Application Programming CA 1**

**Programme:** BSHCCYBE4, BSHCSD4\_BSHCYB4

**Date:** 31st October (Week 7)

Student Name: Moses Ademokun

Student ID: 20126204

# Executive Summary

This report analyses the security implementation of a Java-based authentication system. The original system contained several critical security vulnerabilities, including plain-text password storage, lack of brute force protection, and timing attack vulnerabilities. Through systematic improvements, we implemented industry-standard security measures including BCrypt password hashing, account lockout mechanisms, and constant-time operations to prevent timing attacks. Below is the list of the Vulnerability discovered in the code.

Weak or missing input validation

* No maximum lengths or character restrictions; risks include injection or buffer-related issues if later connected to persistent storage.

No IP-based rate limiting.

* The system tracks login attempts but does not limit requests by IP, enabling distributed brute-force or credential-stuffing attacks that target multiple accounts from one IP (or many IPs).

Account lockout can be abused (Denial-of-Service)

* Locking an account after repeated failures prevents brute-force but allows an attacker to intentionally lock legitimate accounts by submitting bad passwords.

In-memory storage of credentials and sessions

* Users and sessions are stored in JVM memory (HashMap). Data is lost on restart and is unsuitable for multi-instance deployments, no encryption at rest for any persisted form.

Session management is minimal.

* Sessions are single tokens with expiry, but lack session rotation, binding (IP/user-agent), or replay protections. No limit on concurrent sessions per user.

Logging/monitoring absent

* Authentication events (failed attempts, lockouts, suspicious patterns) are not logged or exported for monitoring or alerting.

Partial cryptographic Solutions not fully hardened

* BCrypt is used correctly, but work factor, storage policy and migration path are not externally configurable. No HSM or secret management for token-related operations.

No multi-factor authentication (MFA) or recovery controls

* The system relies solely on passwords; account recovery or MFA are absent.

Potential information leakage via return values

* While the login method attempts to avoid explicit leakages, returning null with different behaviours could still reveal states in some API integrations unless normalized at API layer.

Limited protection against automated enumeration aside from timing measures

* Timing hardening is implemented, but complementary controls (rate limiting, CAPTCHAs) are missing.

**Remediation Explanation (technical Solutions and why)**

1. **BCrypt for password hashing**

Solution: BCrypt (work factor 12).

* Justification: BCrypt is a well-established adaptive algorithm resistant to GPU-based brute-force when the cost factor is increased. It includes built-in salt handling, making per-password salts automatic and reducing implementation complexity. Work factor twelve is a balance between security and usability on typical server hardware; it slows hashing enough to deter brute-force while remaining responsive for users.

1. **Password complexity and validation**

Solution: Enforced minimum length (12) and character class requirements via regex.

* Justification: Stronger passwords increase entropy and slow offline guessing attacks if hashes are leaked. Enforcing complexity at creation prevents weak credentials and reduces the effectiveness of credential-stuffing attacks.

1. **Account lockout after repeated failures.**

Solution: Per-account lockout after 5 failed attempts for 15 minutes.

* Justification: Temporarily locking an account raises the cost for an attacker attempting repeated guesses. The 15-minute window balances security (thwart brute-force) and user inconvenience. Lockout is tracked per-account to contain effects compared to global rate-limits.

1. **Constant time checks and dummy hash**

Solution: Always run BCrypt.checkpw against either the real user hash or a precomputed DUMMY\_HASH.

* Justification: Prevents username enumeration by ensuring the computational cost is the same regardless of whether the username exists. Using a DUMMY\_HASH prevents an attacker from distinguishing valid vs invalid accounts based on response timing.

1. **Cryptographically secure session tokens**

* Solution: 256-bit random tokens generated with SecureRandom and base64 URL-safe encoding.
* Justification: High entropy tokens make brute-force or guessing infeasible. Using SecureRandom ensures tokens are not predictable.

1. **Session expiry**

* Solution: 30-minute session lifetime.
* Justification: Limits the window for stolen-token misuse. A conservative timeout trades off some convenience for better security.

# Trade-off Discussion (security vs usability)

**Account lockout policy (security vs. availability/usability)**

I have chosen to lock an account for 15 minutes after 5 failed attempts. This is good for security because it prevents brute-force attacks by making it impossible for an attacker to keep trying passwords rapidly. However, on the usability side, this can frustrate legitimate users if they forget their password and need multiple tries, have caps lock on by accident or getting locked out by an attacker deliberately failing logins (denial of service).

I made a trade-off Decision to choose five attempts because it gives users several chances to get their password right (more usable than three attempts); but still prevents automated password guessing (more secure than ten attempts). I chose a 15-minute lock out because it is long enough to discourage attackers; but short enough that users are not locked out for too long.