Security Audit and Code Hardening Report

Project: Java Authentication and Session Management System

Student name: Eskandar Atrakchi

Student Number: x23137517

Date: 20th/Oct/2025

Type of document: Report

GitHub link: https://github.com/EskandarAtrakchi/Java-Authentication-and-Session-Management-System

PART A – Security audit

1. Plain text password storage: user passwords were stored as a string in memory which violates confidentiality

```
static class User {
 String password; // Stored in plain text
```

2. User enumeration by early return: because longing in method returned immediately when a username doesn't exist, which allows attackers to know valid usernames based on timing responses which violates defense in depth and confidentiality

```
if (!users.containsKey(username)) {
    return null; // Early return reveals valid users
```

3. Timing attack on password comparison: use of string.equals() means the system will make time differences during measurements which enable potential side channel attacks, and this violates integrity and confidentiality
4. Unlimited attempts: the security flaw violates availability and integrity
5. Session tokens issues: session identifiers were generated from username and timestamps which can be hijacked, violates confidentiality and integrity
6. Session validation weak: only checks for simple prefix which lacks secure token verification, this violates defense in depth

PART B – Remediation and security enhancements

1. Secure password storage: I have replaced plaintext storage with slated password hashing using PBKDF2WithHmacSHA256 with 360,000 iterations and 16-byte random salt
   Justification: PBKDF2 is secured protects passwords even if hashes are leaked, and the high iteration number is to increase the efforts for the hackers, but it won't affect users

```
private static final int Iterations = 360000; // reasonable modern cost
```

2. User enumeration mitigation: the system performs identical cryptographic operations whether the user is valid or invalid using fake salt and random comparisons before returning any feedback
   Justification: consistent execution is important to prevent attackers from guessing the existence usernames through timing or responses

```
// compare to another random value in constant time
byte[] randomCompare = new byte[fakeHash.length];
SecureRandomGenerator.nextBytes(randomCompare);
```

3. Timing attack on password comparison: MessageDigest.isEqual() for password and hash verification
   Justification: this prevents timing-based side channels attacks that may reveal partial hash matches

```
private static boolean constantTimeEquals(byte[] a, byte[] b) {
    if (a == null || b == null) return false;
    return MessageDigest.isEqual(a, b);
```

4. Unlimited attempts: variable account lockout after two attempts after failing twice then 6 minutes lockout knowing all lockouts counters will reset after successful login authentication

```
private static final int maxAttempts = 2;
private static final long lockoutDurationMS = 6 * 60 * 1000L; // 6 minutes
```

5. Session tokens issues: generated 256bit tokens for session using SecureRandom and stored as SHA-256 hashes, each session includes a 30 minutes expiration and stored in server (means as long as the code is running) is also validated by hash lookup
   Justification: this method will prevent token prediction and re-tries attacks, and storing it in server side will lower the risk of memory data exposure
   ```
   sessions.put(tokenHash, new Session(username, now + sessionTimeInMinutes));
   return token; // raw token returned to client server keeps only its hash
   ```
6. Session validation weak: replaced HashMap with ConcurrentHashMap for concurrent access control
   Justification: to make sure shared data stays correct and safe, even if multiple tasks are running together (multi-threaded environments)
   ```
   private final Map<String, User> users = new ConcurrentHashMap<>();
   // store SHA-256(sessionToken) -> Session
   private final Map<String, Session> sessions = new ConcurrentHashMap<>();
   ```

PART C – Design Trade-Off Discussion

One trade-off was security vs usability that is related to lockout user account. While it is true that temporary lockouts stop attacks, it can also lockout a user account if the attackers attack a known account multiple time. The balanced measurements that have been taken are 2 tries with 6 minutes from reasonable user perspective login to strong attacks on a known account. If this project is deployed for production we can have more based components, for example, CAPTCHA, or IP based throttling can also help improve this balance