**Sprint 1 Security Assessment**

**High Impact Vulnerabilities and Proposed Prevention Methodologies**

**Sensitive Data Exposure**

If sensitive data like passwords, credit card information, or private user data is exposed (especially if transmitted over unencrypted channels), it can lead to identity theft, financial loss, or other severe consequences. Exposure of sensitive data can have serious repercussions, such as data breaches, legal liabilities, and trust loss with users. The issue is compounded if proper encryption is not used for data at rest and in transit.

***Proposed Prevention Methodology:*** Always hash passwords using strong hashing algorithms like bcrypt or Argon2. Use HTTPS (SSL/TLS) to encrypt data in transit and never store sensitive data in plaintext.

**SQL Injection**

SQL injection is one of the most critical vulnerabilities because it allows attackers to access, modify, or delete data in the database, which can lead to data breaches, data loss, or unauthorized access to sensitive information. SQL injection allows direct manipulation of the database, making it an extremely high-impact vulnerability. If exploited, it can result in full control over the data and in some cases, even remote code execution.

***Proposed Prevention Methodology:*** Always use parameterized queries or ORM libraries like SQLAlchemy, which prevent SQL injection by handling input safely. Avoid string concatenation in SQL queries.

**Cross-Site Scripting (XSS)**

XSS attacks allow attackers to inject malicious scripts that can steal session cookies, capture user inputs, deface web pages, or spread malware to other users. XSS can lead to session hijacking, credential theft, or spreading malicious content, affecting a large number of users. It's dangerous when sensitive data like cookies or user input is improperly handled.

***Proposed Prevention Methodology:*** Use Flask's built-in escape() function or Jinja2 templating system, which automatically escapes user input to prevent XSS. Avoid directly embedding user-provided content in the HTML or JavaScript.

**Insecure Cookie Handling**

Insecure session cookies can lead to session hijacking, allowing an attacker to impersonate a legitimate user and gain unauthorized access to their data. Cookie vulnerabilities directly impact the integrity and confidentiality of user sessions. If session cookies are improperly configured (e.g., lacking Secure, HttpOnly, or SameSite), they can be stolen or manipulated by attackers.

***Proposed Prevention Methodology:*** Use Flask's secure cookie feature by setting SESSION\_COOKIE\_SECURE=True to ensure that cookies are transmitted over HTTPS. Also, ensure that SESSION\_COOKIE\_HTTPONLY is enabled to prevent client-side access to cookies, and SESSION\_COOKIE\_SAMESITE to restrict cross-site cookie usage.