Module 05

Purwadhika
Digital Technology School

AI Fullstack Software Development

Secure Coding Practices

Job Connector Program

Outline

- Secure Coding
- SAST & DAST
- SonarQube, OWASP ZAP, Trivy

What Is Secure Coding?

- Secure coding is the practice of writing code that protects applications from vulnerabilities and attacks.
- The goal: **prevent unauthorized access**, **data leaks**, and **system compromise**.
- Security should be built into the development process — not added later.

Why Secure Coding Matters

- Over 70% of breaches are caused by exploitable software vulnerabilities.
- Hackers often target coding mistakes like input validation or authentication flaws.
- Secure code reduces risks, maintenance costs, and reputation damage.

What Is OWASP?

- **OWASP (Open Web Application Security Project)** is a nonprofit organization dedicated to improving web application security.
- Provides resources, tools, and standards for secure development.
- Community-driven and widely adopted globally.

https://owasp.org/

OWASP Top 10 Overview

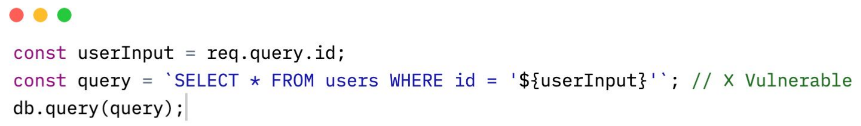
No	OWASP Risk	Meaning	Example Scenario
1	Broken Access Control	When users can perform actions outside their allowed permissions.	A normal user modifies a request to access /admin/dashboard.
2	Cryptographic Failures	Sensitive data is not properly encrypted or is sent insecurely.	Passwords or credit card data are sent over HTTP instead of HTTPS.
3	Injection	Untrusted input allows attackers to execute unintended commands or queries.	SQL Injection: ' OR 1=1 -- lets attacker bypass login.
4	Insecure Design	Security flaws exist in the app's design phase, not just the code.	A money transfer system has no limit on transfer retries or alerts.
5	Security Misconfiguration	Improper configuration exposes sensitive data or admin interfaces.	Default admin panel /admin left open to the public.

OWASP Top 10 Overview

No	OWASP Risk	Meaning	Example Scenario
6	Vulnerable & Outdated Components	Using old libraries or software with known vulnerabilities.	App still uses old Log4j version vulnerable to RCE.
7	Identification & Authentication Failures	Weak login/session handling lets attackers hijack accounts.	Session token remains valid even after logout.
8	Software & Data Integrity Failures	No integrity check on updates or code dependencies.	App installs plugins over HTTP, allowing malicious tampering.
9	Security Logging & Monitoring Failures	Attacks go unnoticed due to missing logs or alerts.	Brute-force login attempts not logged or detected.
10	Server-Side Request Forgery (SSRF)	App fetches user-supplied URLs without validation.	Attacker makes server request internal endpoint http://localhost/admin.

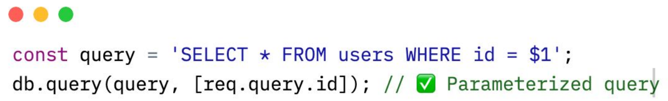
OWASP Example : Injection

Injection occurs when untrusted data is sent to an interpreter.



```
const userInput = req.query.id;
const query = `SELECT * FROM users WHERE id = '${userInput}'`; // X Vulnerable
db.query(query);
```

❌Vulnerable version

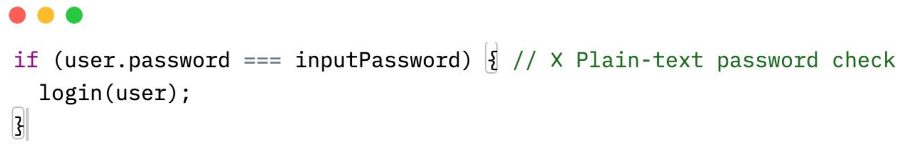


```
const query = 'SELECT * FROM users WHERE id = $1';
db.query(query, [req.query.id]); // ✅ Parameterized query
```

✅Secure version : use parameterized query

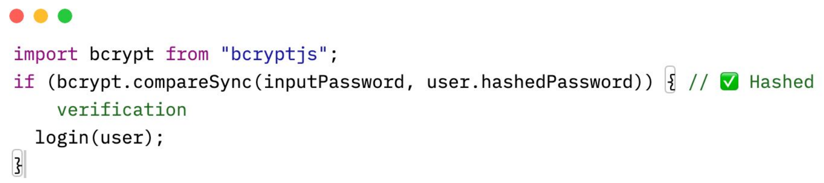
OWASP Example : Broken Authentication

Occurs when authentication is improperly implemented.



```
if (user.password === inputPassword) { // X Plain-text password check
  login(user);
}
```

❌Vulnerable version



```
import bcrypt from "bcryptjs";
if (bcrypt.compareSync(inputPassword, user.hashedPassword)) { // ✅ Hashed
    verification
  login(user);
}
```

✅Secure version : use bcrypt for hashing password

Static Application Security Testing (SAST)

- Analyzes source code without executing it.
- Detects vulnerabilities early in development.
- Integrates with CI/CD pipelines and IDEs.
    - **Example Tools: SonarQube, Checkmarx, Fortify.**
    - **Analogy: Like proofreading your code before running it.**

Dynamic Application Security Testing (DAST)

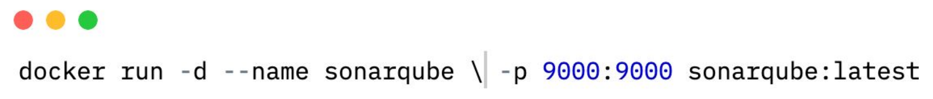
- Tests a running application from the outside.
- Simulates attacks to find runtime vulnerabilities.
- Useful for identifying configuration or deployment issues.
  - **Example Tools: OWASP ZAP, Burp Suite.**
  - **Analogy: Like ethical hacking your own app.**

SonarQube Overview

- SonarQube is a popular open-source SAST tool.
- Detects bugs, vulnerabilities, and code smells.
- Supports TypeScript, Java, Python, and many others.
- Integrates easily with CI/CD pipelines (GitHub Actions, Jenkins, etc.).

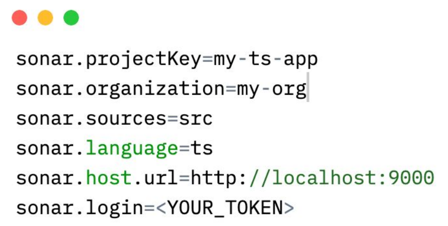
SonarQube Setup

- Run SonarQube locally (Docker): → Access dashboard at: http://localhost:9000



```
docker run -d --name sonarqube \ -p 9000:9000 sonarqube:latest
```

- Login (default: admin / admin) ➜ create a project & token.
- Install scanner locally: **npm install -g sonarqube-scanner**
- Add configuration (sonar-project.properties):



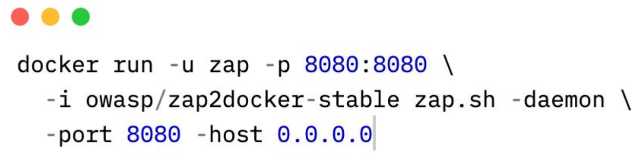
```
sonar.projectKey=my-ts-app
sonar.organization=my-org
sonar.sources=src
sonar.language=ts
sonar.host.url=http://localhost:9000
sonar.login=<YOUR_TOKEN>
```

OWASP ZAP Overview

- OWASP ZAP (Zed Attack Proxy) is an open-source DAST tool.
- Can scan web apps for vulnerabilities such as **XSS, SQL injection, CSRF.**
- Provides automated and manual testing modes.
- Often used in DevSecOps pipelines.

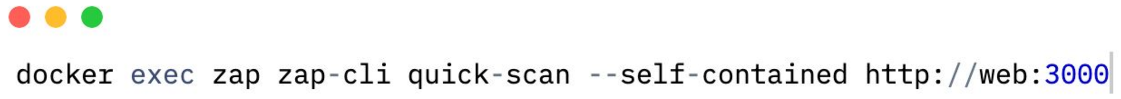
OWASP ZAP Setup with Docker

- Run OWASP ZAP container:



```
docker run -u zap -p 8080:8080 \
  -i owasp/zap2docker-stable zap.sh -daemon \
  -port 8080 -host 0.0.0.0
```

- Scan your running web app:



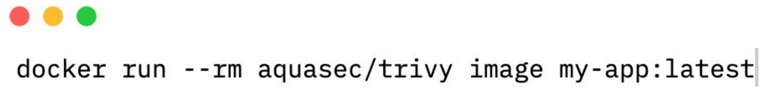
```
docker exec zap zap-cli quick-scan --self-contained http://web:3000
```

Trivy Overview

- Trivy is a vulnerability scanner for containers, filesystems, and dependencies.
- Detects CVEs in Docker images and application libraries.
- Useful for DevOps and secure deployment pipelines.

Trivy Setup with Docker

- Scan Docker images with Trivy:



```
docker run --rm aquasec/trivy image my-app:latest
```

-> Scans OS and dependency vulnerabilities.

Demo

1. Run this app : https://github.com/BagasDhitya/puzzle-todo-vulnerable
2. Trigger github actions for see results

Thank you