Threat Analysis for the Online Shoe Company

Stages:

1. Define Business and Security Objectives

- PCI-DSS Compliance: Since the application will handle financial transactions, it is essential to comply with PCI-DSS (Payment Card Industry Data Security Standard) to ensure the security of credit card data.
- Secure Authentication and Authorization: Implement multi-factor authentication (MFA) to strengthen security, especially during login and transaction processes.
- Data Encryption: Ensure that all sensitive data, such as credentials and financial information, is encrypted both in transit (using TLS) and at rest (using AES-256).
- Access Control: Apply the principle of least privilege to ensure only authorized users have access to sensitive data.

2. Define Technical Scope

- API Review: APIs are essential for interconnection between clients, partners, and employees. They must be evaluated and prioritized due to the sensitive data they handle. Consider using OAuth 2.0 and JWT (JSON Web Tokens) for authenticating and authorizing API requests.
- API Gateway: Use an API Gateway to manage traffic and secure the APIs. Apply measures like rate limiting and authentication to prevent abuse.
- SHA-256: Ensure that passwords and sensitive data are protected using SHA-256 and other robust encryption techniques.
- PKI (Public Key Infrastructure): Implement PKI to secure communication between users and the application, as well as between internal systems.

3. Break Down the Application

- Data Flow Diagram (DFD): Use a detailed Data Flow Diagram (DFD) to illustrate how data moves through the application and identify critical points that may be vulnerable.
- SQL and Prepared Statements: Ensure that SQL queries use prepared statements to prevent SQL injection. Inputs should be properly validated to avoid malicious injections.
- User Input Controls: Implement input validation and sanitization to prevent vulnerabilities like Cross-Site Scripting (XSS) or Cross-Site Request Forgery (CSRF).

4. Threat Analysis

- SQL Injection: Ensure that the database is protected against SQL injection attacks, which are common in web applications interacting with databases. Use prepared statements and ORMs (Object Relational Mappers) to prevent these vulnerabilities.
- Session Hijacking: Session hijacking is possible if cookies are not handled securely. Implement the HTTPOnly and Secure flags to ensure cookies cannot be accessed via JavaScript or transmitted over insecure channels.
- Cross-Site Scripting (XSS): Prevent XSS attacks by validating and properly escaping user inputs, and using security headers such as Content Security Policy (CSP).
- DDoS Attacks: The application may be vulnerable to Distributed Denial of Service (DDoS) attacks. Use DDoS protection services like Cloudflare or AWS Shield.

5. Vulnerability Analysis

- SQL Injection Due to Lack of Prepared Statements: Failure to use prepared statements could open the door for SQL injections. Ensure that all database queries use prepared statements or stored procedures.
- Broken API Token: API tokens should be securely generated and never stored in plain text. Implement a policy for API token rotation and use OAuth 2.0 or JWT for authenticating API requests.
- Insecure Session Handling: If cookies or session tokens are mishandled, an attacker could hijack a user's session. Review session handling policies, implement automatic expiration, and ensure session revocation.

6. Attack Modeling

- Use MITRE ATT&CK and the CVE® List to understand the techniques used by cybercriminals and adapt them to the application's context.
- Conduct regular risk assessments to identify emerging threats, using tools like OWASP ZAP and Burp Suite to perform penetration testing and identify vulnerabilities.

7. Risk and Impact Analysis

- SHA-256 for Secure Hashing: Apply SHA-256 for securely storing passwords and other sensitive data. Use salts and iterative hashing to protect data at rest.
- Incident Response Procedures: Implement incident response procedures to detect, contain, and remediate any security breaches quickly. Ensure that personnel are trained to respond effectively to incidents.
- Password Policy: Establish a strong password policy that requires a minimum length, special characters, and prohibits the use of common passwords. Implement an account lockout mechanism after multiple failed login attempts.

- Security Monitoring: Implement a security monitoring solution that can detect suspicious activity in real time, such as network intrusions or unauthorized access. Tools like Splunk and Elastic Stack are useful for this.

Summary of Improvements and Next Steps:

1. Refining Security Objectives: Ensure that security objectives are clearly aligned with business goals and that appropriate controls are implemented from the outset.
2. Ongoing Review of APIs and Third-Party Components: APIs should be continuously evaluated and audited for new vulnerabilities.
3. Regular Penetration Testing: Conduct penetration testing periodically to identify emerging vulnerabilities before attackers can exploit them.
4. Implement Security Controls: Apply additional controls such as WAFs, API Gateways, and DDoS protection to strengthen the application's security posture.