MLA// EXTENDED ASSESSMENT TASK

Please make a copy of this document

NAME OF SESSION	DevSecOps
STUDENT NAME	Funkeyi Jessica Omoro

ASSESSMENT FOCUS	Secure Development - Application Task

Scenario:

You are a developer on a team building a new e-commerce platform for a clothing retailer. The application handles sensitive customer data, including payment information and personal details. Your team is responsible for implementing security measures to protect this data and ensure the application's overall security.

Task:

1. **Shift-Left Security:** Explain how the concept of "shift-left security" applies to the development of this e-commerce platform. Describe specific security practices you would integrate into each stage of the development lifecycle (e.g., design, coding, testing, deployment) to ensure a secure application.

2. **Vulnerability Awareness:** Based on the lesson, discuss the key security vulnerabilities developers should be aware of when building an e-commerce platform. Explain how these vulnerabilities could be exploited and their potential impact on the application and its users.

3. **SAST and DAST:** Describe how Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) can be used to improve the security of this e-commerce platform. Explain the differences between these testing approaches and provide examples of tools that could be used for each.

1. Shift-Left Security
Shift-left security means integrating security practices early in the development lifecycle rather than treating it as a final checkpoint. For our e-commerce platform, this approach is critical given the sensitive payment and personal data we handle.
Design Phase: Security begins with threat modelling - identifying potential attack vectors like payment data interception, account takeover, and SQL injection. We would architect the system with security principles like least privilege access, data encryption at rest and in transit, and secure authentication mechanisms. Designing separate microservices for payment processing isolates sensitive operations from the main application.
Coding Phase: Developers follow secure coding standards - sanitizing all user inputs, using parameterized queries to prevent SQL injection, and implementing proper authentication/authorization checks. Code reviews specifically examine security

implications, and we use linting tools that flag common security issues like hardcoded credentials or insecure cryptographic functions.
Testing Phase: Security testing runs alongside functional testing - automated scans check for vulnerabilities, penetration testing simulates attacks, and we validate encryption implementations. Every API endpoint undergoes security validation before merging.
Deployment Phase: Automated security scans run in CI/CD pipelines, blocking deployments if critical vulnerabilities are detected. We implement secrets management to keep API keys and credentials secure, use HTTPS everywhere, and configure security headers. This comprehensive approach catches issues early when they're cheaper and easier to fix.

2. Vulnerability Awareness
When building an e-commerce platform, developers must guard against several critical vulnerabilities. **SQL Injection** occurs when attackers insert malicious SQL code into input fields, potentially exposing the entire customer database. For example, entering **' OR '1'='1** in a login form could bypass authentication if queries aren't properly parameterized. The impact is catastrophic - attackers could steal payment details, personal information, and compromise thousands of accounts.
Cross-Site Scripting (XSS) allows attackers to inject malicious JavaScript into pages viewed by other users. A compromised product review could steal session cookies, redirecting users to fake payment pages that harvest credit card details. This erodes customer trust and exposes sensitive data.
Insecure Direct Object References happen when applications expose internal object identifiers in URLs like **/order/12345**. Attackers could modify these IDs to access other customers' order histories and payment information, violating privacy and compliance regulations.
Authentication and Session Management flaws include weak password policies, predictable session tokens, and lack of multi-factor authentication. Attackers exploiting these weaknesses gain unauthorized access to customer accounts, making fraudulent purchases or stealing stored payment methods.
Sensitive Data Exposure occurs when payment information, passwords, or personal details aren't properly encrypted during storage or transmission. Man-in-the-middle attacks on unencrypted connections could intercept credit card numbers in transit. These vulnerabilities not only harm customers but expose the business to massive financial penalties, legal liability, and reputational damage.

3. SAST and DAST
Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) provide complementary security analysis for our e-commerce platform. **SAST** analyses source code without executing it, identifying vulnerabilities during development. It examines code for patterns like SQL injection vulnerabilities, hardcoded credentials, insecure cryptographic implementations, and improper input validation. SAST tools integrate into IDEs and CI/CD pipelines, catching issues before code reaches production. Example tools include **SonarQube**, which scans for security hotspots and code quality issues, **Checkmarx**, which identifies vulnerabilities across multiple languages, and **Snyk**

MLA// EXTENDED ASSESSMENT TASK

Please make a copy of this document

Code, which provides real-time security feedback during coding.
DAST, conversely, tests the running application from an external perspective, simulating real attacker behaviour without access to source code. It identifies runtime vulnerabilities like authentication bypass, session management flaws, and configuration errors that only appear when the application is operational. DAST tools send malicious inputs to APIs and forms, testing how the system responds. Example tools include **OWASP ZAP**, an open-source scanner for web application vulnerabilities, **Burp Suite**, which performs comprehensive penetration testing, and **Acunetix**, which specializes in detecting SQL injection and XSS vulnerabilities.
The key difference: SAST is "white-box" testing examining code structure, while DAST is "black-box" testing attacking the running application. Using both provides comprehensive coverage - SAST catches coding errors early and cheaply, while DAST validates security in the production-like environment, finding configuration and runtime issues SAST might