Open-source supply chain attacks expand to the banking sector

Two banks have been targeted by open-source software supply chain attacks in recent months in what researchers are calling the first such...

4 weeks ago

Dark Reading

PowerShell Gallery Prone to Typosquatting, Other Supply Chain Attacks

Microsoft's PowerShell Gallery presents a software supply chain risk because of its relatively weak protections against attackers who want...

1 day ago

North Korea Leverages SaaS Provider in a Targeted Supply Chain Attack

In July 2023, Mandiant Consulting responded to a supply chain compromise affecting a US-based software solutions entity.

Software Supply Chain Security

Healthcare Supply Chain Attacks Raise Cyber Security Alarm

The healthcare sector has become a popular target for cybercriminals and is one of the most targeted industries by cyber criminals.

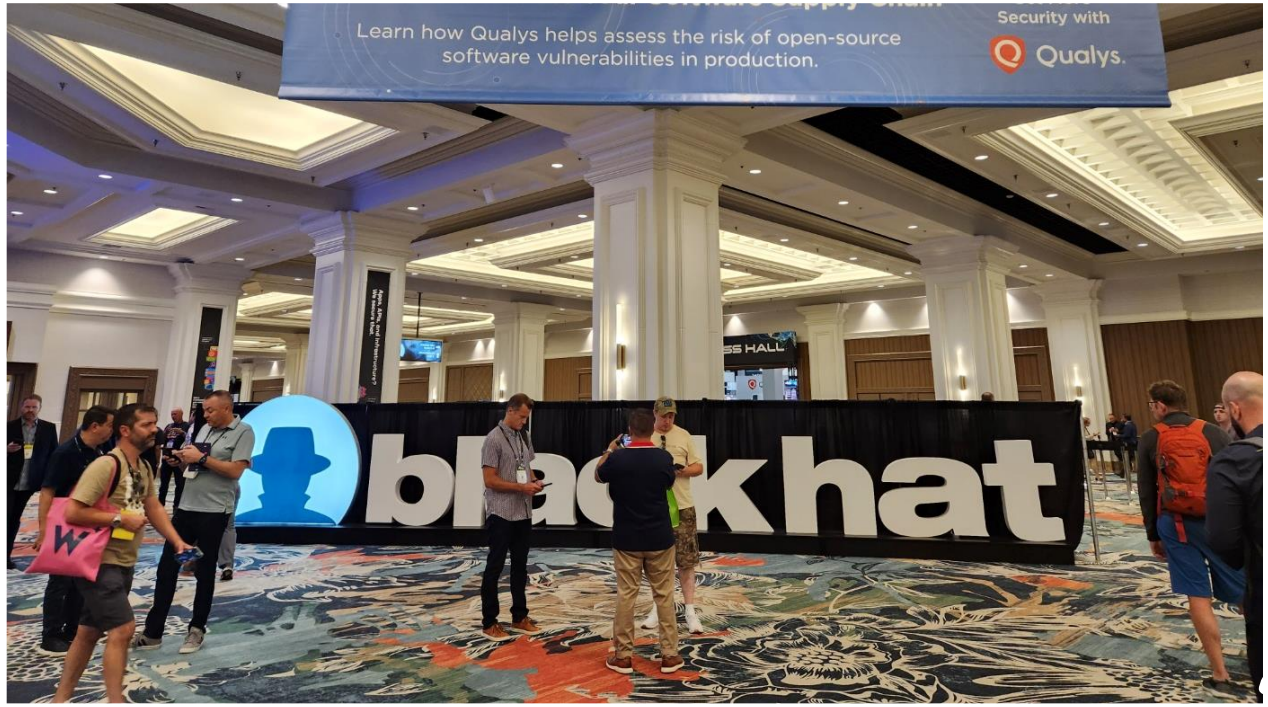
Jan 11, 2023

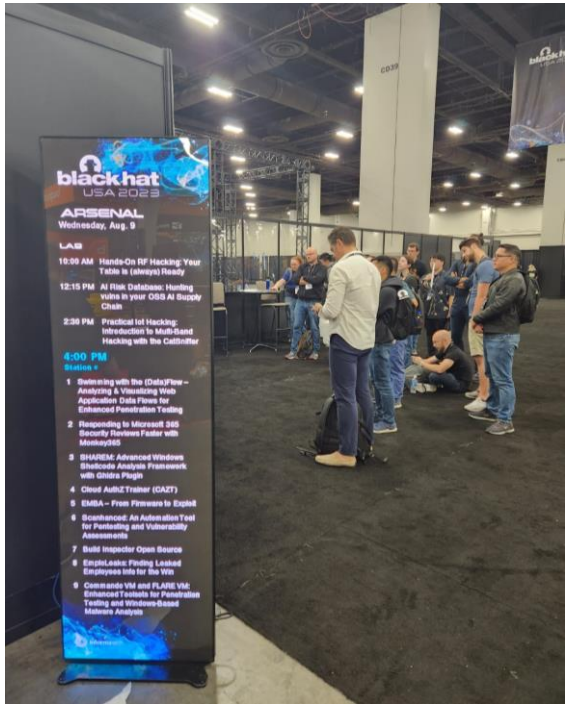
SolarWinds hack explained: Everything you need to know

The SolarWinds hack exposed government and enterprise networks to hackers through a routine maintenance update to the company's Orion IT...

1 month ago

Earlier this month at Black Hat USA





“By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021”

Gartner’s Outlook for Application Security 2023.

Past SW Supply Chain Attacks

Trans-Siberian Pipeline
Malicious code insertion

SimDisk
Auto-update mechanism of file sharing application

SolarWinds
Malicious code insertion

PyPi
Malicious Python Packages

Linux kernel
Backdoor malware insertion

Altair Technologies
Backdoor malware insertion

CodeCov
Compromised Bash upload

Zero Trust & SW Supply Chain security

- Zero trust is “a security paradigm that assumes no user or device is trusted by default, and that access to resources is granted on a least-privilege basis, with continuous verification required throughout the session”

- SW Supply Chain Security: “activities, processes, and technologies that protect the integrity of software throughout its development, delivery, and operation”

Do not trust

Zero Trust Security: Do not trust anything and validate everything	**To protect the network and Resource**
Software Supply Chain Security: Do not trust code you did not build	**Protecting the software**

Key Principles

Zero Trust	SW Supply Chain
Least Privilege	Visibility
Micro-Segmentation	Risk Assessment
Continuous Monitoring	Control
Trust is earned	Monitoring
	Response

EO 14028 Improving the Nation's Cybersecurity

Sec 3.
The Federal Government must adopt security best practices; advance toward Zero Trust Architecture

Sec 4.
The Federal Government must take action to rapidly improve the security and integrity of the software supply chain, with a priority on addressing critical software.

Guidelines on Minimum Standards for Developer Verification of Software

Technique Class	Technique	Description & Reference
Threat Modeling	Threat modeling helps identify key or potentially overlooked testing targets.	Section 2.1. Threat modeling methods create an abstraction of the system, profiles of potential attackers and their goals and methods, and a catalog of potential threats. Threat modeling can identify design-level security issues and help focus verification.
Automated Testing	As testing is automated, it can be repeated often, for instance, upon every commit or before an issue is retired.	Section 2.2. Automated testing can run tests consistently, check results accurately, and minimize the need for human effort and expertise. Automated testing can be integrated into the existing workflow or issue tracking system.
Code-based (static) analysis	Use a code scanner to look for top bugs.	Section 2.3. Static analysis tools can check code for many kinds of vulnerabilities and for compliance with the organization's coding standards. For multi-threaded or parallel processing software, use a scanner capable of detecting race conditions.
	Review for hardcoded secrets.	Section 2.4. Heuristic tools can be somewhat effective checking for hardcoded passwords and private encryption keys since functions or services taking these as parameters have specific interfaces.
Dynamic analysis	Run with built-in checks and protections.	Section 2.5. Programming languages, both compiled and interpreted, provide many built-in checks and protections.
	Create "black box" test cases.	Section 2.6. "Black box" tests can address functional specifications or requirements, negative tests (invalid inputs and testing what the software should not do), denial of service and overload attempts, input boundary analysis, and input combinations.

Technique Class	Technique	Description & Reference
Dynamic analysis	Create code-based structural test cases.	Section 2.7. Code-based, or structural, test cases are based on the implementation, that is, the specifics of the code. Code-based test cases may also come from coverage metrics.
	Use test cases created to catch previous bugs.	Section 2.8. Test cases which have been created to specifically show the presence (and later, the absence) of a bug can be used to identify issues in the absence of more general "first principles" assurance approaches for detecting bugs.
	Run a fuzzer.	Section 2.9. Fuzzers can try an immense number of inputs with minimal human supervision. The tools can be programmed with inputs that often reveal bugs, such as very long or empty inputs and special characters.
	If the software might be connected to the Internet, run a web app scanner.	Section 2.10. If there is a network interface, use a dynamic security testing tool (e.g., web application scanner) to detect vulnerabilities.
Check included software	Use similar techniques to gain assurance that included libraries, packages, services, etc. are no less secure than your code.	Section 2.11. Use the verification techniques recommended in this section to gain assurance that included code is at least as secure as code developed locally. The components of your software must be continually monitored against databases of known vulnerabilities; a new vulnerability in existing code may be reported at any time.
Fix bugs	Fix critical bugs that are uncovered.	Correct critical bugs as soon as possible and make process improvements necessary to prevent such bugs in the future, or to at least catch them earlier in the development process.

Software Composition Analysis

Analyze software to

- Identify open-source in use
- Check dependency
- Identify license information
- Detect known vulnerabilities in OSS
- Generate SBOM

Static Analysis

Analyze source code or binary to

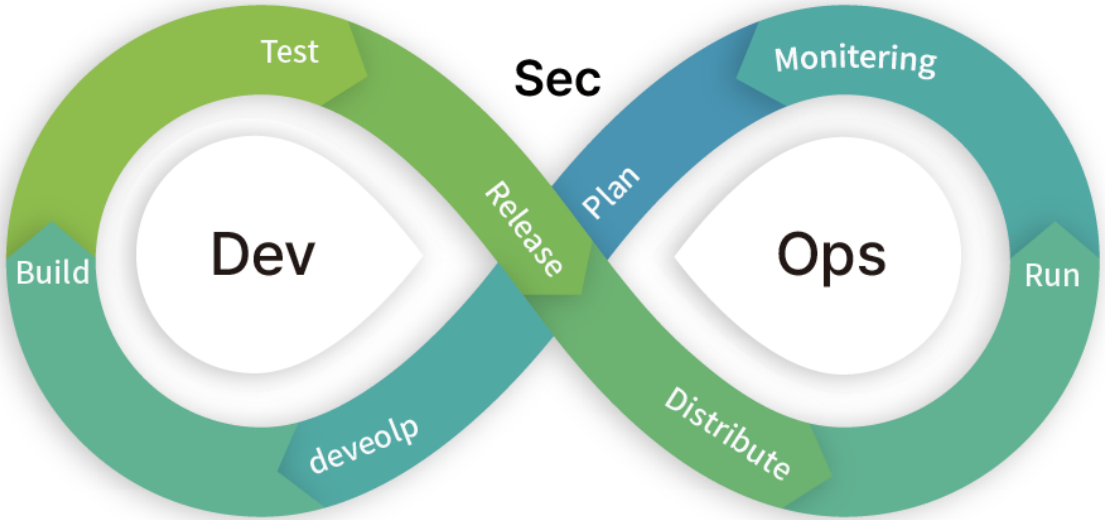
- Detect security vulnerabilities or quality issues in the code without executing it.
- Detect syntax error, logical errors, security vulnerabilities coding standard violations... etc

Dynamic Analysis

Perform security testing on a running application to

- Detect security vulnerabilities(including SQLi, XSS, CSRF...etc)
- Detect issues by simulating attacks and based on responses

Automated Testing and DevSecOps

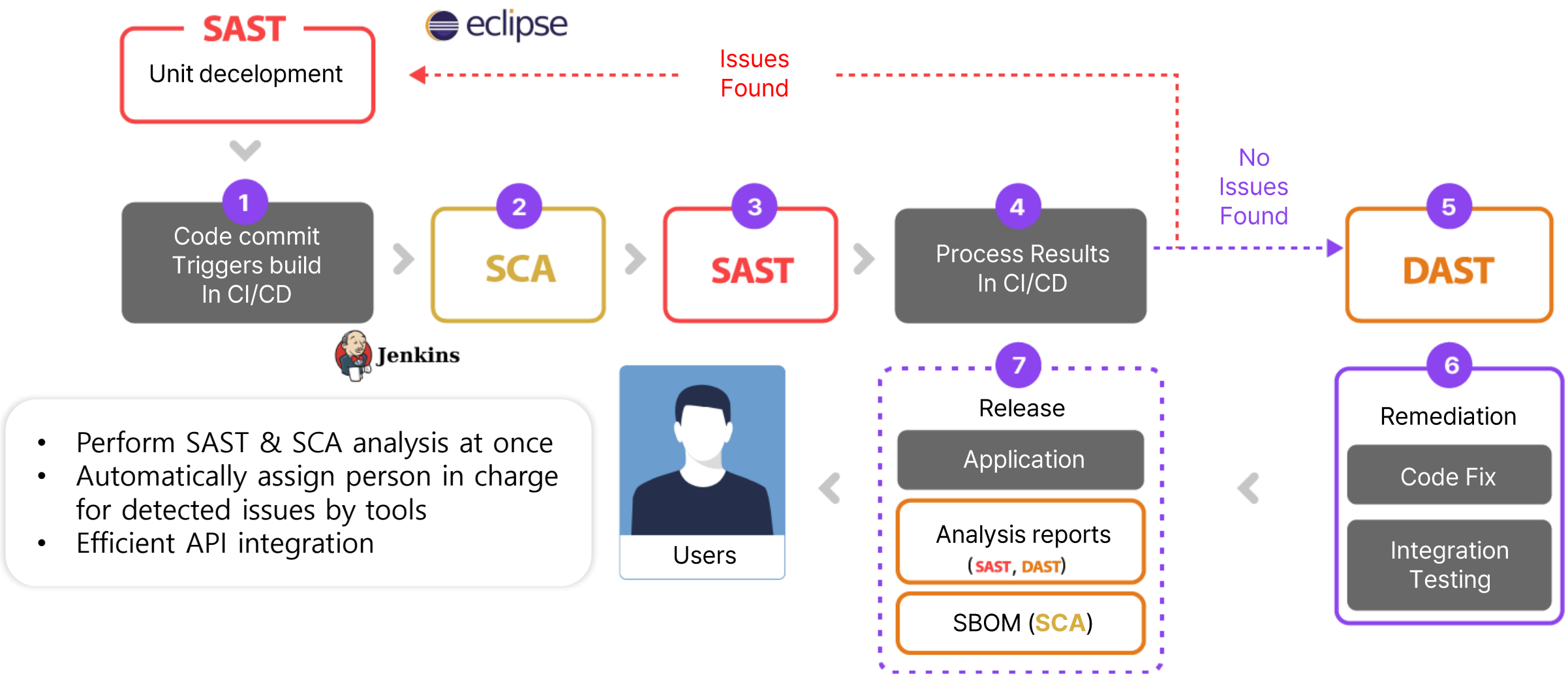


Automated testing is recommended.

- Run tests consistently
- Minimize human effort and expertise
- Can be integrated into existing workflow and issue tracking system

By automating testing and integrating with development process, adopt DevSecOps.

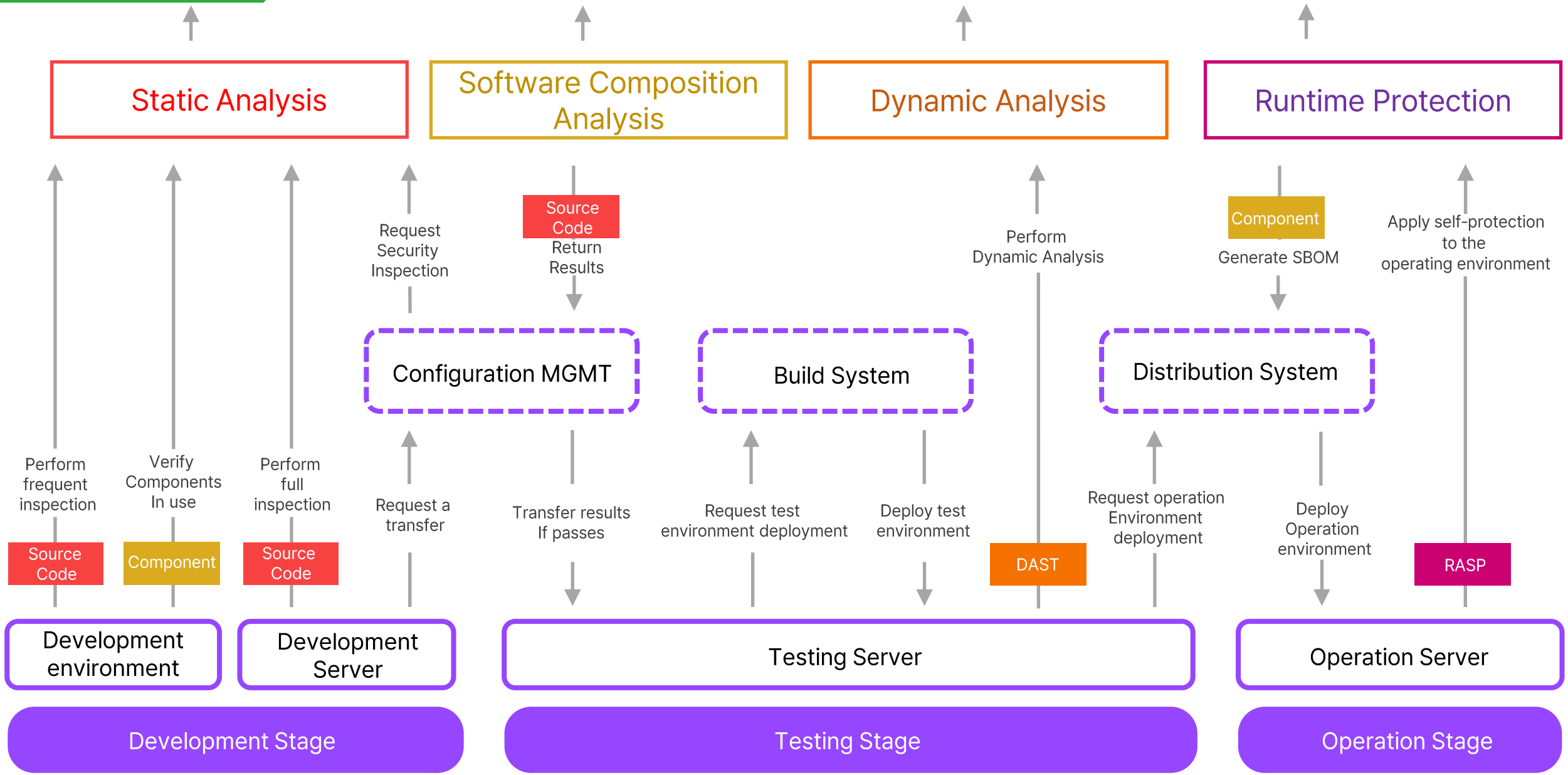
Case Study 1.



- Perform SAST & SCA analysis at once
- Automatically assign person in charge for detected issues by tools
- Efficient API integration

Case Study 2.

Integrated application security platform



How new technology is changing security

AI/ML

Automated patching

Recommending remediation

Identifying new vulnerabilities

Eliminating false positive

Automating repetitive tasks

Prioritizing vulnerabilities

Generative AI

Generating synthetic data

Generating test cases

DevSecOps

Automating testing and process

Increase visibility

Improve integrability

CNAPP