A checklist for AI-powered DevSecOps

Could your codebases use a security boost from AI?

> Use this checklist to help accelerate your remediation times from months to minutes with AI and automation integration.

WRITTEN BY GITHUB WITH ♥

Software security is more important than ever. Yet vulnerabilities continue to find their way into production as the amount of code continues to increase. From patching known issues in dependencies to fixing vulnerabilities to removing accidentally leaked credentials from codebases, security is often just too much for humans to keep up with—especially given the shortage of security professionals.

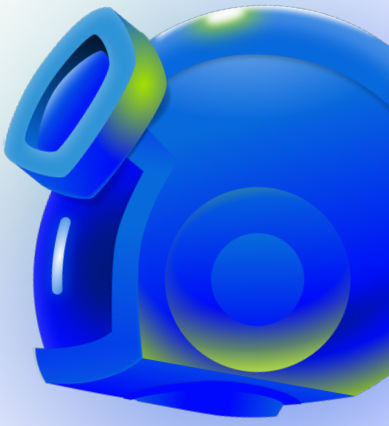
This has led many organizations to adopt DevSecOps, a framework that seeks to integrate security into every stage of the software development lifecycle (SDLC). This often puts developers on the front lines of security, leaving them with the responsibility of fixing vulnerabilities and writing more secure code from the start.

In theory, DevSecOps helps organizations ship more secure software faster. In practice it varies as developers can often face friction in writing secure code and addressing issues. This starts with commonly used security tools that aren't always designed with developers in mind and may provide little context for remediating issues and a large number of false positives, which can cause developers to lose trust in security tools. Meanwhile, developers are often under constant pressure to ship new features before they've had time to deal with existing security issues. Known issues can linger for months or years. With so many competing priorities, security debt can accumulate and leave organizations exposed to threats.

To help manage the onslaught of fixes needed, new generative AI-powered security tools are increasingly being used to help developers identify and remediate risks faster. These tools don't require already overburdened developers to engage with additional manual processes. Instead of "opt-in" processes, AI security tools can automatically run tests and even help with remediation while developers are writing and reviewing code, sharply reducing the amount of time it takes to address issues.

In the following guide, we'll offer you a checklist to help evaluate your organizational DevSecOps and security posture alongside guidance for incorporating AI-powered security tools in your engineering workflows. Let's jump in.

1: ISC2 Cybersecurity Workforce, "How the economy, skills gap, and artificial intelligence are challenging the global cybersecurity workforce," 2023.

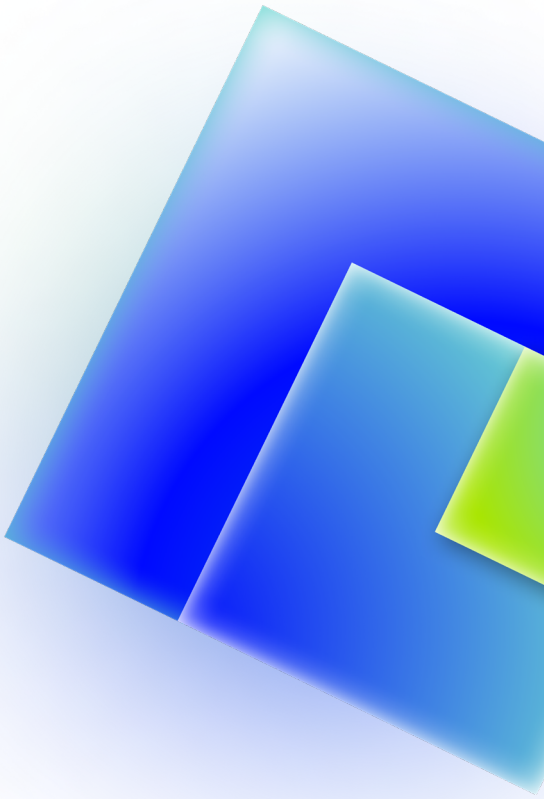


Step 1: Review and implement platform security fundamentals

Secure accounts, and evaluate the security features of your tools and platforms.

Basic security hygiene often begins with simple security best practices such as multi-factor authentication to harden your development platform. A secret exposed for even seconds can be exploited by bad actors to launch broader attacks—much like the SolarWinds attack where its CI/CD systems were targeted. You need persistent vigilance to ensure passwords and other credentials stay out of your codebases.

Many existing security tools detect machine-generated credentials, such as API keys, but they can often fail to detect human-generated secrets, such as passwords, which can leave systems exposed. New generative AI-powered security tools, however, are proving adept at better detecting human-generated secrets to help keep them out of codebases. And they can do it on an ongoing basis, decreasing the likelihood that secrets are introduced into code in the first place.



Common platform security risks and mitigation measures

Risk: Developers accidentally commit secrets to code that ships to production, exposing systems to exploitation.

- [ ] **Prevent secrets from entering code by scanning all code commits for any tokens or passwords that leave your systems exposed.**

  By integrating both traditional security tools and new AI-powered solutions, organizations can proactively guard against this risk. AI-powered tools can detect and alert teams to sensitive data in real time, analyzing code before it even enters production. These tools help enforce a culture of security-first coding and reduce the potential for accidental exposure of critical information across development workflows.

- [ ] **Detect existing secrets by scanning your entire Git repository as well as any other areas where secrets may reside, including issues, descriptions, and comments.**

  Detecting and remediating secrets requires a proactive and comprehensive approach. By scanning all areas where secrets may be stored, including code repositories, issues, and comments, organizations ensure no stone is left unturned. This thorough scanning process provides a safeguard against both new and previously undetected secrets.

- [ ] **Remediate leaked secrets by empowering developers with security expertise with the assistance of AI.**

  Empowering developers with robust security knowledge, supported by AI-driven tools, helps to enable quick and effective responses when secrets are detected. AI coding tools can suggest fixes, flag high-risk exposure points, and even automate certain remediation steps, allowing security teams and developers to work in tandem to mitigate risks. This collaborative approach reinforces security across the development lifecycle, reducing the chance of exposure and fostering a stronger security culture within the organization.

Risk: Unsecured AI tools leak sensitive data to third parties or even the public.

- [ ] **When adopting an AI tool, ensure it meets security best practices, compliance requirements, and has enterprise-grade security features like encryption and data masking.**

  Unsecured AI tools can inadvertently expose sensitive data, creating risks for organizations and users alike. To mitigate these risks, it's essential to vet AI tools rigorously before adoption. Ensuring that these tools meet robust security best practices and compliance requirements helps protect data integrity. Additionally, regular audits and strict data governance policies can support secure AI usage, reducing the likelihood of sensitive information being inadvertently shared with third parties or publicly exposed.

Risk: Your organization fails to meet regulatory and compliance requirements for your industry, or you can't prove that you meet them.

- [ ] **Ensure your platform can compile audit logs and provide easy access to them, so that your organization can better meet compliance standards.**

  It's crucial that your platform supports comprehensive audit logging and provides straightforward access to these logs. Audit logs should capture all relevant events, from user access to system changes, creating a transparent record that can be easily reviewed during audits. Ensuring that these logs are secure, searchable, and readily available enables your organization to demonstrate compliance more quickly and confidently.

2: GitHub, "The enterprise guide to end-to-end CI/CD governance." 2023.

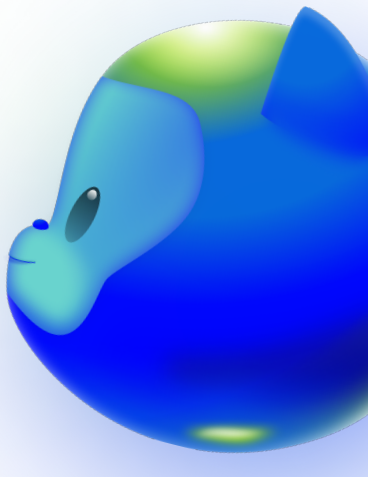
Step 2: Protect your codebase

Isolate the most critical vulnerabilities and use AI to remediate them at scale.

Vulnerabilities can take many forms. Sometimes they're accidentally introduced by developers, and sometimes they come from outside your organization, in your third-party dependencies—both open source and proprietary. The good news is that there are often already fixes for these vulnerabilities. But keeping dependencies up to date can be a challenge, and fixing vulnerabilities in your own code can often get pushed to the bottom of your developers' priority list.

New agentic AI security tools have the potential to transform how we handle remediation, enabling developers with tools to help avoid introducing vulnerabilities in the first place and providing suggestions to fix existing vulnerabilities both in the supply chain and in original code faster than ever. These advanced AI tools act as proactive security partners, scanning for vulnerabilities in real time and offering immediate feedback to developers. By embedding AI-driven security checks throughout the development lifecycle, organizations can shift security left, identifying and addressing potential issues at earlier stages in the workflow.

For example, the AI-powered security tool GitHub Copilot Autofix can help developers fix vulnerabilities an average of three times faster compared to developers who fix vulnerabilities manually.



Common codebase security risks and mitigation measures

Risk: Your applications contain unmonitored software dependencies, exposing you to vulnerabilities and liabilities that you are unaware of.

- [ ] **Create a software bill of materials (SBOM) that includes all software components (libraries, dependencies, etc.) of your product.**

  Unmonitored software dependencies can introduce hidden risks, as vulnerabilities in third-party libraries or components may go undetected until it's too late. By creating a comprehensive Software Bill of Materials (SBOM), your organization gains a clear and detailed inventory of all software components used within applications. An SBOM not only provides visibility but also helps security and development teams quickly assess which dependencies need updates or replacements when new vulnerabilities are disclosed.

Risk: Your application's code includes reported vulnerabilities, potentially with known fixes.

- [ ] **Enable code scanning in your repository to automate monitoring of your codebase.**

  Code vulnerabilities can leave applications exposed to exploitation, especially when known fixes are available but not applied. By enabling automated code scanning within your repository, your organization can continuously monitor the codebase for these vulnerabilities. Automated scanning tools can identify, flag, and even suggest fixes for issues as they arise, giving developers the ability to address vulnerabilities before they reach production. This proactive approach to monitoring makes it possible to bring newly introduced or previously undetected vulnerabilities to attention, significantly strengthening your application's security posture over time and reducing reliance on manual checks.

Risk: You have security scanning tools in place, but have little insight into your security posture.

- [ ] **Use tools like security overview to triage issues and gain a high-level view into how application security efforts are performing over time.**

  Having security scanning tools is a good start, but without comprehensive visibility, it's difficult to assess the effectiveness of these tools or your security strategy overall. By implementing tools like a security overview dashboard, teams gain valuable insights into key metrics and trends over time. This macro-level visibility allows developers, team leaders, and security professionals to understand their current security posture, track progress on remediation efforts, and identify areas for improvement. These tools can filter out lower-severity vulnerabilities, reducing noise and enabling developers to address the most pressing risks without distraction making it easier to allocate resources effectively and address the most critical security issues first.

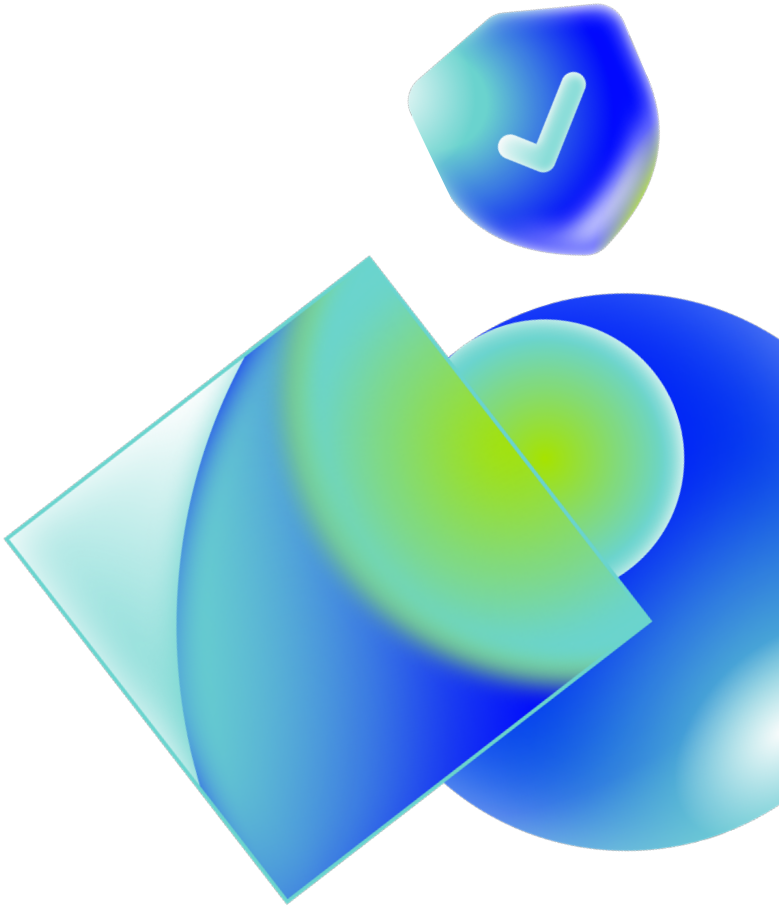
Risk: Your code has detected vulnerabilities that your team has yet to fix.

- [ ] **Automate remediation with code scanning autofix and provide developers, in a pull request, an AI-generated code fix suggestion with every vulnerability alert.**

  Automating remediation further strengthens your security posture by providing real-time, AI-generated code fix suggestions for every vulnerability alert. These fixes, delivered directly in pull requests, empower developers to apply fixes with minimal disruption to their workflow. This not only accelerates the patching process but can also help reduce human error, allowing vulnerabilities to be remediated swiftly and more accurately. With automated triage and remediation, your team can maintain a continuous, agile approach to security, proactively managing vulnerabilities and working to prevent potential exploits before they escalate.

- [ ] **Create security campaigns to prioritize remediation initiatives and track progress.**

  Even with security scanning tools, it can be challenging to fully understand and manage your security posture without organized, targeted efforts. By creating security campaigns, you can systematically prioritize and address specific remediation initiatives, bringing greater focus and accountability to your security strategy. For instance, a campaign targeting the log4shell vulnerability in log4j allows teams to assess the scope of exposure and allocate resources efficiently to eliminate this high-risk issue. Similarly, campaigns focused on high-severity vulnerabilities within key repositories help keep critical codebases secured. Campaigns provide clear objectives and measurable progress, making it easier to rally teams around remediation efforts, track improvement over time, and maintain a continuously evolving defense against threats.



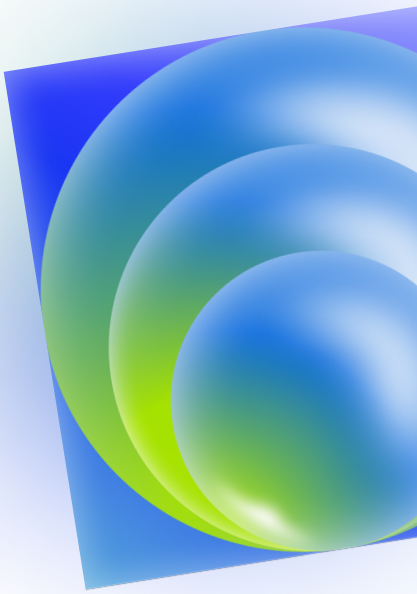
Step 3: Safeguard your build system

Protect the systems used to build and distribute artifacts.

Your build system, though often overlooked, is one of the most critical components in the software development lifecycle. It may not be directly exposed to external threats, but if compromised, it becomes a gateway for attackers to inject malicious code or backdoors without needing to breach personal accounts or source code directly. A secure build process is essential for maintaining the integrity of your application, helps protect artifacts from vulnerabilities or tampering.

Given its importance, it's crucial that the build system is protected with the same level of diligence as other components in your development pipeline. AI-powered tools can help automate security checks and enforce policies across the build process, reducing the risk of human error that could otherwise introduce vulnerabilities.

By implementing best practices for securing the build environment and utilizing AI-driven solutions, you can help ensure that critical gaps are not left in your system, providing a strong foundation for the rest of your DevSecOps efforts. This step safeguards your application from the very start, helping to prevent vulnerabilities, whether introduced accidentally or maliciously, do not make it into production.



Common build system security risks and mitigation measures

Risk: Bad actors modify your build process to insert backdoors or other malicious code.

- [ ] **Only allow authorized personnel to access build environments. Use secure authentication and authorization measures.**

  Unauthorized modifications to your build process can introduce critical vulnerabilities that compromise the integrity of your software. To mitigate this risk, it's crucial to restrict access to build environments to only authorized personnel, safeguarding that sensitive build operations are not exposed to malicious actors. Implementing strong, multi-factor authentication and role-based access controls further enhances security, limiting access to individuals with the appropriate permissions.

- [ ] **Isolate the environment where builds happen to prevent any outside interference or contamination of the build process.**

  Isolating your build environment is another key measure that protects against external interference. By creating a secure, segmented environment for builds, you can prevent unauthorized access and contamination from outside sources, shielding the integrity of your code and dependencies remains intact. This isolation limits the potential attack surface and adds an additional layer of defense against potential threats. Together, these strategies help safeguard your build process from tampering, helping you create software that is more secure and free from malicious alterations.

Risk: You're unsure whether your build processes pull only from the correct sources and the results match the expected outcome.

- [ ] **Use checksums or hash functions to conduct source code integrity checks to see whether the code is coming from a trusted source.**

  Untrustworthy code in your build process can lead to potential security vulnerabilities, as malicious code could be inadvertently introduced. To build trustworthy code, use checksums or hash functions to help verify the integrity of the source code as an essential step. By comparing the checksum of the source code to a known trusted value, you can confirm that the code has not been altered and is coming from a secure and verified source.

- [ ] **Make the build process reproducible.**

  This means that doing the build process in the same environment with the same source code should produce identical results every time. This helps detect inconsistencies that might arise due to unexpected changes in the code or the environment and offers an additional layer of confidence in the integrity of the build.

- [ ] **Sign the builds.**

  Implement a system to sign the builds produced. This will certify that the build has not been tampered with since it was last signed. Implementing a robust signing process ensures that any changes to the build after it has been signed are immediately detectable, offering an added layer of trust and accountability in the build pipeline. Together, these practices create a more secure and auditable process, giving you confidence that your build results are authentic and free from malicious modifications.

Risk: Human-error leads to important tests being skipped, allowing bugs and vulnerabilities into your final product.

- [ ] **Ensure developers use trusted workflows in their build environments by securing automated workflows in the SDLC.**

  Human error is an inevitable part of the development process, but it can lead to significant vulnerabilities and bugs if critical tests are skipped. Automating testing throughout the SDLC reduces the chance of skipping essential tests or using outdated processes. This added layer of security in the SDLC ensures that all code changes go through a rigorous testing process, preventing bugs and vulnerabilities from being introduced into the final product.

- [ ] **Use an AI pair programmer to generate tailored workflows that you can automate with a CI/CD tool.**

  Automating workflows takes time that busy developers and DevSecOps teams often lack. AI pair programming tools can accelerate the process of writing automation scripts and help tighten the security of these processes. Together, AI-assisted workflow generation and automation can create a more reliable and efficient development cycle, improving the overall security and quality of your software.

Take this with you

Application security remains an enormous challenge for organizations of all sizes—but AI is poised to help change that amid a new category of AI-powered security tools that are quickly evolving. These AI-powered solutions offer the potential to streamline and enhance the security process, providing faster, more accurate threat detection, and automated remediation that can shift security practices from reactive to proactive.

For DevSecOps to truly thrive, a strong collaboration between development and security teams is essential. Developers need to be empowered with the security expertise to tackle vulnerabilities directly, without overwhelming security teams with every issue that arises. This is where AI shines. By equipping developers with real-time, automated solutions, AI allows for faster fixes and immediate responses to vulnerabilities. It's this seamless integration of development and security that unlocks the true promise of DevSecOps—enabling teams to code securely while maintaining speed and innovation.

As AI continues to evolve, its integration into your security workflows will not only help improve your ability to safeguard your systems today but also prepare you for the increasingly sophisticated threats of tomorrow. Don't wait for the next breach—leverage AI to enhance your security posture and build a more resilient, agile development process that anticipates and addresses risks with greater efficiency.

Learn more

Learn more about how powerful generative AI-powered features in GitHub Advanced Security can help you better secure your software supply chain more efficiently and faster.

GitHub Copilot Autofix, for instance, has shown dramatic reductions in the amount of time between detection and successful remediation:

- Organizations that used Copilot Autofix **fixed code vulnerabilities more than three times faster** than those who did so manually, reducing time to fix for a pull request-time alert from 1.5 hours to 28 minutes.
- **They were able to fix cross-site scripting vulnerabilities seven times faster**, reducing time to fix to 22 minutes, compared to almost three hours.
- **They fixed SQL injection vulnerabilities twelve times faster**, cutting time to fix to just 18 minutes, compared to 3.7 hours.

"Copilot Autofix takes care of cumbersome security tasks, ensuring our existing and new code is always as secure as possible. Vulnerabilities are flagged immediately and code changes are recommended automatically. It helps our teams to free up time so they can focus on more strategic initiatives.

Mario Landgraf, Community Manager, Security // Otto (GmbH & Co KG)