Lab 2: Designing a Secure Development Lifecycle (SDLC)

Topic: Application Security & DevSecOps

Duration: 60 Minutes

The Scenario

Your startup is releasing a new feature. You must define your new feature to your product of your company.

The Engineering Team is moving fast and deploying code to production daily. Instead of manually checking every line of code, you must design a **DevSecOps Pipeline** that automates security.

Part 1: The Pipeline Architecture (20 Minutes)

Objective: Map security controls to the Software Development Lifecycle (SDLC).

Instructions:

You have a standard CI/CD pipeline with the following 5 stages. **Assign ONE security tool/process from the "Toolbox" to the correct stage where it adds the most value.** (You can use a tool more than once if necessary, but try to find its *primary* home).

The Pipeline Stages:

1. **IDE / Coding:** (Developer writing code on their laptop).
2. **Commit / Build:** (Code is pushed to GitHub and compiled).
3. **Test / Staging:** (Running the app in a mock environment).
4. **Deploy / Production:** (The app is live for customers).
5. **Monitor:** (Day-to-day operations).

The Toolbox:

- *SAST (Static Analysis)*
- *DAST (Dynamic Analysis)*
- *SCA (Software Composition Analysis / Dependency Check)*
- *Secrets Scanning (checking for API keys)*
- *WAF (Web Application Firewall)*
- *Penetration Testing*
- *IDE Security Plugin (Spellcheck for code)*

Deliverable: Draw the pipeline on paper or in a diagramming tool and label where each tool goes.

Part 2: Mini-Threat Model (20 Minutes)

Objective: Apply the **STRIDE** methodology to a new feature.

Instructions:

Using the **STRIDE** model, identify **THREE** potential security threats to this feature and propose **ONE** fix for each.

STRIDE Category	The Threat (What could go wrong?)	The Fix (Control)
S - Spoofing	Example: A user uploads a photo for someone else's account.	Fix: Strict session validation on upload.
T - Tampering		
I - Information Disclosure		
D - Denial of Service		
E – Elevation of Privilege		

(Hint: What happens if I upload a 10GB file? What happens if I upload a virus renamed as 'photo.jpg'?)

Part 3: Vulnerability Triage Simulation (15 Minutes)

Objective: Prioritize bugs based on Risk (Likelihood x Impact), not just technical severity.

The Situation:

Your scanners just finished running. You have **limited resources** (only 1 developer available to fix bugs today). You found the following 3 vulnerabilities.

The Bugs:

1. **Bug A (Critical - CVSS 9.8):** A "Remote Code Execution" vulnerability in a 3rd-party library (Log4j). However, this library is only used in an *internal* tool that is not accessible from the internet.
2. **Bug B (High - CVSS 7.5):** A "Reflected XSS" vulnerability on the *public* login page. An attacker could trick users into clicking a link and stealing their session cookies.
3. **Bug C (Medium - CVSS 5.0):** The server is revealing its version number in the HTTP headers (Information Disclosure).

Instructions:

1. **Rank them** in order of priority (1 = Fix Now, 3 = Fix Later).
2. **Justify your #1 choice.** Why did you pick it over the others?