BLACK DUCK®

GUIDE

ULTIMATE CHECKLIST

SAFE CODING PRACTICES FOR AI-GENERATED AND OPEN SOURCE CODE

Open source software and AI-generated code are both increasingly common in software development. They offer numerous benefits, such as enhanced productivity and faster development cycles. But they also introduce significant risks that organizations must mitigate to ensure the security and reliability of their software products.

OPEN SOURCE CODE RISKS

Common risks associated with using open source code include

- **Unpatched vulnerabilities:** Open source components can contain unpatched vulnerabilities that attackers can exploit. For instance, the Log4j vulnerability exposed millions of systems to remote code execution attacks.
- **Abandoned or unmaintained code:** Some open source projects may be incomplete or become inactive, leaving vulnerabilities unpatched and users unaware of risks.
- **Malicious code injections:** Anyone can contribute to open source code, which means attackers can inject malicious code into repositories or packages, typically using compromised contributor accounts or malicious pull requests.

AI-GENERATED CODE RISKS

Common risks associated with AI-generated code include

- **Insecure or flawed code:** Center for Security and Emerging Technology (CSET) research shows that nearly half of the code snippets produced by AI coding assistants contain bugs that could lead to hazardous software failure or malicious exploitation. AI coding tools are trained on large data sets that include open source code, so they produce code with similar flaws.
- **Lack of transparency:** The lack of transparency in how AI models generate code makes it challenging to identify potential security risks and defects that cause software to fail.
- **Rapid proliferation and deployment:** AI-generated code can be produced and deployed rapidly, potentially outpacing code assessments. This increases the risk of vulnerabilities in production environments.

USING AI-GENERATED AND OPEN SOURCE CODE SAFELY

To address the concerns about AI-generated and open source code, organizations must adopt proactive strategies such as rigorous code reviews, secure coding practices, automated testing, and more.

Conduct thorough and effective code reviews

Rigorous human and automated code reviews are crucial for identifying and mitigating code defects and security vulnerabilities early in the software development life cycle. Repetitive checks for formatting, basic syntax, and style violations can be automated so human reviewers can focus on higher-priority review tasks.

Focus on intent and functionality

- ☐ Review against business requirements, and verify that the software performs properly and adheres to the specified business logic.
- ☐ Ensure that the code is adequately documented and that comments accurately reflect its purpose and functionality.
- ☐ Test functionality with edge cases. Pay close attention to error handling.

Prioritize security

- ☐ Assume AI-generated code, like any other unreviewed code, contains vulnerabilities until proven otherwise.
- ☐ Prioritize human review for security-sensitive sections such as authentication, authorization, and data-handling functions.
- ☐ Validate input and output handling. Carefully examine how AI-generated code processes user input and interacts with external systems to ensure proper validation and sanitization.

Evaluate code quality and maintainability

- ☐ Verify that the code functions correctly and is consistent with coding standards, duplicated logic, and maintainability.
- ☐ Ensure that the code follows established coding standards and style guidelines for consistency and readability.
- ☐ Identify and remove duplicate code, simplify overly intricate logic, remove undocumented workarounds, etc.

Enforce secure coding practices

Adopting secure coding practices can stop security vulnerabilities from being introduced in the first place, preventing the need for remediation after bugs are discovered.

Validate and sanitize all inputs

- ☐ Establish a zero-trust policy for all inputs to prevent SQL injection and cross-site scripting attacks.
- ☐ Implement server-side validation. Client-side validation is useful for user experience but should be supplemented with robust server-side validation to prevent malicious input from reaching the back end.
- ☐ Encode or sanitize all data before displaying it to the user or incorporating it into other systems to prevent injection attacks.

Enforce the principle of least privilege

- ☐ Grant users, processes, and applications only the minimum necessary permissions to perform their required tasks.
- ☐ Implement role-based access control. Assign permissions to roles and then assign roles to users, simplifying access management.
- ☐ Configure systems and applications to deny access by default, and only explicitly grant necessary permissions.

Secure error handling and logging

- ☐ Implement secure error handling and logging mechanisms to prevent information leakage and aid in incident response.
- ☐ Avoid logging sensitive information. Never include sensitive data like passwords or personal identification in log files.
- ☐ Utilize structured logs for easier analysis and set up monitoring and alerts for critical errors or suspicious activities.

Encrypt data in transit and at rest

- ☐ Employ strong encryption standards like AES for data encryption.
- ☐ Use secure communication protocols like HTTPS/SSL/TLS to encrypt data during transmission over networks.
- ☐ Store encryption keys separately from the data they protect, using secure key management systems and regularly rotating keys.

Implement vulnerability management processes

Implement vulnerability management processes aimed at discovering, evaluating, and correcting security flaws to reduce attack surfaces and build more secure and resilient software.

Use patch management systems

- ☐ Define a clear patch management policy. Establish patching criteria, schedules, and responsibilities for different categories of software and devices.
- ☐ Identify all hardware and software assets requiring patching and prioritize them based on their criticality and risk to the organization.
- ☐ Utilize patch management software to automate updates and patch deployment, streamlining the process and reducing manual effort.
- ☐ Maintain detailed records of patching activities, including patch status, installed updates, etc.

Perform regular security audits and penetration testing

- ☐ Schedule security audits at least annually or following significant system changes to ensure compliance and identify new vulnerabilities.
- ☐ Conduct penetration tests quarterly or biannually to simulate real-world attacks and uncover exploitable weaknesses.
- ☐ Use automated scanning tools and manual penetration testing techniques for a comprehensive security posture assessment.

Utilize open source software dependency management

- ☐ Create and maintain a Software Bill of Materials (SBOM) to track all open source components and their dependencies in your applications.
- ☐ Integrate automated vulnerability scanning tools into your CI/CD pipeline to identify vulnerabilities in open source components at various stages of development.
- ☐ Assess the severity and impact of identified vulnerabilities and prioritize remediation efforts, addressing critical issues promptly.

Automate code quality testing

Implement static application security testing (SAST), dynamic application security testing (DAST), and software composition analysis (SCA) to proactively identify and remediate vulnerabilities in their software throughout the entire development life cycle.

Perform SAST scans to analyze source code

- ☐ Track and manage compliance with coding standards.
- ☐ Perform continuous vulnerability assessments of applications in development.
- ☐ Find and fix issues like SQL injection, cross-site scripting, or insecure configurations.
- ☐ Locate the exact location of vulnerabilities within the code, and get detailed remediation suggestions.

Perform DAST scans on running applications

- ☐ Perform continuous vulnerability assessments of applications in QA and production.
- ☐ Discover vulnerabilities such as authentication bypasses, weak passwords, and flawed configurations that only surface during runtime.
- ☐ Evaluate risk and prioritize remediation efforts.

Perform SCA scans on open source code

- ☐ Create SBOMs to meet customer and compliance requirements.
- ☐ Find and fix open source dependencies in source code, files, artifacts, containers, and firmware.
- ☐ Evaluate risk and prioritization and remediation efforts.

HOW BLACK DUCK CAN HELP

To ensure software is secure and reliable, organizations should

- ☐ Understand risks associated with AI-generated and open source code
- ☐ Implement proactive mitigation strategies
- ☐ Deploy effective application security tools including SAST, DAST, and SCA

Coverity® Static Analysis from Black Duck provides comprehensive code scanning that empowers developers and security teams to deliver high-quality software that complies with security, functional safety, and industry standards. Black Duck® Continuous Dynamic rapidly and accurately finds vulnerabilities in live websites and applications. Black Duck® SCA helps teams manage code security, quality, and license compliance; create and manage SBOMs; and manage software supply chain risk. To learn more, book a demo.

ABOUT BLACK DUCK

Black Duck® meets the board-level risks of modern software with True Scale Application Security, ensuring uncompromised trust in software for the regulated, AI-powered world. Only Black Duck solutions free organizations from tradeoffs between speed, accuracy, and compliance at scale while eliminating security, regulatory, and licensing risks. Whether in the cloud or on premises, Black Duck is the only choice for securing mission-critical software everywhere code happens. With Black Duck, security leaders can make smarter decisions and unleash business innovation with confidence. Learn more at www.blackduck.com.