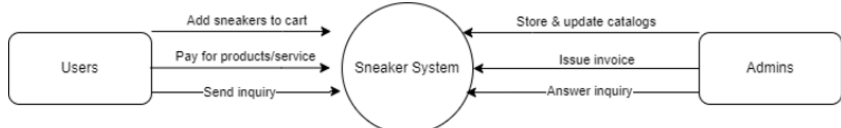
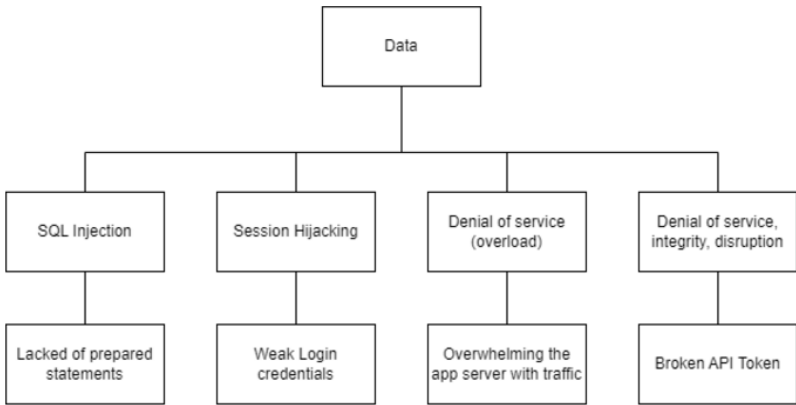


PASTA worksheet

Stages	Sneaker company
I. Define business and security objectives	<p>The app should provide functionality for users to create profiles, either internally or by connecting external accounts, to streamline the onboarding process. It must securely process financial transactions and comply with industry standards, such as PCI-DSS, to ensure user data and payment information are protected. Additionally, the app should facilitate communication between buyers and sellers and include a rating system to enhance transparency and trust. Lastly, the app should be user-friendly while maintaining a high level of security to prevent unauthorized access and potential data breaches.</p>
II. Define the technical scope	<p>List of technologies used by the application:</p> <ul style="list-style-type: none"> • <i>Application programming interface (API)</i> • <i>Public key infrastructure (PKI)</i> • <i>SHA-256</i> • <i>SQL</i> <p>From a security standpoint, prioritizing APIs is crucial as they facilitate the exchange of sensitive data between customers, partners, and employees. APIs connect various systems and users, creating a broader attack surface and increasing the potential for security vulnerabilities. This risk is amplified because the sneaker company has limited control over the security policies of third-party APIs. To mitigate these risks, it is essential to evaluate which APIs are being used, their specific roles within the application, and their adherence to stringent security standards.</p>
III. Decompose application	 <pre> graph LR Users[Users] -- "Add sneakers to cart" --> SS((Sneaker System)) Users -- "Pay for products/service" --> SS Users -- "Send inquiry" --> SS Admins[Admins] -- "Store & update catalogs" --> SS Admins -- "Issue invoice" --> SS Admins -- "Answer inquiry" --> SS </pre> <p>One of the key processes for this application involves utilizing APIs to handle and process incoming requests. These processes include validating the requests, authenticating users, authorizing access, and performing any additional necessary operations to ensure secure and efficient functionality.</p>

IV. Threat analysis	<p>Two potential threats to the application's data include:</p> <p>Session Hijacking: An external threat actor could intercept third-party API communications with users, potentially stealing sensitive personally identifiable information (PII). This attack could compromise user accounts and expose critical data.</p> <p>Injection Attacks: Unauthorized access to the application's database, potentially by an employee without clearance, could lead to the exploitation of vulnerabilities such as SQL injection. This could allow the attacker to manipulate or extract sensitive information from the database.</p> <p>By addressing these risks, the application can enhance its security posture and protect sensitive information from malicious actors.</p>
V. Vulnerability analysis	<p>Two vulnerabilities identified in the PASTA worksheet that could be exploited are:</p> <p>Lack of Prepared Statements: If the application does not use prepared statements for database queries, it could be vulnerable to SQL injection attacks, allowing malicious actors to manipulate or retrieve sensitive data.</p> <p>Broken API Token: A compromised, or improperly secured API token could expose the application to unauthorized access, enabling attackers to bypass authentication and gain access to sensitive user or system data.</p> <p>Addressing these vulnerabilities is critical to ensuring secure interactions with the application and protecting sensitive customer and system information.</p>
VI. Attack modeling	 <pre> graph TD Data[Data] --> SQL[SQL Injection] Data --> Session[Session Hijacking] Data --> Denial1[Denial of service (overload)] Data --> Denial2[Denial of service, integrity, disruption] SQL --> Lacked[Lacked of prepared statements] Session --> Weak[Weak Login credentials] Denial1 --> Overwhelm[Overwhelming the app server with traffic] Denial2 --> Broken[Broken API Token] </pre> <p>Malicious actors could exploit vulnerabilities in the application by</p>

	injecting SQL commands into data input fields, potentially gaining unauthorized access to sensitive database information.
VII. Risk analysis and impact	<p>Implementing input sanitization to prevent injection attacks.</p> <ul style="list-style-type: none">• Enforcing the principle of least privilege to restrict access to sensitive resources.• Utilizing hashing and salting techniques, such as SHA-256, to securely store passwords.• Establishing multi-factor authentication to add an additional layer of security for user access.• These controls, along with robust incident response procedures and a strong password policy, collectively enhance the application's overall security posture.
