## PASTA worksheet

| **Stages** | **Sneaker company** |
| --- | --- |
| **I. Define business and security objectives** | * *Will the app process transactions?*   ***Facilitate Seamless Connection and Commerce:*** *The app's primary goal is to be an* ***efficient and user-friendly meeting point*** *for buyers and sellers, enabling clear and swift transactions.*   * *Does it do a lot of back-end processing?*   ***Ensure User Privacy and Trust:*** *For us, it's a priority to* ***protect user data privacy*** *and ensure users feel secure about how their information is handled.*   * *Are there industry regulations that need to be considered?*   ***Guarantee Proper Payment Management and Legal Compliance:*** *It's crucial that the payment system is robust, offers* ***multiple options****, and is managed correctly to* ***prevent legal issues****.* |
| **II. Define the technical scope** | List oftechnologies used by the application:   * *Application programming interface (API)* * *Public key infrastructure (PKI)* * *SHA-256* * *SQL*   We'd prioritize the API for security evaluation, as it's often the main external entry point.  Prioritized Evaluation & Security Risks  API (Application Programming Interface):  Risks: Vulnerable to injection attacks, poor authentication/authorization, excessive data exposure, DoS attacks, and data tampering. Third-party API vulnerabilities are also a concern.  SQL (Structured Query Language):  Risks: High risk of SQL injection (SQLi) if inputs aren't properly sanitized, leading to unauthorized data access, modification, or deletion.  PKI (Public Key Infrastructure) - AES and RSA:  Risks: Weak key management (generation, storage, rotation), flawed algorithm implementation, private key leaks, and susceptibility to downgrade attacks.  SHA-256:  Risks: If not used with unique "salts," hashed passwords can be vulnerable to brute-force or dictionary attacks. Proper implementation is key to prevent misuse.  Our strategy is to first address the APIs and their SQL interaction due to their direct exposure, then focus on PKI and finally the proper use of SHA-256. |
| **III. Decompose application** | The user accesses the database to view and interact with the different fields that contain the sneaker models and price.  The database receives the request from the user and sends the information back.  **API:** Ensures user search inputs are validated to prevent injection attacks and securely transmits data (queries/results) via HTTPS.  **SQL Requests:** Preventing SQL injection (SQLi) by properly sanitizing user input. Also enforces database access controls (e.g., read-only for search).  **PKI:** Secures the communication channel (e.g., via HTTPS/TLS) for search queries and results, protecting them from eavesdropping.  **Encrypt data:** Primarily protects sensitive user data *at rest* within the database (like hashed passwords), not directly the search query/results in transit. |
| **IV. Threat analysis** | Both SQL Injection and Session Hijacking are highly aggressive risks.  However, we consider SQL Injection generally more aggressive due to its potential to fully compromise the entire application database.  Session Hijacking is also very aggressive as it allows an attacker to take over a legitimate user's account.   * *What are the internal threats?*   *Ex-worker in the organization, system misconfigurations, weak internal control access.*   * *What are the external threats?*   *SQL Injection, Session Hijacking, Phishing, Social Engineering, Malware, DDoS attack, Supply chain attacks.* |
| **V. Vulnerability analysis** | **Lack of prepared statements** (leading to SQL Injection)  Example: CVE-2025-1094, *Improper neutralization of quoting syntax in PostgreSQL libpq functions PQescapeLiteral(), PQescapeIdentifier(), PQescapeString(), and PQescapeStringConn() allows a database input provider to achieve SQL injection in certain usage patterns. Specifically, SQL injection requires the application to use the function result to construct input to psql, the PostgreSQL interactive terminal.*  **Weak login credentials** (contributing to Session Hijacking)   * *Could there be things wrong with the codebase?*   *Yes, absolutely (e.g., insecure APIs, improper error handling, XSS).*   * *Could there be weaknesses in the database?*   *Yes (e.g., weak passwords, unpatched software, misconfigurations, no data-at-rest encryption).*   * *Could there be flaws in the network?*   *Yes (e.g., insecure communication channels, open ports, poor segmentation, vulnerable devices).* |
| **VI. Attack modeling** | [Sample attack tree diagram](https://docs.google.com/presentation/d/1FmWLyHgmq9XQoVuMxOym2PHO8IuedCkan4moYnI-EJ0/template/preview?usp=sharing&resourcekey=0-zYPY7AhPJdcClXamlAfOag) |
| **VII. Risk analysis and impact** | SHA-256 apply, network secure controls, sanitize user input, access user control revising secure connections with HTTPS protocol. |