SECURE CODE WARRIOR

WHITE PAPER

AI Coding Assistants: A Guide to Security-Safe Navigation for the Next Generation of Developers

Large language models deliver irresistible advantages in speed and productivity, but they also introduce undeniable risks to the enterprise. Traditional security guardrails aren't enough to control the deluge. Developers require precise, verified security skills to identify and prevent security flaws at the outset of the software development lifecycle.

Authors:

Pieter Danhieux,
CEO & Co-Founder,
Secure Code Warrior

Dr. Matias Madou,
CTO & Co-Founder,
Secure Code Warrior

Executive Summary

In less than three years since the public debut of generative artificial intelligence, large language model (LLM) coding assistants have become nearly ubiquitous in software development, increasing efficiency and boosting productivity for developers throughout the industry. AI/LLM assistants have fundamentally changed the software development lifecycle (SDLC). While AI models are supercharging output, they have also raised significant concerns regarding the security of the code they generate.

To put it bluntly, the code being generated by AI can't be entirely trusted (yet). A variety of tests and studies, as well as their persistent use by developers, have consistently demonstrated that LLMs produce code with vulnerabilities that can, and often do, migrate into the codebase. Our own experiments into their overall security comprehension found consistent difficulty with critical categories of vulnerabilities. With three-quarters of developers using or planning to use AI tools in their development processes, these tools have created a significant level of risk that enterprise security leaders must now contend with in increasingly complex environments.

No current AI coding tool—whether it's Cursor, Copilot, Windsurf, or Cline —is accurate enough at secure coding and contextual security on its own to be considered "safe" in an enterprise environment today. Even the most up-to-date security programs fall short when

it comes to securing AI-assisted development. Additionally, a murky regulatory landscape, ranging from the EU AI Act's attempts to define risk levels, to the U.S. approach based on voluntary cooperation, adds further complexity in navigating tools that are viable and appropriate for the tech stack. General security performance is also dependent on which LLM is being used in the backend of the AI assistant. And if you don't trust the backend LLM, it's possible you can inadvertently leverage malicious, backdoored examples like "BadSeek" and introduce even more threat layers.

Establishing AI policies is commendable, but guardrails on AI use—whether internal or external—won't prevent developers with low security awareness from producing an endless stream of buggy code. Measured and verified security skills are the only path forward to genuinely reducing developer risk in a scalable manner, and are essential in AI workflows, both for secure prompting of AI and for reviewing AI-generated code. The next generation of developers must be security-aware and able to leverage the benefits of AI coding assistants while ensuring that the code remains secure throughout the Software Development Life Cycle (SDLC).

Introduction

When a demo of ChatGPT, an AI chatbot built on an LLM by OpenAI, first appeared in November 2022, it drew immediate interest. OpenAI quickly followed up with new iterations of ChatGPT and other companies followed with their own AI chatbots. Developers were quick to jump on the train, taking advantage of the potential benefits that AI assistants provide, including increases in speed, productivity, overall code quality, automated documentation, immediate answers to questions and the easy access to a programming assistant. By June 2023, 92% of developers in a GitHub survey reported using AI tools on the job or outside of work.

Since then, the market for AI tools has expanded, with other providers releasing coding-capable tools of their own, such as Cursor AI, Google Gemini, Amazon Q Developer, GitHub Copilot (the software development complement to Microsoft's general-purpose Copilot) and, recently, DeepSeek from China.

It's easy to see why AI tools have become so popular for virtually any use, including software creation. The tools can enhance productivity, reduce repetitive tasks and even improve code quality, acting as intelligent pair programmers, offering code suggestions and bug fixes. In addition to boosting speed, AI coding assistants allow developers to generate entire functions, modules or even applications in seconds. A prompt elicits an immediate response, which has also led to "vibe coding," where a user—even one without programming skills—can create an application by simply asking an LLM to do it. In programming environments, vibe coding enables a more fluid and intuitive workflow, allowing developers to focus on high-level intent rather than syntax.

This has all contributed to an exponential surge in the volume of written code. But that productivity boost comes with new challenges concerning the safety and quality of the code.

In recent surveys of broad swaths of the industry, AI use among developers hasn't quite hit the 90-plus percent rate that GitHub's early survey found, but plenty of developers are avidly using the tools. Stack Overflow's most recent annual survey, of more than 65,000 developers in September 2024, found that 76% of developers either were using or planned to use AI tools, a jump from 70% in 2023. And while 72% said their view of the tools was either favorable or very favorable, that was a drop from 77% the previous year. Overall, 42% of respondents said they trust the output of AI tools in their workflows, while 31% said they didn't trust AI outputs. Some people have yet to form an opinion, with 27% still on the fence about it.

Studies Define the Weaknesses in AI-generated Code

User opinions can vary, but AI tools have come up short when put to the test in recent assessments. The recently introduced BaxBench coding benchmark, created to evaluate the ability of LLMs to generate secure code, found that even when including the best-performing LLMs, 62% of solutions overall were either incorrect or contained a security vulnerability. BaxBench tested 21 LLMs against its benchmark, consisting of 392 security-critical backend coding tasks, and they found that even when the LLMs solutions were correct, about half were insecure.

The BaxBench group said that its results raised questions about current metrics and evaluations that look solely at code correctness. It also noted that security requirements add complexity to coding and can result in making tradeoffs in favor of correct code, suggesting that more targeted efforts to ensure secure coding are needed.

Other tests have echoed those results. An in-depth evaluation of code snippets generated by GitHub Copilot illustrated the challenges of working with all LLMs. The study, conducted by researchers at universities in China, Australia and New Zealand, analyzed 452 snippets generated by Copilot and found that 29.6% of Copilot-generated code snippets have security weaknesses related to 38 different Common Weakness Enumeration (CWE) categories, including several significant CWEs. Eight of the CWEs affected are among MITRE's 2023 CWE Top-25.

The authors also explained how easily vulnerabilities can slip into AI-generated code. LLMs are trained on billions of lines of open-source code, which are likely to contain unsafe coding patterns. Copilot, for example, is trained on untrusted data from GitHub, which, the authors say, is known to contain flawed programs. As a result, it may replicate those security issues, which if not caught can go into production and eventually be used to train other LLMs, thus creating a vicious cycle. Meanwhile, insecure code can proliferate in the software ecosystem, raising the risk for organizations everywhere.

The authors also pointed out, as BaxBench did, that research to date on code generation tools has focused mostly on the correctness of the results in response to prompts while paying less attention to security, which seems to reflect the focus of the tools themselves. In BaxBench's results, to take one example, GPT-4.0o produced correct results 45.6% of the time but was rated Correct and Secure just 29.6% of the time. The highest scoring model, OpenAI o3, had a Correct rating of 65.3% but a Correct and Secure rating of 46.9%.

BaxBench tested different versions of tools from providers, and the results varied. A couple of OpenAI models, for instance, had higher Correct and Secure scores than GPT-4.0o. But all had similar disparities between being correct and secure. For example, DeepSeek R1 scored 51.5% Correct, but only 32.1% for Correct and Secure; DeepSeek V3 scored 39.6% and 19.6%, respectively.

DeepSeek, created by a Chinese AI firm of the same name, turned the market on its ear earlier this year. It was touted as an open-source tool that was developed for considerably less money and resources than other LLMs and operated on lower-power chips, all while reportedly performing as well or better than other LLMs. Shortly after its release in January 2025, DeepSeek's R1-powered chatbot shot to the top of the iOS Apple Store charts as the most-downloaded freeware app, while causing stock market disruption for more established players. But despite the hype and the presence of its proprietary coder tool, DeepSeek has shown the same limitations in coding security (along with presenting other vulnerabilities).

AI coding assistants offer undeniable benefits in productivity and speed, but when it comes to producing secure code, unrestricted use of AI has been demonstrated to be unsafe at any speed, regardless of the tool used.

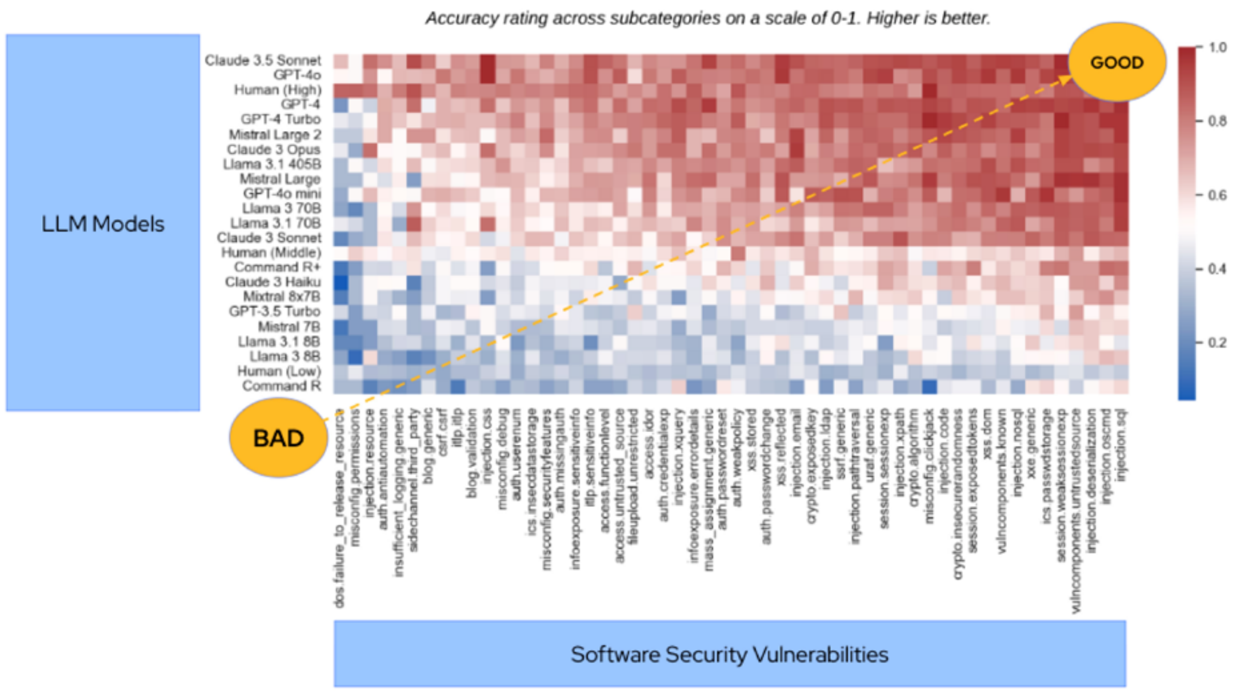
It's not that the tools aren't very useful, nor that they are incapable of creating secure code. But they don't produce secure code often enough to be trusted blindly. A Snyk survey found that 76% of respondents believed AI-generated code was more secure than code created by humans, but 56.4% also stated that AI introduces errors either sometimes or frequently. And those errors can be showstoppers, considering that the same survey found that 80% of developers do not apply AI code security policies when using the tools.

This is deeply concerning, especially in the wake of our own tests into the apparent security comprehension skills of popular LLMs.

Details in the Data: What We Discovered About the Real-Time Security Comprehension of LLMs

The Secure Code Warrior Learning Platform comprises multiple components that form a multifaceted, hands-on, and data-driven approach to secure coding and developer governance. Within this system, we offer thousands of coding challenges based on real code snippets, in over 60 languages and frameworks, covering hundreds of vulnerability categories.

We have years of data on how humans respond to and perform in these challenges, but what about AI agents and LLMs? We conducted an in-house experiment to see just how well they would fare when asked to complete them, just as any developer would when using the platform to upskill:



This graph represents an aggregation of results from several top LLMs (latest versions available in 2024), compared with human users, as they navigate a range of software security vulnerabilities that span from common to more obscure.

Specific categories are relatively more complex for LLMs. More vague or subjective categories, such as “DoS protection”, “insufficient logging”, or “misconfigured permissions”, prove more difficult, while more superficial and straightforward patterns are easier, such as the injection categories.

This is disturbing, as OWASP reportedly found security misconfiguration in 90% of the applications they examined. This is a common attack vector ruthlessly pursued by threat actors, and these small windows of opportunity can have devastating consequences. It is vital that developers, whether AI-assisted or not, upskill adequately to keep security at the forefront of their minds.

How Secure Code Warrior’s challenges work

In the Learning Platform, challenges are broken into different stage types that address key parts of remediating a vulnerability in code:

- In the ***Identify*** stage, the task is to select the correct vulnerability category when presented with a vulnerable code snippet.
- In the ***Locate*** stage, the task is to select a vulnerable snippet of code when given a vulnerability category name.
- In the ***Fix*** stage, the task is to identify the most effective fix for a given vulnerability.

How we utilized prompting

The generated prompts include mostly the same information that human players would be given before they start. In the case of vulnerable code snippets, only the files containing the vulnerabilities are shown, along with only the relevant lines from these files. Relevant lines are the snippet lines themselves, context lines before and after, and any structure or scope-defining lines affecting the snippets. This includes class and function definitions only for those classes and methods that contain the snippets. The rationale behind this is to reduce the context size required, thereby enabling the running of more LLMs.

The prompts were generated in markdown syntax and utilize the Chain of Thought pattern, and all reasoning was disregarded during answer parsing. This was solely used to induce better response quality. We used a single-shot pattern and did not present any extra information in the prompts. All responses will be based solely on the LLMs’ prior knowledge, reasoning, and test-taking ability.

There is reason to believe that better answers might be obtained by using few-shot prompts, but we opted not to pursue this approach since the examples are large, and this would be problematic due to context size. It also would be less representative of how people organically interact with LLMs, so it might be less indicative of actual security considerations in the wild.

Conclusion

In these key experiments, we've examined how LLMs perform on specific code security tasks and compared those results to human performance on the same tasks.

We've shown that the best LLMs perform comparably with proficient people at a range of limited secure coding tasks. However, there is a significant drop in consistency among LLMs across different stages of tasks, languages, and vulnerability categories. Generally, top developers with security proficiency outperform all LLMs, while average developers do not.

One thing that's very clear from all of the recent surveys is that developers who are constantly under pressure to churn out more code are making use of AI tools. The benefits in speed and productivity are too great to ignore. But it's also past the time when we should be taking clear steps toward ensuring secure development. Developers need to have both the skills to be able to prompt AI in a manner that generates secure code, and also (as BaxBench recommends) perform competent security reviews of the generated code. Security skills become even more critical in this world where a junior dev can generate thousands of lines of code per hour: it is vital that developers and organizations significantly raise security awareness and acquire secure coding skills.

Why the SDLC Is Ground Zero for the Future of Security

Software flaws are a top target of cyberattackers, who can exploit vulnerabilities in applications and APIs, misconfigured software and other flaws to gain access, compromise data and carry out attacks. As software has proliferated with the growth of cloud-based systems—and as the industry has gotten better at protecting networks—weaknesses in software security have become increasingly popular as an attack vector.

The types of attacks targeting software run the gamut of attack vectors and complexity, from classic SQL injection, resulting in direct database access, to complex, chained vulnerabilities that enable arbitrary code execution and data exfiltration. Software vulnerabilities are exploited to carry out data theft, ransomware and distributed denial-of-service (DDOS) attacks and contribute to phishing campaigns, supply chain attacks, insider threats, credential thefts, data breaches and a host of other malicious activity. Those types of attacks dominate the lists of the most prevalent attacks, such as the OWASP Top Ten or CrowdStrike's 12 Most Common Types of Cyberattacks.

The threat landscape has evolved as software development itself has grown into a more dynamic, creative and complex environment. The security skills of developers must evolve accordingly, but in most organizations that has yet to happen. Meanwhile, traditional approaches to security have been overwhelmed by a tidal wave of new code.

As more lines of code are produced faster than ever before, the level of scrutiny they get from overworked security teams goes down and the number of vulnerabilities and unnoticed flaws goes up. The technical debt accrued by an organization as buggy software proceeds through the SDLC can quickly grow into a mountain. And as our research has shown, AI only exacerbates the problem because people normally take AI-generated answers for granted, essentially eliminating any critical thinking or scrutiny of most AI responses.

The best way to deter the spread of vulnerabilities is to stop them early, via developer education on secure coding and by emphasizing the importance of code review, testing and secure coding. Implementing secure best practices from the start of the SDLC is critical even when code is being created entirely by humans, but it's especially important when using tools such as GitHub Copilot, OpenAI, Cursor AI, Amazon Q Developer or any other LLM.

The Key Elements of Developer Risk Management in AI-Assisted Environments

A security program that actively targets developer risk should contain specific features to ensure that developers become security proficient in the right way, ultimately reducing the danger they pose when improperly skilled. The program should be agile and flexible, delivering management tools and learning pathways on schedules and in formats that fit developers' working lives. And most critically, it should become the cornerstone of an ongoing cultural shift toward a security-first mindset.

Such programs are also most effective when they involve hands-on sessions addressing real-world problems developers are likely to encounter on the job, and that will ultimately surface in AI-generated code. If properly implemented, it not only benefits developers, whose acquired skills can help their career paths, but also enables the company to better manage software development and ensure quality and security. It can pave the way for security, development and engineering teams to collaborate effectively while improving security enterprise-wide.

Impactful developer risk management should start by helping developers understand the potential pitfalls of AI-generated code, emphasizing the importance of data security, model integrity and ethical considerations. But it also should provide the means for people to make improvements individually and as part of a team. A suite of tools, such as those available on the Secure Code Warrior Learning Platform, provides the full spectrum of capabilities and features organizations need to ensure safe and secure software development. It enables organizations to implement a Secure by Design approach, as advocated by the U.S. Cybersecurity and Infrastructure Security Agency.

Some of the key features include:

Benchmarking the Security Skills of Developers:

Organizations need to establish a baseline of security skills that developers should possess, including the ability to write secure code themselves and to review code generated by AI assistants, as well as code obtained from open-source repositories and third-party providers. This has been a difficult challenge at the enterprise level for some time.

A benchmark like the industry-first SCW Trust Score enables organizations to measure the security posture of developer teams while providing a baseline for their learning programs. It can also be used to evaluate the overall effectiveness of an educational program.

Benchmarks provide visibility into a program's effectiveness, using data-driven assessments to demonstrate progress both within the organization and in relation to industry standards and the performance of industry leaders. In addition to providing a clear view of developers' progress, benchmarks also identify areas in need of improvement and can help organizations to develop new learning methods.

Progress can be measured both internally and against industry standards and best practices. To assess an organization's performance in relation to the broader context, Trust Score employs three benchmarks: a global benchmark that measures progress against all companies across industries, a technology benchmark explicitly tailored to the tech industry, and a banking and financial services benchmark for companies in this sector.

Benchmarks contribute significantly to a learning program that helps organizations manage developer risk, resolve key vulnerabilities and identify developers capable of working on sensitive projects and potentially engaging directly with their AppSec counterparts on threat modeling exercises.

Investigating the Trustworthiness of the AI Assistant:

We've touched on ensuring the backend LLM powering the tool is legitimate and fit for purpose (lest you end up with a malicious, backdoored version, or even one that is weaker on secure coding best practices), but what level of trust are you placing in the entire tool? The drawcard of agentic AI tooling is that it can run autonomous processes and make decisions according to the set parameters; essentially, it needs to be assessed on its trustworthiness with the same scrutiny you would apply to a human developer.

Developers who cannot demonstrate their security proficiency should not be granted access to sensitive repositories, and they should not be trusted to assess the security of AI-generated code using their tools. Ideally, measures would be in place to validate developer security proficiency, and, once proven they can navigate relevant security issues and uphold established policies, the individual and their approved AI agent can be assumed to have passed a baseline level of security prowess and will, in turn, reduce the risk associated with poor security skill and code review.

Enforcement & Validation in the Code Commits:

While Trust Score uses benchmarking metrics to measure and quantify the progress of individuals, teams, and the organization overall, SCW Trust Agent builds on Trust Score, drilling down to assess the specific security skills of developers for every code commit, going beyond merely measuring training performance by showing how well developers are applying their new skills. Combined with the agile learning platform, it enables organizations to implement a comprehensive program for upskilling developers, providing clear visibility to every step of the process.

Observability & Governance Across All Source Code Repositories:

Developer risk management must have a powerful impact at the repository level. It has never been more crucial to implement continuous observability for granular insight into code health, insecure patterns, and compromised dependencies, fully committing to a proactive assessment of the security posture. This should be paired with automated governance that enforces security policies and guardrails directly within the repository workflow, ensuring secure coding practices by default. This approach enables developers who are proficient in the appropriate languages and frameworks to maintain established security policy, and provides essential oversight before deployment.

Developer risk management at the repository level integrates observability for real-time insight into developer activity, and governance to enforce security policies directly within the development workflow, ensuring secure coding practices by default and proactive risk mitigation.

Role- and Language-Specific Learning Pathways:

Depending on a developer's job, training can focus on areas such as front-end web, mobile, back-end or infrastructure-as-code (IaC), enabling developers to work with the exact APIs and code structures they'll use in creating software.

Developers need to be able to upskill in the programming languages they use, whether Rust, Python, Typescript or other languages used for web applications like Kotlin Spring API, Java Enterprise Edition, or Python Web API. A learning platform should also support mobile programming languages like Java Android, Swift or React Native. And many developers also need to train on different platforms or tools like Kubernetes, Docker and Ansible for IaC. SCW's platform, for example, covers more than 60 language frameworks.

Interactive, Ongoing Sessions.

Learning platforms should mirror real-life situations as much as possible. Lab exercises can, for example, simulate instances where an AI coding assistant (or a human coworker) makes changes to existing code, which the developer must then accept or reject.

Developers need to keep up with all of the latest developments in the threat landscape. This can be done by reviewing the most recent OWASP Top 10 for LLM Applications, offering detailed guideline breakdowns and providing case studies, videos and other information on both the advantages and risks of using AI/LLMs in creating code.

Absolute Flexibility:

A crucial aspect of ongoing education is that a program must be able to adapt to changing conditions. As developers' skills improve, for instance, the organization may want to adjust internal goals. The emergence of new cyber attacks will also necessitate new focuses for training. As time goes on, you may need to take a fresh look at how an education program aligns with business requirements (which can also change) and the organization's security posture.

A comprehensive, agile education program also gives leaders the means to scale the program across an entire enterprise, which is something that has proved a challenge for many organizations. Our research has shown that implementing a secure-by-design approach quickly produces measurable improvements at both large and smaller organizations.

Conclusion

In the face of an expanding, increasingly sophisticated and progressively more malicious threat landscape, cybersecurity must launch a new era built on a security-first culture that reflects the postures on national, international and organizational priorities. The most serious threats today are those attacking software, so security needs to start with secure software creation.

Developer and security teams can no longer exist as completely separate entities performing different functions, just as cybersecurity should no longer be viewed as just a technical issue. It's now a business imperative. To support that new paradigm, an agile, comprehensive learning program can give developers the skills they need to ensure secure code early in the SDLC, whether created by themselves, generated by their fast-working AI assistants or acquired from open-source repositories and third parties.

A platform that provides a detailed framework for implementing an effective, ongoing learning program will help bring organizations into cybersecurity's new era. When supported by a robust educational platform, developers can become security-aware and better able to use AI tools to their maximum advantage without compromising the security of their data, applications or organizations.

How are you managing developer risk?

- **Download our all-new white paper, divulging our findings from over twenty interviews with top security professionals discussing enterprise Secure by Design initiatives.**
- **Learn more about coding with AI, and the learning pathways available for your developers to safely leverage the productivity gains of this technology. Download your FREE AI coding rulesets now!**
- **Book your demo today.**