

SECURE CODE

VILLAGE

Presented by
Gaurav Bhosale
Hare Krishna Rai



Introduction

Secure Code Village is a hands-on security learning space focused on helping developers and security professionals build secure software from the ground up. Through workshops, live demos, and real-world challenges, we promote a shift-left mindset—embedding security early in the development lifecycle. Whether you're just starting out or deep into DevSecOps, Secure Code Village is where code meets security, practically.

About us



Gaurav Bhosale

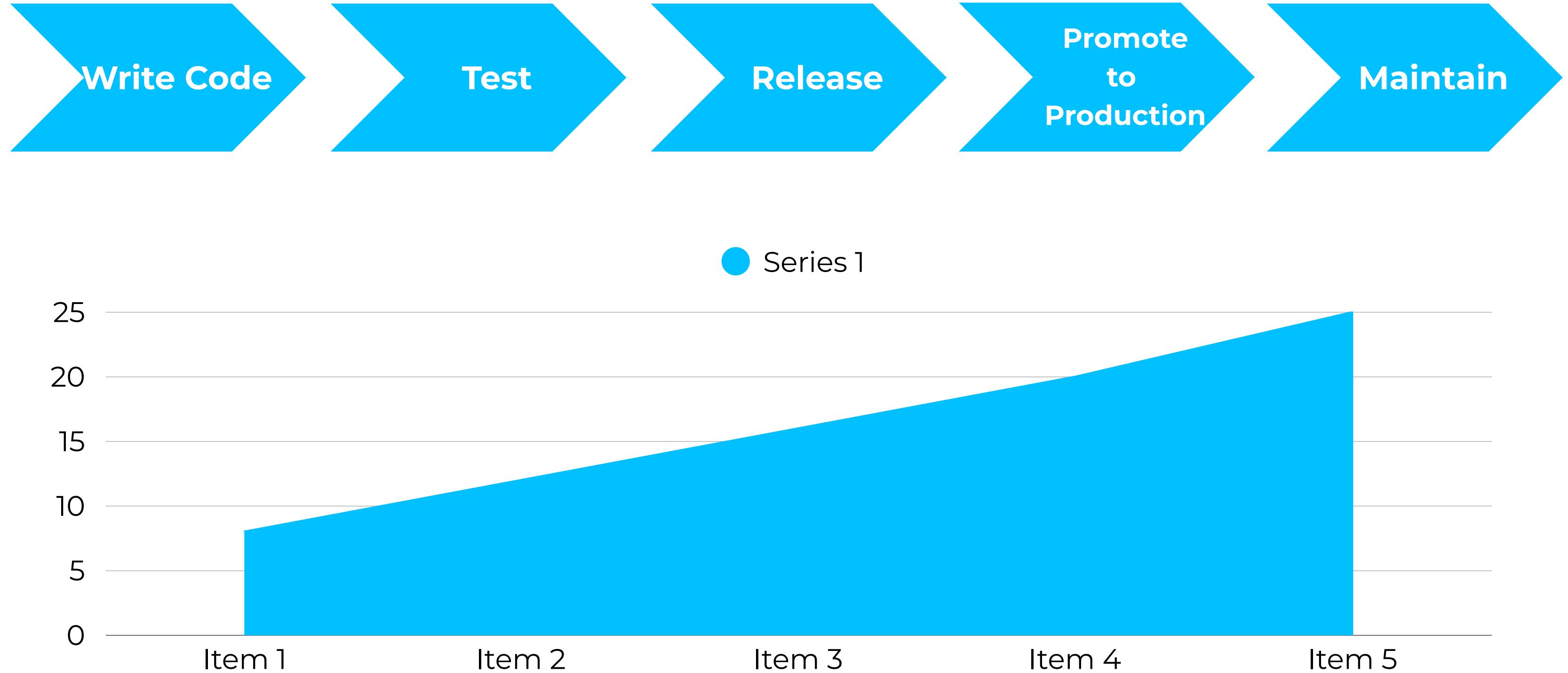
Application Security Enginer
| Ex-mastercard | Ex-Payatu
Speaker



Hare krishna Rai

Security Engineer @ Okta
Arsenal @Blackhat Asia, Europe

Catching Problem Early Means Less People Impacted



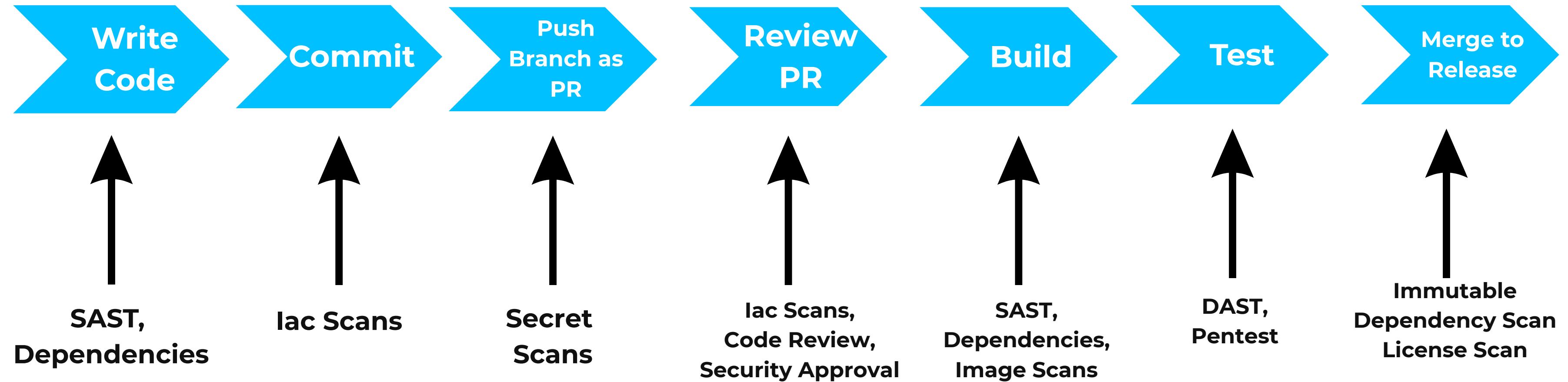
Typical Development Process



Security Integration Points

Integration Point	Integration
Developer IDE	Ability to run SAST & Dependency Scan
Commit Hook	Platform Iac Scanning & Linting
Push Hook	Secret Scanning (Prevents Secret reaching the git history)
Pull Request	Microservice Iac Scanning, code Review, Code Owner for Security Approval
Repository	Secret Scanning
Build Pipeline	SAST (Block High & Medium), Dependency Scan (Block Critical, High & Medium)
Release Process	Immutable Dependency & license Scans
Test Process	DAST Scan, Pentesting

Security Test Integration



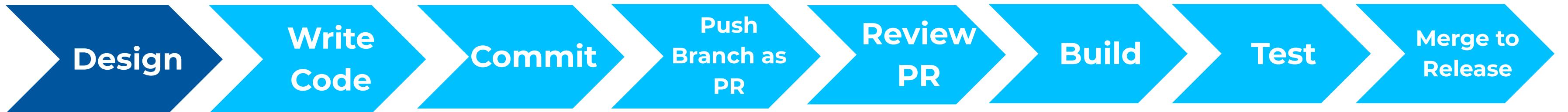
Is this is good?

- its Comprehensive
- Multiple check
- Includes both early advisory checks and later governance checks

But.....

- Only addresses implementation issues not design or concept
- Missing controls only discovered at the end during pentest
- Fixing Problem rather than preventing them

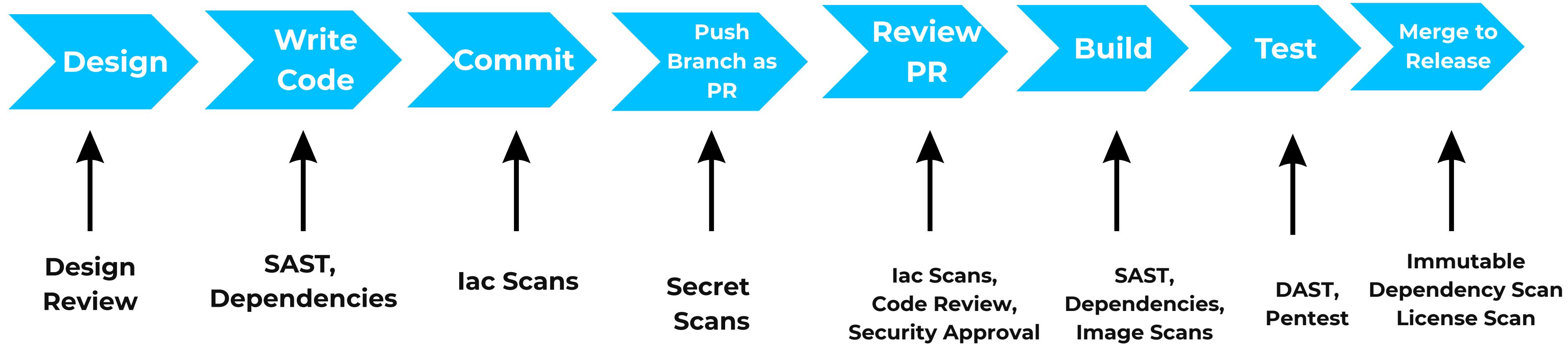
Considering Security During Design



Design Reviews

- Find problems Before implementation
- Can identify missing control early

Security Integration with Design Review



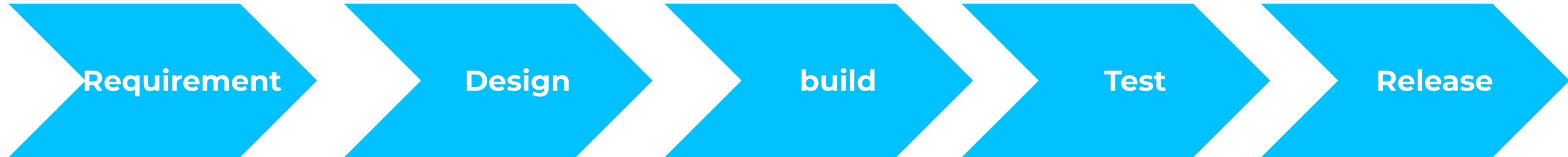
Definitely Better

- All the controls we had before
- Design review Caught problem earlier
- Now identify missing control ealry

BUT...

- Still fixing problems rather than preventing them

Introducing Threat Modelling

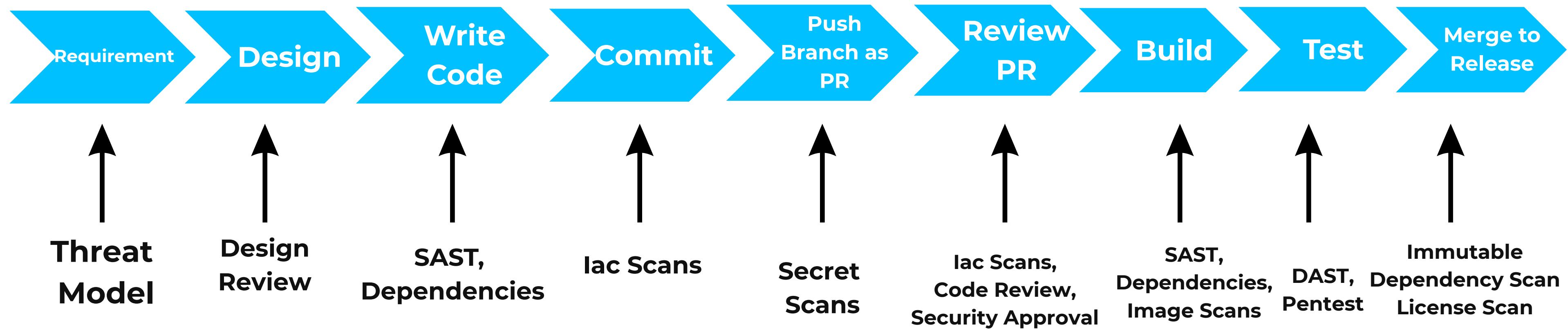


- Consider threats as early as possible
- Can be done before the design to help identify concerns

Light Weigh Threat Modeling

- Anyone can do it
 - Security Champion can facilitate
 - Living Document Not stable Spreadsheets
 - Bring Value much more early
 - No Secret Sauce, it is easy
-
- It's not rocket science - just good ol' overthinking, like my ex at a shopping mall.

End to End Security Integration





Mujhe break lena hai

A DEEP DIVE INTO STATIC APPLICATION SECURITY TESTING

Static Application Security Testing

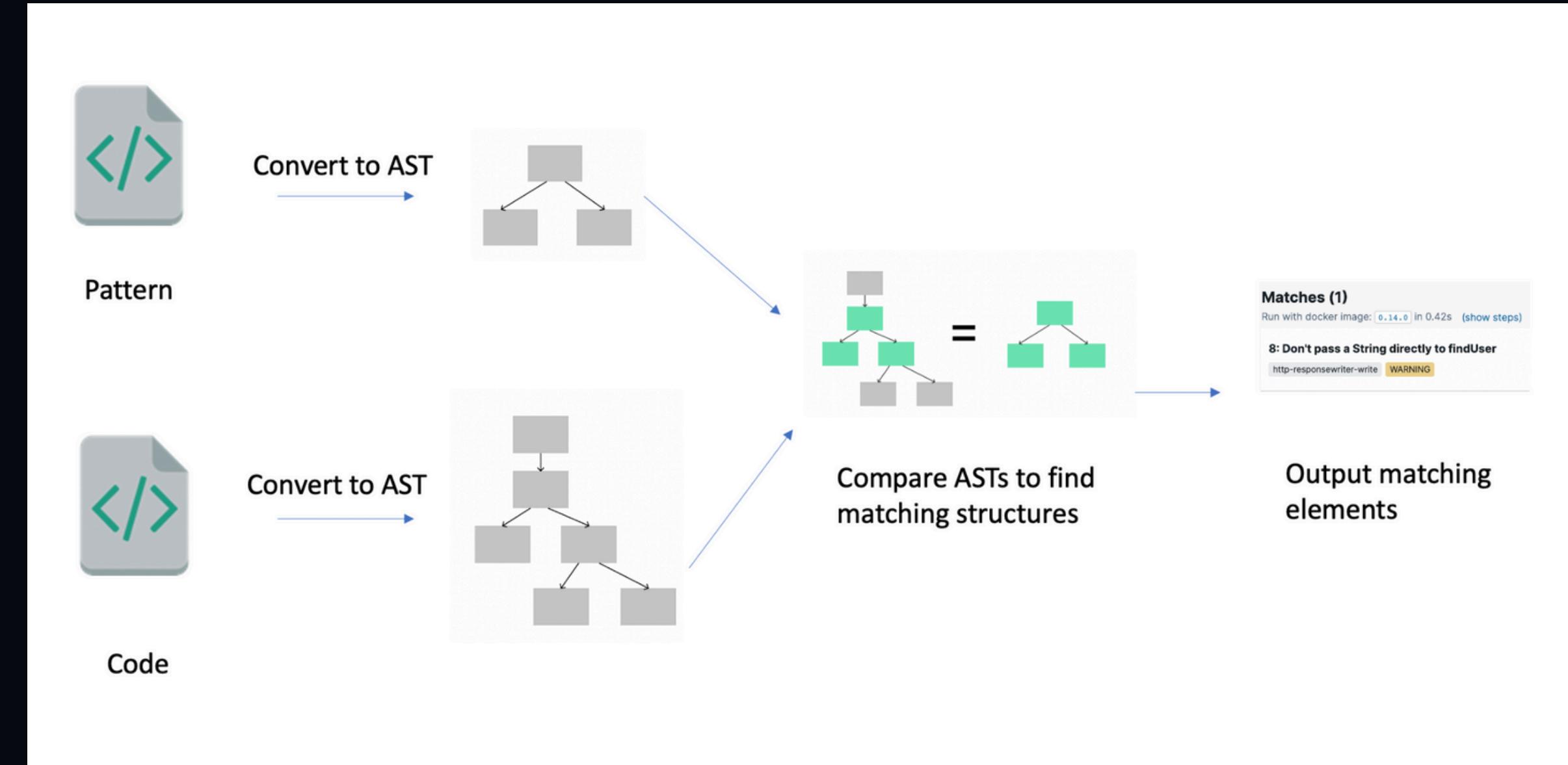
- Analyzes Source Code without execution
- Originally meant for Developers
- Most of OWASP vulns are found at code Level
- Cost effective
- Early identification of Vulnerabilities Various stages of Implementation

HOW is SAST used to be done?

How can you leverage grep,git grep for the SAST in your manual code review?

Abstract syntax tree

- What is AST?
- How AST is done?
- how it benefits SAST tools to identify the vulnerabilities?



How are the different tools different from each other in terms of their working or analysis mechanisms - like checkmarx, fortify, semgrep, opengrep?



Basics of Semgrep - Semgrep Rules?

The screenshot shows the Semgrep Editor interface with the following layout:

- Left Sidebar:** Contains links for Dashboard, Projects, Code, Secrets, Supply Chain, Rules (selected), Policies, Editor (selected), Code Search beta, Registry, Feedback, Settings, Docs, Help, Admin, and a logo.
- Top Bar:** Shows the project name "semgrep" and the current tab "Library".
- Library Tab:** Includes a search bar ("e.g.: python.flask"), a "Create new rule" button with a plus sign icon, a "Recent" section listing a rule named "c.lang.correctness.goto-fail.double_goto" created "2 months ago", and sections for "Examples" (Semgrep vs. grep, Unchecked subprocess return code, Using Semgrep's metavariables) and "Learn" (Tutorial, Text mode, Docs).

SEMGREP RULES CONSTRUCTS

Construct	Description	Example
...	Any number of arguments/statements	func(..., x=1, ...)
\$_	Any single argument/expression	print(\$_)
\$X	Capture a value for reuse	\$FUNC(\$ARG)
pattern-not	Exclude a pattern	Exclude os.system("safe")
pattern-inside	Match only within some code	Inside a def block
pattern-regex	Match via regex instead of AST	pattern-regex: exec\s*\(

SEMGREP RULES EXAMPLES FROM THE PORTAL - DEMO

A DEEP DIVE INTO SOFTWARE COMPOSITION ANALYSIS

“With great power comes great responsibility... and great vulnerabilities”

TABLE OF CONTENTS

.....

securecodevillage.com

01

Introduction

What is Software Composition Analysis? Why do we need it?

02

Types of SCA Scanning & Tools

Understanding Different SCA Approaches
SCA scanning can be categorized into several types based on what and how they analyze

05

Future of OSS Security

AI Driven SCA, Zero Trust Supply chains, Regulatory Pushes

03

The Open Source Boom

Why OSS is Everywhere

04

Mitigation Strategies

Planning and mitigation strategies for the SCA detected issues

06

Conclusion & Q/A

Conclusion on whats next and summing up

INTRODUCTION

Open Source Software (OSS) powers modern applications, from web frameworks to AI models. But with its widespread adoption comes a hidden cost: vulnerabilities.

- 80% of modern codebases contain OSS components.
- OSS vulnerabilities can lead to catastrophic breaches.
- Software Composition Analysis (SCA) is critical for identifying and mitigating risks.

Today, we'll explore OSS vulnerabilities, SCA challenges, and how to secure your software supply chain.

WHY OSS IS EVERYWHERE ?

- Cost-effective: Free to use and modify.
- Community-driven: Rapid development and innovation.
- Flexibility: Integrates into any stack.

WHAT IS SOFTWARE COMPOSITION ANALYSIS (SCA)?

SCA tools identify and manage OSS components in a codebase, tracking:

- Dependencies and their versions.
- Known vulnerabilities (e.g., CVE database).
- License compliance.

Why it matters: Manual tracking is impossible in complex projects.

TYPES OF SCA SCANNING

SCA scanning can be categorized into several types based on what and how they analyze:

- Manifest Scanning: Analyzes package manifest files (e.g., package.json, pom.xml) to identify declared dependencies.
- Binary Scanning: Examines compiled binaries to detect embedded OSS components, useful when source code isn't available.
- Snippet Scanning: Looks for code snippets that match known OSS code, even if not declared as dependencies.
- Reachability Scanning: Determines if vulnerable code in dependencies is actually used by the application, prioritizing exploitable vulnerabilities.

Each type offers unique insights, and advanced SCA tools often combine multiple approaches for comprehensive analysis.

EXAMPLE - MANIFEST SCANNING

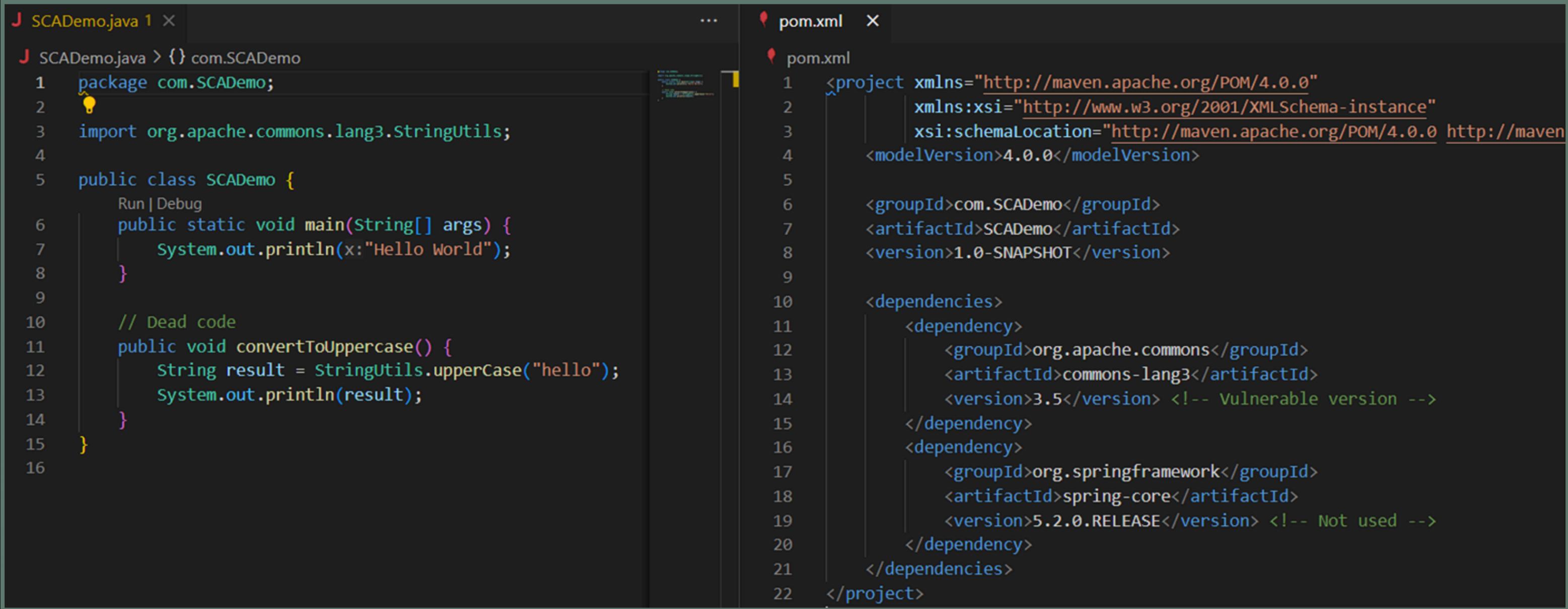
The screenshot shows a code editor interface with two files open:

- package.json**:
A JSON file defining the project metadata. It includes fields for name, version, description, main file, scripts (start command), dependencies (express, lodash, moment), and devDependencies (jest). The dependencies section has annotations: lodash and moment are marked as "Unused".
- app.js**:
A Node.js script that imports express and uses it to create an application. It defines a get route for '/' that returns "Hello, World!". It also listens on port 3000 and logs a message to the console.

A sidebar on the right lists the unused imports found in the code:

- // Unused lodash
- // Unused moment
- // Unused jest

EXAMPLE - REACHABILITY SCANNING



The screenshot shows a Java development environment with two files open:

- SCADemo.java**: A simple Java class with a main method that prints "Hello World". It also contains a commented-out method named convertToUppercase that uses the org.apache.commons.lang3.StringUtils library.
- pom.xml**: A Maven project configuration file. It specifies the project's group ID, artifact ID, and version. It lists dependencies for org.apache.commons/commons-lang3 (version 3.5) and org.springframework/spring-core (version 5.2.0.RELEASE). The commons-lang3 dependency is marked as vulnerable, while the spring-core dependency is noted as not used.

```
SCADemo.java
package com.SCADemo;
import org.apache.commons.lang3.StringUtils;
public class SCADemo {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
    // Dead code
    public void convertToUppercase() {
        String result = StringUtils.upperCase("hello");
        System.out.println(result);
    }
}
```

```
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.SCADemo</groupId>
    <artifactId>SCADemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.5</version> <!-- Vulnerable version -->
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.2.0.RELEASE</version> <!-- Not used -->
        </dependency>
    </dependencies>
</project>
```

SCA TOOLS

 ENDOR LABS



 Checkmarx



 Semgrep



 Kodem

dependency-check/
DependencyCheck

OWASP dependency-check is a software composition analysis utility that detects publicly disclosed vulnerabilities in application dependencies.

302 Contributors 25 Used by 7k Stars 1k Forks

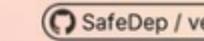


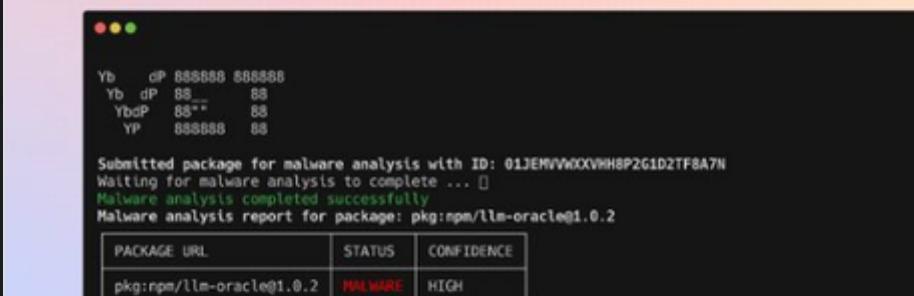
dependency-check/DependencyCheck: OWASP dependency-check is a software composition analysis...

OWASP dependency-check is a software composition analysis utility that detects publicly disclosed vulnerabilities in application dependencies. - dependency-check/DependencyCheck

[GitHub](#)

google/osv-scanner

 Protect against Malicious Code



safedep/vet: Next Generation Software Composition Analysis (SCA) with Malicious Package Detection, Code...

Next Generation Software Composition Analysis (SCA) with Malicious Package Detection, Code Context & Policy as Code - safedep/vet

[GitHub](#)

WHAT ARE THE COMMON ISSUES IN SCA?

- New Vulnerabilities : CVEs identified in the open source package.
- Outdated dependencies: Unpatched vulnerabilities persist.
- Malicious packages: Typosquatting and supply chain attacks.
- Unmaintained projects: No updates, no fixes.
- Transitive dependencies: Hidden risks in nested libraries.

In 2024, 70% of OSS vulnerabilities were in transitive dependencies.

REAL-WORLD OSS DISASTERS

The image displays three overlapping screenshots of news articles from ZDNet, Synopsys, and Akamai, illustrating significant security incidents involving open-source software.

ZDNET Article:

Equifax blames open-source software for its record-breaking security breach: Report

The credit rating giant claims an Apache Struts security hole was the real cause of its security breach of 143 million records. ZDNet examines the claim.

Synopsys Article:

Equifax, Apache Struts, and CVE-2017-5638 vulnerability

It's an Equifax breach / Apache Struts / CVE-2017-5638 vulnerability issue of Open Source Insight this week as we examine how an unpatched open source flaw and an apparent lack of diligence exposed sensitive data for over 140 million US consumers. We discuss what happened, how to check whether you've been affected by the breach, and whether you should replace Struts with another framework.

Akamai Article:

XZ Utils Backdoor – Everything You Need to Know, and What You Can Do

Akamai Security Intelligence Group
April 01, 2024

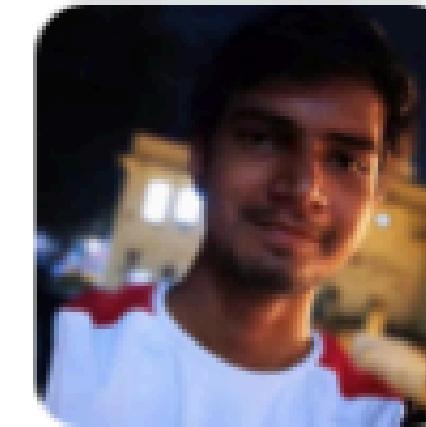
DAMN-VULNERABLE-SCA (SCAGOAT)

The Damn Vulnerable SCA project is a deliberately vulnerable application designed to help users understand and test SCA tools.

- Purpose: Provides a safe environment to learn about OSS vulnerabilities and how SCA tools detect them.
- Features: Includes known vulnerabilities like Log4Shell, malicious packages, and more.
- Usage: Clone the repo, build the Docker image, and run SCA scans to see vulnerabilities in action.

Why it matters: Hands-on experience helps security professionals and developers better understand SCA outputs and remediation steps.

harekrishnarai/Damn-vulnerable-sca



Damn Vulnerable SCA Application

6

Contributors

1

Issue

36

Stars

32

Forks



harekrishnarai/Damn-vulnerable-sca: Damn Vulnerable SCA Application

Damn Vulnerable SCA Application. Contribute to harekrishnarai/Damn-vulnerable-sca development by creating an account on GitHub.



GitHub

MITIGATION STRATEGIES

- Automate SCA: Integrate into CI/CD pipelines.
- Patch promptly: Use tools like Dependabot for auto-updates.
- Minimize dependencies: Audit and remove unused libraries.
- Monitor actively: Subscribe to vulnerability feeds (e.g., NVD).
- Educate teams: Train developers on secure powołanie and dependency management.

FUTURE OF OSS SECURITY

- AI-driven SCA: Predictive vulnerability detection.
- Zero-trust supply chains: Verifying every component's origin.
- Regulatory push: Laws like EU's CRA mandating SBOMs.
- Community efforts: Projects like OpenSSF improving OSS security.

OSS is a double-edged sword: powerful but risky. SCA is your shield, but it's not foolproof.

- Understand your dependencies.
- Leverage SCA tools effectively.
- Stay proactive with updates and monitoring.

"Secure software starts with secure components."

Q&A

THANK YOU



hi@harekrishnarai.me



harekrishnarai.me



linkedin.com/in/harekrishnarai



github.com/harekrishnarai