



National Defense-ISAC

Software Security Automation:
A Roadmap toward Efficiency and Security
January 1, 2020

Authors:

Mike Heim
Paul Keim
John Munsch
Waldemar Pabon

Reviewers:

Jeffrey Malovich
Brad Roberts





Executive Summary

Technology innovation, agility and the sheer growth in lines of code all contribute to an overwhelming threat landscape. The speed of modern software development increases certain feature deployments while exposing a larger attack surface to more capable adversaries. The need to address this snowballing trend in software vulnerabilities has reached a critical point. Despite the need, the cybersecurity industry expects to have 3.5 million security positions unfilled by 2021 (Morgan, 2017). At the scale and speed of modern software development traditional manual security checks become a security liability and a business impediment. With such a demanding environment, security teams must become more efficient in reducing risk, while enabling product teams to drive value.

To counteract the increasing threat landscape and the gap in human capital, security automation and orchestration have taken a center stage in the overall strategy to secure the software. Repetitive processes are better served by automation, while the implementation of specific workflows can be orchestrated to support better the security needed in the Software Development Lifecycle. There are several key tools that must be put into place as part of the automation strategy to protect the software properly: Static Application Security Testing (SAST), Software Composition Analysis (SCA), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), Runtime Application Self-Protection (RASP), and Web Application Firewalls (WAF).



Security improvements require continuous efforts and from that perspective, metrics become an important ingredient in the decision-making process. Being able to measure the security compliance of the organization through the different tools used to secure the software allows teams to be efficient with their remediation efforts and security process improvements. To enable this type of integration through automation, Application Program Interface (API) is needed to provide better control over project creation, user management, and access control. Integration also extends to the developer Integrated Development Environment (IDE). Allowing developers to have security advisors provided as part of the security plugins working in the IDEs minimizes the impacts associated with resource constraints and security skills gap.

As teams work with the security automation implementation, deploying code to production will require runtime protections to provide flexibility when remediating security vulnerabilities while also providing temporary protections for those same issues in production. Adjusting the configuration of the security tools will reduce the false positive ratio as well as the detection capabilities of the security automation. Selecting the right tool for the job will help with performance and scalability.

As teams face the future of automation and transition from a legacy Software Development Lifecycle (SDLC) to a DevOps methodology, automation and build servers will become core components of the strategy. Gated releases will provide a risk mitigation strategy, enabling a



pipeline to leverage APIs and provide proactive protections in production environments through WAF. In addition, virtual patching will enable agility as part of the process. Regardless of the SDLC methodology in play (legacy Waterfall or DevOps), software security automation is a key component of the overall organizational security strategy when protecting software in the SDLC. Multiple security controls are available to be implemented at different stages of the process to guarantee that all security concerns within the software are identified. From SAST tools serving as security advisors, to virtual patching allowing teams to provision temporary fixes while the teams work on remediation of a security finding, automation can be leveraged at all stages of the SDLC. Organizations should embrace automation as an embedded component of the application security strategy to minimize the window of exposure and the risk of a compromise.



Table of Content

Executive Summary	2
Introduction.....	7
Objective	7
Audience	8
Structure of the paper	8
Software Security Automation: A Roadmap toward efficiency and security	9
Automation Overview.....	9
Security Components.....	12
Toolchain Architecture Overview	12
Interactive Application Security Testing (IAST).....	16
What to Look For	17
Waterfall Vs Agile and Human Controls vs Automated Controls.....	18
Strategy	22
Metrics, Analytics & Trending.....	22
Risk Management, Developer Feedback, and Remediation	28
Workflow Integration.....	31
API/CLI Integration	33
Runtime Protections	37



Implementation	39
Use Cases	39
False Positives.....	44
Aggregation & Correlation	47
Performance & Scalability	50
Future of Automation	53
Roadmap to transition.....	53
CI/CD Pipeline Integrations with Security Components	56
Software Gated Releases	63
Conclusion	64



Introduction

Objective

Technology innovation, agility, and the growth of technical capabilities have contributed to a significant increase in vulnerabilities. The fast pace of modern software development methodologies has increased both feature deployments and the number of vulnerabilities deployed to production environments. Identification of such vulnerabilities has reached a critical point, forcing organizations to consume an increasing number of resources requiring human capital with very specific technical skills. Despite this need, the cybersecurity industry expects to have 3.5 million security positions unfilled by 2021 (Morgan, 2017). In such a demanding environment, organizations need to become efficient in the way they remediate security vulnerabilities while adding value to the business. Traditional manual security checks become roadblocks in the cadence of tasks needed to secure the software and prevent threat actors from weaponizing the software.

To counteract the increasing threat landscape and the lack of human capital, security automation and orchestration have taken a center stage in the strategy to secure software. Repetitive processes are better served by automation, while the implementation of specific workflows can be orchestrated to better support the security needed in the Software Development Lifecycle (SDLC). This white paper provides guidance in terms of the security controls needed when automating software security in traditional Waterfall/Agile Scrum methodologies as well as organizations beginning a transition into a DevSecOps practice.



Audience

The audience of this white paper includes security engineers, lead software engineers, product managers, senior managers, and senior executives responsible for the implementation of software security automation initiatives in the organization, as well as those managing the risk associated with threat vectors in software.

Structure of the paper

This paper introduces the importance of security automation and orchestration in the quest to secure software and minimize risk to the organization. The security components needed for a security automation strategy are presented and discussed. Strategies on how to integrate metrics, remediation, risk management, API capabilities, and runtime protections are presented. Software security automation implementation is covered through the discussion of use cases as well as guidelines on how to handle false positives, defect correlation, aggregation, and performance. Finally, this white paper provides an overview on how to transition traditional software security practices into a DevSecOps approach, helping to maximize the benefits associated with agility and efficiency while maintaining a strong security stance.



Software Security Automation: A Roadmap toward efficiency and security

Automation Overview

Organizations always look for ways to save money, improve efficiencies, and remove errors from processes. Technology innovation and agility requirements by organizations are creating an expansion of security threats which, without the right security strategy in place, will become an increasingly difficult challenge to both discover and remediate. Due to the limited technical human capital and the need to support business initiatives at a faster pace, automation has taken a pivotal role in the defense strategy of securing software. From a software security perspective, automation can be used to initiate:

- detection of vulnerable third-party libraries;
- scan of code in a test environment to identify threat vectors prior to the rollout to production;
- automatic creation of tickets in an issue tracking system; and
- proactive deployment of protections to a WAF based on results from a vulnerability scan.

Each one of the previous examples are individual tasks that demonstrate how automation can be leveraged to achieve efficiency and security. Automation takes care of the initiation of an individual task to minimize human error and improve the time it takes for a repetitive action to be completed. In terms of protecting the software in the SDLC, automation efforts are effective when implementing several independent but interrelated steps that can be



used to improve the security stance of the final product. A common strategy used in industry to leverage automation initiatives and help the business improve their security stance, orchestration, involves the creation of a workflow that controls several of the automation steps in a coordinated arrangement of components.

Orchestration involves the integration of multiple automation tasks to control and execute a larger process. The repeatable characteristic of business workflows is one of the key points used by orchestration to eliminate redundancies and provision better agility. From a security standpoint, the use of orchestration allows organizations to streamline and optimize repeatable security workflows (Tillyard, 2018). The use of orchestration empowers organizations with a standardized implementation of security in software efforts at every stage of the SDLC. From scanning third party libraries to WAF protections in the production environment, the process of securing the software becomes an enabler for a better and more accurate control of security threats.

To simplify the discussion in this white paper, the word, automation, is used to illustrate the intrinsic relationship between several automation tasks orchestrated to achieve multiple benefits for the organization. The benefits of software security automation can be summarized as:

- Standardization of the security tasks needed to minimize threat vectors in software;



- Reduction in costs associated with the integration of security processes in the development efforts of software;
- Improvement in the feedback loop to developers as it relates to the identified security vulnerabilities in code;
- Removal of the overhead associated with human intervention in the different security related tasks to enable the agility of teams and the time-to-market of software solutions; and
- Support for the proactive enablement of the organization to defend itself from cyber threats.

Automation is an important part of a continuous improvement process. Not only has software security automation become a key aspect of the business security strategy, but the ever-changing technology landscape imposes a requirement to be flexible in the response to cyber threats. Implementing a security automation strategy in the organization provides an effective roadmap toward efficiency and proactive detections that will help strengthen the security stance of software.



Security Components

Toolchain Architecture Overview

In this section we provide a brief overview of the different tool areas mentioned in a software automation toolchain. Each tool area will be presented as to its purpose in the tool chain, as well as its specific strengths and weaknesses.

We will approach these descriptions from the viewpoint of the standard enterprise web application, though much of the material is also relevant for desktop applications as well. Several of these tool areas depicted at Figure 1 are multi-purpose and may be used for security automation of related technologies, such as application containers. However, specifics on securing related technologies will be left to future papers. Also, outside the scope of this section is the discussion on specific tools. Mention of specific tools should not be taken as an endorsement of the tool by the white paper's authors.

Figure 1. Toolchain Overview

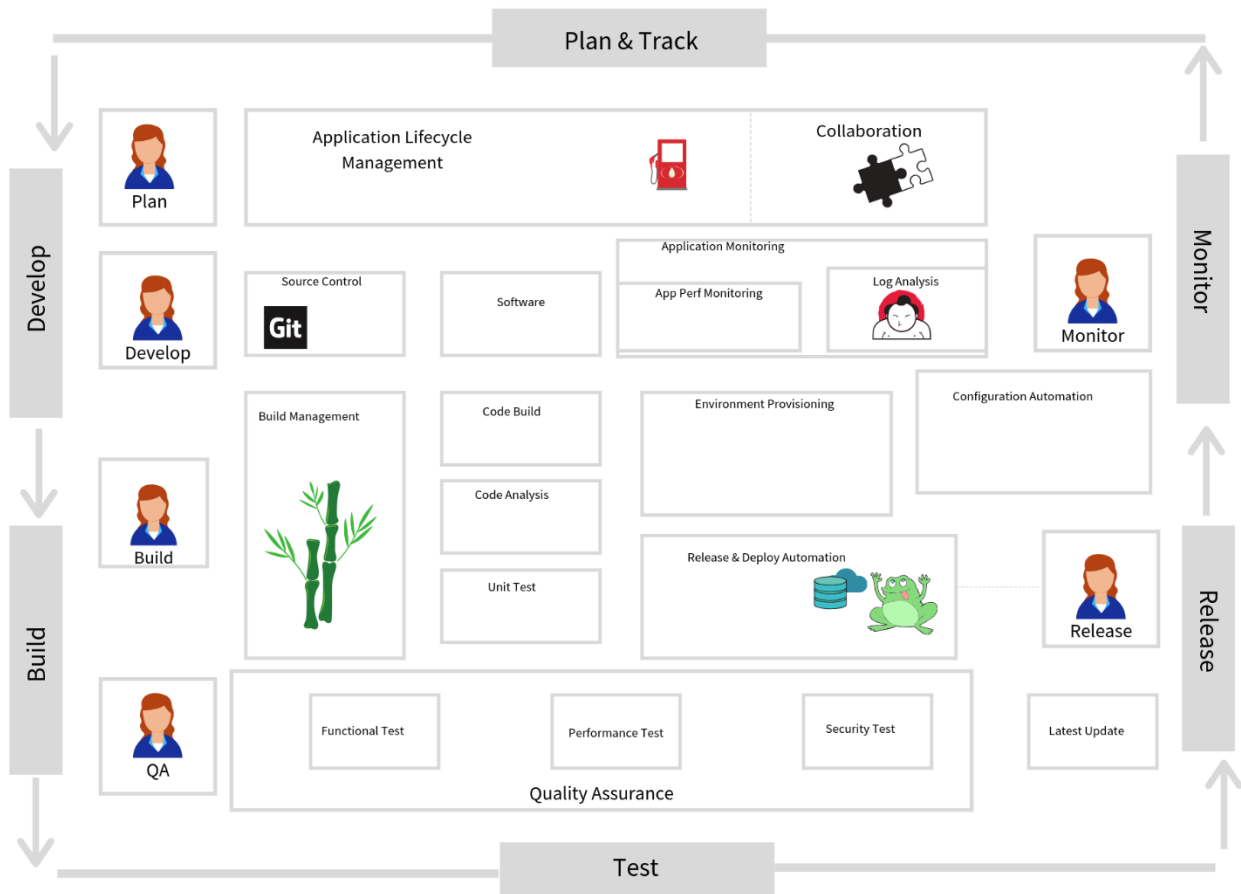


Figure 1. An architectural overview of software automation. The outer ring describes a continuous application lifecycle. Inner regions group roles with software tool categories.

Static Application Security Testing (SAST)

SAST tools, also known as Static Source Code Testing or Source Code Analysis, are white-box tools that review the application's source code or compiled artifacts to identify defects. As the static analysis can be performed after code is checked in or compiled, it is the first location short of an IDE plugin that we can begin to identify defects in the toolchain.

Strengths:



- Scales well due to not requiring an active running application
- Can directly identify problematic lines of code
- Can be used to enforce some code quality checks that dynamic testing would be unable to identify.

Weaknesses:

- Unable to identify business logic, authentication, or authorization issues
- Looks at only one link in the chain, does not identify potential issues further down the chain (i.e., a webservice called by the application)
- False positive analysis often requires access to and familiarity with the source code and the application's environment
- Generally, COTS vendors will not make source code available for scanning. Check with the vendor for 3rd party scan results.

Software Composition Analysis (SCA)

SCA tools examine the open source libraries imported by an application to determine if there are known vulnerabilities in those dependencies. These tools typically draw from the National Vulnerability Database (NVD) as well as several proprietary data sources purchased or maintained by vendors. An SCA implementation allows a user to build a quick inventory of what open source libraries teams are using; identify potential vulnerabilities and licensing restrictions; and assist to determine potential impact from newly published vulnerabilities. More mature tools include an analysis to determine if the vulnerable component is actively



being used in the application's execution flow, negating one of the primary weaknesses of SCA.

As a note, asking for a Software Bill of Materials (SBOM) or a list of all the different components that comprise an application is becoming more commonplace. Software Composition Analysis is one method of creating these SBOMs.

Strengths:

- Able to easily validate an insecure library is being used in the application, either by itself or by a dependency further up its build chain
- Analysis is performed prior to or immediately after build, results are almost immediate.

Weakness:

- Tools may not be able to identify if a vulnerability is exposed, only that a published vulnerability is present in the artifact.

Dynamic Application Security Testing (DAST)

Contrasting with SCA and SAST, DAST is black-box testing that utilizes a running version of the application to examine its runtime behavior. An automated DAST tool will crawl through an application submitting malicious requests and use the application responses to attempt to identify if the attacks were successful.

Strengths:



- Results include actual exploitability of findings, and provides payloads used to produce the finding
- Lower rate of false positives than static analysis tools
- Internally deployed COTS tools can be scanned.

Weaknesses:

- Unable to identify business logic, authorization, and authentication flaws
- Often requires heavy configuration on a per-app basis, making it less easily scalable
- False positive analysis may require manual reproduction of the finding
- Findings do not indicate location in problematic code, requiring a knowledge of the code base and environment prior to offering a fix
- Remediation of findings in COTS tools requires vendor support.

Interactive Application Security Testing (IAST)

IAST is an agent-based testing mechanism that supplements dynamic testing. IAST can be deployed in an *active* or *passive* method. In Active IAST, the tool actively submits requests as the agent monitors the runtime of the application. This form can be considered an expansion or extension to DAST. In Passive IAST, the agent monitors the runtime continuously as other QA and security testing is performed.

Strengths:

- Able to find runtime vulnerabilities that would not normally be exposed to an attacker



- Once baked into the build process, no additional configuration is required
- Provides more detailed feedback than DAST, able to better identify the affected code locations
- May be used with other security tools or the team's regression testing to eliminate false positives and strengthen validity of findings.

Weaknesses:

- Newer technology that is not as developed, thus supports fewer technologies and environments.

What to Look For

Though each organization will have different selection criteria for a tool, consider the following areas when performing tool selection:

- API Capabilities, CLI & Connectors. The key functionality (creating projects in the scan tool, starting scans and returning scan results) of the tool should be available through APIs, and it should also support common enterprise use cases such as automated user access provisioning. Freely available connectors between the tool and your chosen development tools (such as your bug tracking systems) are also an important consideration; anything that isn't out of the box will require additional development to implement. For SAST and SCA, CLIs allow for scan execution to be offloaded to CI/CD pipelines.



- Single Sign On (SSO) Support. This makes it easier for both the operations and development teams to access the tool.
- Development Environment Support. The tool should support a majority of the development teams' languages and frameworks. Having an established standard for languages and frameworks will serve as a strong baseline for this tool.
- Enterprise Environment Support. Understanding how the tool integrates with other enterprise tools helps plan the implementation, but also identifies any potential future shortcomings.
- Reporting and Remediation Guidance. The tool's reports should be clear and descriptive providing relevant remediation guidance to developers. False positives should be minimized within the tool's capabilities, or the tool should be able to be tuned to ignore known false positives.
- Scalability. The tool should be able to scale to the expected workload.

In a follow-on paper, we intend to further describe the toolchain above and provide detailed recommendations for each area.

Waterfall Vs Agile and Human Controls vs Automated Controls

Automated software deployment environments are made of integrated tools that are most often described in technical terms. However, for security to be correctly built into an automated development lifecycle, special focus is needed on more than just tools.



Supporting policies, processes and procedures must be re-evaluated. Software development methodologies have evolved from waterfall to Agile, and Agile is further operationalized by DevOps. Traditional security processes tend to be heavy with manual steps performed periodically. Security architecture reviews, code reviews, pen tests, or manual code analysis are not processes that lend themselves to continuous development and deployment.

In an automated release process, risks that are addressed by manual steps or traditional human controls must not be left unaddressed. Some human controls will become automated, while others will be addressed with abstractions. The software release aspects that require human intervention, will be performed parallel to the continuous release process. The policies, processes and procedures needed for addressing the risks may look entirely different in a heavily automated, continuous release software lifecycle.

There are three steps necessary to reduce risk that is associated with manual or traditional human controls:

Step 1: Document the current manual security controls, or procedures, in the traditional development process.

Step 2: Clearly describe the security risk that each of these manual security controls is designed to address. Identify and understand and clearly document the originating purpose



of each manual security control. In large IT or Information Security organizations, it may be difficult to drive change in policies and processes. Identifying the originating purpose of any manual step to support the need for fundamental changes where a new automation or abstraction addresses the same originating purpose.

Step 3: Identify what newly introduced automation or tool addresses the same requirement that the legacy manual control addresses. This new control may not be a tool or process; it may be a security best practice, or a behavior that is derived from the new automated development pipeline. The new control may simply be a benefit of using common libraries and frameworks across the portfolio of applications.

Controls tend to map in groups to general purposes. For example, risk profiling and threat modeling are manual controls that generally serve a security architecture purpose. While these controls are not activities that can be fully automated, they can be optimized by using risk profiling and threat modeling processes that leverage reference models which is a common taxonomy and a well-developed index of standard threats with standard mitigations. Use of standard patterns enables rapid development of risk profiles and threat models streamlining the analysis and review of these deliverables. Initially, manual controls are run as separate periodic activities outside the continuous release process. Our long-term goal is to make these activities as iterative as possible, bringing them into the continuous development process.



Technical Security testing should be fully automated. SAST, SCA, DAST, IAST, RASP are areas where we can expect to leverage automation to address a security purpose more effectively than we have previously done with manual controls. Fully automation of DAST can be a challenge for complex applications; however, once configured in the DAST tool, scan execution can be automated.

Code Reviews should be incorporated into development. With agile development methodologies, and modern Application Lifecycle Management (ALM) tools, developers review code security continually during development and Agile development norms should include reviews as part of the merge request process. Automated testing feedback is continuous and detailed, and technically relevant. Security requirements with test cases are driven out of the optimized Risk Profiling and Threat Modeling activities.

Manual Gated Reviews, Authorizations and Approvals for release are minimized. These mechanisms serve assurance, attestation, transparency, and acceptance purposes. All of these purposes are better served by the enhanced monitoring and reporting in an automated pipeline. Similarly, use of common technology stacks and reference models based on standardized tech stacks allow many releases to proceed with prior-authorization, or no-authorization.



The exercise of mapping control-to-purpose also serves to develop a scoping criterion. Not all applications and application teams fit well in a continuous, Agile, or DevOps development practice. If applications cannot adopt the controls that support security automation, they may be more clearly identified as a 'legacy' application. This legacy status highlights risks including: security risk, technical debt, and business risk (cost). Beyond reducing risk, the new application security paradigm becomes a cost saver as it highlights additional return on enterprise efforts to retire and consolidate applications.

Strategy

Metrics, Analytics & Trending

Organizations need to measure how effective their software security automation process is and how automation is supporting not only the identification of vulnerabilities but also how teams deal with them. Application Security metrics allow the business to measure the risk associated with the identified vulnerabilities and understand its level of organizational compliance, as well as to identify the support needed for making strategic business decisions. Trending becomes an important component in the overall continuous improvement strategy as it helps teams identify patterns in the data analyses.

Those patterns can later be used by the business to formulate security strategies to tackle common issues:







- Reduction of costs by improving the automation process (identifies a common problem and integrates a solution in the automation strategy to prevent future teams from spending time dealing with the issue).
- Supporting better awareness and educational efforts (being able to identify areas where teams are failing to address as a whole and implement training sessions and awareness campaigns).

Metrics, Analytics and Trending are effective data collecting mechanisms used to maintain a continuous improvement environment throughout the different stages of the software security automation process. In addition, these mechanisms provide greater visibility into the cadence of protection efforts and efficiency. However, there are two main concerns to consider: organizational and process compliance. From an organizational compliance perspective, having this type of visibility from a metrics standpoint allows executives to derive business requirements decisions as well as evaluate the compliance level of the organization against the established risk appetite. At a lower level, the process compliance metrics allows the management structure of development teams to continuously evaluate the state of the security strategy over time, which will support a better continuous improvement process. Table 1 provides a list of some basic common process compliance metrics that should drive the improvement efforts behind software security automation in development teams.

**Table 1**

Process Compliance Metrics

Metric Component	Rationale	Trend Objective
Security Defect Density	Identifies the number of security defects found in every thousand (millions, depending on the codebase) lines of code. This measure helps teams evaluate the security practices implemented. Over time, this metric should show a downward trend.	
OWASP Top 10	Allows the identification/classification of vulnerabilities per the OWASP Top 10 categories of threat vectors. This metric supports awareness, training, and proactive protection efforts. Over time, this metric should show a downward trend.	
Tooling Automation	Presents the number of applications covered by software security automation efforts (SCA, SAST, DAST, RASP). This metric demonstrates if the software automation investment is being used and to what extent.	
Average Resolution Time	Average time spent by the team to resolve vulnerabilities. This metric exposes the development team's agility to deal with security findings. Over time, this metric should show a downward trend.	






Metric Component	Rationale	Trend Objective
Flaw Creation Rate	Measures the rate at which vulnerabilities are created as detected by the tools in the security automation. This metric should be compared against the average time to resolve a security flaw. Over time, this metric should be lower than the Average Resolution Time.	


Table 2 presents the list of some basic common organizational compliance metrics that should support strategic business decisions.

Table 2

Organizational Compliance Metrics

Metric Component	Rationale	Trend Objective
Window of Exposure	Measures how much time it takes for the organization to remediate a vulnerability from the moment it gets disclosed, to the moment the vulnerability is patched. Software Security Automation plays a vital role in the identification process. Over time, this metric should show a downward trend.	
Criticality	Measures how many high, critical, medium and low vulnerabilities are identified at the different stages of the security automation process. Over time, this metrics should show a downward trend. CISOs can use this metric to measure	



Metric Component	Rationale	Trend Objective
	whether the organization is getting better or worse at releasing secure software.	
Program Participation Levels	Measures the participation within the organization including how many business units/projects have full/partial/no software automation in place. Over time, this metric should show an upward trend toward full coverage.	
Root Cause	Measures whether the vulnerability is associated with third party libraries vs custom code. This metric should help understand where to focus investments decision over time.	N/A
SDLC Phase Detection	Measures in which phase of the SDLC the most of vulnerabilities (higher percentage of issues) originate.	N/A

The aggregation of data in a Business Intelligence (BI) platform becomes an important aspect of Software Security Automation efforts. Securing the software is a continuous improvement process. Therefore, organizations need to be able to capture the information provided by all the different components (static code, software composition, dynamic code analysis, etc.) and aggregate the data in a structure which will support the decision making. To properly implement a BI architecture supporting the analysis process, there are two main components needed: measures; and the elements to use when slicing and dicing the data. Figure 2



provides an idea of the BI structure needed to support analytics and the analysis of trending (presented for reference only to provide a general understanding of the BI strategy).

Figure 2. High Level Overview of a BI structure for App Sec Metrics

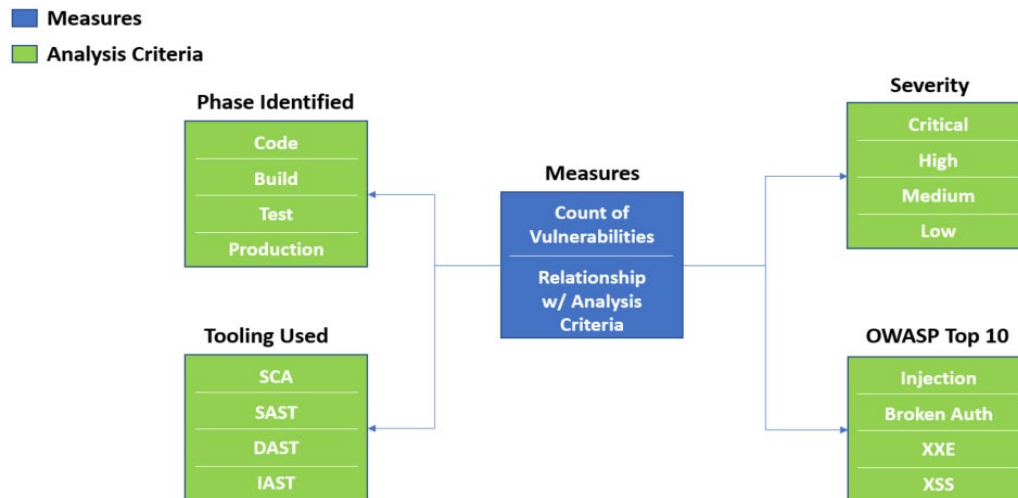


Figure 2. The measures component will capture and store the relationship against all the analysis criteria in every single vulnerability identified through the Software Security Automation effort. From a database referential integrity perspective, those relationships will represent foreign keys of the different analysis criteria components. The Analysis Criteria components will hold the definition of the elements to use when measuring the overall vulnerabilities. For example, in the case of the Severity component, the structure will have a row per each of the different severities the organization is trying to measure with a unique id that will be referenced in a measure component.

Organizations need to make sure they select a set of metrics that will improve their security stance, understanding the importance of context. Legacy applications included for the first time in a vulnerability scan process will have a high probability of reporting multiple vulnerabilities than an application already engaged in cyber hygiene throughout the SDLC. This type of exception must be understood to avoid the overreaction of teams to deviations. Another important aspect to consider when dealing with metrics involves the approach



toward the implementation of a BI architecture. Organizations need to use an approach of “think big, start small.” Trying to aggregate a large set of analysis criteria from the get-go may create a non-feasibility scenario from a cost perspective. At the start of implementation, it is important to begin with a small subset of analysis criteria, and as the software security automation process matures, new components can be integrated.

Risk Management, Developer Feedback, and Remediation

Prior to implementing an automated toolchain, an important consideration is how to address the risk of the findings it will identify. Each organization has its own approach and methodology towards risk appraisal and management. These methodologies often are focused on individual findings and thus have the downside of needing to understand the finding, its environment, and to be reviewed in that context. While this is suitable when the environment has a relatively low number of findings, it becomes a significant manual effort as the toolchain grows.

To deal with the risk assessment in an automated fashion, we recommend implementing a parallel approach to complement the risk appraisal process. The automated tools will generate findings in a variety of ways, but will almost always be tied back to a Common Weakness Enumeration (CWE) number. Instead of categorizing each finding based on severity or exploitability, consider categorizing them by CWE, and use the organization’s standards and compliance obligations to determine which CWEs will need to be addressed. Alternatively,



there are scoring systems such as the Common Weakness Scoring System (CWSS) that can assist in producing different categorization buckets.

To provide a quick example from common SAST findings, consider a SQL Injection finding (CWE-89) versus a CRLF Log-injection finding (CWE-117). As SQL Injection is a high-risk OWASP Top 10 finding, categorize the CWE-89 findings as “mandatory fix.” Then, when a scan identifies a SQL injection finding, it fails the scan policy. Contrasting that, the CRLF injection finding is generally considered low risk and not an OWASP Top 10 finding. As such, it can be considered a “discretionary fix” and would not have it fail the scan.

This approach is intended to complement and simplify the existing risk appraisal process, not replace it. The categorization of mandatory and discretionary is generally insufficient for communicating the full risk of a finding, but it does reduce the complexity of informing developers about the findings and what fixes are required.

The next step in an effective risk management approach is to determine how to relay findings to the developers. To simplify the discussion around feedback, consider breaking it down into two categories: active feedback and passive status.

Active feedback are notifications (e-mail, slack messages, desk drive-by) that relay the status of a scan to the team. These are vital to encouraging self-service as the developers will need



to be informed of any actions they need to take. However, be cautious with the amount of active feedback. Too many notifications and developers will begin to tune them out, suffering from alert fatigue. To combat alert fatigue, evaluate how many notifications will be sent, what the content will be, and if the notification really needs to be sent.

To allow teams to see their scan results on-demand, rather than digging through their notifications, a passive status mechanism can be implemented. This can be thought of as a dashboard where the development team can see their overall status, as well as the results broken down by each layer of the tool chain. Another advantage of a passive status mechanism is that a single pane of glass dashboard enables self-service reporting for the development teams, reducing the security team's workload.

While each development organization will think differently, a good starting point for the content of the active and passive mechanisms is to include:

- Relevant build information (version ID, build number, environment (if applicable), etc.)
- Status (Pass / Fail)
- Number of findings (if applicable, broken down into mandatory and discretionary as above)
- Next Steps: Team should review report and remediate findings
- Contact Information for questions



Workflow Integration

Traditional security processes and tools have been poorly integrated into software development workflows. General security requirements may be identified early, but they are perceived as high effort, low value work and are pushed off to later sprints. Often software security becomes a last-minute check to get approval for release. While software security lifecycle frameworks call to “shift left,” the tooling and procedures to do this are so poorly integrated that the strategy is rarely executed well.

As security practitioners, we must put the developer’s workflow first if we expect to integrate our security requirements into their product development. When selecting security automation tools, prioritize these fundamentals of security integration:

Do not break the developer’s context: Wherever possible allow a developer to address a security requirement or defect within the coding tool that they are working in. Security features can be best integrated into an Integrated Development Environment (IDE).

Difficulties occur when a developer is forced to switch from their coding environment to check a website for a process, or a dashboard for status, or run a separate analysis tool. That switch breaks the developer’s focus, slows down development, and most importantly relegates security to the many activities that a developer might get to *after* the coding is done.



Integration is priority as tool vendors share their new interface or dashboard. Developers need the security functionality plugged into the interfaces already used in the work environment. Secure coding feedback needs to be continuous, immediate feedback as a plugin to the IDE.

Secure development is not ‘owned’ by Security, but by the software development team. A

centralized, Information-Security-owned, vulnerability management system is not an appropriate system for development teams to manage their security work. ALM tools are used by software teams to manage resources, backlogs, source code repositories, work items or issues. ALM systems provide development organizations all the traceability to understand project risk, cost, billing and workload management. A software team’s ALM is the appropriate system to store that team’s security work.

Application security requirements and defects that do not appear in the developer’s work management system will never be prioritized. Security defects must be pushed into the same bug tracking systems that the software team uses for all other bug tracking. Any Information Security system that identifies and aggregates software security defects needs to have first-class support for the ALM’s application programming interface (API).

Avoid Tool Fatigue: Similar to above, avoid yet another dashboard or tool fatigue. Any human interface added to the developer’s workflow requires additional training and support.



These extra interfaces increase cost, slow adoption, and make upgrades or product changes more difficult.

Don't spam the backlog. Automated code analysis tools must intelligently integrate with ALMs. Each discovered defect should not be individually pushed into a backlog as a unique user story or task. Scanning tools must intelligently aggregate and correlate defects. Defect correlation across assessment types reduces noise and helps prioritize severe issues. Assessment systems must be appropriately tuned to bundle and aggregate defects into work packages that make sense for the developers to work them. How the security analysts understand and associate these defects is secondary to how developers' intake and assign this work. Consider grouping defects by software class, module, or by defect type. Favor analysis tools with aggregation and correlation that can be tuned by application team.

API/CLI Integration

Day-to-day operational management of the toolchain can be challenging—especially for larger organizations. One challenge that large organizations can face is maintaining appropriate access control. Large organizations often have multiple business or functional areas, and hundreds or thousands of developers and applications. Managing project creation and access control can quickly become unsustainable. Many SAST and SCA tools include a CLI that facilitates auto-creation of projects the first time a scan is executed. This works well for small operations, where developers are authorized to access all projects. However, this approach



falls short for organizations with multiple functional areas or where developers' access must be controlled within the organization's hierarchy and structure.

Another challenge for larger organizations is related on non-unique project names. There are several projects with the same name in multiple repositories. The potential exists for Project B's CLI to overwrite the scan results from Project A, and unintended access control issues can occur. A mature tool's API will provide better control over project creation, user management and access control. The API also provides the ability to abstract a CI/CD pipeline's integration implementation.

A mature tool's Application Program Interface (API) will provide better control over project creation, user management and access control. The API also provides the ability to abstract a CI/CD pipeline's integration implementation.

Ultimately, abstraction is key. Consider developing a DevSecOps API which abstracts the security tool's API integrations. The code that manages projects for the security tools is only located in one place. Relatively generic DevSecOps API interfaces can be called from CI/CD tools such as GitLab, TFS and Jenkins. Examples include:

- The DevSecOps API has control over a project naming strategy that guarantees uniqueness. Meta data associated with the API call provides information needed to create a unique project name. e.g.: myproject_repouri_collection



- If the project already exists in the downstream security tool, the API simply returns the unique project name to the CI/CD tool
- If the project does not exist, the project will be created, and the API returns the project name to the CI/CD tool
- If the developer who triggered the CI/CD pipeline does not exist in security tool's project, the API adds the developer to the tool with the appropriate access control.

Assumption, if the developer is authorized to check code into the source code repository, the developer is also authorized to review and remediate scan findings in the security tool.

The abstraction layer described above will save the security tools operations teams untold hours of labor.

Additional opportunities for abstraction can save developers several hours of labor every time they set up a new project's CI/CD pipeline. For example, a developer needs to setup SAST scanning for a project in GitLab. GitLab's CI/CD pipeline is based on Docker Runners. Prior to work; the integration team creates a Docker image with the appropriate scan CLI and add the unique project name. The Python script then executes the SAST scan via the CLI. The scan results are forwarded to the SAST tool.



Figure 3. Process to enable SAST scanning for the project.

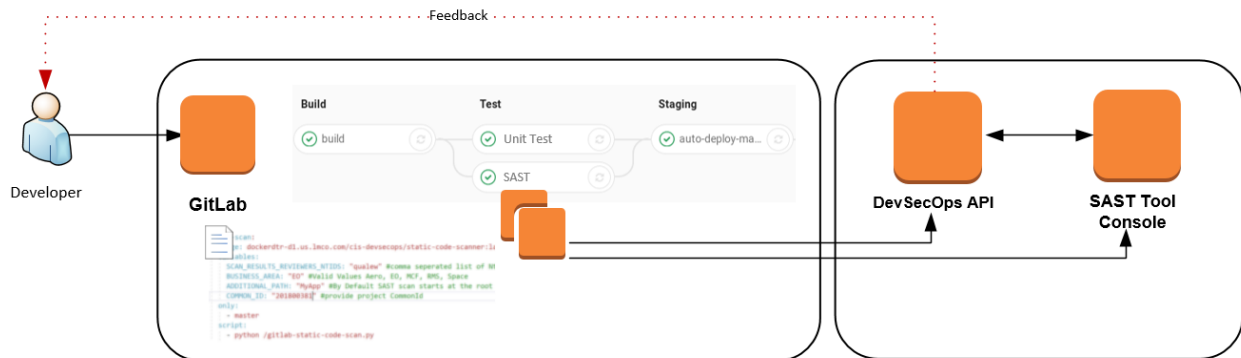


Figure 3. Developer checks code change into repository. A SAST Test Pipeline spins up, requests the Project ID from the DevSecOps API. If the Project does not exist in the SAST tool, the API automatically creates the Project and returns the Project ID to the Pipeline. The SAST Test Pipeline executes the SAST scan and sends the results to the SAST Tool.

The above described scenario becomes a 10-minute task for the developer to enable SAST scanning for the project. The developer simply adds a few lines of text to his project's .gitlab-ci.yml file. The value of abstracting the security tool's API, goes beyond reducing labor costs for operations and developers. Envision a scenario where management has decided it is time to switch scan tools or add a second scanning tool. IT deploys the new scan tool into the environment.

Remaining Tasks:

1. The DevSecOps API team adds support for the new scan tool to the API.
2. The integration team adds the new scan tool's CLI to the Docker container and modifies the Python script support the new tool.
3. If the abstraction layer is well-designed, the developer may not need to do anything in their CI/CD pipeline configuration.



API/CLI related items to consider when selecting security scanning tools (*Applicable to SAST, SCA, IAST and DAST tools*):

1. Can projects be created through the API?
2. Can users be added through the API?
3. Is access control manageable through the API?
4. Can scans be executed through the API or CLI?
5. Can scan results be retrieved through the API?
6. Can scan reports be retrieved through the API?
7. Does the API provide visibility into remediation activities?
8. Does the API support system configuration?
9. Some tools have the concept of public vs private project visibility. If the tool has this option does the API support setting the projects visibility?
10. Does the tool support policy configuration?

DAST projects typically require the operations team to configure traversal and logon sequences. However, integrations with DAST tool APIs can reduce the man hours required for the operations team to set up a project in the DAST tool. It is not appropriate to kick off a DAST scan every time code is check into a repo, but it might make sense to kick the scan off via the API when the pipeline deploys the code to a server that is dedicated to DAST scanning.

Runtime Protections

Runtime protections exist outside of the automated assessment toolchain but are still important to understand in the context of application security. When vulnerabilities are identified in a runtime environment, the remediation efforts may either inflict unacceptable



cost (for example, downtime in a critical application) or require more time to implement than is acceptable. To address that, the concept of virtual patching can be applied. Virtual patching is the implementation of a mitigating control at a layer other than the application code. While there are multiple ways of implementing this, the primary methods used are two similar technologies, Web Application Firewalls and Runtime Application Self Protection (RASP).

Web Application Firewall (WAF)

Web Application Firewalls proxy traffic between a user and the application servers, inspecting the traffic to determine if malicious attacks are being sent. By default, these are typically used to implement signature-based detections, although custom rules can be written to address specific vulnerabilities.

Runtime Application Self Protection (RASP)

Similar to IAST in the automated toolchain, RASP is an agent-based tool instrumented into the application's runtime. There, it monitors the runtime behavior for potential attack using a similar signature-based detection ruleset to the WAF.

While there is no real automation mechanism for creating virtual patches via these tools (as the rules will need to be custom created for each situation), the APIs and implementation of the patches can be automated. Utilizing the existing deployment mechanisms and scripts allow these patches to work in multiple environments with minimal overhead.



Implementation

Use Cases

The use of automation as part of the software security strategy provides organizations with an efficient, cost-effective approach toward a proactive posture regarding the identification and remediation of vulnerabilities. There are several use cases under which software security automation will help organizations minimize the risk associated with the software they manufacture: early detection, production elasticity, regulatory compliance, and metrics consolidation.

Early Detection

Traditional security automation efforts focus primarily in the Test cycle. DAST solutions are used to detect vulnerabilities as a gated approach to the prevention of rolling out to production potential security vulnerabilities that could put the organization at risk. The issue with such approach is the effort and impact of dealing with security vulnerabilities so late in the development process. Studies reveal that fixing a security vulnerability in a Test cycle will be 10 times more effective than if the fix is enforced at the early stages of the SDLC. An early detection strategy will require the implementation of two main automation tools to be successful before the build of the software is completed prior to test deployment: SAST and software composition analysis tools (SCA) (see Figure 4 below).



Figure 4. Security Controls needed to support security automation adopting an Early Detection strategy

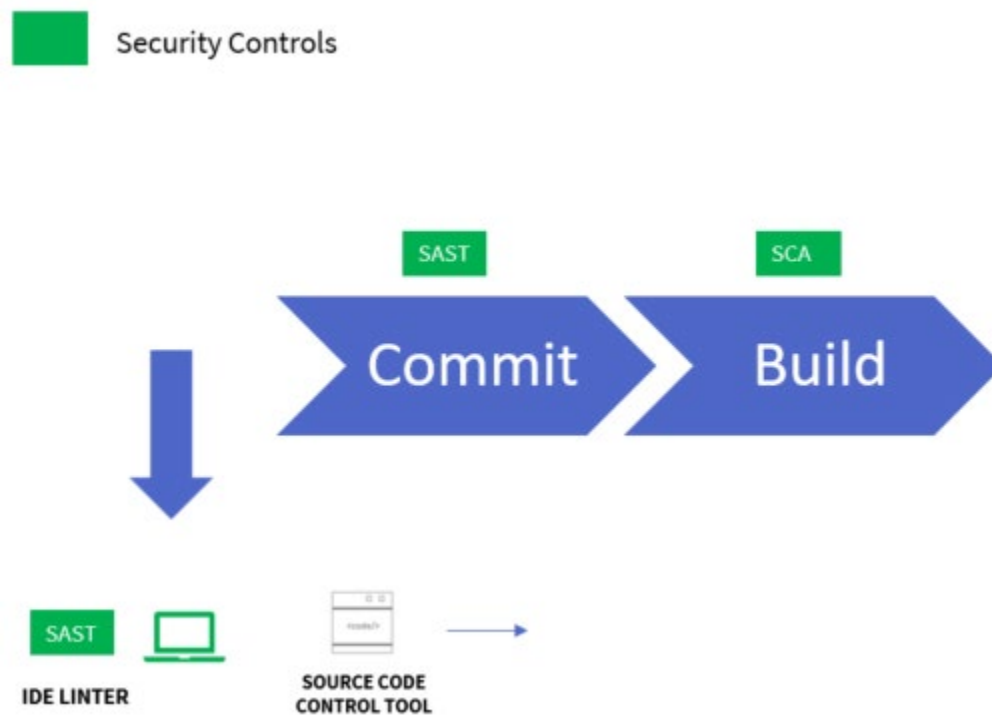


Figure 4. Early detection requires the implementation of security controls in the initial stages of the process. Three (3) common areas used to enforce early detection encompass the IDE environment, the commit process against a source code repository and the build stage.

SAST tools should be implemented as an integral component of the developer IDE. Much of today's SAST tools serve the role of security advisors, providing details and remediation strategies to allow the developer to fix the issue before code gets committed to the source code repository. SAST scanning should also be executed in the CI/CD pipeline when code is merged with the master branch. Implementing such an approach will create an environment where safe code takes precedence as one of the guardrails of software.



The second component that will be needed to implement in the early detection strategy is the SCA tool. With open source code presence in at least 95% of the software created globally, the identification and remediation of vulnerabilities associated with open source must be part of the early stages of the SDLC. SCA tools provide the capability to detect known vulnerabilities in open source code used in software projects. Some advanced SCA tools will provide enhanced features such as checks to determine if the function associated with a vulnerability is used in the code. This provides a more efficient approach toward the minimization of false positives in the scan results.

Regardless of whether organizations are using legacy waterfall, Agile Scrum or other methodologies, implementing SAST & SCA tools become integral components of the early detection strategy. Most of the tools will allow legacy methodologies to integrate SCA tool as part of their build process as a task. In a CI/CD pipeline, teams will have flexibility to configure risk tolerance associated with release gates while improving their vulnerability detection capabilities.

Production Elasticity

To be able to scale in both speed and quality without sacrificing the security stance of the software, automation strategies should allow teams to be able to roll out code to a production environment while providing intermediate protections. Such an approach allows



development teams to be able to work with a vulnerability fix while temporary security protections are deployed and in place. As part of this approach, technologies such as RASP, WAF, and Virtual Patching become key components in the production elasticity needed to enable flexibility without sacrificing agility and efficiency in the software delivery process.

One of the key components of the strategy is to leverage APIs. There is an important relationship between vulnerability scanning when checking software at the “Test” state/cycle and temporary protections in the production environment. For vulnerabilities where the organization may not have the bandwidth to immediately take action and remediate, having the capability of leveraging APIs in WAF and Virtual Patching is key. Organizations have the flexibility to feed the vulnerabilities identified by a DAST scan to security controls such as the WAF and create a layer of temporary protections for the production environment. This kind of strategy will provide flexibility to the software development cadence while protecting the organization against identified vulnerabilities.

Regulatory Compliance

An important aspect of today’s security posture in every organization is related to compliance and regulation. Automation can play a vital role in the organizational capability of policy enforcement and regulation compliance. Security tools provide features that allow an organization to create security policies in the tools and then enforce their implementation as part of the automation strategy. This way, security controls in the SDLC (e.g., SAST, SCA, DAST,



IAST, etc.) provide efficient guardrails to ensure the organization's software stays secure and in compliance.

SCA tools extend compliance requirements by allowing an organization to manage an inventory of valid open source libraries that have been vetted and regulated. This guardrail automatically enforces security compliance in projects and teams, extending the capability to the point where a build process can be stopped if a non-compliant library is found. In addition, SCA tools can identify licenses used in a project which could potentially expose the organization to legal risks. SAST tools could potentially detect hardcoded credentials in code and prevent a commit if a sensitive information policy looking for this type of scenarios is triggered. The same holds true for scenarios under which proper encoding is not used and executing a commit of the code could potentially introduce a Cross-Site Scripting attack vector. Using automation to implement these controls in the security controls put in place as part of the SDLC provides a cost-effective approach toward policy enforcement and security.

Metrics Consolidation

Smart decision making will require the availability of information associated with all the security controls put in place to protect the software. To understand the overall compliance, organizations are faced with the challenge of having multiple silos of information spread across the different software projects. There are two main trends in the consolidation of security related information and the generation of smart indicator to help the organization



with remediation as well as the planning needed to improve the security stance of the software.

On one hand, vendors are now trying to provision all the security controls (SCA, SAST, DAST, etc.) in a single platform to facilitate the centralization of the associated data and enable the much-needed view of the overall organizational security compliance. These new platforms are providing out-of-the-box dashboards and reports with the most common metrics used by teams to improve their processes. This kind of embedded automation, which is natively provisioned by the platform, creates efficiencies as teams are not required to allocate labor and license costs to capture security metrics from projects. On the other hand, independent security controls provide data export capabilities to serve as a data source for the aggregation of security information. Here, the use of automation can be leveraged to enable input feeds from security controls into a centralized repository which can be configured to provide the metrics needed by the teams. Due to the integration effort needed by this type of automation implementation, license and labor costs must be evaluated for feasibility.

False Positives

Regardless of which tools an organization implements (or how good their marketing team says they are), there will be false positives. Most tools will allow an organization to mark findings as false positives, but every application will generate a number of false positives that require addressing. There are several different ways to handle this depending on the



development team's level of training, how much trust security teams can extend the development teams, and how much time and resources the organization has available. Regardless of which of the methods chosen to implement for false positive handling, two initial steps are recommended in order to reduce overhead:

1. Provide read-only access to the source code repositories and, scan tool consoles to the Application Security team in order to enable review of SAST findings
2. Maintain lower environment test accounts for the Application Security team in order to enable review of DAST findings

Arranging this access for the Application Security team will decrease the time required for review while also allowing for the team to perform spot-checks and audits.

Once the applications have been onboarded and the reports are being generated an organization has several options for false positive handling:

1. Full Security Review
2. Flagged Security Review
3. Self-Service & Audit

In the first model, the Security team performs a full review of each report to identify false positives. Working with the developers and their knowledge of the application, they will either perform a code review or manual assessment to validate the finding. The development team is then given a remediation recommendation or the false positive is flagged for future



reference. The benefit to this model is that each report is reviewed and validated. The downside to this is that this method requires significant time and energy investment and as your development organization moves faster you will need more and more resources to keep up with the reports. As such, this model is not scalable outside of very small and slow development shops.

In the second model, the Development teams perform the initial triage of the report findings. The developers then flag the relevant findings as false positives and submit them to the Security team for review. The Security team gets the request, reviews with the developers as needed, and either provides a remediation next step or approves the false positive flag. Contrasting to the first model, this reduces the overhead for the Security team while moving to semi-self-service for the developers. However, the approval is still limited by the Security team's time and resources.

The final model is a full self-service approach, supported by regular auditing. Here, the developers are given the full ability to review, triage, flag, and approve issues as false positives. Under this model, the Security team will only be engaged if the team has questions, minimizing the amount of overhead time for handling. Security's role is then to periodically audit submitted false positives and monitor flagging rates to determine if there is any tool tuning or training for developers to be done. In the event of the Security team identifying developers purposefully marking false positives as a way to avoid remediation effort, teams



should be moved back to the flagged false positive model until they are able to rebuild the trust.

Aggregation & Correlation

Correlation and Aggregation are newer capabilities in security automation made necessary by the growing number of security assessment tools, the high volume of findings reported by these tools, and the continuous nature of modern software development.

Correlation is the association of a security finding discovered in one assessment type or tool, with a security finding in another assessment type or tool. For example, a static analysis tool may discover a pattern in code that indicates a vulnerability to SQL injection. A dynamic analysis tool may discover a similar finding by performing an injection on that same function in the running application. While the security analysis tools have discovered two findings, from a developer's perspective these are a single defect. Ideally security automation supports the developer perspective by correlating these findings and creating a single work item or issue for work.

Correlation reduces false positives and manual analysis by using more than a single testing methodology to verify defects. Correlation may also demonstrate higher exploitability and may be used to adjust the severity of a defect. Correlation also gives more context to a reported defect, by combining the analysis of more than one finding, reducing the analysis required by developers.



Aggregation is the bundling of many security defects into a single backlog item, or issue for work. For example, several SQL injection findings in a single application may be combined into one backlog item or ticket. Aggregation is especially important when performing analysis periodically rather than continuously during development, or when new analysis is performed on already-developed applications. When security analysis is not done while coding, the organization is likely to generate many findings that will be worked at some later time.

Any automatable application security testing tool should be expected to integrate with popular ALMs or issue tracking systems. If the goal is to shift-left and design-in rather than bolt-on security, then security remediation work must be automatically pushed into the developer backlog to appear alongside all other development work. On the other hand, producing a unique backlog item for every security finding will overwhelm development teams. Security findings need to be aggregated in some intelligent way before they are pushed to a product backlog for later analysis and remediation.

Examples of Grouping Factors used for Aggregation

- Finding Type. Group all Cross-Site Request Forgery findings into a single backlog item
- Finding Severity. Group all critical findings into one backlog item
- Code File. Group all findings found in a single file
- Code Library. Group all findings discovered in one library or module



A security automation tool supporting aggregation should enable organizations to determine the factors used for grouping findings into a single backlog item. These grouping factors should be further tunable by each product team to fit their development workflow and these grouping factors should be usable in combination. A combination of finding type and software module is a likely good starting place for many teams and should result in a normalized number of security defect backlog items across projects with few or many security defects.

For small development organizations with a few greenfield applications, correlation and aggregation are low value features as most of the remediation work is done continuously during development. However, for an enterprise with a large number and variety of applications, correlation and aggregation become critical features as the Security team integrates their tools into the developer workflow and the backlog is relied on to plan work. Automatically pushing analysis results into a product backlog will not work without some transformation of the analysis into workable packages.



Performance & Scalability

Software Security Automation Tool requirements differ based on the type of tool. Tool vendors implement varying approaches to address performance and scalability, and it is important to understand these differences in order to select the right tools for the job.

When selecting the right tool:

- Understand your organization's needs, the number of applications, and the number of developers
- Size your tools appropriately
- Review each tool's system requirements. Under sizing the server, can impact performance on the system and on dependent systems. For example, if a CI/CD pipeline is sending scan results to your Security Tool (and the tool is undersized), performance impacts can affect build and scans for other projects in the CI/CD pipeline.

When selecting tools, determine if the tool supports connecting to a remote database. Some tools such as, out of the box installations, result in the database being installed on the local file system. The database should be externalized to a database server located near the security tool.



A SAST tool analyzes an application's source code, binaries and/or byte code for design patterns and security flaws. Most SAST tools identify findings as defined in CWE, and the CWEs are mapped to other industry standards such as OWASP. SCA tools examine the application for open-source products. SCA tools support a couple Use Cases:

1. Open-source component approval processes and license compliance.
2. Detect an application's usage of open-source modules, frameworks and libraries to determine if the application is using components with known Common Vulnerabilities and Exposures (CVE).

Large applications SAST and SCA scan activity can be extremely resource intensive and time consuming. It is recommended to integrate security at multiple points in DevOps CI/CD pipelines. The scans should be implemented in ways that are largely transparent to developers and preserve the teamwork, agility and speed of DevOps and Agile development lifecycle. Performance and scalability of SAST and SCA tooling are critical in achieving these goals.

SCA scans typically are executed after a build, at the point when a CI/CD pipeline is deploying the application to QA and/or to production server. SAST scans are typically executed in a CI pipeline when a developer merges code changes with the master branch in their code repository.



When selecting an SCA or SAST tool, determine if tool has a CLI that can execute the scan in a distributed fashion. This is especially important to larger companies. For example, if a company has multiple business areas, each business area may have its own CI/CD pipelines implemented in TFS, GitLab and Jenkins. The distributed CLIs off-load the scanning activity to the CI/CD pipelines. Off-loading this activity to the pipeline runners and/or build servers mitigates potential bottle necks on the SCA or SAST tool's console.

When evaluating tool providers, take time to understand how their CLI works. The goal is to offload the scan to the CI/CD pipeline. In some cases, tool vendors have implemented CLIs that just send the application to a single scan server—resulting in a bottleneck. If the company has selected a tool that doesn't off-load the actual scanning to the CI/CD pipeline, make sure the product supports deploying the scanners behind a load balancer.

DAST performs black box testing against a running web application to discover security vulnerabilities. While DAST scans are not typically kicked-off from a CI/CD pipeline, they can still benefit from a distributed architecture. A DAST scan sends HTTP requests to a system to discover information about the system and exploit vulnerabilities. Every parameter, form field, and REST API discovered is repeatedly probed with tests to expose the vulnerabilities. The scan potentially sends tens of thousands of HTTP requests to a web application.



When selecting a DAST tool, determine if the tool includes Scan Agents that can be deployed throughout the Wide Area Network (WAN) and into your Cloud environments. The Scan Agents perform the scan and send the scan results to the DAST tool's centralized console. The Scan Agents should be installed near the web applications. One or more Scan Agents may need to be deployed in each of your data centers, and one or more agents may need to be installed in your cloud environments. When scanning web applications that are hosted in a cloud environment, make sure scan activity is coordinated with the cloud provider and adhere to policy and contract permissions. Scan activities that are not coordinated with the cloud provider, may interpret the DAST activity as a Denial-of-Service (DoS) attack and the cloud account could be shut down.

Future of Automation

Roadmap to transition

Greenfield companies can more easily embrace Agile, DevOps and Security Automation (DevSecOps). In contrast most, established companies have a large portfolio of applications that were built when Waterfall methodology was popular. As a result, a significant percentage of these applications may be using SCM tools, which are not capable of supporting automation. It is essential to migrate this technical debt to tools which support automation. Unfortunately, migrating these applications to CI/CD-DevSecOps capable tooling cannot happen overnight. During the transition, there are things that can be done to mitigate security risks:



1. Make Application Security training for developers a priority
2. Implement a Security Champion program. A Security Champion is typically a development team member that focuses on the project's security posture. The Security Champion is empowered to make security decisions for the team. The Security Champion mentors' developers and assists in triage of security findings. He or she is the liaison between the Security Team and the development team and recognizes when functional changes in a given Sprint merit threat modeling or additional security controls.
3. Provide developers with Security Linters for their IDE. A Security Linting tool works like a spell checker. The tool recognizes and highlights common security flaws while the developer is working with the source. Many of the SAST vendors provide linters that integrate with popular IDEs.
4. If the SAST vendor provides scan plugins for the developer's IDE, make these tools available to the developer. Keep in mind, the plugins should be light weight enough that a developer can install and use the tools with a low level of effort. If the plugin is difficult to use or impedes the use of the IDE while the scan is running, a busy developer will not embrace the tool



Figure 5. DevOps Methodology and cadence relationships

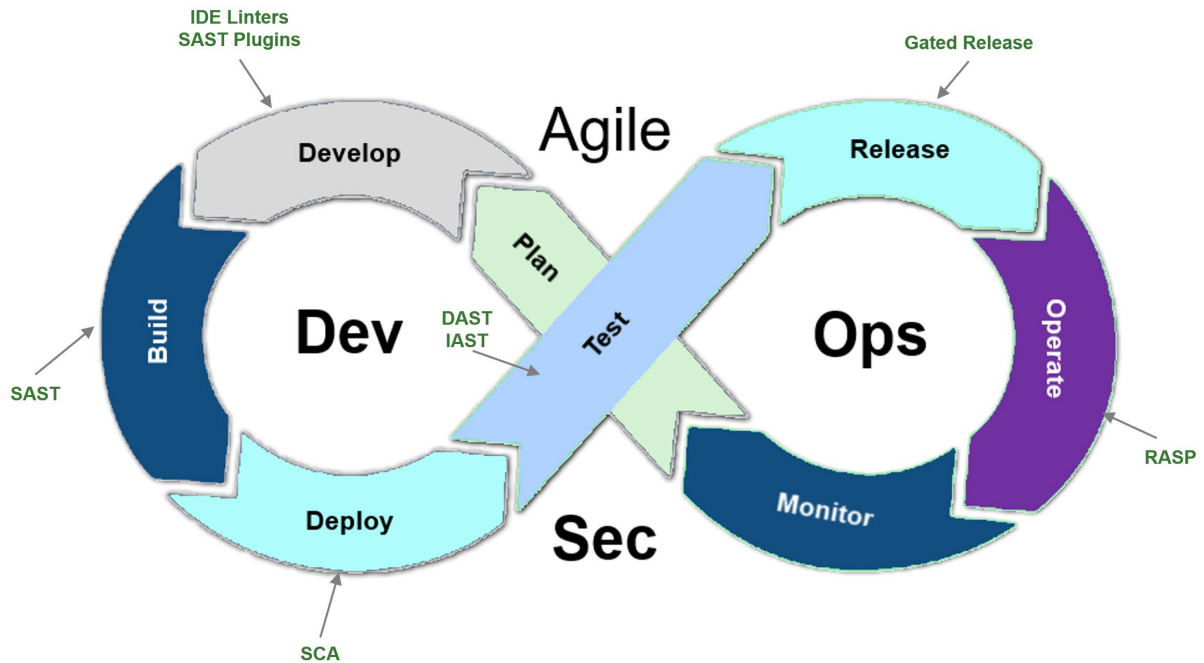


Figure 5. Agile DevOps, with Security Champions, Automated Security Scans, Gated Releases, Runtime and Continuous Monitoring

Not all IDEs support SAST plugins, and even fewer support SCA (Software Composition Analysis). A solution is to implement Self-Service SAST and SCA scanning capabilities. Most SAST and SCA tools have CLIs and APIs that can be leveraged to stand up a Self-Service capability for the developer. For a SAST scan, the developer zips up and uploads their source



code to a workflow. For an SCA scan, the developer builds the project, and zips and uploads the build artifacts to the workflow.

The Self-Service upload screen and queue were developed internally. As projects transition to CI/CD-DevSecOps capable tooling, they no-longer need the Self-Service capability. However, Self-Service will continue to be of value for Ad-hoc scenarios.

Figure 6. Docker container that was developed to execute scans for a CI/CD pipeline

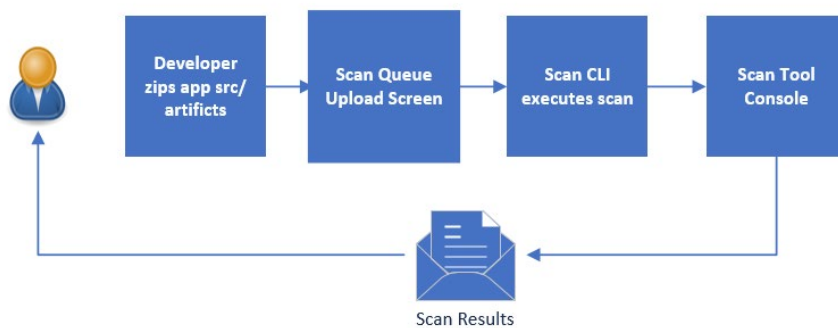


Figure 6. In this example the same Docker container that was developed to execute scans for a CI/CD pipeline was re-used for Self-Service.

CI/CD Pipeline Integrations with Security Components

As teams are required to be more agile in their software delivery process, adopting a continuous integration (CI) and continuous delivery (CD) approach provides a set of operating principles which will support a more frequent and reliable code change deployment process. With a CI/CD strategy in place, smaller changes are deployed at a faster pace. Automation becomes a key component of the software delivery process as the need to orchestrate the



software vulnerability identification and the provisioning of proactive protections for such vulnerabilities become extremely important aspects of the cadence of events.

Without the proper security controls in place to provide protections, a faster cadence will increase the probability of security vulnerabilities getting introduced into the software without any meaningful mechanism to identify or provision temporary protections. As the industry embraces more and more early detection as a standard practice in the Software Development Lifecycle, applying security controls at all stages of the process becomes an absolute must. With early detection, teams employ different tools as part of their pipelines to make sure security vulnerabilities are identified prior to reaching a test environment. Figure 7 provides a general overview of a CI/CD pipeline with the corresponding security controls which must be considered if teams are to be able to identify security vulnerabilities and provision meaningful proactive protection.



Figure 7. CI/CD pipeline showing security controls

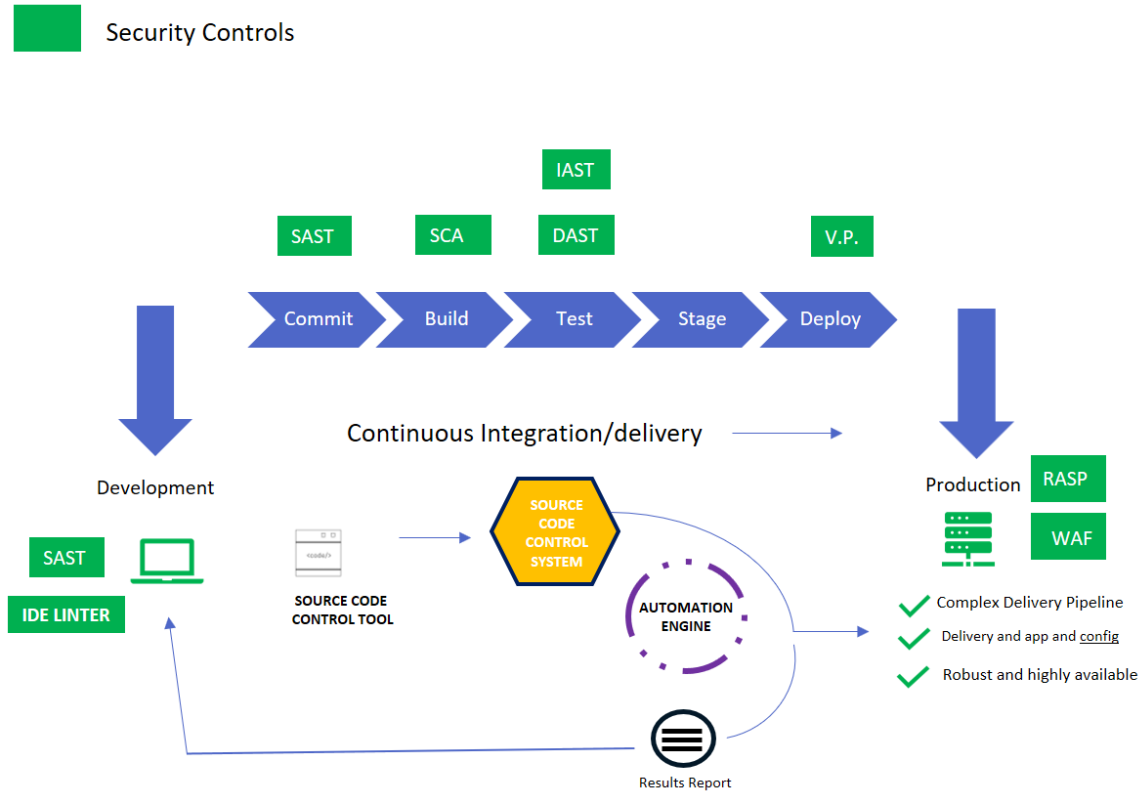


Figure 7. The CI/CD pipeline will encompass multiple stages. To provide a comprehensive security coverage, security controls will need to be added in some of the major steps in the workflow. Source code will have to be evaluated while executing a commit of the code to the source code repository. Open source libraries will have to be checked when building with an SCA tool. Blackbox testing will be conducted when the application moves to a test stage using DAST & IAST tools. Finally, when preparing to deploy the application to a production environment, tools such as WAF, RASP and Virtual Patching become an important ingredient in the proactive protection strategy.

The first basic component in the CI/CD pipeline is the SAST tool. SAST is configured to trigger when developers commit their code changes to the repository. This kind of interaction enables the examination of the source code for security defects and supports a secure code practice, providing teams with the flexibility to even stop the commit process if high/critical findings are identified. The integration of SAST in the CI/CD becomes a critical step in the



implementation of a sustainable program driving efficiency through the early detection of security vulnerabilities.

The second basic component in the CI/CD pipeline is the software composition analysis (SCA) tool. With an estimated 95% of applications using open source libraries per industry studies and having an estimated 67% of all open source libraries holding security vulnerabilities, dealing with their associated risk becomes another key aspect of the early detection strategy in the CI/CD. SCA tools allow development teams to identify security vulnerabilities in open source libraries even before a build is created. Risk appetite configuration with SCA tools will allow development teams to determine the level of risk they are willing to accept as part of their code change delivery process, which includes the capability of stopping a build in the CI/CD pipeline. Transitioning to the test environment, the importance of DAST tools as well as the IAST tools becomes evident.

DAST tools, in contrast to SAST tools, evaluate the behaviors of the application during runtime without the need to evaluate the code (which is one of the core functions of SAST solutions) for security vulnerabilities. From that perspective, DAST tools serve as black-box security testing tools, mimicking the mechanisms used by hackers to attack the software. During the test stage in the CI/CD, one of the necessary steps is to configure a DAST scan after the application gets automatically deployed to a test environment. DAST tools allow development



teams to define in their pipeline how much risk they are willing to accept before stopping a build.

IAST tools, on the other hand, combine the tests strategies from white-box as well as black-box testing. The effectiveness of IAST tools increase when combined with automated functional tests in the pipeline, as those tests will exercise the features of the application, exposing its behaviors to the IAST tools for evaluation. Just like with DAST, the proper configuration of IAST in the pipeline is to wait until the application gets deployed to the test environment, configure the IAST scan to trigger as soon as automated regression testing gets executed. IAST will provide real-time findings and a great API testing capability which could potentially support microservice testing.

As the CI/CD pipeline moves to the deployment and productions stages, proactive protections become important components of the risk-based decision-making. When deploying the solution to a production environment, teams will face a key decision point of whether to allow a deployment to continue even if a security vulnerability is identified during the test cycle. If the organization has the resources to implement a combination of virtual patching, RASP and WAFs, then the deployment can continue. Virtual patching provides a protection layer preventing the exploitation of a security vulnerability identified as part of the DAST testing. This type of protection becomes a great scalable solution as the implementation takes place



in few locations while there could be code used by multiple applications hosting the vulnerability.

WAF also provides development teams with flexibility when providing a remediation for the issue. Even though virtual patching is considered a valid intermediate solution for a security vulnerability, development teams should really focus in fixing the threat vector in code.

RASP is another security control which could be provisioned as a proactive protection in the runtime environment. RASP tools have the capability to identify real-time attacks and even control application execution if needed. When automating this tool, leveraging API features exposed in the RASP solution becomes an important component needed while enabling automation in the CI/CD pipeline through the configuration and protections capabilities.

Figure 8 shows the process executed when RASP gets implemented as a proactive protection.



Figure 8. RASP identification & protection process

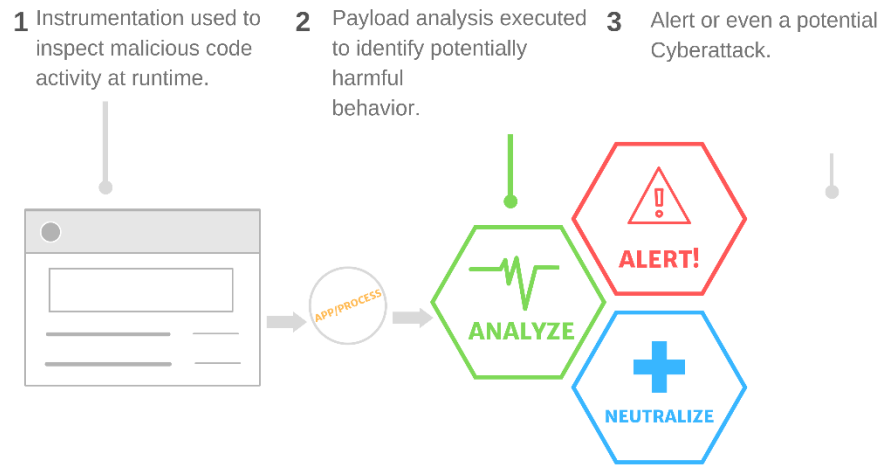


Figure 8. RASP protection gets provided through instrumentation. When a payload is processed in memory, the RASP tool will have the capability to inspect the code and identify if there is any potential harmful behavior. Once identified, the RASP tool provides the capabilities to block the behavior or alert.

Another similar solution which is associated with web applications is the web application firewall (WAF). WAF provides a protection layer which will also allow development teams a time flexibility when fixing security vulnerabilities. Just like virtual patching, the protections provided by the WAF should not be considered long term solutions; development teams need to fix the security vulnerability in the associated code. When running in the CI/CD pipeline, and just like with RASP, the API capabilities in the WAF will be needed to properly integrate the protections for the web application.

One final component in the CI/CD is the mechanism to provide a quick root cause analysis and reporting capability back to the development teams to plan for their incident response. The



strategy needs to be focused in an immediate feedback loop to the development team. With a feedback loop, development managers can efficiently invest their time and effort. The strategy should support diagnosing and addressing key security concerns identified by the different security tools embedded through automation in their CI/CD pipelines. A feedback loop guarantees that development team will have all the information needed when establishing priorities and remediation plans. Embedding security tools in the CI/CD pipeline alone is not enough; reporting back the results so teams can be efficient in their incident response is key.

Software Gated Releases

A common question by management when presented with an automated toolchain is, “How will we use this to prevent flaws from reaching production?”. One of the most common ideas is to block production deployments if scans are not completed or have failed.

While the writers of this white paper believe that this is a valid approach, we want to offer a quick word of caution about taking this route. This is a significant step and should only be considered by the most mature teams.

To prepare for implementing this control, the security team should be comfortable enough that they can be removed from the manual gating process. That is, the development teams have been given all the access and training needed to review and address findings, and



Security's approval should be implicit. Likewise, the target applications should have no backlog of security items. Finally, all the tools should be able to automatically report findings so that the deployment job can retrieve the application's status from either the tools or from a centralized reporting tool.

Do not forget to include a bypass mechanism into this control. If an urgent fix must be deployed to production, the above toolchain will still likely not be sufficiently fast for the business' need. In that event, the control should raise a warning and provide notifications to the relevant teams but allow the development organization to deploy.

Conclusion

Regardless of the SDLC methodology in play (legacy Waterfall or DevOps), software security automation is a key component of the overall organizational security strategy when protecting software in the SDLC. Multiple security controls are available to be implemented at different stages of the process to guarantee that all security concerns within the software are identified. From SAST tools serving as security advisors, to virtual patching allowing teams to provision temporary fixes while the teams work with an official remediation for a security finding, automating all aspects of software security is not just a practice, but a comprehensive strategy that must be embraced by organizations to properly minimize the window of exposure and the risk of a compromise.



References

Morgan, S. (2017, June 6). Cybersecurity labor crunch to hit 3.5 million unfilled jobs by 2021.

CSO. Retrieved from <https://www.csoonline.com/article/3200024/cybersecurity-labor-crunch-to-hit-35-million-unfilled-jobs-by-2021.html>

Tillyard, J. (2018, October 9). Security Automation vs Security Orchestration - What's the

Difference? *DFLABS*. Retrieved from <https://www.dflabs.com/blog/security-automation-vs-security-orchestration-whats-the-difference/>