Threat Modeling & Static Analysis Review

# 1. Introduction

Threat Modeling and Static Analysis are two crucial components of Secure Software Development Life Cycle (SSDLC). Threat Modeling helps in identifying potential security threats during the design phase, while Static Analysis involves analyzing source code or binaries to detect vulnerabilities without executing the program.

# 2. Objectives

- Understand the importance of Threat Modeling and Static Analysis in application security.  
- Explore real-time examples and techniques.  
- Demonstrate integration of static analysis tools into CI/CD pipelines.  
- Discuss common findings and mitigation strategies.

# 3. Threat Modeling Overview

Threat Modeling is a proactive approach to identifying potential threats and designing countermeasures. It involves understanding what you're building, identifying security risks, and addressing them early.

Common frameworks:

- STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege)  
- DFDs (Data Flow Diagrams) to visualize data movement  
- Attack Trees and Risk Matrices

# 4. Real-Time Threat Modeling Example

Example: An online banking system  
- Assets: User credentials, transactions, PII  
- Entry Points: Login page, API endpoints  
- Threats:  
 \* Spoofing login sessions  
 \* Tampering with request data  
 \* Information Disclosure through verbose errors  
- Mitigations:  
 \* Enforce MFA  
 \* Input validation  
 \* Secure error handling

# 5. Static Analysis Overview

Static Analysis involves inspecting source code, bytecode, or binaries for security vulnerabilities without executing the program. It helps in identifying coding flaws, security misconfigurations, and dangerous functions early in development.

Popular tools include: SonarQube, Fortify SCA, Checkmarx, Semgrep, and ESLint (for JavaScript).

# 6. Real-Time Static Analysis Example

Example: Node.js Web Application  
- Tool: ESLint with security plugins  
- Finding: Use of `eval()` function  
- Risk: Code injection vulnerability  
- Fix: Replace `eval()` with safe alternatives like JSON.parse or template engines

# 7. Integration with CI/CD

Static analysis tools can be integrated into CI/CD pipelines to automate security checks:  
- Stage 1: Code checkout  
- Stage 2: Static scan using tools like SonarQube or Semgrep  
- Stage 3: Fail build on high-severity findings  
- Stage 4: Generate and email scan reports to developers

# 8. Common Vulnerabilities Found

- SQL Injection  
- Cross-Site Scripting (XSS)  
- Hardcoded secrets or credentials  
- Insecure APIs  
- Misconfigured permissions or access controls

# 9. Recommendations & Best Practices

- Use threat modeling early in SDLC  
- Conduct regular static code reviews  
- Integrate static analysis tools with version control and CI/CD  
- Train developers on secure coding practices  
- Track and fix findings with JIRA or other ticketing tools

# 10. Conclusion

Combining Threat Modeling with Static Analysis provides a powerful defense against software vulnerabilities. By identifying and fixing security issues early, teams can ensure robust, secure applications with reduced risk and faster delivery.