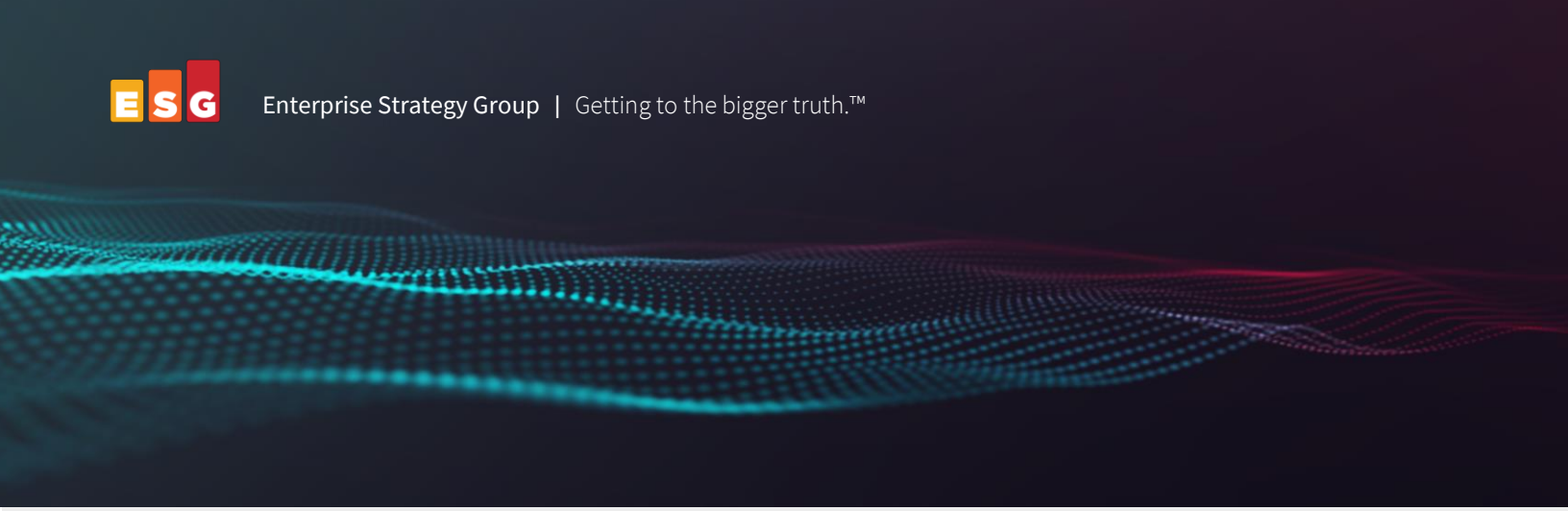


ESG WHITE PAPER

The Application Security Testing Imperative

A Pragmatic Approach to Reducing Production Vulnerabilities with an Integrated Approach

By Doug Cahill, Vice President and Group Director

November 2020

This ESG White Paper was commissioned by Checkmarx and is distributed under license from ESG.

Contents

Executive Summary

The central role of applications is such that they represent business-critical infrastructure and thus a prominent feature of every company's attack surface. For organizations that build and deploy internally developed applications, the accelerated rate of innovation has challenged cybersecurity objectives, creating an application security testing (AST) imperative.

An integrated approach to application security testing supports seemingly mutually exclusive objectives: agile software development and risk mitigation.

Fortunately, foundational best practices employed to protect production runtime environments apply to the implementation of an AST program, principally, reducing attack surface and remediating exploitable vulnerabilities. An integrated approach to application security testing supports seemingly mutually exclusive objectives: agile software development and risk mitigation.

This paper provides practical guidance for CISOs, CIOs, and DevOps leaders for designing an effective application security program to secure modern application development via an integrated approach. The paper also aims to equip application security practitioners with research data to support building the business case for AST investments.

Competing Objectives of the Application Economy

The transformation to the digital enterprise has resulted in business applications serving critical front, middle, and back office functions. In addition to underpinning business operations, applications are often a primary point of competitive differentiation, creating pressure on internal application development teams to accelerate the rate at which new code is developed and new builds are deployed to production. In fact, many organizations with a cloud-first orientation push to production multiple times per day.

While the application stack comprises a heterogenous mix of technologies, cloud-native applications based on a microservices architecture have accelerated rapid application development initiatives, fueling innovation. Modern methodologies are also being brought to bear against the need for speed. Agile software development and the continuous integration and continuous delivery (CI/CD) processes of DevOps have enabled an iterative approach to application definition and delivery.

In contrast to managing risk in the context of the more deliberative approach of waterfall, in which phases and gates serve as checkpoints, the constant rate of change in today's software development (SDLC) paradigm often creates a security readiness gap.

The Implications of Deadline-driven Decisions

But just how commonly do the pressures on application development project teams to move fast result in security implications? In research conducted by ESG, 79% of respondents shared their organizations regularly or occasionally push code to production with known organic vulnerabilities.[1]

These teams are clearly feeling pressure to knowingly deploy vulnerable code, with 54% citing the need to meet a critical deadline with a plan to remediate the issue in a later release as the reason to deploy with known issues. The timing of

[1] Source: ESG Master Survey Results, *Modern Application Development Security*, to be published. All ESG research references and charts in this white paper have been taken from the master survey results, unless otherwise noted.

discovery was also cited as a factor for deploying faulty code, with 45% noting that the issues were detected too late in their cycle for them to be resolved (see Figure 1).

Figure 1. Reasons for Pushing Code to Production with Known Organic Vulnerabilities

Source: Enterprise Strategy Group

As noted by 49% of the respondents, the lack of an exploit and thus a low severity rating may be another reason to deploy. But what happens if a low severity vulnerability becomes a high severity issue when an exploit is known to be in the wild? According to the same research, 81% report production applications have been exploited in the last 12 months.

Patching may require downtime, impacting service level agreements (SLAs) and further delaying addressing the vulnerability. Some organizations may look to virtual patching solutions that detect and block exploit behavior while they plan the deployment of a new build that doesn't impact SLAs. But as is the case with web application firewalls (WAFs), such controls are often deployed in passive mode, creating a false sense of security.

The vulnerability discovery and patching cycle is a perennial cybersecurity challenge. As such, vulnerability discovery and remediation should not be relegated to, or exclusive to, runtime environments. For organizations that internally develop applications, vulnerability management needs to be viewed through the lens of reducing the attack surface, which starts in the development stage. The componentry of an application represents its attack surface, including the use of third-party and open source software (OSS) libraries.

The Open Source Software Dilemma

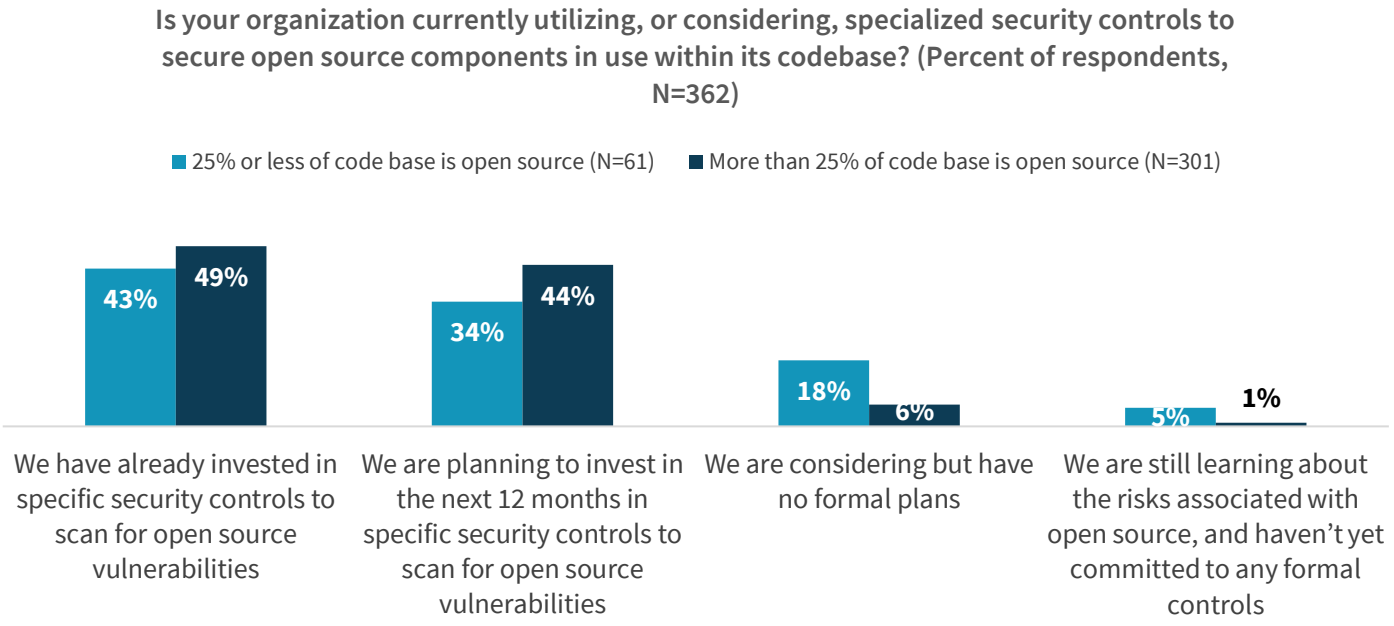
Time pressures often lead to the use of off-the-shelf third-party components, including open source software (OSS). A notable 80% of research participants report significant use of OSS, with 43% stating more than half of their code base is composed of open source.

Vulnerabilities in a source code branch transcend those inadvertently introduced organically to those that may be present in OSS. Therein lies the dilemma: Open source software allows project teams to leverage reusable code to expedite the deployment of a new build but doing so creates the risk of expanding production attack surface with vulnerable source code. How are application development project teams assuring their use of OSS is secure?

Slightly less than half (49%) of those who report a quarter or more of their code base is based on open source software have invested in controls to scan that code for vulnerabilities. While this is inadequate, it is somewhat encouraging that

another 44% of those who report 25% or more of their code base is OSS plan to invest in such controls over the next 12 months (see Figure 2).

Figure 2. The Use of Controls to Scan Open Source Software for Vulnerabilities



Source: Enterprise Strategy Group

In addition to risk associated with licensing that governs the use of OSS libraries, which may not have been taken into consideration by the developer who chose to use it, it is worth noting there is a potential for a long tail of technical debt incurred by using open source software that is no longer being updated by the community.

Demystifying DevSecOps

DevSecOps is often cited as the means by which security objectives can keep pace with the rate at which software is developed, integrated, and delivered. But the term is frequently misused and misunderstood, marginalizing the intent of the approach. As such, a clear definition is in order.

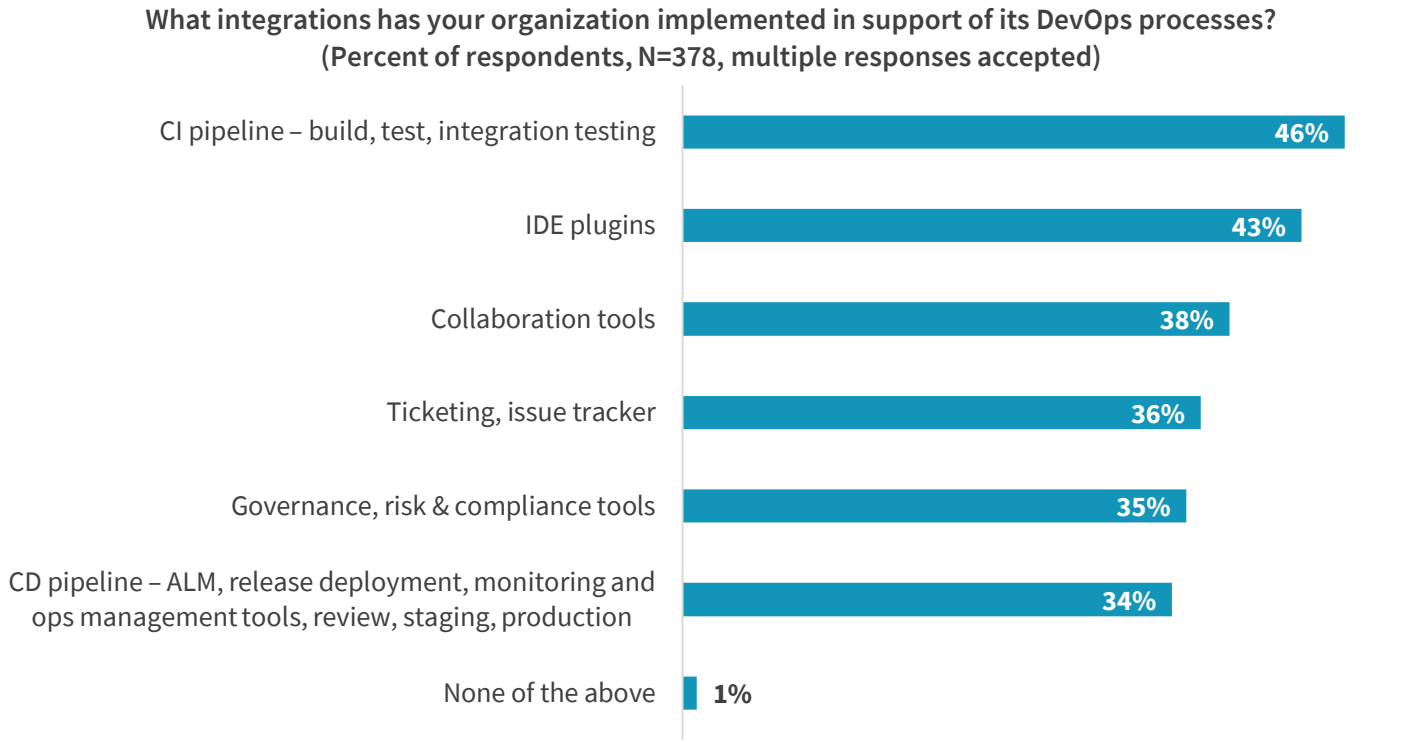
DevSecOps is the automation of cybersecurity processes and controls via integration with the continuous integration and continuous delivery (CI/CD) toolchain that orchestrates the application lifecycle.

For the purposes of this paper, DevSecOps is the automation of cybersecurity processes and controls via integration with the continuous integration and continuous delivery (CI/CD) toolchain that orchestrates the application lifecycle. The application lifecycle starts with the definition stage and spans into the development, integration, delivery, and production stages. As such, DevSecOps starts with the definition stage in which security user stories are authored for implementation in a near-term sprint. Some user stories, such as code scanning, should then be treated as a best practice for all sprints.

Just as was the case with DevOps, a successful approach to DevSecOps is predicated on culture, one that treats security as a priority and a shared responsibility among project team members. And because developers are often deciding on the composition of modern applications inclusive of organic code, APIs, open source software, and third-party libraries, they are a key stakeholder in the success of a secure DevOps program.

While a majority of research participants, 56%, shared their organization is utilizing an integrated set of security controls throughout their DevOps process, less than half, 46%, have done so in their continuous integration (CI) pipeline (i.e., build, test, and integration), and even less in their developer's interactive development environments (IDEs), 43% (see Figure 3).

Figure 3. Application Security DevOps Integrations



Source: Enterprise Strategy Group

Notably, two-thirds of those who have integrated security in the pre-deployment stages report their company has a more effective application security program. But what exactly are those pre-deployment stages?

The Stages of a Shift-left Approach

DevSecOps is often associated with the notion of shifting security practices left into the pre-deployment stages of the application lifecycle. The furthest left stages are actually before developers start to code.

The Planning Stage

- Definition: The authoring of application security testing user stories as part of the agile software development process is the first step for the implementation of a Secure DevOps program to assure the secure development of code.
- Assignment: The prioritization of these user stories for each sprint is essential to security being codified.
- Awareness: Training the development team on secure coding practices, which should include IDE-integrated reinforcement of such best practices, makes every member of the development team vigilant.

The Development Stage

Some of the foundational user stories that developers will be charged with implementing for all sprints and code branches include:

- Software composition analysis (SCA) to inventory the elements of a code branch.
- Static analysis to evaluate source code within the developers' IDE.

The Integration Stage

Testing new builds as part of the integration stage should include the use of application security testing controls, which will evaluate runtime context for possible vulnerabilities. DevSecOps use cases in this stage should also include the automated introduction of cybersecurity controls to protect production environments such as cloud workload protection platforms (CWPP).

All Are Equally Important

As evidenced by the aforementioned research conducted by ESG, time-to-market pressures too often result in code with known vulnerabilities getting deployed to production. As such, these pre-deployment stages are not mutually exclusive.

After all, the more vulnerabilities that are discovered and corrected in the development stage, the less that are then found during the build process, and those that are vetted in the integration stage do not make it to production.

After all, the more vulnerabilities that are discovered and corrected in the development stage, the less that are then found during the build process, and those that are vetted in the integration stage do not make it to production. Unfortunately, per ESG's research, there is work to be done, with less than half of the participating organizations integrating security in their CI pipeline and less still as an IDE plug-in.

Workflow Integration

The rinse and repeat, continuous nature of such automated secure DevOps practices requires a feedback loop. The project management, ticketing, and issue tracking tools employed by project teams orchestrate the workflow across all stages of the application lifecycle, including these pre-deployment stages. As such, ticketing and issue tracking systems serve a central role in the feedback loop.

As issues are detected from scanning source code management (SCM) repositories and at build time, tickets are automatically opened and assigned to the appropriate developer to streamline remediation. Such workflow integrations also enable measuring key performance indicators (KPIs), such as the reduction of organic vulnerabilities over time, so that application security testing programs can be measured and improved based on metrics such as vulnerabilities detected by stage, mean time to resolution, and more.

Critical Success Factors for an Effective AST Program

Reducing an organization's attack surface is a core tenet of any cybersecurity program, a best practice challenged by multiple factors: the heterogenous nature of hybrid, multi-cloud environments, the velocity at which applications are developed and delivered, and the fact that a remote workforce needs access to seemingly any application from any device at any time.

While reducing infrastructure attack surface is centered on IP-based firewall rule sets, vulnerability management and patching, east-west segmentation, the principles of least privilege access (LPA), and more, reducing the attack surface of internally developed applications requires identifying and eliminating software vulnerabilities as a core tenet of an

organization's software development practices. The implementation of an application security testing (AST) program that instills this approach as an imperative should be based on a set of critical success factors, including:

- Policies. Formally documented best practices and organizational standards must be created, communicated, and reinforced within the development community as best practices.
- Automation. Integration of AST into the software development and DevOps tool sets is critical to support time-to-delivery objectives.
- Security training. Just-in-time security training for developers should be integrated within the development toolset.
- Metrics. KPIs and reporting allow development managers to track the continuous improvement of developer skills and where to fine tune processes, as well as to keep cybersecurity leaders and stakeholders informed.
- Issue tracking. Tracking security issues during code development is essential to gain visibility, establish metrics, and provide a feedback loop to the individual developer and team at large.

Requirements for an Integrated Approach to AST

Like many cybersecurity product categories, application security testing (AST) has been populated by a range of point tools focused on addressing a specific requirement. In fact, 43% of organizations are using 11-20 separate application security tools, with 84% noting the proliferation of these tools is problematic. Indeed, the use of a disparate set of controls in multiple cybersecurity domains has increased both cost and complexity, leading many organizations to consolidate by purchasing more controls from fewer vendors. This dynamic holds true for application security testing, per the roughly one-third of respondents, 34%, who cited an intent to focus their application security investments on the consolidation of tools to simplify their AST processes.

To support this market requirement, some application security testing vendors offer multiple, integrated products to address the broad range of security vulnerabilities and testing approaches. Organizations interested in an integrated approach to AST vis-à-vis a one-stop-shop vendor should consider the following requirements.

Code Analysis

Inspecting Source Code with Static Application Security Testing (SAST)

Static application security testing (SAST) represents a foundational element of an AST program. Employed iteratively during the coding stage, SAST evaluates static, uncompiled source code for potential issues. This automated approach to white-box testing is highly performant relative to manual code reviews. Key facets of a complete SAST solution include:

- Interdependency discovery. SAST tools will learn a code base, including the interdependencies between different elements of the base as well as data flows, a notable capability aimed to preventing data loss.
- Extensibility. Built-in and custom rules will identify issues such as input validation errors, path traversals, race conditions, injections, and more.
- Scalability. SAST solutions must be highly performant with respect to providing expedited results from scans to support rapid application development objectives.

- High fidelity. Since false positives will slow the development process and frustrate developers, ongoing out-of-the-box accuracy improvements and permissible query preset tuning is critical with SAST technologies.
- Streamlined onboarding. The successful onboarding into the development ecosystem is essential, making capabilities such as application risk analysis, scan engine tuning, and incremental scanning capabilities important.

Spotlight: The Case for Interactive Application Security Testing (IAST)

Some vulnerabilities do not reveal themselves as exploitable until evaluated in a runtime state. As such, interactive analysis also needs to be in scope for identifying organic software vulnerabilities. IAST is often considered as *grey box testing,* which is indicative of having similar attributes of both static and dynamic testing approaches.

Its coverage is as wide as the testing, since IAST is always active, waiting to analyze the application's activity during functional testing.

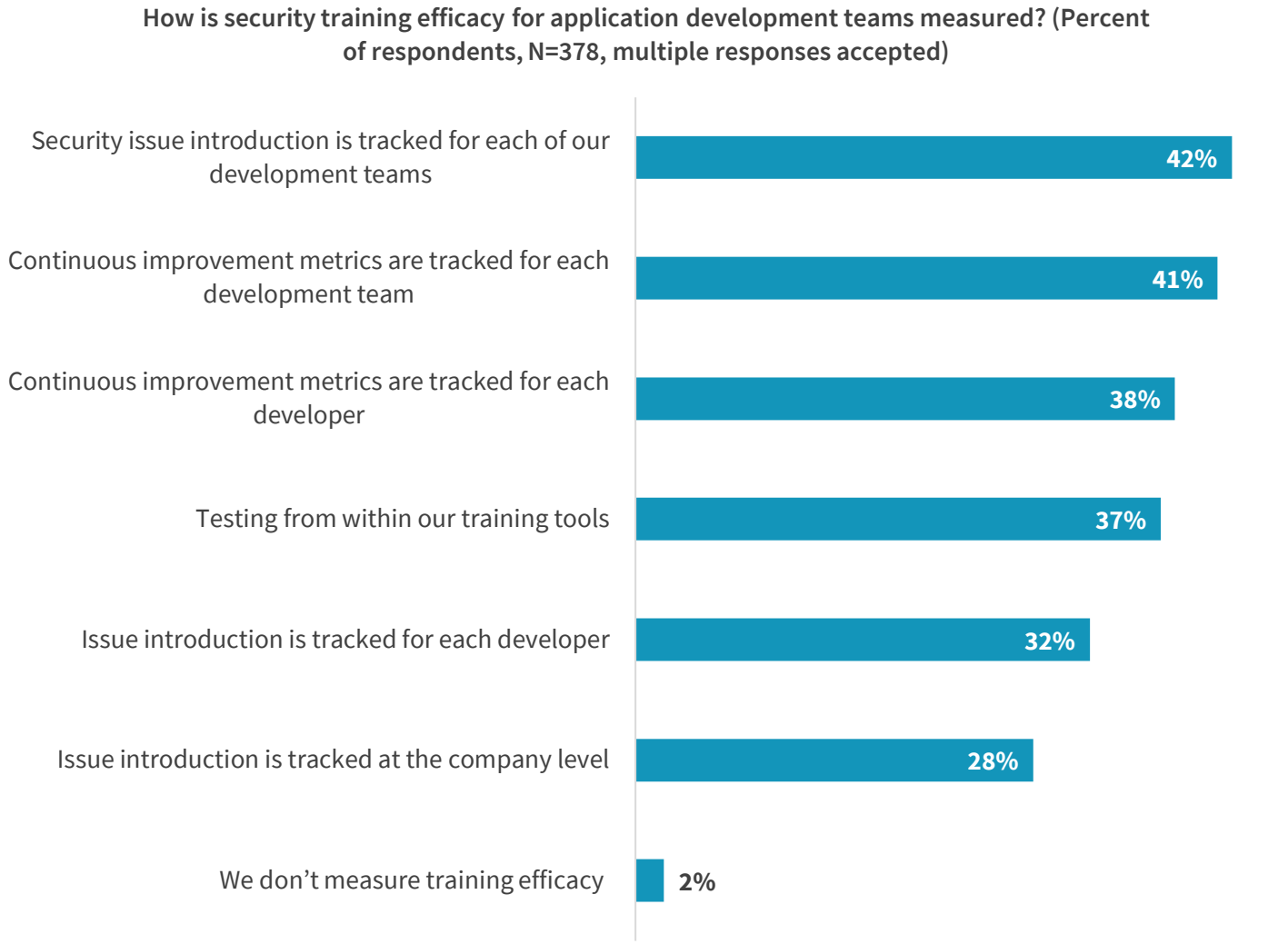
Testing from both inside-out, and outside-in, IAST can understand much of the internal details of the applications' code itself. Using an agent orchestrated into the application, IAST monitors and analyzes the applications but doesn't require testing on code or binaries to detect vulnerabilities. Its coverage is as wide as the testing, since IAST is always active, waiting to analyze the application's activity during functional testing. IAST also has the ability to cover third-party modules and its fast and immediate results don't impact or delay the CI/CD process. Participants in ESG's research agree with the value of an integrated approach, with 37% of organizations citing IAST as one of the top areas in which they plan to prioritize application security investments over the next 12 months.

IDE Integrated Developer Security and Awareness Training

Just as end-user awareness training is essential to making workers more vigilant against spear phishing attacks, so too do developers need to be educated on secure coding practices essential to protect against a range of application attacks including SQL injections (SQLi), cross-site scripting (XSS), cross-site request forgery (CSRF), parameter tampering, and more. And as is also the case with end-user awareness training, developer training should be repeated and integrated into their daily workflow, which means directly in their IDE. However, only a little over half, 53%, of organizations provide security training for developers once per year or less.

Tight IDE integration to make training iterative and contextual includes the ability to access a lesson specific to a class of vulnerability without leaving the IDE. Such integrated tutorials should be prescriptive by identifying the vulnerable lines of code, attacks types that could exploit it, and how to best correct the issue. Finally, IDE integrated developer security and awareness training should include metrics around training performance for KPIs and reporting (see Figure 4).

Figure 4. Metrics for Measuring Developer Security Training Effectiveness



Source: Enterprise Strategy Group

Software Composition Analysis

Another tried and true cybersecurity adage also applies to the need to perform software composition analysis (SCA): You can't secure what you cannot see. In this case, that is the code base. As such, software composition analysis plays a foundational role in application security testing. That is, just as cybersecurity teams need to keep a current inventory of the elements of their company's infrastructure, so too do project teams need to know the composition of their applications, a software bill of materials. And because of the iterative nature of modern software development, the composition of a code base is ever changing, requiring an automated approach to SCA so the bill of materials is always current. Of note is the applicability of software composition analysis with the aforementioned use of third-party libraries and open source software (OSS) so those elements of a source code branch are also scanned for vulnerabilities and to surface other issues such as licensing considerations.

Just as cybersecurity teams need to keep a current inventory of the elements of their company's infrastructure, so too do project teams need to know the composition of their applications, a software bill of materials.

Toolchain Integrations

Native integrations with the tools already used by the project teams for each respective stage of the application lifecycle is critical, including:

- IDE plug-ins for context-based scanning and developer tutorials.
- Source code management (SCM) repositories so scans can be initiated upon commit or a pull request as well as performed retrospectively via integrations with leading SCM systems.
- CI plug-ins to automate AST scans at build time.
- Ticketing and orchestration systems to automate the issue tracking feedback loop.

Comprehensive Coverage of Programming Languages

Akin to the requirement for host-based security controls, such as endpoint security protection platforms (EPPs) and cloud workload protection platforms (CWPPs) to support a range of operating systems, an integrated set of AST products must support a range of programming languages and frameworks. Support for multiple generations of programming languages includes traditional languages such as JavaScript and Python as well as modern languages such as Ruby and Go Lang.

The Bigger Truth

The reliance on business applications, including those being internally developed, has yielded competing objectives like expediting time to market and managing the associated risk. Competitive pressures to further accelerate the rate at which software is developed and delivered has further complicated security considerations. As such, the charter for cross-functional teams building and delivering modern applications is to assure security can keep pace with the rate of innovation.

To do so, the adage "do it right the first time" aptly conveys the strategic role of application security testing (AST) in eliminating organic vulnerabilities before they become a feature of production applications. The principles of AST parallel many of those employed to protect enterprise environments, including attack surface reduction via vulnerability management. Integrated AST solutions enable the automated implementation of this approach to the pre-deployment stages, providing an essential complementary set of controls to a defense in depth strategy.

Enterprise Strategy Group is an IT analyst, research, validation, and strategy firm that provides market intelligence and actionable insight to the global IT community.

www.esg-global.com contact@esg-global.com 508.482.0188