DYNAMIC APPLICATION SECURITY TESTING

DevSecOps Basics

What is Dynamic Application Security Testing?

The Dynamic Application Security Analysis (DAST) is performed to identify the possible run-time vulnerabilities or security issues. Unlike static analysis, dynamic analysis is performed on a running project.

The following components are there in this phase:

- DAST tools i.e. OWASP ZAP, BDD Security, Arachini, Nikto, Radamsa, FuzzDB

People involved: Developers and Testers

External sources

- What is Dynamic Application Security Testing? https://www.synopsys.com/software-integrity/application-security-testing-services/dynamic-analysis-dast.html
- Static vs Dynamic Code Analysis https://blog.overops.com/static-vs-dynamic-code-analysis-how-to-choose-between-them/

Why is it important in DevSecOps?

The Dynamic Application Security Testing can unravel the runtime-security issues before project/application deployment (or test deployment). DAST can be automated using tools and can be added to the DevSecOps pipeline for continuous security testing.

What will you learn in this section?

The user will learn to perform the following tasks

- Perform the dynamic analysis on provided web applications for issues

Tools Covered

- OWASP ZAP
- BDD Security
- Arachini
- Nikto
- Radamsa
- FuzzDB

Labs

- OWASP ZAP: Detecting Vulnerabilities in WebApps
  - A Kali machine is provided to the user with OWASP ZAP available on it. Three example of vulnerable web portals are also provided.
    Objective: Analyze the web applications with OWASP ZAP and identify the vulnerabilities!
- BDD Security: Behaviour Driven Development
  - A Kali machine is provided to the user with BDD-Security on it. The source code for the web application is provided in the home directory of the root user. The WebGoat instance can be reached at the 'test-server' endpoint at port 8080.
    Objective: Scan BDD security to find issues in the code!
- Arachini: Automated Vulnerability Scanning
  - The Arachni instance is running in the lab. Three example of vulnerable web portals are also provided.
    Objective: Identify the vulnerabilities in web applications using Archini.
- Nikto: Automatic WebApp Scanning

Objective: Identify the vulnerabilities in web applications using Nikto!

- Radamsa: Automated Fuzzing
  - A Kali machine is provided to the user with Radamsa installed on it. The two sample web applications are provided in the home directory of the root user. The user will run these applications and then test these with Radamsa.
    Objective: Run Radamsa on provided web applications and find issues!
- FuzzDB: Fault Injection Testing
  - A Kali machine with OWASP ZAP and FuzzDB ZAP plugin is available to the user. A vulnerable web portal is also provided for testing.
    Objective: Use the OWASP ZAP tool with FuzzDB to identify issues in the web application!

OWASP ZAP: Detecting Vulnerabilities in ...

Start

Arachini: Automated Vulnerability Scanning

Start

Nikto: Automatic WebApp Scanning

Start

Radamsa: Automated Fuzzing

Start

FuzzDB: Fault Injection Testing

Start

BDD Security: Behaviour Driven Development

Start

Prog Pilot: Statically Scanning PHP Code

Start