SANS | Research Program

White Paper

Shift Left to Shift Everywhere: Continuous Development's Impact on Security

Written by **Chris Edmundson**

March 2024

BLACKDUCK®

Introduction

Dramatic changes in the application development landscape are introducing new and dangerous security vulnerabilities that enterprises must urgently address. The continuous integration/continuous delivery (CI/CD) pipeline now extends the potential attack surface across the entire enterprise and beyond, downstream from DevOps to the enterprise's industry ecosystem of partners, customers, and end users, making the development process an increasingly attractive vector for malicious activity. And these problems are made even more serious by widespread, well-established, but rapidly evolving technological developments like automation, artificial intelligence (AI), and open-source software (OSS) that are used by both developers and the hackers targeting them.

The development process is now so complex, so widespread, and so fast-moving that it's difficult for enterprises, their directors of DevOps, or their security organizations to fully understand it and manage it effectively, much less successfully defend it against attacks. Complex applications under development may require input from multiple teams—some inevitably more security-conscious than others—that are typically unaware of one another's activities, and it's difficult for security to scale to keep up with development. They may work across heterogeneous multicloud environments on platforms from different service providers with disparate security requirements. They may use third-party code from outside organizations that hasn't been thoroughly vetted. And that code may be written using tools whose impact isn't yet fully understood. Automation and AI, currently the most high-profile examples of such tools, clearly deliver real-world value to developers, and there's no question that they're here to stay. But if organizations do not put appropriate security controls around the output of these tools, they can readily be manipulated by bad actors or misused by developers, resulting in weak code, vulnerable applications, and damage across the software development life cycle and software supply chain.

Critical Development Security Risks

The ongoing changes in the development process may introduce complexities that directors of DevOps, other enterprise stakeholders, and even experienced security practitioners may not fully recognize, making establishing security practices difficult and inconsistent. Some issues that challenge DevOps security programs are:

- **Insecure access controls**
- **Poor credential management**
- **Vulnerable software dependencies**
- **Unsecured development pipelines and supply chains**
- **Lack of visibility into the end-to-end DevOps process**

It's important to recognize how damaging a security failure in the DevOps process can be. Developers work on projects that affect internal users, external partners, enterprise customers, end users, and other third parties. This dramatically extends the scope of their influence, even into areas where they don't directly have access rights and permissions. The result is that malicious or low-security code introduced in development can end up literally anywhere in the enterprise. And if that bad code remains undetected, it's likely to end up in the final product, leaving customers, end users, and the business open to data breaches, ransomware, and other fast-evolving attacks.

Malicious, weak, or vulnerable code introduced in development can end up literally anywhere in the enterprise. And if that bad code remains undetected, it's likely to end up in the final product.

It's clear that a new approach to securing the end-to-end software development life cycle (SDLC) is urgently required—one that applies continuous testing to continuous development. Point-in-time assessments that present little more than a snapshot of current-state security are no longer adequate. Threats and vulnerabilities must be, and can be, identified, remediated, prevented, and mitigated across every step of the development process. This approach means building on, and moving beyond, the widely employed shift-left security framework to an approach that's increasingly being called "shift everywhere."

The shift-everywhere approach—which combines and extends shift-right and shift-left techniques—isn't simply a new methodology for application security. It represents a fundamental change in the enterprise's approach to assessing risk and implementing efficient communication across teams to shrink the window of opportunity for an attack. There's likely to be cultural or organizational resistance from individuals and teams that aren't used to seeing security as an integral part of their functions or to communicating and collaborating with others. New security technologies will have to be evaluated and implemented in ways that don't conflict with existing development technologies and processes. In a shift-everywhere approach, it's important to instill an awareness of the critical roles developers and many other stakeholders play in security, as well as to cultivate their security skills to quickly fix or preclude risks.

Shift Right, Shift Left, Shift Everywhere Defined

Shift right—Performing security testing, quality assurance, and performance evaluation within, or immediately preceding deployment to, the production environment to uncover real-world issues that manifest at runtime

Shift left—Integrating quality checks—including security analysis—throughout the entire SDLC, beginning as early as possible (e.g., in development, at build) to identify potential problems before they are pushed downstream

Shift everywhere—Taking a holistic approach that prioritizes early detection, continuous monitoring, and rapid response to quickly detect and fix issues at any point from development to deployment

Development Pipelines Are Becoming Critical Attack Vectors

The application development landscape is undergoing a radical transformation, one whose complexity and impact are almost impossible to overstate. Teams of developers can work in parallel on software composed of containers, functions, microservices, and more. Code and binary repositories exist in the cloud as hosted development and build pipelines ingest assets from across the enterprise. The benefits of this transformation are unmistakable: The widespread use of CI/CD methodologies—replacing slower waterfall and iterative models—dramatically increases the speed of development and sharply reduces time to market. Automation, AI, and OSS are also playing ever-greater roles in code integration, testing, deployment, and, crucially, reducing some aspects of human error. These ongoing changes all deliver significant advantages, but they can also introduce serious security challenges that demand urgent attention.

Development may involve multiple teams using different tools—integrated development environments (IDEs), package managers, build pipelines, and repositories. Some of these teams may be outside the enterprise, and all may have limited knowledge of what the others are doing. These complex pipelines can obscure risk visibility and make it difficult to assign clear security expectations with each tool. The sheer scale of today's development environments, which often rely on multicloud implementations, presents further challenges to historically manual security assessments and risk mitigation. Lastly, some development teams will also be more security-conscious than others. This challenge is exacerbated because developers are likely to be using new and untried automation technologies, AI tools such as ChatGPT, and OSS from unknown third-party sources—all of which can introduce new threats, vulnerabilities, and attack vectors.

Let's take a look at how this transformation, and its security impacts, could play out in the real world. It's first thing Monday morning, and Katherine, CISO of a major game developer with operations on three continents, has just asked for an emergency meeting with Markus, the company's director of DevOps. She's been planning to meet with him for some time because of her concerns about the security implications of DevOps' growing reliance on automation, AI, and OSS in the CI/CD pipeline. But what's driven her to call this urgent meeting is something else entirely: emerging threat intelligence reports of a highly sophisticated North Korean supply chain attack targeting the development process—an attack that's rapidly spreading beyond the original target to its supply chain partners, its customers, and its end users.

Markus shares Katherine's alarm at the news, and their discussion of the CyberLink hack (see inset, right) and how to respond to it leads to a wide-ranging conversation about the relationship between the DevOps and security organizations. They come away from the meeting with two major conclusions: The first is that neither of them knows nearly enough about the other's challenges, requirements, and concerns. The second—perhaps even more important—is that the rapid changes in DevOps processes affect many roles and organizations other than their own, and they need to immediately bring in many other stakeholders if they're to understand and address the risks those changes introduce.

The North Korean Supply Chain Attack

In late October 2023, CyberLink, a Taiwan-based developer of audio, video, and photo editing applications, discovered that its systems had been compromised. The attack—attributed to the notorious North Korean government–sponsored Lazarus group—was a sophisticated supply chain attack targeting the development process. The hackers modified an installer for one of CyberLink's applications so that, after a predetermined period, it downloaded a hard-to-detect trojan on systems using or installing the application. The full extent and purpose of the attack remain unclear, but in the past, Lazarus has been known to steal sensitive data and infiltrate build environments, moving downstream to exploit third parties and establish advanced persistent threats. The CyberLink hack is known to have infected more than 100 devices, systems, and applications in Taiwan, Japan, Canada, and the United States.[1]

[1] "North Korea-Backed Hackers Target CyberLink Users in Supply-Chain Attack," November 22, 2023, https://techcrunch.com/2023/11/22/north-korea-backed-hackers-target-cyberlink-users-in-supply-chain-attack

Katherine and Markus approach the CEO and ask permission to create a governance committee that will be tasked with identifying and prioritizing the most critical security risks and vulnerabilities and looking for ways to address them before they get pushed into production and leave the final product open to attack. The CEO—who has been reading reports about the CyberLink incident in the mainstream media—immediately grants their request, instructs them to involve a broad range of stakeholders, and tells them to come back to him with a preliminary report from the committee within 10 business days.

The result of the meetings of the governance committee is a heightened understanding by all stakeholders of the importance of integrating security into DevOps workflows. But that isn't a simple matter, especially given the challenges and limitations of traditional testing methodologies that offer only point-in-time assessments and typically have significant gaps in their analysis of risks. A consensus view quickly emerges that continuous testing across the entire SDLC is the only way to deliver effective DevOps security.

DevSecOps Stakeholders

DevSecOps stakeholders—the individuals and organizations with a compelling interest in the security of the application development practice—come from all areas of the enterprise, and in many cases beyond the enterprise itself. These stakeholders will all have different needs, expectations, and challenges, and informed, deliberate actions will be required to create a successful DevSecOps program. Some of the most important stakeholders include:

- **Chief information security officer (CISO)/chief security officer (CSO)**
- **Chief information officer (CIO)/chief technology officer (CTO)**
- **Chief compliance officer (COO)**
- **Chief data officer (CDO)/chief data privacy officer (CDPO)**
- **Legal counsel**
- **Director of DevOps**
- **Application security director/manager**
- **Quality assurance (QA) managers**
- **Product owners/managers**
- **Their counterparts with third-party partners and technology providers**
- **Their counterparts with customers who may seek security attestation**

Note that these roles may not be explicitly defined in every enterprise, so it may be necessary to identify or designate—or in some cases even hire—individuals with responsibility for these functions.

Katherine and Markus are fully on board, and so are the other stakeholders, especially when the other benefits of continuous testing become clear. These benefits include not only early detection and mitigation of security vulnerabilities but also improved software quality and reliability, accelerated delivery cycles, and enhanced collaboration among security, development, and operations teams. This shared understanding of the benefits of continuous testing makes it far simpler to gain senior-level commitment to—and funding for—the transition to the "shift-everywhere" approach to DevOps security.

Katherine and Markus, collaborating closely and drawing on the insights they've received from the other stakeholders, develop a strategy for transitioning to the "shift-everywhere" approach that will eventually take into account the following methodologies.

Continuous testing across the entire software development life cycle is the only way to deliver effective DevOps security.

Shift Left

This methodology includes:

- Security training for developers, helping them write secure code from the beginning, introducing fewer weaknesses and accelerating remediation for issues detected later
- Security unit testing, incorporating security tests into unit testing frameworks
- Threat modeling to identify possible threats and vulnerabilities during software design
- Automated application security testing to discover weaknesses and vulnerabilities during development and the build pipeline, leveraging tools including:
    - Static application security testing (SAST) to discover weak or insecure proprietary code
    - Software composition analysis (SCA) to detect known vulnerabilities in open-source components or libraries within projects
- Enabling developers to perform immediate application security testing on active files or whole projects as they work on them within the IDE, helping to avoid the introduction of new risks

Shift Right

This methodology includes:

- Lightweight, automated penetration testing (for example, Dynamic Application Security Testing [DAST]) on mission-critical applications, to discover exploitable conditions and vulnerabilities during production runtime
- Continuous security monitoring of applications and their running environments, with real-time feedback and monitoring of application and infrastructure logs and discovery of anomalies
- Supplementing static and dynamic analysis with runtime application self-protection (RASP), providing security measures within applications to prevent attacks in production
- Incident response plans, enabling the security organization to be prepared, agile, and ready to respond to security incidents and other events

Shift Everywhere

This methodology includes:

- Accelerating remediation by automatically alerting security, development, and DevOps teams to newly published vulnerabilities affecting previously scanned or deployed applications
- Providing risk-relevant fix guidance and prescribed, curated, developer security training directly to developers, eliminating the gap in security capability and cultivating a stronger culture of DevSecOps over time
- Performing security testing integrated directly into the SDLC and CI/CD pipelines, ensuring automated security gates at each stage, including:
    - Scanning assets within the source code management (SCM) repository with SAST or SCA to detect issues that weren't tested at the developer's desktop or that entered the CI/CD pipeline through unapproved or unsecured secondary workflows
    - Testing at or after the build—typically using SCA, because compiled artifacts are now available—so potentially vulnerable transitive dependencies or malicious artifacts from third parties can be identified if they have been resolved into the project
    - Turning automated functional tests into security tests without additional cycles, using interactive application security testing (IAST) to detect insecure configurations and anomalous activity that manifests in pre-production runtime
- Creating policies to govern integrated, automated security testing based on various project, environment, or business contexts to ensure that only necessary tests run at appropriate points in the pipeline to balance coverage with efficiency

The governance committee begins implementing these practices and technologies. Recognizing that this will necessarily be a long-term process, they ensure that they implement tools and processes that can accommodate changes to the pipeline and the business without requiring a complete rework of the DevSecOps initiative. Katherine is more confident of her understanding of the security requirements of DevOps, and Markus is reassured that new security controls won't necessarily slow down product development. Meanwhile, the other stakeholders are beginning to realize the benefits of the DevSecOps transformation, with a stronger, more resilient security posture and greater confidence in an effective, efficient remediation process. There's a long road ahead, but everyone involved is seeing opportunities for improvement, not just in security but also in efficiency and cost-effectiveness.

Key DevSecOps Technologies: A Closer Look

Let's take a look at some of the tools that can help enterprises secure the end-to-end development process.

- **Static application security testing (SAST)—**This "white box" technology works from the inside out, scanning an application's code to identify and remediate the root causes of security flaws and vulnerabilities. SAST is a useful tool for educating developers about security issues because it gives them real-time access to risk information and fix recommendations, whether they have performed IDE-based analysis on their own or they ingest the results of pipeline-based scans through issue management tools. SAST is a useful tool for security and compliance teams because it helps them align policies to risk tolerance thresholds, compliance standards, or best practices guidelines such as the Open Web Application Security Project (OWASP) Top 10.
- **Software composition analysis (SCA)—**SCA is a security process that automatically detects open source components declared within source code or resolved during builds, providing insight into known vulnerabilities that may be used to exploit the application. SCA provides a secondary benefit to compliance and legal teams by identifying associated open source and third-party licensing that may threaten intellectual property if used improperly. SCA is the foundation of software supply chain security, supporting the creation of software bills of materials.
- **Software bill of materials (SBOM)—**An SBOM documents the third-party components composing software and serves as an attestation of security to partners, investors, and customers. The SBOM enables SCA solutions to provide immediate notification of new risks affecting deployed software as vulnerabilities are published, without the need to perform additional scans.
- **Interactive application security testing (IAST)—**IAST is a "gray box" technology that's applied in the QA stage of the SDLC. It deploys agents to continuously analyze the interactions of running applications in pre-production environments, paired with the source-level insight required to completely analyze application activity and data flow. IAST provides great insight into security risks without burdening runtime environments with weighty, resource-intensive scans and without additional testing cycles, because existing functional tests can be used by the IAST tool for its analysis. Some IAST tools have incredibly low false-positive rates, performing active verification of detected risks to validate exploitability and assist in risk prioritization.
- **Dynamic application security testing (DAST)—**This is a "black box" method of application security testing that looks at an application from the outside while it's running and tests it against simulated attacks. The testers have no access or visibility into the source program itself. More advanced DAST tools can automate this testing, negating the need for manual review by cybersecurity experts and allowing greater scale and flexibility to the analysis. DAST solutions should be lightweight to avoid encumbering runtime environment resources.
- **Runtime application self-protection (RASP)—**Gartner defines RASP as "a security technology that is built on or linked into an application runtime environment, and is capable of controlling application execution, and detecting and preventing real-time attacks."[2] When an attack is detected, RASP can issue alerts, block application requests, and—in some cases—even patch the application. RASP tools can be a somewhat heavy lift for production environments where performance and scale are priorities for DevOps and cloud operations. For this reason, many organizations are replacing legacy RASP solutions with more complete DevSecOps implementations that include the aforementioned tools, orchestrated by a unified security platform and aligned to policies for consistent coverage.

[2] "Runtime Application Self-Protection (RASP)," www.gartner.com/en/information-technology/glossary/runtime-application-self-protection-rasp

Ensuring the Security of the SDLC: A Checklist

A set of basic steps are needed to enhance the security posture of your software development life cycle:

- ✓ Perform regular scans for code security issues using SAST and SCA tools, and implement remediation measures to address identified vulnerabilities. Run those tests as early as possible, including during developer coding in the IDE.
- ✓ Implement the right tests to balance efficiency and coverage, accounting for things like the changes that have been made in a given release, the sensitivity of the information being handled, the deployment environment and its security configurations, and more.
- ✓ Regularly scan projects at stages across the CI/CD pipeline to detect weaknesses and potential vulnerabilities that may not have been detected by earlier security testing or that may have changed in risk status since the last scan was performed.
- ✓ Establish and enforce secure access control, least privileges, and secrets management best practices. This includes enforcement in the final application within the pipeline itself to avoid malicious configuration changes.
- ✓ Establish proper flow control mechanisms to prevent unauthorized data access or manipulation.
- ✓ Build and maintain robust logging mechanisms for comprehensive visibility into system activities.
- ✓ Regularly review logs to detect and respond to security incidents, leveraging a formalized incident response process.
- ✓ Monitor for newly published vulnerabilities impacting previously scanned and deployed projects. An SBOM can be leveraged to assist with this, combined with risk tolerance policies to minimize noise.
- ✓ Centralize visibility of security risks across projects and testing tools for efficient assessment and triage.
- ✓ Integrate automated security testing and controls—aligned with application security policies and using tools including SAST, SCA, and secrets detection—to stop risk propagation downstream and initiate earlier remediation workflows.
- ✓ Establish security policies and testing automation processes that can scale and pivot with cloud-native deployments and multicloud environments.
- ✓ Provide developers with risk-relevant security training and remediation guidance that enables them to fix issues quickly and avoid introducing them again. Encourage them to conduct vulnerability and dependency scans before committing or pushing code changes.
- ✓ Strengthen policy compliance by continuously documenting security measures and ensuring transparency.
- ✓ Work to foster a culture of shared responsibility and security, taking into account the working requirements and success criteria of every contributor to the development process.

Key Metrics

Ensuring the ongoing success of application development security initiatives—and, of course, receiving senior-level support and funding—requires actionable metrics that measure real-world successes and failures. Some important metrics to consider in defining configurations and parameters for a DevSecOps program are:

- **Open security issues and vulnerabilities—**This is often the simplest metric to establish, counting whole numbers of issues detected for each project. This may be heavily influenced by the lines of code within an application, the open-source libraries selected, or the speed at which fixes may be applied before deploying into production. Often, adequately trained developers will introduce fewer security issues over time, as they become more aware of risks and better able to avoid them before checking in code.
- **Mean time to repair (MTTR)—**The average time required to rectify a detected defect. Often, a shorter time to repair (TTR) is required for mission-critical applications. Some enterprises may choose to set a phased improvement plan for TTR as developers become more security-capable and accelerate remediation. This time can be abbreviated by providing faster, closed-loop communication of issues and remediation guidance to developers through integrated workflows or earlier risk detection.
- **Test coverage—**The percentage of code covered by automated tests. This may vary depending on the tests being performed (for example, SAST vs. SCA) or the project's fundamental technologies (for example, language, frameworks, or architecture).
- **Deployment frequency—**The count of deployments within a specified time frame (for example, per day, per week, per month). This metric, although not exclusively tied to security, can be combined with a project's metrics in the aforementioned categories to help establish an assessment of priority for new issues.
- **Lead time—**The duration from code commit to deployment. This can provide a standard for how long a team may have to accomplish a required fix and will often vary by the business criticality of a given project. A comparison of lead time and MTTR can help identify potentially large windows of exposure to compromise that exceed a risk tolerance threshold.

The Bottom Line

The stunning changes in application development processes represent a double-edged sword for enterprises, their DevOps organizations, and their security teams. CI/CD methodologies and other agile practices, automation, AI, OSS, and cloud computing all offer the potential for faster, more efficient development, with cost savings and improvements in customer satisfaction. A proactive approach that embeds security tests and controls continuously across the entire SDLC is required to ensure that these benefits don't come at the cost of security. By embracing a shift-everywhere development security methodology, raising developers' security awareness and skills, and implementing advanced DevSecOps technologies, enterprises can take advantage of the speed and agility of the new era in DevOps while still ensuring the safety and integrity of their applications, their IT environments, and the sensitive assets of their business partners and customers.

Sponsor

SANS would like to thank this paper's sponsor:

BLACKDUCK®