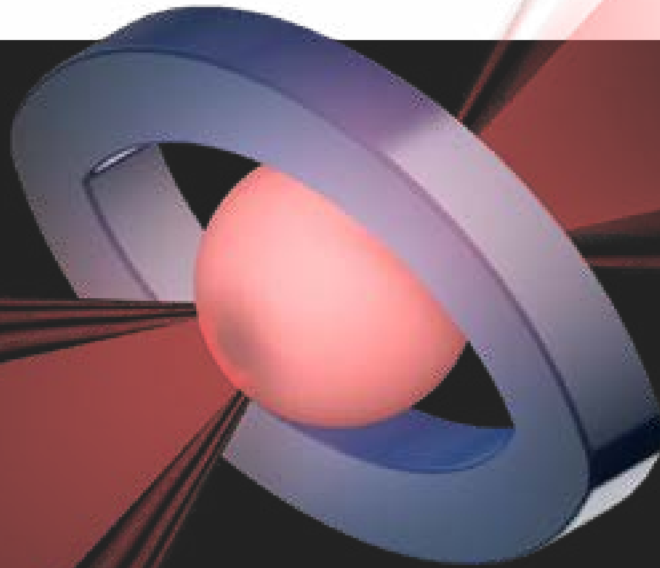
Pentest Tools



The shrinking validation window

What 241 developers told us about vulnerabilities in AI-assisted coding (and how it impacts security work)

A 2026 survey from Pentest-Tools.com

Index

01. Foreword: the gap that AI didn't create, but did widen

02. Methodology and sample

03. Headline findings

04. The state of AI-assisted coding inside development teams

05. The anatomy of the validation gap

06. What practitioners are actually seeing

07. The compliance angle: from "we tested it" to "we can prove it"

08. What successfully adapting teams are doing differently

09. Closing the gap with audit-ready evidence

10. Your concerns, answered

Foreword: the gap that AI didn’t create, but did widen

Validation has been falling behind code deployment for years. Every team that has shipped under deadline pressure knows the feeling:

→ a pull request that should have a longer review
→ a test suite that doesn’t quite cover the new path
→ a scan that runs after the deploy because the deploy couldn’t wait.

AI-assisted coding tools didn’t create that gap. They’ve widened it.

We surveyed **241** developers about **how AI coding tools have changed their work over the past 12 months.** The patterns are consistent across organization sizes and tech stacks:

- Adoption of AI coding tools is near-universal
- The volume of code reaching production has gone up (and keeps increasing)
- Validation processes that catch security issues before they ship have not scaled to match.

This isn’t a story about AI generating vulnerable code (although that happens too). It’s a story about the time between writing code and someone confirming whether it’s safe to deploy.

That window is shrinking.

The vulnerabilities that slip through aren’t always the obvious kind. And the teams closest to the work know it.

Throughout this report, we paired the quantitative survey data with verbatim responses to an open-ended question about how AI-assisted coding has changed the vulnerabilities developers see. The qualitative answers tell a more layered story than the percentages alone: fewer obvious bugs, vulnerabilities that compound across changes rather than appearing in one, and a recurring concern about review fatigue when code arrives faster than anyone can read carefully.

A note for the security teams reading

This report is written with you specifically in mind. The validation gap the data describes is the same one you've been closing manually, maybe without enough recognition for the work involved, so use this dataset accordingly.

The numbers give you something to point to in the next budget conversation, the next prioritization meeting, or the next time someone asks why offensive security work matters when developers are "already reviewing the code." A 9% pace-with-development number isn't an indictment of your team, it's evidence the validation discipline you bring is a structural need, and the volume coming at you is orders of magnitude larger than it was twelve months ago.

Offensive security teams are the function that keeps the validation gap from becoming an incident.

This report is a working document for making that case, externally and internally. The numbers and quotes are here to help you locate your own organization in the data, identify where your validation processes are most exposed to the shift, and decide what to do about it.

Methodology and sample

We ran the survey in March 2026 with **241** respondents across the United States, the United Kingdom, and continental Europe. Participants were screened for active development work and were asked to confirm they use AI-assisted coding tools as part of their job. We only included participants with relevant, confirmed experience in this dataset.

Primary coding languages used

Python
17.9%

Java
14.8%

TypeScript
13.7%

JavaScript
13.3%

C#
12.1%

C and C++
9%

Smaller cohorts work primarily in PHP, Ruby, Go, Rust, and Kotlin.

Survey participants

124 respondents
48.8%
Large enterprise
250+ employees

47 respondents
18.4%
Medium-sized enterprise
50 to 249 employees

44 respondents
16.8%
Small enterprise
10 to 49 employees

27 respondents
10.5%
Micro enterprise
1 to 9 employees

What we asked

The survey covered eight closed-ended questions about adoption of AI coding tools, frequency of use, time pressure, code review practices, post-deployment findings, and how well vulnerability testing keeps up with development speed.

It closed with one open-ended question asking how AI-assisted code has changed the types or frequency of vulnerabilities respondents see in their deployments.

What this report includes

The report walks through the headline findings, then digs into adoption, the validation gap, and the qualitative responses about how the nature of vulnerabilities has changed. We close with a section on what teams who report better outcomes appear to be doing differently, and what that implies for compliance teams who keep being asked the same question: **can we prove this is safe to ship?**

We won't pretend the survey settles the debate about AI-assisted coding. It doesn't. What it does do is locate the seam where most security work currently fails - **between code reaching production and someone confirming whether a finding is exploitable** - and show how AI is putting more pressure on that seam.

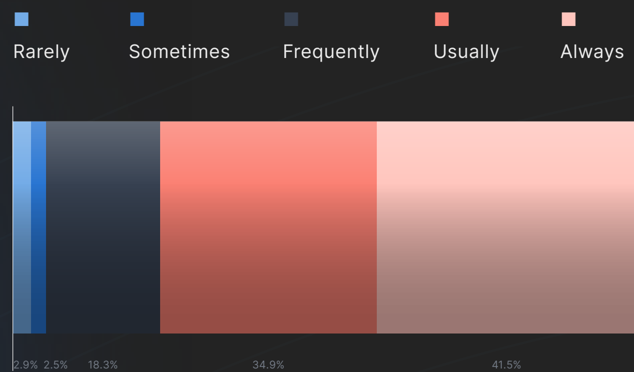
Headline findings

Five findings stood out.
Each one connects to the next.

AI-assisted coding is now infrastructure, not experiment.

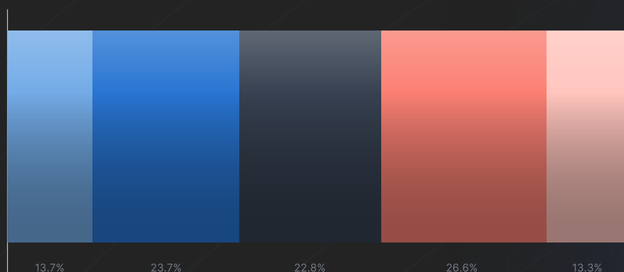
76.4% of respondents use AI coding tools "always" (41.5%) or "usually" (34.9%) in their development work. Only 5.4% use them rarely or sometimes. Adoption isn't trending - it's already happened.

How frequently do you use AI-assisted coding tools (e.g., Claude Code, GitHub, Copilot, Cursor) in your development work?



To what extent do you agree with the following statement: "I feel increased pressure to deliver more code, than before AI tools were introduced."

Rarely Sometimes Frequently Usually Always



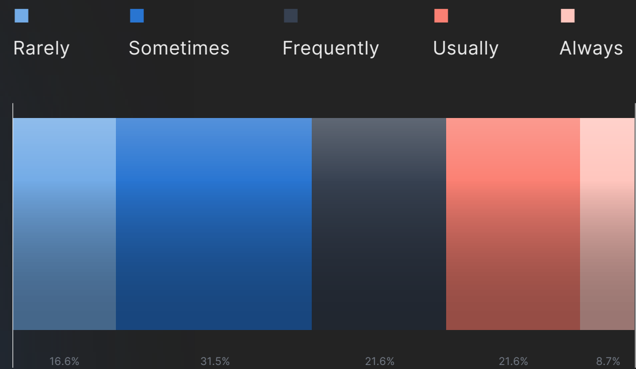
Most organizations are actively pushing this.

82.2% of respondents work at organizations that enforce, strongly encourage, or informally encourage AI tool use. The "allowed but not encouraged" cohort is 8.3%. Cautious encouragement sits at 9.5%. There is no meaningful resistance cohort in this sample.

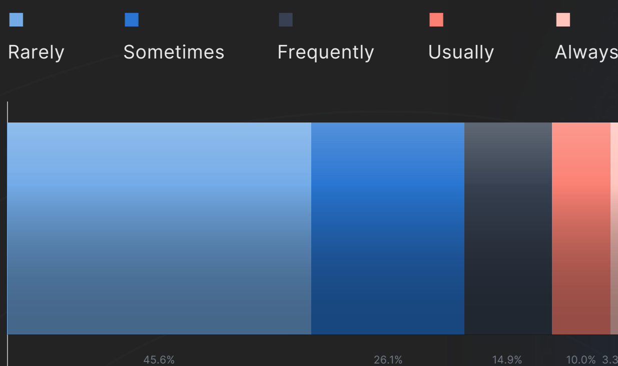
Roughly ⅓ developers say they don't have enough time to review AI-generated code thoroughly.

30.3% disagree or strongly disagree with the statement "I have sufficient time to thoroughly review AI-generated code before deployment." 48.1% say they do have enough time. The remaining 21.6% are neutral. Reviews still happen. But not always to the depth required.

To what extent do you agree with the following statement: "I have sufficient time to thoroughly review AI-generated code before deployment."



When using AI-generated code in development, how often do you examine it for potential vulnerabilities before integrating it into your application?



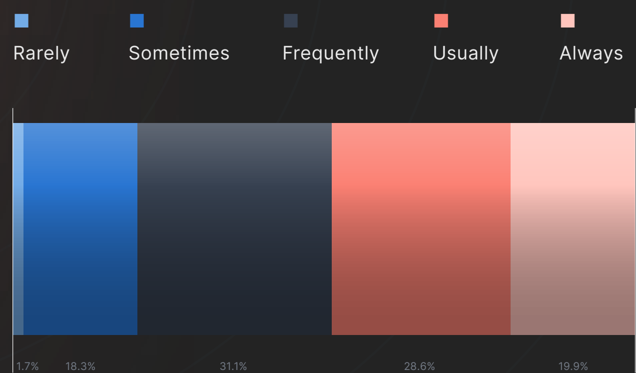
About half of practitioners see vulnerabilities surface in AI-assisted code after deployment.

20% see post-deployment vulnerabilities in AI-assisted code "always" or "often." Another 31.1% see this "sometimes." Combined: 51.1% of respondents experience this regularly. Just under one in five (19.9%) say it never happens to them.

Vulnerability testing rarely keeps full pace with development.

Only 8.7% of respondents say testing keeps pace completely. 41.5% say it keeps pace most of the time. 35.7% say it sometimes falls behind, and 9.5% say it frequently does. The validation window is real, and most practitioners feel it.

In the past 12 months, how frequently have vulnerabilities been discovered in AI generated code (or code snippets) you helped develop, after it was released or deployed?



These five findings work together. AI coding tools are now infrastructure. The organization wants more output from them. The time available to review thoroughly hasn't grown. Vulnerabilities surface regularly after deployment. Testing rarely keeps up.

> "Fewer basic implementation mistakes, but more copied patterns with weak auth checks, unsafe input handling, insecure defaults, and risky dependencies. It's shifted vulnerabilities from obvious bugs to harder-to-spot review failures."

—Survey respondent

That's not an AI problem. It's an operational one.

AI just made it harder to ignore.

The state of AI-assisted coding in development teams

To understand the validation gap, you have to understand how integrated AI tools have become, and how that integration changes the inputs security teams have to work with.

Adoption is near-universal

76%
always or usually use AI for coding

The combined "always" and "usually" cohort represents 76.4% of respondents. Add "frequently" (18.3%) and you reach 94.7%. The "rarely" and "sometimes" cohort, combined, is 5.4%.

This isn't a surprise to anyone who has been watching the developer tooling space, but it's worth stating plainly: AI-assisted coding has moved through the experimental phase and into daily practice for the overwhelming majority of working developers.

Organizations are actively pushing it

82%
work in orgs that encourage AI use in coding

The picture on organizational stance reinforces this. 7.1% of respondents work at organizations that enforce AI tool use. 50.2% are at organizations that strongly encourage it. Another 24.9% report informal encouragement. Together, that's 82.2% of respondents working in environments where the cultural and operational signal is "use these tools."

The cohort that is "allowed, but not encouraged" is 8.3%, and the "cautiously encouraged" cohort is 9.5%. Outright restriction wasn't represented in the sample.

This matters because it shifts the framing. The pressure to use AI tools isn't coming from individual developers cutting corners. It's coming from the organization, often in the form of explicit policy or strong cultural expectation. It's an organizational problem, not an individual one.

Asked whether they feel increased pressure to deliver more code than before AI tools were introduced, 37.4% of respondents agreed or strongly agreed. 39.9% disagreed or strongly disagreed. The rest were neutral.

What that pressure looks like in practice

The split is interesting. It suggests that the pressure isn't uniform. Some teams have absorbed AI tooling without ratcheting up output expectations. Others have explicitly tied AI adoption to higher delivery targets.

The qualitative responses, which we cover later in this report, suggest that where the pressure exists, it's where validation practices most often break down.

One respondent put it directly:

"Overall I think it has increased the frequency of vulnerabilities that appear in my deployments. Because I am expected to deliver more code now with AI, there are times where I get exhausted from reviewing so much AI generated code and let some code through that causes bugs after deployment."

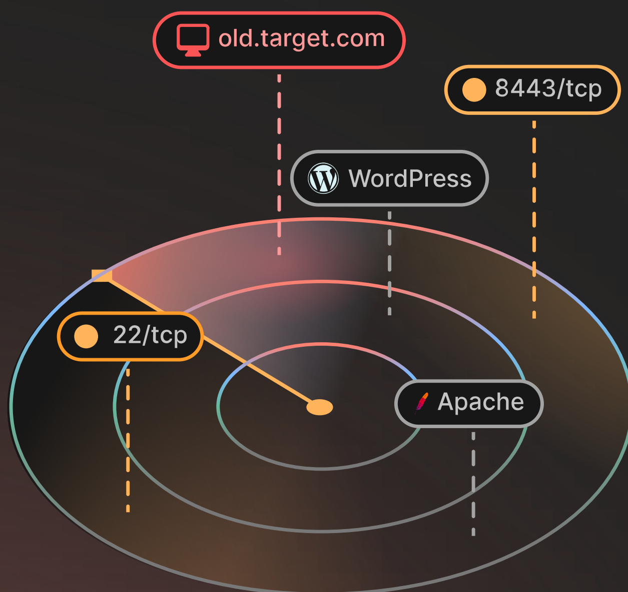
That's the operational reality behind the percentages. Not "AI made the code worse." Not "developers got lazy."

Just: more code arrived faster than the reviewer could carefully read.

More code means a larger attack surface

The implication is structural. When developers ship more code, they ship more endpoints, more integrations, more configuration paths, more dependencies. Each one is something a security team eventually has to reason about, whether for a routine assessment, an audit, an incident response, or a board-level risk conversation.

This isn't new. It's the same shift that microservices and CI/CD created a decade ago, accelerated. What's different now is the pace of generation has outrun the tooling and processes that catch issues before they reach production. The next section examines exactly where that breakdown happens.



The anatomy of the validation gap

We use the word “gap” deliberately, mainly because validation hasn’t stopped - it’s just operating on a delay relative to development. The survey isolates four places where that delay shows up.

The pre-deploy review gap

71%
always examine AI-generated code for vulnerabilities

71.7% of respondents say they always (45.6%) or often (26.1%) examine AI-generated code for vulnerabilities before integrating it. That’s a strong number: practitioners are clearly trying to do the right thing.

But intent isn’t the same as time. When asked whether they have sufficient time to review thoroughly, 30.3% disagreed. Another 21.6% were neutral. So roughly half of respondents report either insufficient time or ambivalence about whether they have enough.

The implication: code reviews happen, but the depth varies. A senior engineer with three open pull requests can give one of them careful attention. The other two get a faster pass. AI-generated code makes that triage worse, because the suggestion looks reasonable. Subtle issues(the kind we cover in section 06) are exactly the issues a fast review will miss.

The vulnerability-discovery gap

31%
sometimes find vulnerabilities in it

Once code ships, the question becomes how often something gets caught after the fact. 20% of respondents say vulnerabilities surface in AI-assisted code post-deployment “always” (1.7%) or “often” (18.3%). 31.1% say it happens sometimes. 28.6% say rarely, and 19.9% say never.

The “never” number is worth noting. One-fifth of the sample report never found post-deployment vulnerabilities in AI-assisted code. There are two ways to read that:

- either those teams have validation processes that catch everything before deployment (possible but unlikely at scale)
- or their post-deployment monitoring isn’t surfacing what’s already there (more likely, given the rest of the data)

The testing-pace gap

8%
believe vulnerability testing keeps up AI-generated code

This is the cleanest signal in the data. Only 8.7% of respondents say vulnerability testing keeps pace with development completely. 41.5% say it keeps pace most of the time. The remaining cohort, a strong 50.5%, describes some flavor of validation arriving late, and accounts for “sometimes falls behind,” “frequently falls behind,” and “not sure.”

Half of respondents are describing organizations where testing is structurally behind. That’s not a tooling problem alone and it’s not an AI problem alone. We would argue AI’s contribution here is to compress the timeline further, because the inputs arrive faster.

The release-before-review gap

34%
agree that dev speed sometimes ships vulnerable code

Asked whether development speed sometimes results in code being released before vulnerabilities are fully explored, 34% agreed or strongly agreed. 41.5% disagreed. The rest were neutral.

A third of the sample acknowledging that code ships before review is complete is a meaningful number. It maps directly to the “review fatigue” theme in open-ended responses. And it suggests the gap isn’t theoretical, but an active operational state for at least some part of every working day.

What “the gap” actually is

Putting it together: AI tools are widely used, organizations push their use, the time to review hasn’t grown, vulnerabilities surface after deployment in roughly half of teams, and testing rarely keeps full pace with development.

The gap isn’t a single broken process. It’s the cumulative effect of speed-up at the front of the pipeline meeting the same human and tooling constraints at the validation end.

Closing it requires either slowing the front (unlikely) or scaling the back which is what the rest of this report focuses on.

What practitioners are actually seeing

The closed-ended questions tell us the shape of the problem. The open-ended responses tell us the texture.

We asked respondents how AI-assisted code has changed the types or frequency of vulnerabilities they see. The responses fell into a few clear themes. Quantitatively, opinions split on whether the number of vulnerabilities had gone up. Qualitatively, almost everyone agreed on the changes in the kind of vulnerabilities they see.

Theme 01

Fewer obvious bugs, more subtle ones

This was the dominant pattern. Practitioners reported that AI tools handle boilerplate cleanly and reduce common syntax mistakes, but introduce a different class of issue.

“Honestly, it’s made things faster, but also a bit sloppier in ways that aren’t always obvious at first. Before AI, most bugs or vulnerabilities came from human mistakes: misunderstanding edge cases, missing validation, that kind of thing. Now, with AI-assisted code, I see fewer ‘basic’ errors, but more subtle and systemic issues slipping through.”

"AI has increased the productivity and reduced basic and common errors, but depending on the AI model capability, it can introduce subtle, harder to detect vulnerabilities that make things tricky some times."

"AI hasn't increased the number of vulnerabilities significantly, but it has made them more subtle and easier to overlook, which makes code review and validation even more important."

"Fewer basic implementation mistakes, but more copied patterns with weak auth checks, unsafe input handling, insecure defaults, and risky dependencies. It's shifted vulnerabilities from obvious bugs to harder-to-spot review failures."

The shared point: the floor came up, and the ceiling came down.

Fewer trivial mistakes reach review. The mistakes that do reach review are harder to spot.

Theme 02

The “gaps“ pattern

Several respondents described the issue as not really a vulnerability in the classical sense, but an inconsistency between the code that exists and the assumptions AI made about existing code.

“They seem to be more about inconsistencies between what really is set up and what the AI thinks it’s set up. They’re more ‘gaps’ than outright vulnerabilities like XSS or sql injection.”

“AI tools have shifted our focus from catching manual syntax errors to auditing for “hallucinated” insecure library versions and logic flaws where the AI suggests technically functional but architecturally insecure configurations.”

“The main shift is from syntax errors to logic flaws. AI is great at boilerplate but often misses our specific architectural context, sometimes suggesting insecure defaults or outdated patterns.”

Traditional scanners struggle with this. Static analysis catches problems in the code itself - bad patterns, unsafe functions. It doesn’t catch problems that emerge from how the code fits together: a route handler that skips an authentication check the rest of the codebase implicitly relies on, or a configuration that’s reasonable on its own but conflicts with the rest of the system. Each piece is fine. The combination isn’t.

Theme 03

Stacking, not standalone

Another sub-pattern: vulnerabilities aren't always individual issues. They compound across multiple changes.

"The types of vulnerabilities are much more subtle and more of stacking problems than individual problems. In other words we don't realize that there is a problem until a few PRs later when our code changes don't scale properly."

"AI-assisted code tends to reduce simple bugs but increase subtle logic and security misconfigurations, and it often spreads the same vulnerability patterns across multiple parts of a system."

This pattern matters for compliance and audit work specifically. A finding that only manifests when multiple changes interact is hard to capture in a single scan. It also doesn't fit cleanly into a CVE list. Auditors who ask for proof of remediation expect a finding to map to a specific code change, but a stacked vulnerability doesn't always have one.

Theme 04

Speed scales risk

A consistent theme is that risk scales with speed. Not because AI generates worse code, but because the throughput is higher.

"I've noticed a shift more than a simple increase or decrease... vulnerabilities show up faster because development is faster. You're producing more code in less time, so unless review processes evolve too, risk scales with speed. It's less about worse code, more about unchecked acceleration."

"It is like two sides of a coin, writing code is now much faster but the reviewing time now takes longer. So it kinda evens out."

"Vulnerabilities tend to arise when deployments are rushed, as this can bypass the human review process that would otherwise catch potential issues. However, such cases occur less frequently than one might expect. This is largely due to the structured workflow that we use in AI-assisted development."

The last quote is worth pulling out. The respondent describes an explicit structured workflow (planning, evaluation, specification, implementation) with security guardrails at each stage. Their experience is that AI-assisted development can be safe, but only with deliberate process. Where the process doesn't exist, the speed-up bypasses validation by default.

Theme 05

Dissent

Not every respondent reported a worsening picture. Roughly a third of free-text responses described stable or improving conditions. Common reasons:

- Strong code review culture that didn't change with AI adoption.
- Using AI primarily for narrow tasks (test scaffolding, helper functions) where the security blast radius is limited.
- Developers use AI tools to find vulnerabilities, not just to write code.

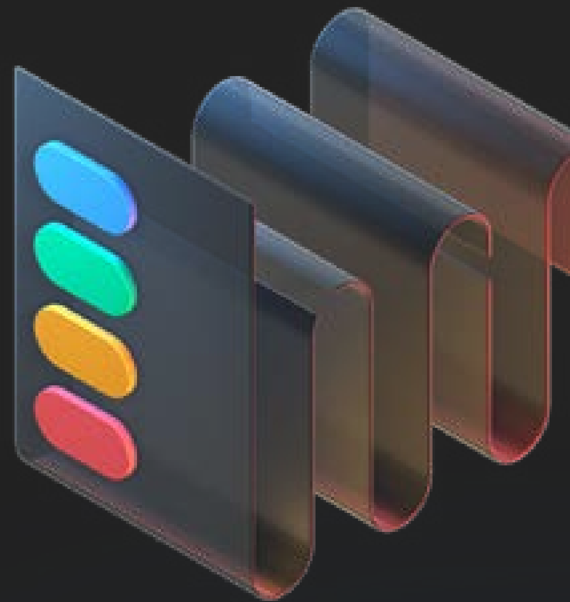
"For most cases it's better because we have automated PR review with Claude which finds additional vulnerabilities so they are less severe and less frequent."

"AI assisted code didn't change the types or frequency of vulnerabilities... we still do manual peer code reviews that catch most of the vulnerabilities before they get deployed."

"Vulnerabilities exists in AI generated code, but AI tools have also helped to identify Vulnerabilities that would have been difficult or time consuming to identify, without using AI. In general using AI tools in conjunction with static and dynamics code analysis tools have reduced the frequency and severity of Vulnerabilities."

This dissent is meaningful. It tells us the validation gap isn't an AI problem inherent to the tools. It's a gap that opens when AI tooling meets review processes that haven't been adapted. Teams that have adapted - typically through more rigorous PR processes, AI-assisted review, or limiting AI to lower-risk code - report better outcomes.

The compliance angle: from “we tested it” to “we can prove it”



The compliance question isn’t separate from the engineering one. It’s downstream of it.

Auditors care about three things, regardless of how the code got written:

- proof a vulnerability existed
- proof it was remediated
- proof the testing was repeatable.

None of those are satisfied by “an AI suggested this and the developer accepted it.” None of them are satisfied by “the pull request had a passing CI build.” And none of them are satisfied by raw scanner output.

The survey didn’t directly ask about compliance maturity but the implications are visible across multiple questions.

The “we tested it” claim is now harder to defend

When code ships before validation completes (and 34% of respondents acknowledged this happens at least sometimes) the documentation trail is weaker.

A pull request that was approved without thorough review doesn’t produce evidence that survives audit.

A vulnerability fixed quickly without a retest doesn’t produce before-and-after artifacts.

A scan that ran after deployment doesn’t prove the code was secure before users got to it.

These aren’t AI-specific failures. But AI accelerates the conditions that produce them.

What auditors actually accept

Compliance frameworks don't reward speed. They reward consistency, reproducibility, and proof. SOC 2, ISO 27001, PCI DSS, DORA, and HIPAA all expect to see:

- Evidence that a vulnerability existed at a specific point in time.
- Evidence of who validated it and how.
- Evidence that remediation occurred.
- Evidence that the fix held - typically a retest with comparable artifacts.

A scan output that flags "potentially vulnerable Apache version" doesn't satisfy any of those criteria on its own. A finding that includes a request and response showing exploitability, a screenshot of the scanner's interpretation, a tracked remediation, and a retest result does.

The implication for teams using AI-assisted coding heavily: the validation step has to produce more evidence than it did before, not less. If the codebase changes faster, the audit trail has to keep up, which means evidence capture has to be a normal part of testing, not a separate audit-prep activity.

How this connects to the survey findings

We can see this concretely in the responses. Teams that reported stable or improving conditions weren't "lucky". They described structured workflows, rigorous reviews, or layered automation. The teams that reported worsening conditions described a process in which speed and pressure outpaced everything.

The compliance distinction doesn't show up in the headline numbers, but it shapes the consequences. A team with a 30% post-deployment vulnerability rate and strong evidence capture is in a different position from a team with the same rate and no evidence trail. The first can defend their work. The second can't.

What successfully adapting teams are doing differently

This survey provides enough signal to identify what tells apart the teams reporting better outcomes from those reporting worse ones. We don't claim this is a controlled comparison, the sample isn't designed for that. But the qualitative responses describe a few practices that repeat across the "things are stable or improving" cohort.

They treat AI-generated code as untrusted by default

Several respondents described workflows where AI suggestions are explicitly treated like third-party code: reviewed with the same skepticism a developer would apply to a copied-in dependency or a pull request from outside the team.

> "The clients we work with are mature enough to see that AI does not bring only benefits and does not fully trust generated code. I personally view it as a powerful search engine and advisor. I rarely let it generate code for me since most of the tasks are pretty complicated."

This framing matters. It changes the review question from "does this code work?" to "do I understand what this code does, and can I defend it?"

They've moved validation closer to the merge

Validation that runs after deployment is too late. Validation that runs at the merge point (e.g. automated scans gated to the pull request, scheduled retests on changed paths, vulnerability diffing across versions) catches issues while there's still time to address them without an incident.

This was implicit in many of the responses describing strong outcomes. The teams doing well had validation at the merge boundary, not after it.

They capture evidence routinely

The teams least worried about audit aren't the ones who do less testing. They're the ones whose testing produces evidence as a routine output: request and response data, screenshots, retest artifacts, scan timestamps. Audit prep stops being a separate workstream when the evidence is already there.

They use AI for review, not just generation

Some respondents described using AI tools to review AI-generated code: a layer of automated PR analysis that surfaces likely issues before a human reviewer sees them.

> "For most cases it's better because we have automated PR review with Claude which finds additional vulnerabilities so they are less severe and less frequent."

AI reviewing AI has obvious limits, so this isn't a complete answer - but it's a sensible use of the same tooling that created the volume problem. Automated first-pass review can partially partially absorb the volume of generated code a, freeing human reviewers to focus on the issues most likely to matter in the organization's context.

They limit AI to lower-risk surface area

Several responses described limiting AI's role to areas with a smaller security blast radius (test scaffolding, helper functions, internal tooling) while keeping high-stakes code (auth flows, payment paths, data access) under stricter human authorship.

This is a defensible default for security-sensitive teams. Not every part of a codebase carries the same risk profile, and not every part needs the same level of careful authorship.

Closing the gap with audit-ready evidence

The validation gap that the survey describes isn't going to close from the development side alone. The pace of code generation will keep increasing. The pressure to ship will keep increasing. What can change is the validation side.

This is where **Pentest-Tools.com** fits, and we'll be direct about it. AI-assisted coding doesn't just produce more code. It produces more deployed attack surface, faster than anyone can validate it - that's the layer we work on. We won't claim it's a complete answer, but it's part of one.

What we keep working on is closing the time between an application reaching production and someone confirming whether it has an exploitable weakness. We test running systems (deployed applications, exposed services, networks, and the dependencies they bring in) because that's the layer attackers reach, and the layer where validation evidence actually comes from.

Validation built into the scanning workflow, not bolted on after

Most scanners stop at detection. They tell you something might be exposed. The validation step - confirming whether the exposure is exploitable, under conditions an attacker would actually use - is left to the security team to perform manually, often days or weeks after the scan ran.

This is the gap **Adversarial Exposure Validation (AEV)** addresses. AEV is the offensive security layer that turns "we detected something" into "we confirmed attackers can exploit it and how - here's the evidence." We keep improving **Pentest-Tools.com** around that distinction.

Case in point, Sniper Auto-Exploiter closes the validation step automatically for high-impact CVEs, returning request and response data, exploit traces, and supporting artifacts that hold up to audit review. The output isn't a "potentially vulnerable Apache version." It's a confirmed exploitation path with the evidence attached.

Less noise, faster review

Building on our focus on accuracy, the Machine Learning classifier we built into our Website Vulnerability Scanner cuts fuzzing false positives by up to 50%. For teams already absorbing more deployed code than they can review, fewer false-positive findings means more time spent on issues that actually matter.

Looking at the network layer, benchmark testing across 167 vulnerable environments ranked our Network Vulnerability Scanner first for overall and remote detection accuracy. Additionally, the Password Auditor identified valid credentials in 84% of test cases (compared to 15% for Hydra, the most common open-source alternative) meaning that "weak credential" findings come with proof of exploitability rather than a probabilistic flag.

Continuous validation, not one-off scans

Compliance work rarely fails because of missing data. It fails because the data is fragmented, outdated, or scattered across tools.

Vulnerability monitoring runs scheduled scans, captures diffs between runs, and routes validated findings into Vanta, Jira, or wherever the team already manages remediation. Evidence is captured as a normal part of testing, including before-and-after artifacts on retest, so the audit trail keeps up with the codebase.

For teams using AI-assisted coding heavily, that continuous loop matters more, not less. The validation window keeps shrinking. **Evidence capture has to keep up, and offensive security validation, run continuously, is what produces evidence that survives both an attacker and an auditor.**

Sniper Auto-Exploiter

Website Vulnerability Scanner

Network Vulnerability Scanner

Password Auditor

Machine Learning classifier

Vulnerability monitoring

AI-enhanced offensive security testing capabilities

What this means in practice

1. A pull request lands with AI-generated changes.
2. A CI/CD action triggers a scan run against the affected paths.
3. Findings come back with exploit confirmation and evidence attached.
4. The remediation gets routed to development teams via Jira.
5. The retest produces before-and-after artifacts that map to the original finding.
6. Compliance evidence is generated as a by-product of the validation work, not as a separate audit-prep activity.

That's the pattern.

While this doesn't solve every gap the survey describes, it offers a safety net: when something slips past a code review - which the data shows happens regularly - there's a structured process focused on catching it before an attacker does, and a reliable trail of evidence to defend the work afterward.

Your concerns, answered

Some of the questions we hear most often from teams thinking through their AI-coding validation strategy.

"We already use a static analysis tool. Doesn't that cover this?"

Static analysis reads source code and catches syntactic patterns. It runs before deployment, against the code as written. The qualitative responses in the survey underline that the most subtle AI-introduced issues don't follow syntactic patterns, instead they show up when the code runs, when systems interact, when an attacker reaches the deployed application. Static analysis is necessary, not just sufficient. The validation that comes from exercising a running system the way an attacker would is a different layer of evidence, and it's the layer the survey data points to as most exposed.

"We have code review. Isn't that enough?"

Code review is essential. It also has limits: 30% of developers say they don't have enough time to review thoroughly, and 34% acknowledge code ships before review completes at least sometimes. A validation layer behind review catches what it misses, not as a critique of reviewers, but as recognition that pressure has consequences.

"How do we prove our AI-assisted code is safe?"

You don't prove it's safe. You prove it's been validated. You prove that detected exposures were confirmed or dismissed with evidence, that fixes were retested, and that the audit trail is intact. The first claim is impossible to make, honestly. The second is what auditors actually accept.

“We’re worried about adding another tool to our stack.”

Reasonable concern. The starting point for most teams isn’t “replace what you have” but “validate the highest-value findings from what you already use.” Pentest-Tools.com integrates with Vanta, Jira, and more. Use the validation layer as a complement to coverage, not a replacement.

“Our AI use is well-controlled. We don’t think we have this problem.”

Possibly. The survey’s “things are stable” cohort is real. The differentiator across responses wasn’t AI usage volume - it was the quality of the surrounding workflow. If your team treats AI suggestions as untrusted, validates at the merge boundary, captures evidence routinely, and limits AI to lower-risk parts of the codebase, you may genuinely be in the clear. If any of those four practices is missing, the survey suggests you’re more exposed than you might assume.

“Won’t validation slow us down?”

The opposite, usually. Validated findings remove the back-and-forth between security and engineering about whether something is real. They let remediation start immediately. They give audit teams what they need without a separate prep cycle. The cost is the validation step itself, which is automated for the high-impact cases. The benefit is the time not spent debating whether a finding deserves attention.

“What if our auditors aren’t asking for this kind of evidence yet?”

They will. The trajectory is clear across SOC 2, ISO 27001, DORA, and the other frameworks that govern most regulated software work. Building evidence capture into the validation step now is cheaper than retrofitting it under audit pressure later.

Working to close the gap is a choice

The validation gap isn't a problem AI created. It's a problem AI made impossible to ignore.

The teams in this survey reporting stable or improving conditions weren't lucky. They had structured workflows, rigorous review practices, and validation processes that produced evidence as a routine output. The teams reporting worsening conditions had the same AI tools, but without the surrounding discipline.

This is a choice - and it matters. It's not about whether to use AI-assisted coding; that choice has effectively been made. It's about what kind of validation infrastructure to build behind it.

The survey can't tell you what to do for your specific environment. It can tell you that 241 working practitioners across the United States, the United Kingdom, and continental Europe are seeing the same pattern: more code, less time to review, subtle issues that surface late, and testing that doesn't quite keep pace.

If that pattern matches your team, the question isn't whether to close the gap. It's how. We'd suggest starting where the evidence is weakest, which is usually at the boundary between detected findings and confirmed ones, and working outward from there.

AI-enhanced offensive security testing **for teams who need real proof**

Europe, Romania, Bucharest
48 Bvd. Iancu de Hunedoara
support@pentest-tools.com
pentest-tools.com

Join our community of ethical hackers!
LinkedIn | Youtube | Reddit