Structuring Application Security Tools and Practices for DevOps and DevSecOps

Published 18 June 2020 - ID G00451296 - 80 min read

By Analysts Frank Catucci, Michael Isbitski

Initiatives:Security Technology and Infrastructure for Technical Professionals

Security and risk management technical professionals responsible for development, security and/or operations of applications must adapt practices to support modern DevSecOps. This assessment covers strategies and tooling that can be used to integrate application security throughout a DevOps cycle.

Overview

Key Findings

- DevSecOps provides great benefit to development, security and operations by promoting consistency and speed through automation. However, certain manual tasks within DevSecOps remain, and therefore, overall process is planned and considered carefully to ensure success.
- Reusable components are fundamental in DevSecOps, particularly with respect to open-source software (OSS). OSS reuse can introduce vulnerabilities into an integrated codebase, resulting in additional security burden of vetting third-party OSS components.
- Many tools offer native integration or web APIs to enable DevSecOps processes, but integrating the disparate security and nonsecurity tooling is a significant hurdle. Growing adoption of cloud and container delivery often increases complexity and can create potential security gaps.

Recommendations

As a technical professional focused on application security, you should:

- Use application security requirements and threat management (ASRTM) tooling to partially automate secure design activities. Refine as many of your security requirements as possible so that they can be verified programmatically and implemented using externalized security.
- Employ software composition analysis (SCA) tools to analyze packages for known vulnerable OSS components, as well as to manage component use throughout the DevSecOps cycle.

- Use infrastructure automation capabilities and cloud workload protection platform (CWPP) solutions to deploy applications to hardened systems or containers and monitor the application throughout its life cycle.
- Select security tools that provide native integration or full-featured APIs to connect with a broad range of DevOps environments. Use application security orchestration and correlation (ASOC) tools and capabilities to integrate security and nonsecurity tools.

Analysis

DevSecOps is not only relevant to organizations creating their own applications, but also important to organizations acquiring software. Organizations use a varying combination of acquisition, outsourcing and custom development. Production implementation also often requires creation of connecting software or custom code to make applications function as expected or required.

Security and risk management technical professionals responsible for development, security and/or operations of applications must adapt practices to support modern DevSecOps trends. This analysis covers application security strategies and tooling that you should integrate and automate within DevOps. These run from the initial-requirements analysis phase to the maintenance phase. A heavy emphasis is placed on structuring application security around development, build and deployment activities. This is where applications are being built in homegrown use cases or where software packages are being configured in acquisition use cases.

Absent a mature application security culture, teams will likely desire different outcomes that can impact priorities and tool selection, possibly even at the expense of overall security. Development teams require accuracy and speed. Operations teams want stability and resiliency of architecture. Security teams want thoroughness, broad coverage and assurance against weaknesses or vulnerabilities. A successful DevSecOps program has to account for all these aspects in processes and tooling. Ultimately, all teams must share the same goals and concerns.

> **DevSecOps cannot succeed without an overarching application security program. Some processes and technologies cannot or should not be fully automated. Manual practices may still help, or be required, to enforce or reinforce application security structure, culture and measurement.**

DevSecOps is about finding ways to tightly integrate application security, and other security practices, into development and operations processes and tooling. Ideally, that is with a high level of transparency and automation to avoid impact to teams. A standardized software development life cycle (SDLC), from process and system perspectives, is ideal but not always feasible,

depending on the needs of the product in addition to the technology stacks. One security solution may not necessarily be right for all applications because the SDLC used by each team — for example, Scrum, Kanban, Safe, LeSS or disciplined agile — may be different for each product or delivery team. However, a standardized SDLC will be essential to minimize the effort to integrate security and reduce the likelihood of security gaps.

Multiple continuous integration and continuous delivery (CI/CD) build pipelines may be a reality for your organization, especially where multiple technology stacks are in use. Each pipeline includes an associated umbrella of tooling such as integrated development environments (IDEs), CI/CD servers, version control systems (VCSs) and binary repository managers.

You are likely also dealing with a variety of infrastructure supporting your applications, including virtual machines and containers beyond just physical servers. These may be deployed on-premises, in cloud providers or both. You will need to integrate the security tooling and processes into each. In some cases, this can present challenges with vendor selection, where a given vendor may not integrate well with a given DevOps system or resource type.

As much as you can standardize or streamline DevOps practices and tooling (from early design phase through to later operations phase), you will improve the likelihood of success with secure DevSecOps. However, security should be careful not to force products or product teams to change tech stacks to conform with tooling or standards that don't apply to the application or product. This impacts a number of areas to an overall program as well as development, operations and security activities beyond the operational burdens and vendor technical challenges, including:

- Centralization and management of artifacts, the latter of which includes vulnerabilities (that should be tracked as bugs or defects)
- Metrics rollup for the purpose of reporting intrateam, interteam and to executive levels
- Promoting security and nonsecurity collaboration and awareness

Details on the various elements of an application security program are covered in depth in "A Guidance Framework for Establishing and Maturing an Application Security Program," as well as in "The Keys to DevOps Success" (webinar) and "Extend Agile With DevOps for Continuous Delivery." If you are unfamiliar with DevOps concepts and tools, the DevOps Practices and Technology section below can serve as a primer for some of the nonsecurity concepts and tooling.

Review Patterns of Application Security Focus

In talking with organizations and Gartner clients, four patterns of application security focuses emerge. All are critical to an overall application security program. But organizations may put heavier emphasis on a particular area due to a variety of factors, including organizational politics or structure, budgets for staffing or tooling, or presence of preexisting tooling that may be repurposed.

The focus areas are commonly one of the following:

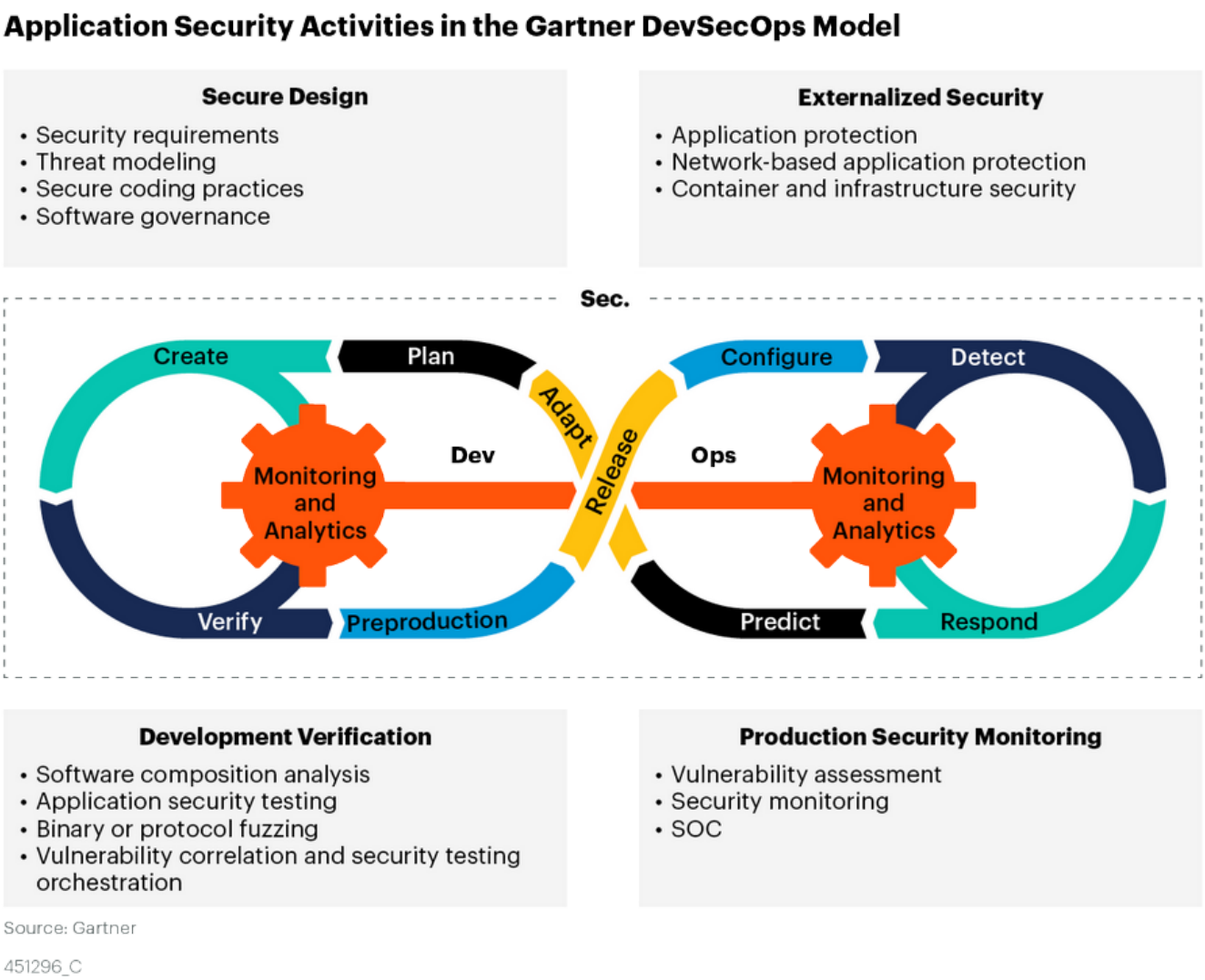
- **Secure design:** Here, there is a heavy emphasis on process and "building security in." Security requirements gathering and enforcement, threat modeling, secure coding practices and promoting use of trusted, externalized components are common elements. Standardize when possible.
- **Development verification:** Validation of secure-coding practices has been performed, security requirements are satisfied, and the application is reasonably free of weaknesses or vulnerabilities. Tooling like software composition analysis (SCA) and application security testing (AST) is used to verify.
- **Externalized security:** The focus here is on securing applications in production or runtime, mostly outside of the codebase. A variety of technologies come into play, including web application firewalls (WAF), application self-protection, API gateways, bot mitigation and application shielding. It may also include workload-specific security mechanisms, such as ensuring the hardening of server builds via continuous configuration automation.
- **Production security monitoring:** This involves continuous discovery and monitoring of applications and systems, within on-premises networks or CSPs. This is also the realm of security operations centers (SOCs), vulnerability assessment and vulnerability management.

All activities are critical for a successful DevSecOps approach. Where you choose to focus your efforts initially impacts which tools you need to procure upfront, as well as what you can integrate or automate.

Map Application Security to the Gartner DevSecOps Model

Figure 1 illustrates where these application security practices and technologies across the four focus areas map to the Gartner DevSecOps model.

Figure 1. Application Security Activities in the Gartner DevSecOps Model



Build Security Into the Design Phase

Activities here include security requirements, threat modeling, secure-coding practices and open-source software governance. The focus is on "building security in," ensuring that developers:

- Create secure code from the beginnings of a project
- Make use of vetted, trustworthy components
- Leverage externalized security as much as possible
- Keep the codebase free of known vulnerable components.

Details on the various elements, all of which are fundamental in an application security program, are covered in depth in "A Guidance Framework for Establishing and Maturing an Application Security Program."

Translate Security Requirements

Security requirements can come from a variety of sources in document form, including resources such as Center for Internet Security (CIS), National Institute of Standards and Technology (NIST) and Open Web Application Security Project (OWASP). Not all elements are specific to application

security and often impact other aspects of security including network, mobile, cloud and Internet of Things. Your own internal company compliance and security policies, unique to your organization, will be an additional source of requirements. Also, regulations specific to your organization's sector or industry are a source of security requirements. Though mandatory for affected organizations, organizations that are not affected may also elect to repurpose regulatory materials for their own use cases.

Unfortunately, while these materials can be great sources of application security guidance at points, they often read more like legal documents than technical references. There is a great deal of overlap between the numerous frameworks, in some cases also cross-referencing preexisting guidance like those from OWASP. There is also a largely manual step of translating these often-lengthy documents into nonfunctional requirements for applications you are creating or acquiring.

Without tools to aid in digesting the numerous security requirements and regulations for your specific industry/sector, you will need to invest time and effort in parsing these documents to create relevant nonfunctional security requirements. These nonfunctional requirements should live within other SDLC systems — namely application development life cycle management (ADLM) and integrated development environments. Once you've created the baseline library of nonfunctional requirements, you can associate them with current and planned application development or acquisition, refining over time as necessary.

Determine Security Requirements Specific to Business Logic

Many security requirements are domain-agnostic or not specific to business logic of the target application. Domain-agnostic issues manifest themselves as vulnerabilities that can be exploited and are easier to test for programmatically as well as protect against with current security tooling. Domain-specific issues are essentially abuses of business functionality, which are highly unique to a given organization. When performing the translation of security requirements, you will need to manually account for these differences with an eye toward verification. In other words, how will you programmatically verify that a security requirement affecting business logic has been fully satisfied (such as running through negative/abuse cases for a given functional requirement)?

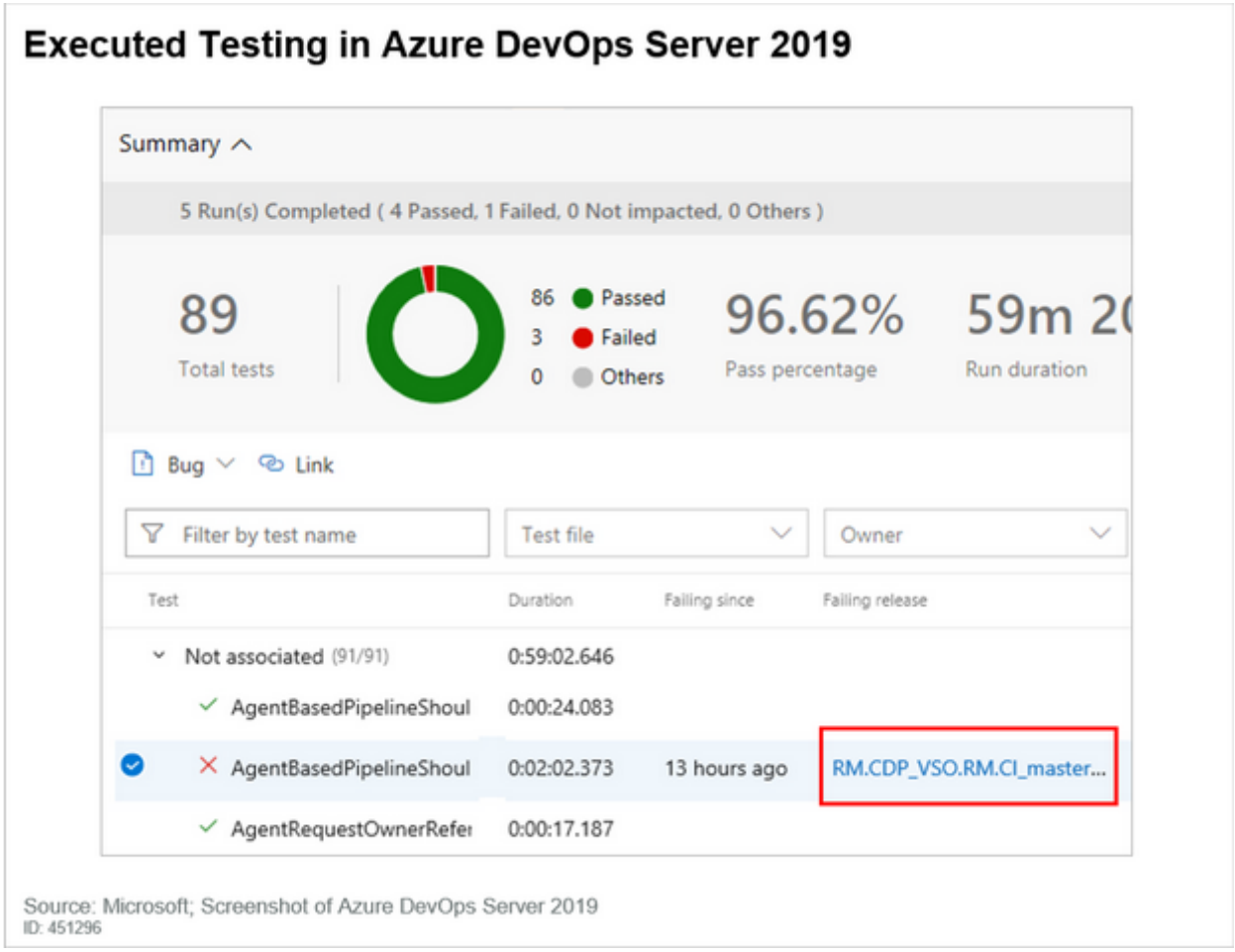
Domain-specific issues tend to be a weak spot for most security tooling and automation. This is due to uniqueness in application design and lack of awareness by the respective tool or protection. In the absence of machine-digestible test cases, test automation scripts and API schemas, effectiveness of security testing (such as AST) and application protections (such as with WAF) is somewhat limited outside of exploit detection.

Some vendors are investing in areas of behavior analysis and artificial intelligence/machine learning to improve product ability to "understand" baseline application behavior to proactively identify application abuse. However, solutions may be more reactive or technology is still emerging. Bot mitigation solutions, covered in the Deploy Network-Based Application Protection section, are one such example.

Track and Maintain Security Requirements With Appropriate Tools

Organizations sometimes attempt to build elaborate spreadsheets or dedicated web applications to publish and track security requirements. However, these efforts become difficult to maintain for larger application portfolios. It's a more effective strategy to create and maintain them within application development life cycle management (ADLM) tools that your organization is likely already using. An example of a security requirement being monitored is in the native DevOps tooling in Azure DevOps Server 2019. Executed testing can be useful for adherence to the set project requirements and tests. This can be seen in Figure 2.

Figure 2. Executed Testing in Azure DevOps Server 2019



Source: Microsoft; Screenshot of Azure DevOps Server 2019
ID: 451296

Dedicated security tooling that can aid you in this area is in the category of application security requirements and threat management. This tooling is focused on security requirements and is limited to a handful of vendors, including Continuum Security IriusRisk and Security Compass SD Elements, as well as the open-source OWASP Security Knowledge Framework. In some cases, the tools are also positioned as aiding the process of threat modeling because they identify potential threats and vulnerabilities.

The tools are focused on security requirements gathering and enforcement that are facilitated through self-questionnaires. They are designed to present application teams with simplified, dynamic GUIs that allow application owners or developers to rapidly select elements of their application and system design. Other security or compliance teams may preconfigure the tools to ensure that the appropriate corporate policy and regulatory requirements are also enabled by default. The tools parse the provided metadata about the application and system, perform the

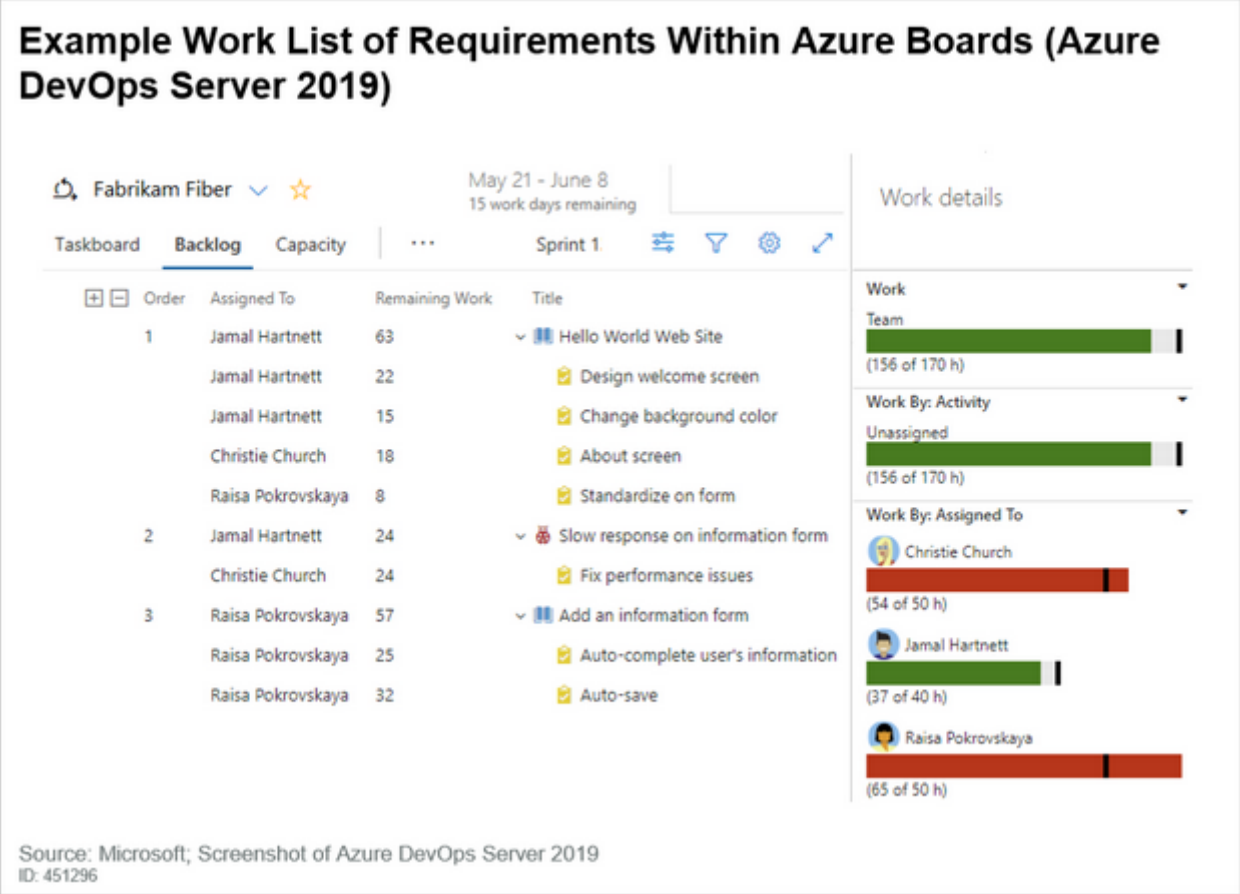
mapping to appropriate and/or necessary security requirements, and return a list of the relevant security requirements, possibly with secure-coding guidance, that should or must be satisfied for the given application design.

The dedicated application security requirements and threat management (ASRTM) tools provide GUIs that display results. But the reality is that, for DevSecOps, these requirements must exist within the systems that design and development are occurring in, namely ADLM and IDEs. Most dedicated ASRTM tools provide integration with the major ADLM and defect tracking systems, where these requirements can be tracked as nonfunctional security requirements. This can enable your organization to centralize in the SDLC ecosystem to help ensure application teams account for requirements as part of design and development.

Application teams can determine owners of security requirements. Also, security professionals must be part of this team. They must work with product management and the team members to identify the security requirements for the product, as well as for the processes, practices and tools, to ensure those requirements are met.

Owners can be identified programmatically if such mapping exists within the ADLM. An example of a work list of requirements in Azure DevOps 2019 is shown in Figure 3.

Figure 3. Example Work List of Requirements Within Azure Boards (Azure DevOps Server 2019)



The presumption is that the owner of a given requirement will satisfy that requirement fully and implement the appropriate control (or controls). But in practice, it may be tempting for a product owner to skip the necessary work. You can also configure the build process so that satisfaction of

security requirement work items is a prerequisite to initiating a software build. Realistically, you still need to verify all security requirements with a test tooling to account for factors such as human error or improper implementation.

Adapt Threat Modeling Practices

Unfortunately, the process of threat modeling is largely manual, even with available tools. It is certainly time-consuming for complex implementations as well, creating a challenge to fit into a DevSecOps program. The challenge is exacerbated with modern application design, where you may be dealing with hundreds of interconnected web APIs, as well as virtualized systems distributed on-premises and in the cloud. A given architecture diagram may look like a "spaghetti" or "Death Star" diagram and be less than human-readable.

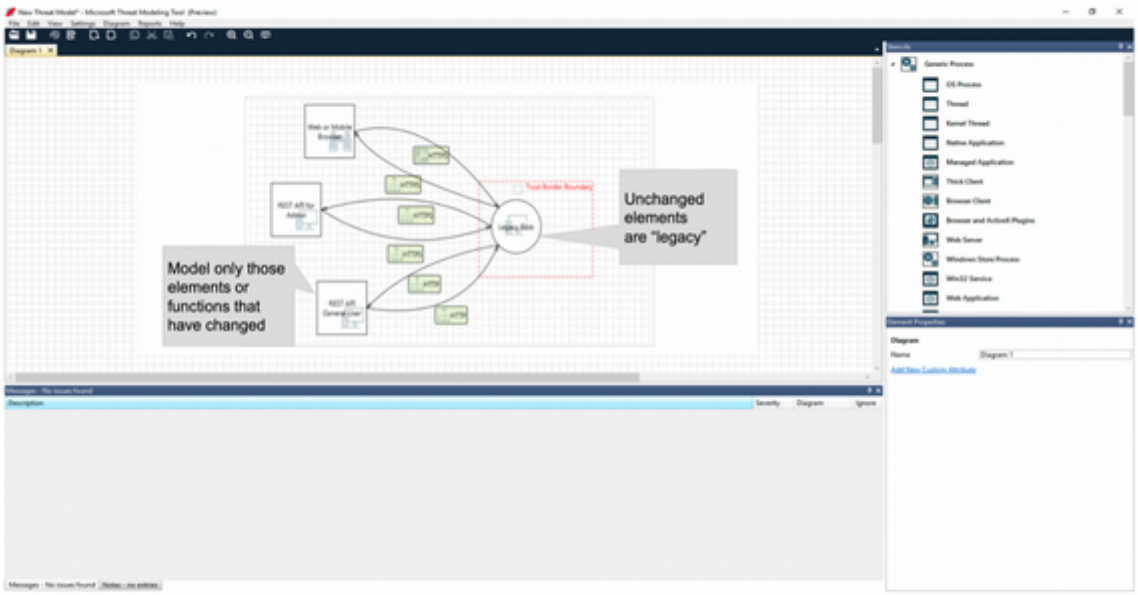
> **Threat modeling, while a valuable application security exercise, is not easily integrated into a DevSecOps toolchain. It is largely a manual process, and dedicated tooling does not yet exist to programmatically analyze an application design and all its interactions, end to end, with accuracy.**

As a result, many organizations skip threat modeling entirely, even in traditional waterfall development and non-DevOps approaches. However, the resulting knowledge of the target system and development of skill in threat/vulnerability identification are important processes to adopt into an organization's overall DevSecOps program and not just the automated pipeline. Enforce the principles of secure design, and increase awareness of teams by providing application security training. If an individual can rapidly identify a weakness as they're creating or configuring an application, he or she can proactively mitigate it without the need to test for or protect against it later.

To account for rapid, iterative changes that typically occur in agile development, incremental threat modeling is an option. [1] Using this technique, you model only new or changed pieces of the application or system, as opposed to the entire architecture, which would otherwise necessitate a complete threat model. It facilitates this through use of a "legacy blob" to represent preexisting system elements, as visualized in Figure 4. This can help reduce the amount of time to generate threat models because it confines the activity to only the changes.

Figure 4. Example Incremental Threat Model for Web Browser and API Interactions

Example Incremental Threat Model for Web Browser and API Interactions



Source: Microsoft; Screenshot of Microsoft Threat Modeling Tool (Microsoft Security Development Lifecycle 2019)
ID: 451296

You should still generate complete threat models at some point in the life cycle of the application or system, at least for your organization's most critical or sensitive assets. However, as you incrementally model the changing parts of the design over time, you are collectively building a threat model for the entire system, assuming active development of all features.

Although there is no tool available today to merge and combine incremental threat models into a full model, it is conceivable through manipulation of the raw XML data. Still, the knowledge and skills gained with the incremental threat modeling approach reduces the time it takes to generate or review complete threat models. As with traditional threat modeling, incremental modeling is also still a manual process and difficult to integrate or automate as part of the DevSecOps toolchain. However, it does result in simpler, more easily digestible threat models. This can translate to further time savings when translating the identified vulnerabilities into nonfunctional requirements or bug tickets.

Use Application Security Requirements and Threat Management Tools in Threat Modeling

You can use ASRTM tools that focus on textual security requirements if a visual model is not necessary for your purposes because most vendor tools perform a type of lightweight threat identification. ASRTM tools that focus on traditional threat modeling and produce visual data flow diagrams include foreseeti securiCAD, the Microsoft Threat Modeling Tool, Mozilla SeaSponge, MyAppSecurity ThreatModeler and OWASP Threat Dragon. Microsoft's tool is one of the most well-known and free to use, while the other vendors provide more advanced functionality or integration with SDLC systems.

The open-source ThreatModel SDK enables parsing of models created with the Microsoft tool to facilitate integration with defect tracking systems (that is, to ensure identified vulnerabilities are mapped to nonfunctional requirements or bug tracking tickets). [2]

The Mozilla and OWASP projects are open source, web-based and may lack some critical features that your apps or teams may require. However, they may be useful for producing simple models or, being open source, you may adapt the code to fit your needs. An area of U.S. Department of Homeland Security-funded R&D for a small set of application vulnerability correlation (ASOC) product vendors is to enable dynamic threat modeling of applications using instrumentation techniques and/or output of application security testing. Although the idea may sound promising, tooling beyond current ASOC capability still needs to materialize for this to become reality — that is, something you can employ for dynamic threat modeling in your DevSecOps program.

Distribute and Promote Secure Coding Practices

There are numerous security requirements that help define the types of controls you should put in place for a given application or system. However, they often provide little technical guidance on how to go about accomplishing this. Your secure coding practices must provide code-specific guidance to address security challenges that arise in application development. They should recommend trusted functions such as native secure functions, trusted third-party libraries or externalized security mechanisms over custom code.

One such example is secrets management. Secrets such as authentication credentials, Amazon Web Services (AWS) access keys or encryption keys typically should never be stored in codebases or public repositories. What mechanisms are you providing to development teams to facilitate secrets management, and are you providing corresponding code-level guidance so that development teams can implement the functions correctly?

Secure-coding practices and technical guidance come from a variety of sources, including the Computer Emergency Response Team (CERT) division of the Software Engineering Institute, OWASP and each respective vendor for a given technology. Others can be adapted from AST tools such as static application security testing (SAST). Security may be incorporated as part of design (presuming a development team hasn't purposefully circumvented) or, more commonly, inadvertently passed because most developers do not fully understand deep secure coding practices. Security has traditionally been something that is validated near the end of the delivery cycle when it is difficult to change.

> **You must account for the potentially numerous languages and technology stacks in use in your organization. Seek to standardize as much as possible to ease DevSecOps efforts overall.**

This is very much a matter of software development life cycle development maturity, specifically formalization of the technology that development teams use and the associated development processes. Once you have made those decisions, you can draft secure coding practices using the variety of technical reference materials available.

Ideally, this is a collaborative effort among DevOps teams to account for language and technology stack complexities. Additionally, these practices promote application security awareness among product management, security, development and operations teams. This is a manual process similar to the translation of security requirements from document form into nonfunctional requirements as part of your application development life cycle management. However, information on secure coding is less linked to the SDLC ecosystem, and integration with developer IDEs and ADLM is not well-supported today. It requires some creativity to get guidance to developers where it is most essential, such as storing and linking information within ADLM or some other knowledge management system. Further information may be found in "How to Build Successful Communities of Practice for Knowledge Management."

Employ AST Tooling to Promote Secure Coding Practices

Ideally, IDE tooling would advise the developer how to create secure code or utilize secure functions of the given language or framework, like how IDEs suggest functions and variables as a developer codes. Application security testing tools fit the bill here and are not limited to just verification activities.

When integrated properly, AST tooling, especially SAST, can provide immediate feedback to development teams as they build applications. AST tools have the added benefit of providing details on why vulnerabilities arise, recommendations on how to resolve vulnerabilities, and code samples to help understand the concepts and mitigations.

The three methods of delivery here are:

- **"Spellchecker" SAST:** This tool advises developers of weaknesses or vulnerabilities as they code in the IDE. This is limited to a subset of vulnerabilities because the tools do not have full context of the complete codebase or the plug-ins' focus on high accuracy over scanning depth. Some vendor products in this space include Micro Focus (formerly HPE Software) DevInspect and Synopsys SecureAssist.
- **SAST ad hoc scans:** Initiate ad hoc SAST scans via an IDE plug-in and receive results nearly immediately, depending on complexity and lines of code in the given codebase. Results may be less accurate (that is, higher false positives and false negatives) than running SAST during CI as part of build scripts because it may not include the entire codebase or all integrations. Most, if not all, SAST vendors provide this capability.
- **DevOps-friendly SAST ad hoc scans:** Initiate ad hoc SAST scans via an IDE plug-in with a focus on higher accuracy, precision and actionable guidance. Results may be returned within the IDE or in a separate web application. Some vendor products in this space include Veracode Greenlight and WhiteHat Scout.

Automate Governance of Open-Source Software

Externalized security, beyond those vetted libraries you build or other externalized security capabilities you implement, includes the use of trustworthy open-source software components.

Software composition analysis (SCA), which now encompasses open-source software governance (OSSG) tools, provides functionality to facilitate management and use of OSS components throughout the life cycle of an application. To put it another way, detection and reporting of a vulnerability in an OSS component — as with software composition analysis alone (point in time or continuously) — simply isn't enough. Other mechanisms need to be in place that can advise teams of OSS issues during development, build and distribution phases, integrated with the numerous SDLC systems. While useful as part of secure design, the capabilities are beneficial for verification and production security monitoring focus areas.

SCA solutions now provide multiple integrations with SDLC systems and support additional workflow around OSS component use to fill a variety of requirements in DevSecOps, including:

- Detect and expose outdated OSS component versions during source code commits and throughout the life cycle of the application.
- Detect and expose OSS components with known vulnerabilities, common vulnerabilities and exposures (CVEs) or otherwise during source code commits and throughout the life cycle of the application.
- Uncover OSS use that violates open-source licensing.
- Provide visibility of OSS component use enterprisewide (that is, for a given OSS component, they expose which applications in an organization's portfolio make use of it and may be impacted).
- Advise development teams of recommended OSS components and latest versions as they browse for them, fetch them within IDEs, or pull components in from public repositories.
- Alert on or block development teams from pulling in known vulnerable components and storing in organizational code or artifact repositories.
- Alert on or fail software builds within the continuous integration and continuous delivery build pipeline if known vulnerable components exist.

Vendor products in this space include Black Duck by Synopsys, Revenera FlexNet Code Insight, JFrog Xray, Sonatype Nexus IQ Server, Snyk, Tidelift and WhiteSource. JFrog and Sonatype leverage and integrate with their respective artifact repository products to facilitate the full SCA functionality, which are Artifactory and Repository Manager, respectively.

It is worth emphasizing that the use of and standardization on version control systems and binary repositories is especially critical in DevOps and DevSecOps. This minimizes the amount of integration work you need to do to integrate verification tooling. It also will reduce the likelihood of vulnerable components or those that have not been vetted creeping into your application codebases.

As a general best practice, you should promote the use of the latest versions of OSS components whenever possible and where other code dependencies permit, which the SCA tools help facilitate. Vetting the numerous OSS libraries and versions can be a fool's errand given the rate of change, and leaning on tooling is essential, if not mandatory. Some organizations will vet OSS libraries (typically with the aid of tooling), store them in a centralized repository for developers and block access to other sources.

Restrictions are often implemented via:

- **SCA blocking mechanisms:** Block OSS components as they are imported into the IDE, checked into a VCS or binary repository, or committed for build as part of CI/CD
- **Secure web gateways (SWG):** Block access to public repositories
- **Cloud access security broker (CASB):** Block access for reading/writing of code or artifacts in public repositories

This can be a resource-intensive endeavor, even with appropriate tooling. Without the aid of tooling, guaranteeing a level of security in OSS components will be difficult, if not impossible.

Analysis of the social media conversations on application security testing confirms the year-over-year increasing mentions of software composition analysis from 2017 through 2019 (see Figure 5). Conversations on mobile application security testing also grew in popularity on social media with focus on its ability to reduce risk of data breaches, to test and to identify all potential vulnerabilities, as well as to ensure safety requirements of a mobile application meet all adequate security compliance.

Figure 5. SCA Importance and Focus Shift From 2017 Through 2019

SCA Importance and Focus Shift From 2017-2019
Top 10 Conversation Drivers Across Years

▲ Increase ▼ Decrease ■ New Addition ● No Change

2017	2018	2019
1. DevOps	1. Web Application Testing ▲	1. DevOps ▲
2. Web Application Testing	2. DevOps ▼	2. Software Composition Analysis ▲
3. Interactive Analysis	3. Software Composition Analysis ▲	3. Vulnerability Management ▲
4. Penetration Testing	4. Interactive Analysis ▼	4. Web Application Testing ▼
5. Vulnerability Management	5. Penetration Testing ▼	5. Penetration Testing ●
6. Software Composition Analysis	6. Code Review ▲	6. API Testing Security ▲
7. Code Review	7. Vulnerability Management ▼	7. Mobile Application Testing ■
8. API Testing Security	8. Bug Bounty Program ■	8. Code Review ▼
9. Threat Management	9. Defect Detection ■	9. Bug Bounty Program ▼
10. Machine Learning	10. Security Vulnerabilities ■	10. Security Vulnerabilities ●

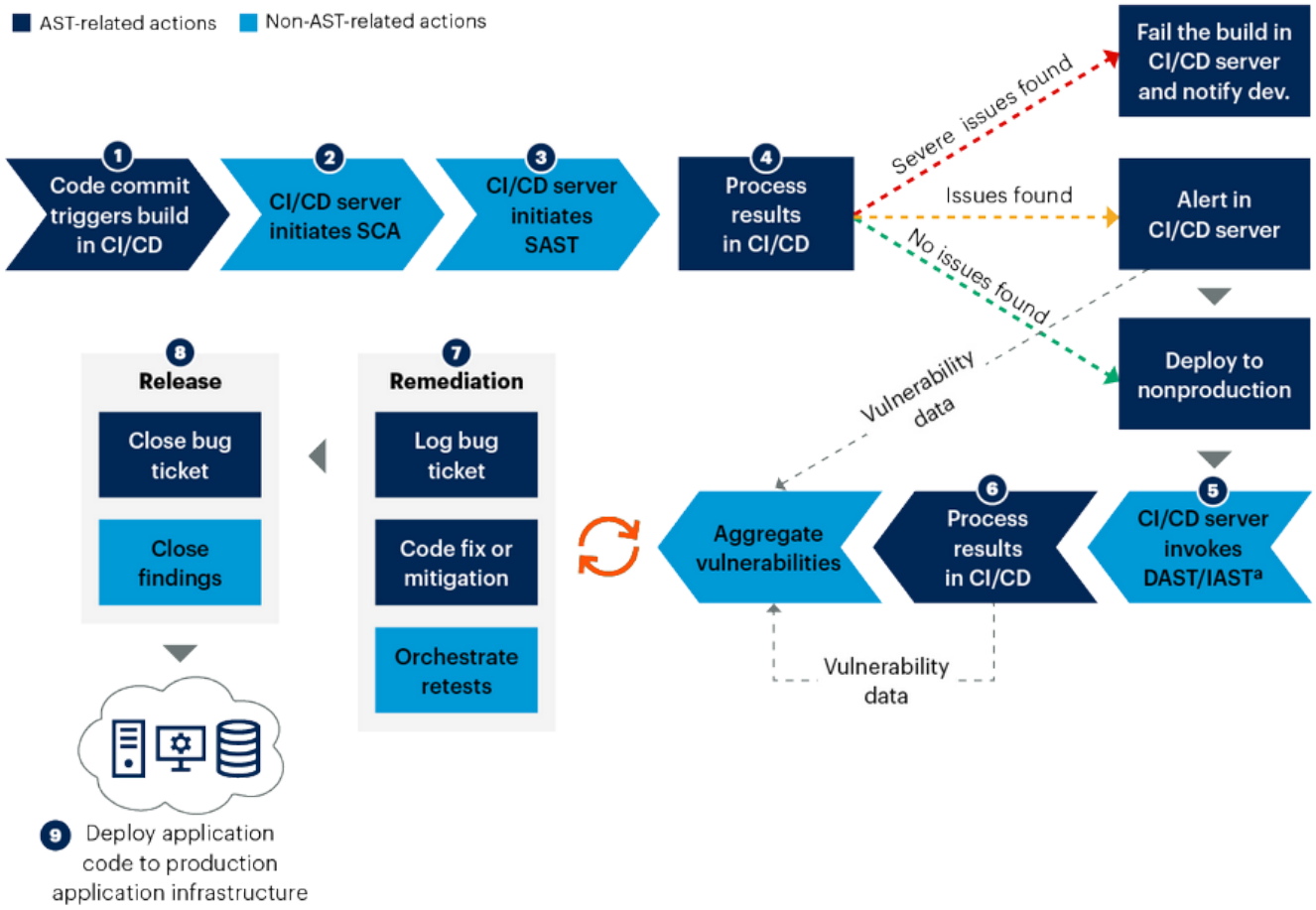
Source: Social Media Listening Tool; Date Range: 1 January 2017–25 December 2019.
451296_C

Choose Relevant Tooling for Development Verification

These activities are a dominant focus within a DevSecOps program. Technology options are more plentiful and readily integrated into development systems. Relevant tooling consists of SCA, AST and fuzzers. These capabilities are covered in depth in "How to Integrate Application Security Testing Into a Software Development Life Cycle." Figure 6 provides a conceptual visualization of continuous testing within a CI/CD build pipeline and of handling output of tools.

Figure 6. Integrating Security Testing Into a CI/CD Build Pipeline

Integrating Security Testing Into a CI/CD Build Pipeline



Source: Gartner

IAST = Interactive Application Sexurity Testing; CI/CD = Continuous Integration/Continuous Delivery; DAST = Dynamic Application Security Testing; SAST = Static Application Security Testing; SCA = Software Composition Analysis

451296_C

Integrate Application Security Testing Into DevSecOps

The type and number of AST tools that you will require are a factor of your application portfolio. As a prerequisite to selecting tools, you need at least a base-level understanding of what applications exist within your organization as well as what is being actively developed or procured.

In the context of DevSecOps, certain types of AST technology are easier to integrate or automate than others:

- SAST is most easily integrated into CI/CD build pipelines.
- DAST is heavily dependent on test automation.
- Interactive application security testing (IAST) is useful for real-time testing independent of the CI/CD pipeline.
- Mobile application security testing (MAST) is the least automation-friendly due to the mixture of techniques.

Embrace Iterative Testing Over Full Testing

Integrating and automating AST requires that you incorporate tool configuration and execution into the build pipeline. With agile development and DevOps, there is a higher frequency of incremental changes to an overall application, as opposed to traditional waterfall or monolithic approaches where releases might be monthly or quarterly. Ideally, you want to target only those functions and sections of the more monolithic codebase that have changed. Iterative scans need to be initiated more frequently, possibly weekly or even daily depending on the pace with which development teams are modifying code.

This is accomplished through:

- Using web APIs, native integration or command line interfaces (CLI) to enable communication between security and nonsecurity tools
- Adjusting policies and configurations (ideally programmatically) in each AST tool to restrict scope to just the relevant changes in the application and by running only the appropriate checks

Full scans should be performed on a regular basis, regardless of company policy or industry regulation. Scans should leverage integration with the CI/CD pipeline to ensure they are always performed. They may also need to be run asynchronously outside of production release windows or scheduled to run ad hoc based on defined security policy. The traditional "kitchen sink approach" with scanning can and will stall the CI/CD build pipelines, making the scheduling aspect important. No production release should occur without some type of security testing, but it is a question of whether an iterative or full scan is most appropriate.

Restricting scope doesn't equate to testing only for subsets of issues such as those contained in the OWASP Top 10. [3] Rather, it's more about scanning for issues relevant to what was changed in code. If no code that impacts application input was modified in the latest code change, it should be unnecessary to rerun checks for vulnerabilities like cross-site scripting (XSS) or SQL injection. This is also where ties back to functional and nonfunctional requirements (and integrations with those systems) can help guide automated testing.

Leverage Requirements, Test Cases and Test Automation

Reusing test automation processes and tools for security testing should be high on the priority list for DevSecOps. Technically, these are critical if not necessary for DevOps, let alone DevSecOps. Gartner research in the area of application test automation includes:

- "Use Test-First Development Processes to Jump-Start Your SDLC"
- "Solution Path for Testing Software Applications"

For information about concepts for building continuous delivery and security, see:

- "Solution Path for Achieving Continuous Delivery With Agile and DevOps"

- "Best Practices for Securing Continuous Delivery Systems and Artifacts"

Clearly defined requirements and unit tests should be reused to help address some of the challenges that arise with incremental SAST. And test scripts from functional testing should be reused to help alleviate some of the complexities of incremental DAST, particularly obtaining authenticated context and exercising applications. Factor these into your AST tool selection, where it becomes a requirement that the tools accept the various test cases or test scripts as input to control scanner behavior. When integrating with these traditional development and quality assurance (QA) practices and tooling, there is the additional benefit of aligning skill sets across development, QA and application security teams. QA teams are also adapting to the realities of DevOps, similar to how security teams are being challenged.

Integrate SAST Tooling Into Development Systems and CI/CD Build

SAST tools are often marketed as one of the easier fits for development environments and a CI/CD build pipeline due to where they integrate with code commit processes and build tools such as Ant, Maven and Gradle. They also typically complete scanning within a reasonable time window for a given release, barring cases of extremely large or overly complex codebases. Unfortunately, while SAST may be relatively easy to integrate and run quickly, it has sometimes earned the reputation of generating too many findings or too many false positives.

Higher volumes of findings are largely inherent in product design because SAST tools evaluate the codebase itself, as opposed to the compiled application. If you desire further vetting of results to verify exploitability, you will need to leverage DAST or perform (out-of-band) manual testing. False positives may also be symptomatic of detecting issues in external dependencies like OSS components or reporting duplicate instances of vulnerabilities. Address the OSS drawback with dedicated software composition analysis or open-source software governance tooling. Use application vulnerability correlation capabilities to handle the issue of duplicate findings, which is covered in further detail in the Use ASOC to Aggregate Verification Results section.

With multiple code releases and incremental updates prevalent in agile and DevOps, it also becomes an issue of whether the SAST tool is evaluating only the relevant changes or deltas as opposed to the entire codebase. Otherwise, you may be uncovering issues already detected in the full codebase and wasting time and resources scanning code that has already been evaluated. This can be satisfied with incremental SAST scans.

Repurpose Test Automation and Tune DAST for CI/CD

Unfortunately, dynamic application security testing has a downside of being very unpredictable in scan completion times. This is particularly true with modern, dynamic, single-page applications (SPAs). These SPAs technically consist of some JavaScript presentation layer that interacts with any number of back-end web APIs (typically representational state transfer [REST]) to send and receive data.

Organizations fall in one of two camps in how they employ DAST in their CI/CD process:

- Run DAST scans asynchronously, triggered as part of CI/CD but not blocking for a release.
- Run DAST scans synchronously in the CI/CD build pipeline, blocking the release until scanning is complete.

With the second scenario, the DevSecOps pipeline becomes an impediment to code, app, services or script releases. Some organizations choose to take this approach. However, running DAST scans synchronously may impact the business, depending on timing of when “final” code is presented for verification and the ability for the DAST scanner to analyze the compiled application in a timely manner. Security processes (or latency of them) may be questioned as a result, especially if your DAST implementation is not fine-tuned and automated. Production release gating is usually implemented as part of CI/CD by failing builds, as well as in change control processes. Variances in scan completion time can make DAST challenging to squeeze into a CI/CD build pipeline.

Reduce DAST scan times by restricting URL scope, tweaking scan policies for specific categories of vulnerability or limiting scanning to relevant code changes. All are a form of incremental scanning and can be implemented programmatically with most vendor products.

The third approach is more complex, requiring tighter integration between development, QA and application security teams. In those cases, repurpose functional test automation scripts or employ test-driven development (TDD) or behavior-driven development (BDD) concepts to limit testing to only the relevant changes. It is also recommended, ultimately, to rearchitect monolithic applications into more modular/componentized applications. This will speed all development and testing activities by isolating change and reducing risks.

The use of wrappers or middleware that employ BDD concepts and technology for the purpose of DAST is still nascent. There are a small handful of OSS options with limited functionality that exist as starting points. Unfortunately, most are in maintenance mode and lack integrations with the majority of commercial tools. However, they can be customized for use with other security testing tools in most cases, provided the tool accepts input and provides output through some scriptable means. The topic of a framework or orchestration layer for AST is further analyzed in the Orchestrate Application Security Testing section.

Evaluate Interactive Application Security Testing If DAST in CI/CD Is Problematic

IAST vendors market their products as ideal fits for DevSecOps, justifiably for the most part. Theoretically, IAST can detect and report issues as the application is running, addressing the “continuous” aspect that is fundamental in DevOps. Coincidentally, hooks into repositories or continuous integration/continuous delivery systems become less necessary because the technology tests continuously and automatically.

IAST will be limited to languages and modern-technology stacks because it depends on instrumentation and installation of agents onto application servers to power scanning. This means your Java and .NET applications are likely supported, along with the possibility of Ruby, Python or Node applications, depending on the vendor. Keep vendor selection limited to self-

testing IAST variants in order to avoid the challenges of having to automate separate DAST scans. As users or testers perform nonsecurity testing against a given application, IAST observes application calls and identifies issues within the instrumented application code. Drive application behavior using functional-test-automation techniques and tooling (as in DAST), or piggyback on other manual testing that happens within your SDLC.

IAST providers also often provide some type of SCA-like capability to detect known vulnerable components. These tools have this visibility by virtue of sitting in the application stack and seeing what OSS components are being invoked as part of the application codebase. Results of IAST testing are more real-time and not be triggered by ad hoc scans or builds as part of CI/CD. As a result, findings may come in sporadically during regular FT, UAT, and so on and as a side effect of changed code or components. Performance testing may be the right time to do this. Automated functional testing executed via CI may work as well.

The IAST tools commonly provide a GUI for managing findings, similar to what you would find in SAST and DAST tools. Like those other testing technologies though, the results are best output to defect tracking systems so that vulnerabilities can be tracked as part of the SDLC and mitigated accordingly.

Automate Elements of Mobile Application Security Testing

Of all the AST types, full MAST is typically more of a manual effort due to the number of component interactions that are inherent in mobile application design. Mobile applications may involve the use of:

- A web application back end (as with web applications formatted for mobile device form factors)
- A lightweight or heavyweight on-device binary (as with hybrid mobile or native, rich-client mobile applications)
- Collections of web APIs to facilitate business functionality

SAST and DAST automation challenges still apply, with additional hurdles of behavioral analysis on-device or in an emulator. If you desire in-depth analysis of all components and interactions for a given mobile application, you will likely need to reserve it for out-of-band activity outside of CI/CD.

Pieces of MAST can be automated depending on the vendor, but unique tooling is required to perform in-depth analysis against pieces like a native, rich-client component. In some cases, these tools cannot be automated and may still require manual operation. "Market Guide for Mobile Threat Defense" provides details on vendors in the MAST space. Vendors providing mobile threat defense (MTD) and mobile application reputation service (MARS) also sometimes compete here. Strategies for mobile security are examined further in "Advance and Improve Your Mobile Security Strategy."

Use API Schemas and Test Automation for Effective Web API Testing

Use SAST tools to test source code for web APIs regularly, provided the original source is available and the language is supported by the scanner. Your DAST tools may encounter issues navigating and testing SPAs and the corresponding web APIs due to higher complexity and variation in how web APIs function. REST is a style of web API development and not a standard like that of SOAP. A given web API can be designed in a large number of ways. What looks like a directory path in the URL might actually be a function, a variable, an attribute or something else. If the REST API also exchanges data via JavaScript Object Notation (JSON) payloads, understanding the structure of the data for requests and responses is yet another hurdle to overcome.

API schema definitions (typically Web Services Description Language [WSDL] with SOAP, API Blueprint, OpenAPI/Swagger or RAML with REST) provide machine-readable data so that a given tool understands how to exercise a web API. Development teams often model APIs with special-purpose tools, and resulting schema definitions can be exported for use in other tooling. The topic is covered in detail in "A Guidance Framework for Creating Usable REST API Specifications." Your DAST tool needs to be able to accept these schema definitions as input in order for it to be useful for guiding the security testing. Not all DAST vendors support these today, and OpenAPI/Swagger is probably the safest bet if they do support it.

Performing effective DAST scanning directly against a web API is difficult at best. Web APIs can't be crawled or spidered like a traditional web application. In the absence of schema definitions for web APIs, you will need to leverage some interfacing application that consumes and exercises the web API. The DAST scanner would observe these requests and responses, and then analyze and manipulate the traffic to uncover vulnerabilities. If test automation scripts, such as Selenium, are present, then these should be used to drive the DAST scanner.

IAST does not suffer from these same issues because it instruments the application code as it is running and does not rely on URL endpoints. This is presuming that you are using a self-testing IAST tool that is not dependent on a DAST inducer, and that the IAST agent supports the given technology stack powering the web API. Testers still need to use the applications that make use of the APIs so that IAST can observe the internal application calls. Or, you could leverage test automation tools and scripts that exercise the API. If a DAST inducer is required by the IAST solution, then the same hurdles present in traditional DAST remain.

Integrate Fuzzing in DevSecOps

Integration of fuzzing into DevSecOps is feasible to an extent. It may be possible to initiate a corresponding fuzzing tool as part of the CI/CD build pipeline, having the fuzzer itself run without intervention. But fuzzing tools require significant configuration upfront to be effective. The complexity is similar to DAST, which is essentially a type of HTTP fuzzer itself, but the wider range of potential variables worsens the problem. This variation includes application behavior, file formats, communication methods and input types, among others. As a result, fuzzing may produce different results over a series of test runs because of the high number of variables. Like

DAST, scan completion times are unpredictable, meaning fuzzing will likely need to be performed asynchronously so as not to impede a code release.

Outsourcing or leveraging managed services to perform fuzzing is recommended in the absence of internal subject matter expertise and staffing. Nonetheless, some tools in the fuzzing space include the commercial tools Peach Tech Peach Fuzzer and Synopsys Defensics. SaaS options have also started to emerge, such as Microsoft Security Risk Detection (formerly Project Springfield) and OSS-Fuzz (operated by Google for analysis of open-source projects). Finally, some organizations also leverage specialized debuggers to analyze binaries. Some vendor products in that space include Hex Rays' IDA Professional (aka "IDA Pro") and Rogue Wave Software's CodeDynamics.

Handle Output of Verification Tooling

Integrating verification tooling into a CI/CD build pipeline is one part of the equation to check for vulnerabilities systematically. However, the issue of how to respond when you do uncover a vulnerability must be addressed.

If vulnerabilities of high-enough severity are found during automated testing, you need to make a choice whether to:

- Accept the risk temporarily until a code fix or mitigation can be developed and deployed
- Fail the build and revert code if necessary

You must configure these choices programmatically as build rules within the CI/CD server for the given project leveraging integration with the AST tools so that the appropriate decision can be made based on scan results.

The choice in how to proceed with a build is largely a factor of one or more of the following:

- Development cadence or frequency of code sprints
- Compression of the production release schedule and whether there is enough "wiggle room" to slow down the CI/CD pipeline momentarily
- Frequency of production releases
- Security sensitivity of the organization

As an example, if a vulnerability of high-enough severity is detected during scanning, and code releases are performed multiples times a day, then it may be reasonable to assume that the vulnerability can be reported and fixed in a later release that same day. In that case, it may be acceptable to complete the build and release the application. Conversely, if code releases are weekly or monthly, then it may be undesirable to proceed with the build and deployment. If your organization has zero tolerance for vulnerabilities of any severity and/or no other mitigating

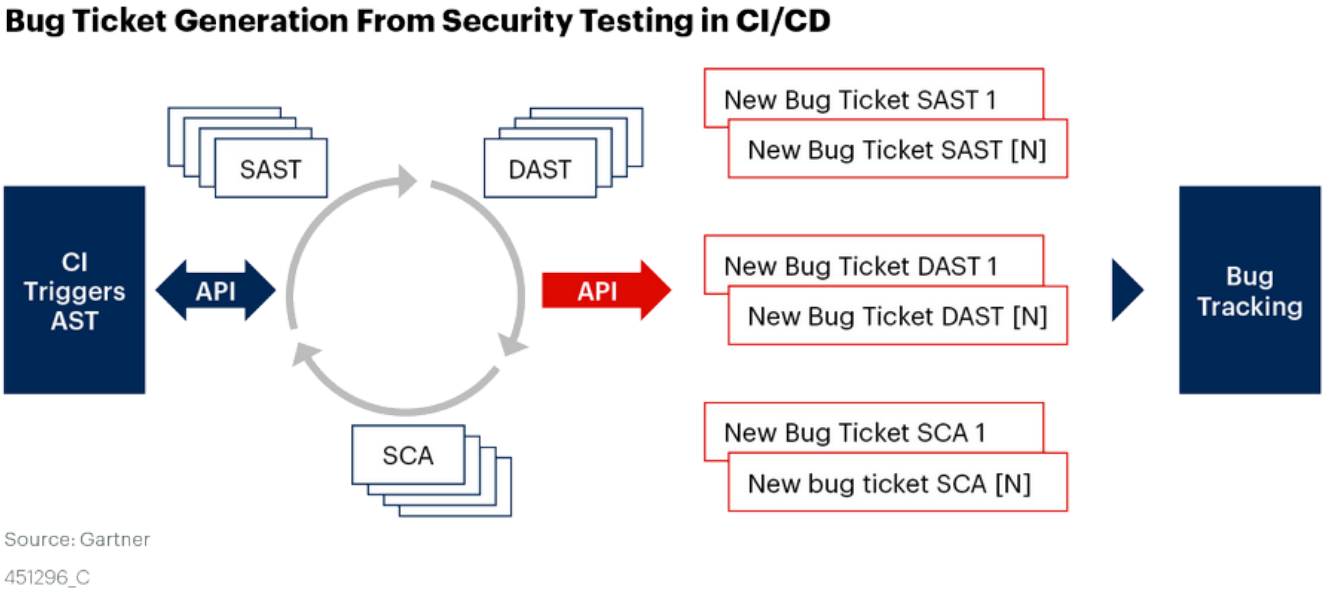
controls, such as a web application firewall, are in place, you will likely want to fail the build if any vulnerabilities are detected. In either case, vulnerability findings must be tracked accordingly.

> **The traditional security practice of generating and emailing vulnerability reports in document form will not scale in a DevSecOps pipeline.**

You should track any findings from verification tooling as bugs using the same SDLC systems and process for defect tracking and reporting. In most cases, they would be higher-severity bugs, particularly if the vulnerability itself is higher-severity. Verification tooling integration with bug tracking systems is fairly common and is a critical evaluation criterion when selecting an AST vendor. Integration with Jira Software is common, with Bugzilla and Microsoft Azure DevOps Services often in the mix as well.

Although vendors often support integration with bug tracking systems, the level of support is important to dig into. If a given AST tool can simply log bug tracking tickets for every vulnerability from a current scan, it is not an ideal method of integration. For most organizations, this will start to generate high volumes of tickets, especially with SAST. Figure 7 illustrates the concept and the resulting volume of tickets that the processes generate.

Figure 7. Bug Ticket Generation From Security Testing in CI/CD



Communication with defect tracking systems may be unidirectional as opposed to bidirectional. In the unidirectional model, there is typically no linkage to existing bug tickets. If a vulnerability is uncovered in scanning, a bug ticket will be created even if a ticket has been logged from a previous scan. Additionally, if any employee updates the bug ticket in the defect tracking system, there is no communication back to the AST tool that the issue has been potentially corrected and

needs to be verified. More vendors are starting to add support for bidirectional models, but it typically requires some level of ASOC capability to accomplish correctly. Specifically, the AST tool needs to track vulnerabilities from scans over time and update bug tickets as necessary, not just create new ones for every scan. The issue is compounded if you are running multiple AST tools from different vendors, where tool output (raw data and not generated reports) is in varying formats.

Use ASOC to Aggregate Verification Results

Use application security orchestration and correlation capabilities to prioritize high volumes of AST results and orchestrate AST tooling in the build pipeline. Feed aggregated results into application development life cycle management or bug tracking to drive remediation.

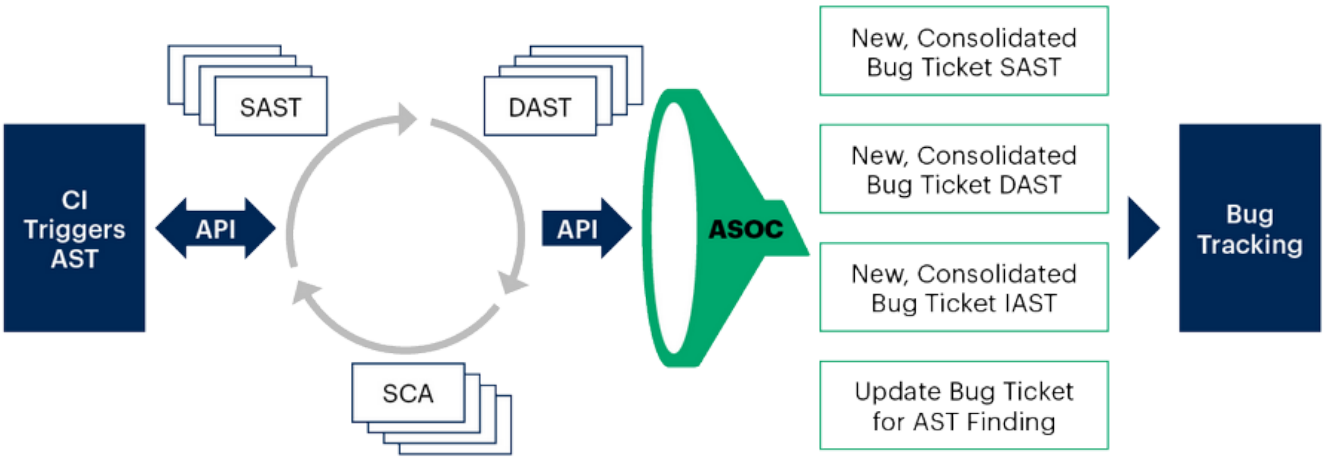
ASOC capabilities attempt to consolidate scan data across scans. Correlation attempts to address some of the issues that arise as a result of performing AST at scale, including:

- **Same AST tool output correlation:** Aggregate and deduplicate historical scan data using similar AST tool types (for example, SAST to SAST) that have been run against a given application.
- **Disparate AST tool output correlation:** Aggregate and deduplicate scan data across different AST tool types (for example, SAST to DAST) that have been run against a given application.
- **Centralized metrics:** Consolidate findings data into one central location and/or for export into bug tracking systems.
- **Vulnerability regrading:** Regrade findings programmatically based on criteria or metadata about a given application such as business criticality, data sensitivity, exposure and functionality. In some cases, this may be based on additional threat intelligence based on vendor implementation.

As similar AST scans and rescans are run against a given application, the output and historical data needs to be normalized to avoid duplicate entries (and duplicate bug tickets) for already reported issues. Figure 8 is a visual depiction of ASOC acting as a funnel and filter for AST results to control the volume of tickets that testing processes generate.

Figure 8. Use Application Vulnerability Correlation to Handle Security Testing Output in CI/CD

Use Application Vulnerability Correlation to Handle Security Testing Output in CI/CD



Source: Gartner
451296_C

This tends to be well-supported within the same category of AST technology (such as SAST with SAST or DAST with DAST), but merging data across AST types proves challenging. Not all vendors provide the same level of correlation. Integrated ASOC, often bundled with AST tool suites, may only support correlation of output for the vendor's own AST tools. Correlation may also be more basic, or it may not provide sufficient, granular control over scan output to aggregate results from high volumes of scans. Dedicated ASOC vendors use more advanced techniques to correlate data, and their products also provide integrations with disparate AST tools from multiple vendors. However, it results in more point solutions and higher complexity.

Ultimately, the decision to use integrated or dedicated ASOC will be a factor of whether a given vendor and its AST capabilities can meet all of your organization's needs so that you can standardize on their suite of tools alone. Otherwise, you may require a dedicated ASOC capability to aggregate AST output. Integrated ASOC typically comes bundled with any AST suite — on-premises or SaaS. Dedicated ASOC options are more limited, with Code Dx and Denim Group being the two commonly named vendors. There is some overlap with vulnerability prioritization technology (VPT) and security orchestration, automation and response (SOAR), where vendors like Kenna Security take a similar approach. But those products primarily focus on aggregation of network security scan and vulnerability assessment output.

Explore Remediation Options Beyond Just Code Fixes

Though a fair number of pieces of a DevSecOps pipeline can be automated, remediation of issues is not currently one of them. When issues are detected, you can certainly log bug tracking tickets to ensure a fix is ultimately put in place. This need not be a code fix and may entail other remediation measures like a virtual patch or architectural mitigation. With API enablement of externalized security and production runtime controls, it becomes possible to automatically create and implement a virtual patch. However, few organizations do this today largely due to concerns about availability impact and the small percentage of risk that a patch could introduce other issues in the production environment. As organizations fully embrace cloud-based infrastructure, infrastructure as code, test automation and other DevOps principles, it becomes

easier to deploy and roll back changes quickly if a mitigation does introduce problems. This is a rare case today though and typically seen only within technology organizations.

Realistically, you will need to track vulnerabilities over time, making use of ASOC capabilities and following traditional vulnerability management best practices. As issues are uncovered, you may opt to fail a build in the CI/CD build pipeline, but there will be cases where you will need to proceed with deployments despite vulnerabilities. The most obvious case would be severity of the issue uncovered. Truly critical vulnerabilities should certainly result in a failure. Medium-severity issues should result in a logged bug tracking ticket but not necessarily failing the build. The decision to fail a build could also be a combination of multiple lower-severity issues or a threshold that needs to be met. One medium-severity issue may not result in a failed build, but maybe 10 separate instances of medium-severity issues do result in a build failure. These decisions will depend on the application in question and such factors as data sensitivity, business criticality, exposure and your organization's risk tolerance.

Code fixes are one of the more expensive remediation options in terms of time and developer resources. If you are outsourcing development, this may be further amplified. Organizations fully embracing DevSecOps need to get comfortable with the concept of virtual patches and other mitigations over code fixes. Externalized security controls like WAF or runtime application self-protection (RASP) can be leveraged in most use cases (excluding some application platform as a service [aPaaS] and function platform as a service [fPaaS] use cases) to address design weaknesses and vulnerabilities in applications. Virtual patches may be short-term or long-term depending on the root cause and target application. It is not uncommon for an organization to be operating applications and systems in maintenance mode. These applications are still actively used, but no resources are invested in updating the codebase. Development teams may also have been long since disbanded, or development of the original codebase was completely outsourced with no access to source code. In these cases, code fixes are unrealistic, if not impossible.

Once a vulnerability has been identified in a given application through security testing, it is mostly a trivial process to create a corresponding rule in an externalized security control (if present) to protect against exploitation of it. Similarly, if the issue is infrastructure-related, and the asset is virtualized (such as with infrastructure as code), then patching and redeploying the affected virtual machine, container or container cluster is another remediation option.

Orchestrate Application Security Testing

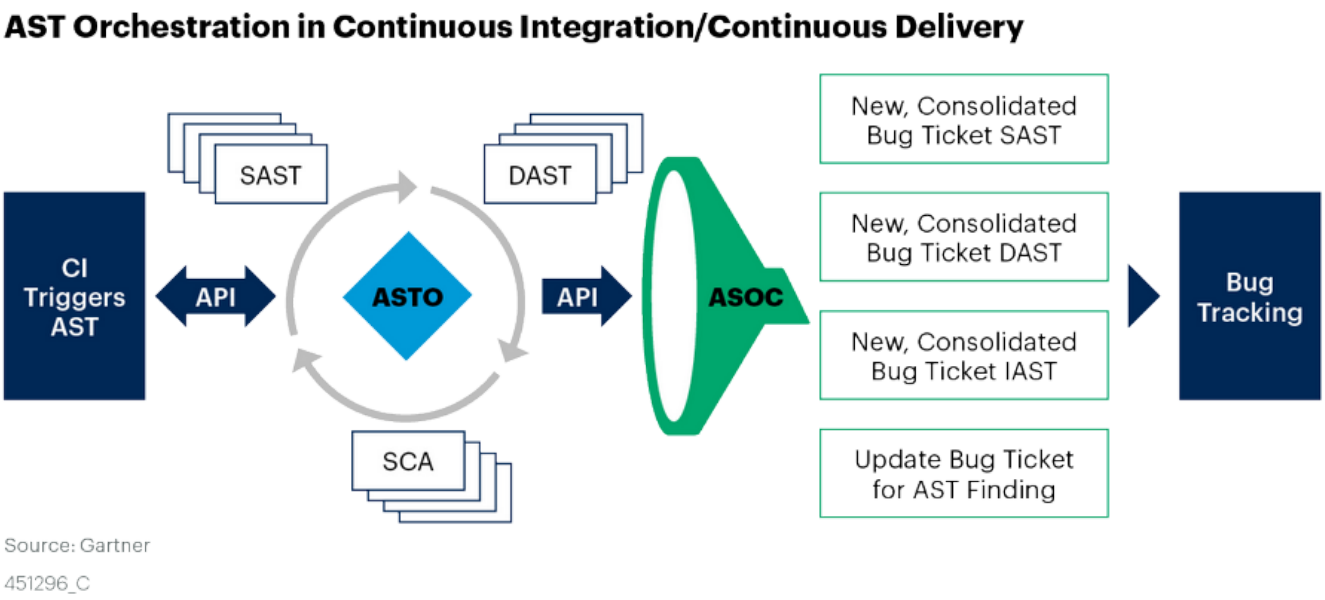
Like the CI/CD process itself, which relies on layers of orchestration to initiate process and control systems (largely controlled by a CI/CD server), coordinating the running of security testing tools across the different testing types also requires orchestration. Rarely, if ever, will you run an AST tool using the native GUI in a DevSecOps program, because it is a manual process. AST tools typically offer integration mechanisms to facilitate automation in the form of command line interfaces, native integration, or web APIs for more extensive customization. Native integrations for popular CI/CD servers may exist with more-established security vendors, but they may not provide enough control over scans and scan results (such as with incremental scanning). Web APIs among vendors, while often well-documented, are largely unstandardized. It is possible to

"stitch" together the various tooling through shell scripts or custom coding initiated as part of CI/CD process. However, this is, at the very least, resource-intensive, if not beyond the subject matter expertise of many application security practitioners. It is likely a better-suited task for development teams (or high-maturity security teams possessing development expertise).

Vendors are starting to address the gap to some degree, which Gartner categorizes as application security testing orchestration (ASTO). Similar to what we see in the ASOC space, integrated capabilities may be limited to just the vendor's specific tools, and dedicated capabilities are still an emerging market. Also, like the ASOC space, Code Dx and Denim Group are the main vendors seeking to address the gaps by enabling invocation of AST as part of vulnerability correlation and SDLC integration. ZeroNorth takes a slightly different approach, spinning up virtual test environments that mimic production and running AST tools against those ephemeral, nonproduction assets.

Figure 9 provides a conceptual view of how ASTO functions as a coordinator of the various AST activities when integrated in a CI/CD process.

Figure 9. AST Orchestration in Continuous Integration and Continuous Delivery



A handful of large organizations further along in adoption of DevSecOps have created their own security orchestration tooling. Some examples of this that are also open-source include Mozilla's Minion and Salesforce's Vulnreport (and the closed-source Salesforce Chimera for Salesforce independent software vendors). The OWASP Offensive Web Testing Framework is yet another open-source option. [4]

These tools are often designed more as platforms that combine an automation layer with self-service security testing where development and QA teams can initiate scans ad hoc, proactively and outside of the CI/CD process. This makes them fundamentally similar to many AST SaaS options, which have a predominant focus on security testing orchestration, output correlation and centralization of results in a web application.

Use Externalized Security Options

Externalized security mechanisms are those controls that exist outside of the codebase. These can be trusted components or libraries, as well as external-security protections and production runtime controls. In cases such as cryptography or secrets management, it is a monumental task — even for seasoned developers — to create sufficiently secure mechanisms, emphasizing the need to leverage externalized security.

> **Not all security can or should be built into the application code. Many attacks targeting applications don't exploit vulnerabilities within the codebase. Rather, attackers abuse functionality provided by the application or take advantage of misconfiguration.**

We also see attacks originate at the application layer that effectively target weaknesses in the operating environment, which gets into container security and securing infrastructure as code. Mitigation requires externalized architectural components to reduce risk of abuse.

Apply Application Protection Before Delivery

At this point, the application has been developed, built and verified. Any additional application protections need to be applied prior to delivery to production application servers. This includes:

- **Application wrapping:** Provide additional secure functions such as encrypting device-side application usage artifacts or enabling integration with enterprise mobility management.
- **Application shielding:** Provide code obfuscation, encrypt secrets and data (aka white-box cryptography), anti-tampering, anti-debugging, and some RASP-like protections.
- **Application authentication:** Provide seamless authentication of the application itself as opposed to just user authentication.
- **Code signing:** Enforced by most modern operating systems and web browsers as part of their security model to ensure authenticity and integrity of the compiled code and/or application package. Code signing and validation mechanisms are present in Apple Mac OS, iOS and Safari, Google Android and Chrome, Microsoft Windows, Mozilla Firefox and Linux/UNIX distributions. Artifact repositories and package managers like apt (or secure apt), Artifactory and Nexus use a form of public-key cryptography (typically GNU Privacy Guard [GnuPG] or some implementation of OpenPGP) to accomplish the same outcome.
- **Runtime application self-protection:** Instrument the application in runtime, providing detection and prevention of exploits and abuse.

The capabilities are described in detail in "Key Considerations When Building Web, Native or Hybrid Mobile Apps," "Building Authentication, Authorization and SSO Into API-Driven Apps" and "Protecting Web Applications and APIs From Exploits and Abuse." These controls exist mostly outside the scope of the codebase. Configuration and monitoring often also involve interaction with some external service, excluding the cases where the application doesn't or can't communicate to some back end (often where application shielding and obfuscation are required).

In cases where the control is added after the application is built, these should be included as part of your CI/CD process. Once you have performed development verification of an application or web API, you can apply these additional protections and deliver the packaged application to any number of hosts where it needs to live.

Some organizations may opt to perform development verification once again, but at this stage of delivery, it would be more of an assessment of the vendor's layered control as opposed to the original codebase. The presumption is that the layered control is implemented and configured correctly because you are applying it programmatically. If you desire to perform this type of second round of testing, the recommendation is to perform it out of band and outside the CI/CD process because it will likely require manual testing and verification.

Some potential pitfalls to be aware of include:

- Some controls, such as application authentication or RASP, may not be purely external to the application codebase. Depending on the vendor implementation, you may need to include additional libraries as dependencies within the codebase prior to compilation. You may opt to create additional custom checks within SAST to verify that the appropriate libraries are included in your application codebases.

- Code signing may require a specific type of code-signing certificate that is different from traditional web server certificates used in Secure Sockets Layer (SSL)/Transport Layer Security (TLS). Depending on the platform, this may need to come from a trustworthy root certificate authority such as DigiCert, or you may need to generate it using the platform provider's tools. You will typically not be able to use self-signed certificates if the application is intended for external distribution.

  In some cases, organizations use these types of certificates for development work, which itself often requires bypassing or suppressing security checks within the runtime environment. This has sometimes resulted in the self-signed certificate and suppressed certificate check making their way into production releases, weakening the security of the application. The best practice is to procure a valid-signing certificate and keep code-signing checks enabled within the given platform.

- Code signing requires manual development tool configuration. In cases of Java applications, Oracle's keytool and jarsigner may be used. [5,6] For mobile application development, each provider defines a process for how the certificate needs to be provisioned or configured within

the IDE as well as how it will be included as part of the application build. You must parse the relevant platform or technology documentation to satisfy the code-signing requirements.

- Code signing should be the last step you perform prior to deployment. If you are making use of an external control that is referenced within code or appended to the compiled application, this will modify the resulting binary, which impacts hashes and code signatures.

Deploy Network-Based Application Protection

The network-based application protections described in "Protecting Web Applications and APIs From Exploits and Abuse" can also be leveraged within a DevSecOps pipeline. This predominantly includes not only API gateways and web application firewalls, but also cloud access security broker for some use cases. CASB and respective vendor products are covered in detail in "Best Practices for Planning, Selecting, Deploying and Operating a CASB," "How to Secure Cloud Applications Using Cloud Access Security Brokers" and "Magic Quadrant for Cloud Access Security Brokers."

Also included are cloud web application and API protection (WAAP) vendors such as Akamai, F5 and Imperva that provide combined capabilities of content delivery network, distributed denial of service mitigation, bot mitigation and/or WAF. These are beneficial for protecting against a wider range of application attacks that occur outside of the codebase and beyond just exploits.

Web API enablement again is the main facilitator for integration. In the context of network-based application protections, they can be deployed or tuned in an automated way. Tuning may involve reconfiguration of the network control itself, or it may involve updates to custom rules to block certain senders, receivers and known exploit patterns. The rule updates may be a result of verification testing you've performed, or perhaps where a virtual patch can be put in place as a temporary or long-term corrective action. Vendors have also virtualized their products in some cases so that they can be delivered as a virtual machine, container or SaaS.

Although virtualization and API enablement theoretically create opportunities for automation, there are some significant pitfalls to be aware of. These include:

- Complexity in coordinating across the entire application, infrastructure and network stack makes this a bit more unrealistic. Orchestration tooling is still maturing and emerging in the ASTO and SOAR spaces. ASTO and SOAR capabilities facilitate integration and automation of tools in their respective domains, but they don't necessarily support integration with each other. Absent an adequate vendor solution, organizations would have to construct these connections themselves, which would be a massive undertaking. Even if all technology barriers were removed, organizations are often structured in such a way that controls are segregated and owned by different teams.

- WAF integration with AST is present with many vendors, but not fully automated. Making use of this type of function typically requires an extra step of vulnerability data export from the AST tool and then importation into the WAF for rule creation. This would require further

improvements to native integration or web APIs within the respective tools to make the process truly automated.

- Virtual patching can help alleviate the burden on development teams to generate code fixes. But implementation of virtual patches also needs to be verified so that they do not break application functionality. Few organizations make use of virtual patches in production for fear of impacting availability for users. Theoretically, with a fully automated DevSecOps pipeline, it becomes possible to rerun test cases to ensure no impact. However, many organizations may not fully trust tooling.

- Deploying high numbers of virtualized, network-based application protections such as virtual WAFs may create an unnecessary level of complexity. They may become difficult to manage at scale, making identification of security misconfigurations or troubleshooting overly challenging. It is still recommended to make use of cloud WAAP (or centralize on-premises network security controls if regulation dictates that cloud can't be used).

Address Container and Infrastructure Security

Organizations are employing containers in greater numbers to enable greater levels of scalability in cloud deployments and flexibility in choice of runtime environments of applications. An application need not run on a physical server or virtual machine; it can also run within a container, providing operating system abstraction and application compartmentalization. Containers have essentially democratized application delivery, freeing application and development teams from some of the hurdles of deploying applications in a traditional enterprise data center. The reality is still such that container ecosystems must live within and interact with traditional environments. This results in mixed or hybrid infrastructure environments, which inherently increases complexity.

For details on containers, container orchestration and cloud delivery of containers, see:

- "How to Prepare for Containers and Kubernetes"
- "Using Kubernetes to Orchestrate Container-Based Cloud and Microservices Applications"
- "Containers: 11 Threats and How to Control Them"
- "Decision Point for Securing Application Architecture"
- "Guidance Framework for Securing Kubernetes"

Running containers at scale requires some form of container orchestration engine, such as Docker Swarm, Kubernetes or Apache Mesos. The orchestration engine is focused on scheduling, deploying and monitoring containers and container clusters for operational purposes. This includes functions such as performance monitoring, container health checks, container instantiation and container termination. When packaged within a container as a service (CaaS) or container infrastructure as a service (ClaaS), some security features such as secrets management and identity and access management may be included as part of the offering.

There also are dedicated container security solutions to address some of the gaps that arise. Security functionality relevant to containers for build and runtime phases include:

- **Secrets management:** Facilitate sharing of certificates, cryptographic keys, passwords and other sensitive data with containers without hardcoding them into the container image.
- **Microsegmentation:** Define network boundaries at a "micro" level, providing monitoring and control of communication between containers and other traditional assets like bare-metal or virtual machines, on-premises and in the cloud. The topics of microsegmentation are covered in "Solution Comparison for Microsegmentation Products" and "Decision Point for Postmodern Security Zones."
- **Container configuration management and hardening:** Ensure container host and image build scripts are hardened, running only those services necessary to fulfill their function. Leverage continuous configuration automation (CCA) and infrastructure as code (IaC) concepts and technology.
- **Container vulnerability assessment and software composition analysis:** Analyze components of the container image prior to instantiation. Or analyze the instantiated container during runtime to ensure no known CVEs exist initially or over the life cycle of the running container.
- **Container profiling and threat detection:** Analyze behaviors such as communication paths and file accesses of a given container to identify anomalous behaviors, which might indicate an unhealthy container or pattern of attack.
- **Container firewalling:** Detect/block traffic patterns considered to be exploits at the container level using virtualized Layer 3-network-firewall- and Layer 7-web-application-firewall-type capabilities.

Availability of a given feature may vary across container types such as App Container (appc). Container security vendors are sometimes grouped into the larger category of cloud workload protection platforms (CWPPs). Adding to the confusion, there is overlap of cloud security and container security functionality, but disparity between vendor feature sets.

The concept of "securing a workload" translates to different things, depending on the vendor. Depending on their heritage, CWPP vendors focus on microsegmentation, endpoint protection, compliance auditing, monitoring and/or vulnerability assessment. Also, a vendor may support container instances or cloud IaaS instances exclusively, but not both. The container orchestration engine you employ can also further complicate what is supported by the vendor. Security functionality may be agentless. Or it may require installation of an agent on the container host, make use of a privileged container running in parallel to other containers, or require inclusion of security controls within container builds.

The "Market Guide for Cloud Workload Protection Platforms" can help demystify some of the vendor variation. Being that these are newer vendor controls and often focused on container

technology, web API enablement and integration with DevOps tooling is essentially a given. This makes automation within a DevSecOps pipeline feasible, but it will still require a great deal of expertise to integrate and operate across a landscape of diverse assets.

Ultimately, your container security strategy will require a mix of native container controls, native orchestration layer controls and supplementary container-focused vendor solutions.

Secure, Test and Audit Infrastructure as Code

Server build scripts, or infrastructure as code, should be tested like other code and put through a similar verification sequence within the CI/CD build pipeline. The topic, along with further analysis of CCA and IaC technology, is covered in detail in "Solution Path for Infrastructure Automation." Auditing build scripts helps ensure verified application code is delivered to security-hardened targets — whether bare metal, virtual machine or container and whether located on-premises or in the cloud. If the organization has fully embraced the concepts of continuous configuration automation and infrastructure as code, any asset identified as vulnerable can be quickly torn down and replaced with a hardened asset.

What constitutes a hardened asset will vary according to each organization, but it is a combination of internal policies, regulations and common server-hardening guidelines, including:

- Center for Internet Security Benchmarks
- Defense Information Systems Agency (DISA) Security Technical Implementation Guides (STIG)
- NIST Common Configuration Enumeration (CCE)

The process of verifying against these guidelines and integrating with CI/CD can look similar to development verification. However, auditing infrastructure as code and enforcing infrastructure security requires different sets of tooling. Server build scripts and running infrastructure can be audited using special-purpose testing frameworks such as Chef InSpec and Serverspec. You will need to create compliance testing scripts in the appropriate format to audit your infrastructure builds, though the scripts are more human-readable and make use of BDD-like syntax. Once you build the scripts, they can be executed as part of CI/CD.

Alternatively, commercial solutions are materializing to aid with test script generation or to provide prebuilt scripts that can serve as compliance profiles. Chef Compliance and Chef Automate are two such examples. Compliance profiles are built against the aforementioned hardening guidelines. Any customization of infrastructure builds or additional server-level controls unique to your environment requires customization of the test scripts. CCA tools like those from Chef can also be used to audit running production instances according to baselines and to reconfigure or redeploy assets accordingly.

CWPP vendors, and even some CSPM vendors, also provide compliance auditing of assets and, in some cases, can also harden them as they are running upon detection of a misconfiguration. This is usually facilitated via a host-based agent installed on every asset or via direct integration with

native cloud service provider web APIs. Vendors can vary in support of assets across internal data centers, private cloud and public cloud. Support for AWS is usually a given, with Azure being a close second. Google Cloud Platform and IBM Cloud are less supported. Additionally, support for aPaaS and fPaaS support may be nonexistent, have limited functionality or be part of product roadmaps. Some representative vendors include Alert Logic, CloudPassage, Dome9 and Symantec.

Continuously Monitor Applications With Production Security Monitoring

In a DevSecOps approach, deployed applications and systems will have gone through a pipeline similar to what has been described thus far. Organizations embracing this strategy should find that performing AST against production applications is less necessary because verification is integrated as part of CI/CD build process. Regular AST might still be part of your process (as out-of-band activity) to account for:

- Verification tooling improvements such as replacing or adding tools for DAST or fuzzing, or for regular scanning engine updates provided by the vendor
- Assessment of legacy applications or applications in maintenance mode where no code changes occur that would otherwise trigger verification as part of CI/CD
- Satisfaction of regulatory compliance or corporate policy that mandates regular, point-in-time audits

Security monitoring of applications postdeployment is more about catching potential misconfigurations, identifying unsanctioned implementations or uncovering new patterns of application attack. Security monitoring can also serve as a fail-safe to account for cases such as security tooling inefficiency or mixed modes of development and operations within the organization. The production security monitoring focus of DevSecOps gets into security configuration assessment, vulnerability assessment and SecOps.

Relevant research in these areas include:

- “Adapting Vulnerability Management to the Modern IT World of Containers and DevOps”
- “A Guidance Framework for Developing and Implementing Vulnerability Management”
- “How to Use Threat Intelligence for Security Monitoring and Incident Response”

Vulnerability assessment (VA) tools can be automated much like software composition analysis and application security testing. They can be executed as part of CI/CD, if so desired, to verify a bare-metal server, virtual machine or container has been configured securely and without known-vulnerable services. However, VA tools are more likely to be run continuously as part of regular production maintenance to identify issues that emerge over the life cycle of a given application and the systems that power it. Output destination of results is more likely to be VPT as opposed to ASOC, but some linkage back to defect tracking systems like Jira may still be desirable.

Vendors in the VA space also focus on API enablement to provide integration with disparate systems, and orchestration capabilities come in the form of SOAR as opposed to ASTO.

Externalized security mechanisms also play a role here. Network-based application protections (such as bot mitigation, RASP and WAF) and container security (such as container-level composition analysis, profiling and threat detection) provide benefit to production security monitoring beyond just mitigation of application attacks. Additionally, some of the CWPP products provide capability to continuously monitor assets postdeployment. The tools integrate with SIEM systems natively in some cases, or logs can be another data source for SIEM systems and process. If applications are undergoing attack, these mechanisms are a source of rich data to feed security monitoring. You can use this data to inform and initiate separate verification activities as appropriate, such as when attack patterns emerge or new vulnerabilities in acquired software are disclosed.

Ensure You Haven't Missed Something

Security incidents have highlighted the fact that not everything will go through established processes or build pipelines. At times, project initiatives may be fast-tracked, bypassing established business and SDLC processes. These are sometimes broadly categorized as unsanctioned IT.

Even with the converse of sanctioned IT initiatives, it can be difficult to maintain inventory of all your applications and APIs over time, especially when accounting for on-premises and cloud-hosted applications. Tooling is also difficult to implement, integrate and tune over time, which can result in security gaps. A DevSecOps program should employ some level of continuous discovery, leveraging other tooling as needed to ensure there's visibility of an organization's entire application portfolio.

Some examples of leaks that have impacted a number of large organizations include:

- Misconfigured or unpatched internet-facing, on-premises servers or cloud instances
- Sensitive data, code or secrets inadvertently or deliberately stored in public repositories such as Amazon S3, GitHub and Reddit
- Offshore assets that may not be as secured or hardened as onshore equivalents

This is often more of a side effect of inadequate or inconsistent log analytics and security monitoring for both on-premises and cloud assets. Some positive behavior patterns observed within organizations employing DevSecOps programs include:

- Leveraging secure web gateways and a CASB to expose sanctioned and unsanctioned SaaS use and highlight where sensitive data about applications or original source code may be leaving the organization
- Tapping into web-based network scanner repositories (such as Shodan or Censys) or SSL/TLS auditing and URL reconnaissance-type services (such as Qualys SSL Labs or ImmuniWeb) to

expose data about the organization's publicly accessible assets

- Analyzing data from customer experience management and marketing analytics platforms to expose commonly browsed URLs and usage patterns (Commonly accessed applications often become higher-value targets that may warrant additional rounds or deeper levels of security testing to focus on what's being actively accessed or attacked.)

Strengths

Practitioners should take note of the following strengths associated with DevSecOps:

- The growing trend of web API enablement of security and nonsecurity tooling makes it possible for organizations to integrate disparate development, security and operations processes. Tooling integration and automation produces greater consistency and enables detection of issues earlier in the SDLC. This can reduce likelihood of expensive (in terms of time and money) application code changes that resulted from traditional point-in-time testing in later stages of the SDLC.
- Application security vendors are embracing the concept of DevOps and ensuring their tools integrate within the SDLC ecosystem through command line interfaces, web APIs or native integration. This is true for verification tooling (such as SCA and AST) as well as externalized security (such as network-based application protections and container security). Dedicated ASOC and ASTO solutions also exist to help integrate and automate security testing. This helps alleviate some of the API integration work needed to coordinate testing as part of a CI/CD build pipeline.
- Tooling to mitigate the risk of known, vulnerable open-source software components is plentiful. A number of options exist, focused on either the software composition analysis aspect or a more complete open-source software governance solution. The vendors have also done a great deal of work to ensure integration with IDEs, VCSs, binary repositories and CI/CD systems where teams may inadvertently introduce vulnerable open-source components into an application codebase.
- Technology has evolved substantially to protect the application runtime environment. CCA and IaC make the traditional, manual process of building and configuring a server fully scriptable and auditable, whether the target is bare metal, virtual machine or container. This helps ensure consistent and hardened operating environments for applications. Cloud and container providers also offer some security capabilities natively for virtualized assets. Dedicated CWPP and container security vendors have also emerged to address aspects of application security in virtualized environments.
- Externalized security mechanisms offer significant improvements to application security beyond what is possible in code alone, alleviating some of the heavy reliance on development verification. Application protections can provide mitigations against other types of application

attacks like abuse, as opposed to just exploits. CWPP and container security solutions also help address protection and monitoring of cloud or container deployments.

Weaknesses

Practitioners should take note of the following weaknesses associated with DevSecOps:

- Security practitioners need to adapt to the new realities of DevOps, learning some development techniques themselves or working in tighter collaboration with development teams. Even still, integrations may be fragile where a change to an API of a given tool can cause a failure in the overall toolchain.
- Integration and automation of tooling for most organizations is still overly complex. This is especially true for organizations utilizing multiple point solutions or managing multiple CI/CD build pipelines due to different technology stacks or organization structure. GUIs aren't readily used in the world of DevOps, or they don't provide the right level of granularity to configure tooling. Native integration with SDLC systems is also often limited to the popular vendors. Integration of disparate security and nonsecurity tools requires an individual skilled in scripting and/or basic coding to communicate with and integrate web APIs.
- Although consistency is a strong suit of security tooling automation, accuracy and code coverage of AST scan engines continue to be an issue, resulting in either high false positives or false negatives. This can be exacerbated by modern application design, which can throw off scan engines that were designed to work with older technology. In the absence of human testing, barrages of different AST technologies and vendor products are sometimes used to compensate for the inherent weaknesses in tooling, but this also generates higher volumes of vulnerability data to parse.
- Business logic testing continues to be a pitfall for most AST tools. This type of testing often requires well-defined, highly custom test cases if an AST tool even supports it. Special-purpose tools designed for business logic testing can also be difficult to configure and use, which will likely introduce manual work into the DevSecOps process. Business logic issues are best prevented as part of secure design activities or mitigated with network-based application protections (such as with bot mitigation).
- ASTO and ASOC help integrate and automate the AST process. VPT and SOAR help do the same for the VA process. However, there is currently no unified tooling or integrations between the two to facilitate full-security orchestration, which many organizations desire. The respective vendors that offer these capabilities are also still relatively young and maturing.
- The rapid growth of DevOps tooling and practices has resulted in some significant security gaps for organizations racing to adopt the wide range of related technology. This has resulted in a number of negative outcomes for organizations, such as exposed web APIs, vulnerable OSS components embedded in code and secrets stored in unsecure locations. In some cases, these have been a contributing factor to incidents or breaches. Security practices and vendor

tooling have struggled to keep pace at times. Many organizations are still early in the learning curve on DevOps concepts or are scrambling to establish a DevSecOps toolchain to deal with the technology sprawl.

Guidance

Establish a Basic DevSecOps Pipeline When Just Starting

Start with a baseline of simple tooling that satisfies the various capabilities, possibly leveraging OSS options initially if budget is restricted. Focus on security tooling that you can easily integrate within your existing SDLC ecosystem, minimizing impact to application teams. Consider initial stages of the pipeline a working prototype as you work through the learning curve and adapt to the new processes.

For DevOps (and agile), it is recommended that, when possible, applications should be able to automatically be built as a result of code being committed to the VCS, and the application should be automatically deployed. With that basic pipeline in place, you can add statics analysis, SAST, DAST and test automation (unit, functional, integration and performance)

To prove out the prototype, work with a small number of application teams (and subset of the applications they own) — or even one to start — that are flexible with changing process or tooling. Once you've established a working DevSecOps toolchain, the next decision point should be which product owners and applications and associated teams to involve, work with and onboard. Identify new projects and forward-thinking development and operations staff to grow your user base and tweak the DevSecOps toolchain.

Again, selecting newer projects as opposed to legacy applications will guarantee a higher level of success. Attempting to onboard too many teams and applications will overload your teams, the DevSecOps tooling and/or the computing assets that enable the toolchain. Once the DevSecOps toolchain and processes are established, prioritize onboarding of teams and applications that represent the highest potential impact to the organization if a security incident were to occur.

Standardize Your SDLC Processes and Tooling First

Standardize SDLC processes and tooling critical to DevOps before attempting to establish a DevSecOps pipeline. Critical elements include the following processes and systems:

- Application development life cycle management
- Defect tracking
- Version control for source code and binary artifacts
- Continuous integration and continuous delivery
- Test automation
- Continuous configuration automation

In the absence of standardization in any of these areas, integrating application security will be overly complex and may result in gaps.

Select Security Tooling That Integrates With SDLC Systems

Security tooling and processes need to be integrated into the SDLC. Favor vendors that readily integrate with DevOps systems and processes. If possible, avoid too many point solutions. Select vendors that can offer a wider range of security capabilities across the DevSecOps toolchain. This will reduce the amount of integration work as well as the learning curves of disparate tools and their associated UI/UX. Running the tools and handling output should be as transparent as possible to teams, triggered as part of standard SDLC activities and the CI/CD process. Note that, in some circumstances, you may have no choice but to use different tools in different pipelines due to technology stack, language or domain.

Tooling also does not need to be a financially expensive endeavor. OSS options exist in most categories, which can be used for basic maturity approaches or until budget can be allotted. They are also useful for learning the processes and techniques prior to committing to a large purchase. It is certainly possible to mix and match OSS and commercial off-the-shelf (COTS) products to build your DevSecOps toolchain.

Resign Yourself to Semiautomation With Secure Design Activities

Avoid the use of spreadsheets for tracking security requirements if possible. Ideally, your security requirements should be implemented within an ADLM suite as nonfunctional requirements. This helps emphasize that development teams should satisfy security requirements as they build applications. However, you still need to lean on security verification tooling to catch issues and ensure requirements were followed. Invest some effort in providing secure coding practices for development teams because these will be the prescriptive guidance in how they can satisfy security requirements. Host them in ADLM or knowledge management systems to ensure the guidance is readily available.

Reserve threat models for high-sensitivity applications as an out-of-band process, or emphasize incremental threat models because it is difficult to create and maintain models over time as code changes. Modern application design and use of APIs can result in “spaghetti” diagrams that become unreadable. Threat modeling may also be more beneficial within application security training and awareness efforts, as opposed to attempting to fit it into a DevSecOps program.

Avoid Excessive Focus on Secure Design

The DevSecOps approach consists of multiple focus areas beyond just secure design and development verification. Application security practices also include externalized security and production security monitoring, which is where infrastructure, operations and network teams often share responsibility.

Collaborate with these teams in your organization as well, and don’t limit interactions to just development or QA. Adopt new security tooling as appropriate to build up the other aspects of the DevSecOps toolchain. The implementation of externalized security and production security

monitoring alleviates some of the burden on development teams and latency in development-time application security, especially for manual activities like security requirements gathering or threat modeling. You also shouldn't mitigate all modern application attacks with code-level mitigations alone.

The Details

DevOps Practices and Technology

DevOps has a number of implied meanings and impacts to application security. The concepts are covered in detail in other Gartner research, such as:

- "The Keys to DevOps Success" (webinar)
- "A Guidance Framework for Continuous Integration: The Continuous Delivery 'Heartbeat'"
- "Solution Path for Infrastructure Automation"
- "Extend Agile With DevOps for Continuous Delivery"
- "Solution Path for Achieving Continuous Delivery With Agile and DevOps"
- "How to Automate Your Network Using DevOps Practices and Infrastructure as Code"
- "Choose the Right Metrics to Drive Your Agile, DevOps and Continuous Delivery Initiatives"

Developers and operations staff are likely working with many of the following:

- **Application development life cycle management** for tracking life cycle of code and the activities around it, such as functional and nonfunctional testing, defect tracking, and project scheduling. More recently, and to support agile and DevOps trends, Gartner has relabeled this category of technology as "enterprise agile planning (EAP) tools." See "Magic Quadrant for Enterprise Agile Planning Tools."
- **Version control systems** for storing and organizing uncompiled source code. The VCS is sometimes also referred to as a code repository or source control management (SCM) system. Developers push or pull code from it, as well as commit code to trigger build and CI processes facilitated through hooks (or webhooks in the case of HTTP) with CI/CD servers.
- **Binary repository managers** for storing and organizing any type of binary artifact, including compiled applications or libraries, server builds, container images, and other objects. These may be artifacts created as output from builds, or they may be artifacts needed for builds to complete.
- **A structured continuous integration/continuous delivery build pipeline,** with a focus on orchestration and automation with other SDLC systems. See "A Guidance Framework for Continuous Integration: The Continuous Delivery 'Heartbeat'" and "How to

Architect Continuous Delivery Pipelines for Cloud-Native Applications." CI/CD facilitates the compilation and delivery of software and is a combination of:

  - Continuous integration centered on the build process, pulling in necessary components and compiling the application
  - Continuous delivery centered on distribution of the application to infrastructure (web, application and/or database servers, etc.) to ensure that it is fully runnable and accessible
  - Continuous deployment (sometimes mixed with continuous delivery) as an extension of continuous delivery where delivery to production is fully automated without a manual gate prior to deployment

- **Test automation,** which provides the ability to execute user interface (UI), user experience (UX) and business logic tests programmatically as part of CI. See "Solution Path for Testing Software Applications."
- **Continuous configuration automation and infrastructure as code** for automating server operations and building hardened server images for deployment to bare metal, virtual machines or containers (or container clusters). See "Solution Path for Infrastructure Automation."
- **Abstraction and virtualization of workloads,** where applications or pieces of them run on physical or virtualized assets on-premises or in the cloud. Related cloud trends include greater adoption of application platform as a service or function platform as a service, as opposed to the more traditional infrastructure-as-a-service model.

Practitioners sometimes confuse agile development methodologies and DevOps practices, but they are distinctly different. Agile development methodologies have more to do with how development teams are structured and how developers create code, as opposed to how code is compiled and released in DevOps. Agile methodologies include Scrum and Extreme Programming, and the result is essentially that iterative code changes at a faster cadence. This further necessitates automation and DevOps practices, including the integrated and automated CI/CD build pipeline, in order to be successful. Technically, DevOps practices and tooling can exist without agile development methodologies, but the reverse situation is less true.

Open-Source Software Options in a DevSecOps Toolchain

Though OSS can equate to "free" in some cases, there are still hidden costs of employing OSS in enterprise environments. Tools still require resources to install, configure and operate. In some cases, the tools may be harder to configure and maintain than their COTS counterparts. They may also require more custom code to integrate and work as expected.

There is a great deal of commoditization of development languages, frameworks and libraries. Many are OSS, creating a low barrier to entry. In some cases, the tools are "freemium" (limited functionality in free model, but pay for advanced functions) but mostly lower-cost.

Some examples of OSS options for SDLC systems can be found in Table 1.

Table 1: Example Open-Source Options for SDLC Systems

SDLC System ↓	Open-Source Software Options ↓
Application Development Life Cycle Management	GitLab Community Edition (CE) and GitHub
Continuous Integration andContinuous Delivery	Jenkins Community
Defect Tracking	Bugzilla
Continuous Configuration Automation	Ansible, Chef and Puppet
Version Control System	■ On-premises: Apache Subversion (SVN) and git ■ Cloud: GitHub and GitLab
Binary Repository	■ On-premises: JFrog Artifactory and Nexus community ■ Cloud: GitHub and GitLab
Test Automation	Cucumber, Puppeteer, Selenium and PhantomJS

Source: Gartner (June 2020)

Conversely, application security tooling is less commoditized. Tools exist in many of the categories, but support (if existent) may be limited and security practitioner communities may be nonexistent. The OWASP community is one of the largest and most well-known. Some tools are unmaintained, alpha/beta projects or feature-limited. This is getting better in some cases, especially with freemium options. Some of the options are listed in Table 2.

Table 2: Example Open-Source Options for Application Security Tools

Application Security Tool Category ↓	Application Security Focus ↓	Open-Source Software Options ↓
Security Requirements	Secure Design	OWASP SKF

Application Security Tool Category	Application Security Focus	Open-Source Software Options
Threat Modeling	Secure Design	■ Microsoft Threat Modeling Tool ■ Mozilla SeaSponge ■ OWASP Threat Dragon
Software Composition Analysis	Development Verification	■ On-premises: OWASP Dependency Check, Retire.js and VersionEye ■ Cloud: GitLab Gemnasium, nsp Live and Snyk
Static Application Security Testing	Development Verification	■ On-premises: OWASP Find Security Bugs, Salesforce Providence and SonarSource SonarQube ■ Cloud: SonarSource SonarCloud and Software Assurance Marketplace (SWAMP)
Dynamic Application Security Testing	Development Verification	Arachni, OWASP ZAP and w3af
Interactive Application Security Testing	Development Verification	Hdiv Community
Binary orProtocol Fuzzing	Development Verification	American fuzzy lop, libFuzzer, Radamsa and Sulley (and the forked boofuzz)
Application Vulnerability Correlation	Development Verification	■ Code Dx Standard Edition ■ Denim Group ThreadFix Community Edition
Application Security Testing Orchestration	Development Verification	BDD-Security, Gauntlt and Mozilla Minion
Code Obfuscation	Externalized Security	ProGuard

Application Security Tool Category	Application Security Focus	Open-Source Software Options
Continuous Configuration Automation Auditing	Externalized Security	Chef InSpec, Serverspec
Runtime ApplicationSelf-Protection	Externalized Security	Hdiv Community
Web Application Firewall	Externalized Security	ModSecurity

Source: Gartner (June 2020)

Gartner Welcomes Your Feedback

We strive to continuously improve the quality and relevance of our research. If you would like to provide feedback on this research, please visit Gartner GTP Paper Feedback to fill out a short survey. Your valuable input will help us deliver better content and service in the future.

Evidence

[1] "How to Fit Threat Modeling Into Agile Development: Slice It Up," Irene Michlin, DevSecCon London 2018.

[2] ThreatModel SDK, GitHub.

[3] OWASP Top 10 2017: The Ten Most Critical Web Application Security Risks, Open Web Application Security Project.

[4] Offensive Web Testing Framework, OWASP.

[5] Keytool — Key and Certificate Management Tool, Oracle Java SE Documentation.

[6] Jarsigner — JAR Signing and Verification Tool, Oracle Java SE Documentation.

[7] Social Media Analytics (SMA) Methodology: Gartner conducts social listening analysis leveraging third-party data tools to complement or supplement the other fact bases presented in this research. Due to its qualitative and organic nature, the results should not be used separately from the rest of this research. No conclusions should be drawn from this data alone. Social media

data in reference is from 1 January 2017 through 25 December 2019 in all geographies (except China) and recognized languages.

[8] Additional research contributions were provided by Ritesh Srivastava and Ayush Saxena from the Gartner Social Media Analytics team.

Document Revision History

Structuring Application Security Practices and Tools to Support DevOps and DevSecOps - 28 November 2017

Recommended by the Authors

A Guidance Framework for Developing and Implementing Vulnerability Management

A Guidance Framework for Establishing and Maturing an Application Security Program

How to Deploy and Perform Application Security Testing

Protecting Web Applications and APIs From Exploits and Abuse

Containers: 11 Threats and How to Control Them

Best Practices for Securing Continuous Delivery Systems and Artifacts

Solution Path for Forming an API Security Strategy

Recommended For You

How to Deploy and Perform Application Security Testing

Decision Point for Postmodern Security Zones

Best Practices for Choosing Network Security Controls Between EFW, SWG and CASB

5 Core Security Patterns to Protect Against Highly Evasive Attacks

Decision Point for Deploying WAFs for Application Protection

or influence from any third party. For further information, see "Guiding Principles on Independence and Objectivity."