Testing Your Code

Marcin Hoppe

@marcin_hoppe marcinhoppe.com

Overview

- Automated security testing
- Preventing use of unsafe functions
- Detecting prototype pollution
- Detect vulnerable third-party libraries

Detect vulnerabilities

Prevent from being added to the code

Scale

- Large code bases
- Multiple developers
- Fast feedback

Security Testing Techniques

SAST

Static application security testing

DAST

Dynamic application security testing

IAST

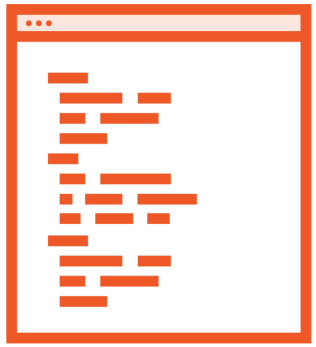
Interactive application security testing

Techniques Compared

SAST	DAST
Source code	Running application
Known bad code patterns	Malicious payloads
Safe	May be destructive
Compilers, linters, scanners	Automated tests and scanners

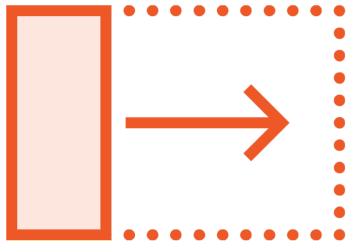
Finding Unsafe Code Using ESLint

Linters are lightweight code analysis tools that flag bugs and coding errors



Code

Parse code and analyze the abstract syntax tree (AST)



Extensible

Comes with a set of bundled checks

Fast

Can be used by IDEs and build pipelines

Demo

Basics of ESLint

- Installation
- Rule configuration

Preventing use of `eval`

Reliable and repeatable

Easy payload delivery

Inspect program state after attack

- Input violating assumptions about types
- Code injection effects
- Detect prototype pollution

Demo

Running tests with Mocha

Detect prototype pollution

- Payload
- Detect successful attack

SAST

- ESLint
- GitHub Code Scanning and LGTM
- semgrep

OWASP ZAP

- Many commercial alternatives

Dependency management

- npm audit
- Retire.js
- Dependency-Track
- Snyk

Demo

Dependency management

Detect vulnerable library with `npm audit`

Summary

Security testing techniques

- SAST
- DAST
- IAST

Preventing vulnerabilities with automated tests

- ESLint
- Unit tests
- npm audit