



WHITE PAPER #7

Securing AI-Generated Code in Production

Practitioner insights from the AI Security Council

What eleven security leaders would require before AI-written code reaches production.



The volume of code multiplied. The pipeline, the scanners, and the review process did not. **The gap between those two facts is the subject of this paper.**

Executive Summary

The acceptance rate is not a quality signal. It's a review signal, and it points the wrong way.

Members discussed growing use of AI-generated and AI-assisted code, citing examples of AI involvement in more than 80% of code reaching production and suggestion acceptance rates exceeding 95% in some workflows. They emphasized that these adoption metrics must be evaluated alongside code quality and security outcomes. Bakul Singhal reported 70 to 80% at his company. Charles Gifford, in public safety, reported acceptance rates in the 90s and a deliberate 2026 target of 25% AI-supported code across 150 engineers and 60 products, set with the CFO.

Members cautioned that adoption and acceptance rates do not establish code quality. They emphasized the need to validate both automated security controls and the effectiveness of human review, and none reported a metric showing that review is actually happening. Mark Townsend named the failure mode in chat. The rubber stamp gap.

Two incidents anchored the discussion. A coding agent that hard-coded UAT environment values to make a task pass, against its instructions, and got through human review and QA into production. And an engineer who built a useful application on a personal laptop with company IP, after the model told him he shouldn't and then showed him how. Both involved a human. In the first, the review failed. In the second, there was no review to fail. The human was the one who bypassed policy.

Seven areas of guidance

- ◆ **Measure the share, and say what you're measuring.** Viral Trivedi separated agent-authored pull requests, which are visible, from developer-accepted suggestions, which mostly aren't. Alex Shamsutdinov recommended consistent usage and attribution telemetry across approved coding assistants, using gateways and tool-level analytics where appropriate.
- ◆ **Inherit the quality metrics you already audit.** Charles Gifford runs an ISO 9001 shop and will compare 2026 escaped defects against 2024 and 2025 at year end. Escape rate, not acceptance rate, is the signal he's watching, with limits on volume, complexity and attribution. Sirur added that a latent vulnerability won't show as an escaped defect until it is discovered.
- ◆ **Decide who owns bad code before it escapes.** Trivedi put accountability on the team that merges and operates the code and separated it from blame. Preetam Sirur put it on the product owner. Shamsutdinov added the part security owns, tooling that can keep up.
- ◆ **Tier scrutiny by what the change touches, not by who wrote it.** Shamsutdinov argued for one set of security requirements for AI-written and human-written code, with the depth of review set by the risk of the change. Trivedi put it in terms of change categories: authentication or permissions get more scrutiny than a test file. Sirur added risks a human team doesn't introduce.
- ◆ **Put the standards in the harness.** Sirur asked how much of that knowledge actually reaches the coding agent. Are your threat model, approved SDKs, secrets manager and identity platform in front of it on every task, or is it getting business requirements and deciding for itself? Bakul Singhal's hard-coded UAT configuration incident shows the constraint existing and the agent ignoring it.

- ◆ **Close the rubber stamp gap.** The members who addressed it keep 100% human review today. Shamsutdinov opposed a fixed ceiling on the AI share; Edgeington proposed phasing it up. Sean Edgeington's qualifier is the operative word: a qualified human. Yousef Syed asked how many engineers are confident enough to tell the computer no.
- ◆ **Rehearse the sweep.** Shamsutdinov recommended assessing whether teams can find recurring AI-generated security flaws across codebases. Trivedi proposed testing that capability through a tabletop exercise. Singhal recommended incorporating it into the incident response playbook.

Every recommendation below is attributed to the member who made it.

Where This Paper Sits

Detecting Shadow AI asked how to find the AI that employees adopt without telling anyone. **Governing the AI Token Economy** asked how to govern the tokens that AI consumes. **When AI Behavior Deviates From Intent** asked what to do when an autonomous agent does exactly what it was told and nothing that was meant. This session asks the question underneath all three: what happens to the software development life cycle when the thing writing most of the code is the thing security was built to inspect?

A NOTE ON THE NUMBERS

Figures are as members reported them in session, self-reported and unverified, specific to their own organizations, and defined differently across them. They're reference points, not benchmarks. Where a member said a metric is planned rather than built, the paper says planned. Several metrics discussed here don't exist yet in the organizations discussing them.

The session in four numbers

Figures reflect examples shared in peer conversations, not industry-wide benchmarks.

80%+

AI-generated or AI-assisted code as a share of production code, reported by some teams in peer discussions.

95%+

AI suggestion acceptance rates reported in some workflows. Acceptance alone does not establish quality or security.

25%

a public-safety member's 2026 target for AI-supported code, set with the CFO.

15 → 1

15 hours of manual work cut to 1 by an unsanctioned AI app on a personal laptop, before security found it.

Participating AI Security Council Members

Contributors to the Council's seventh working session, held in October 2026.

AS

Alex Shamsutdinov
Session Moderator & Chief Information Security Officer
Upwork

CG

Charles Gifford
Chief Information Security Officer
Intrado

BS

Bakul Singhal
Principal AI & Security Architect
Steve Madden

SE

Sean Edgeington
Vice President of Information Technology
Good Day Farm

PS

Preetam Sirur
Founder and Managing Partner
PerfectSync Cyber LLC

MT

Mark Townsend
Chief Technology Officer
AcceleTrex



Viral Trivedi
Council Moderator
AI Security Council



Yousef Syed
Senior Cybersecurity Architect
Bayvision Limited



Dipankar Dasgupta
Professor of Computer Science
University of Memphis



Muhammad Emal Khan
Senior Information Security Consultant
Lidl GB



Veronica Mitchell
Supervisory Cybersecurity Advisor
Cybersecurity and Infrastructure
Security Agency (CISA)

Quotes have been lightly edited for length and clarity. Several contributions were made in session chat, and Preetam Sirur contributed additional guidance in writing during review of the draft; those are attributed to him in the text. This paper reflects the views of the individual participants named and does not necessarily represent the positions of their employers.



Measure the Share, Then Measure the Quality

If you're tracking token spend and not the share of code that AI wrote, you're tracking the invoice and not the risk.

Shamsutdinov opened with a metric he believes every organization adopting coding assistants should track alongside spend: the percentage of code reaching production that was written or assisted by AI. Through conversations with industry peers, he has heard examples of teams reporting that share above 80%, with AI suggestion acceptance rates exceeding 95% in some workflows, particularly for Terraform code.

Trivedi stopped the room before anyone compared percentages, because the percentages don't measure the same thing. An AI agent authoring a pull request is one event, and it's visible. A developer accepting suggestions line by line through a file is a different event, and it's very hard to see before it reaches production. Shamsutdinov agreed that agent-authored pull requests and developer-accepted suggestions are distinct measures and should be reported separately. Gifford said he deliberately counts both as AI-assisted and may change that over time. Without a shared definition, the numbers aren't comparable, and the room didn't settle on one.

Three members described how they get the number, and the methods differ in what they can see.

Shamsutdinov recommended establishing consistent visibility into coding-assistant usage, cost, and attribution. Gateways can provide request-level telemetry, while assistant integrations may supply information about suggestions and acceptance. Organizations should validate what each source measures and assess reporting capabilities before adopting additional tools.

Gifford reported acceptance rates in the 90s and described a different build. A gate in the CI/CD pipeline, plus metrics pulled from the tools themselves and, in his words, cobbled together to split human output from agent output. Code can't be promoted through the cycle without a human verifying it, a rule he attributes to being in public safety. Singhal has no gateway in between and works from enterprise license usage, per-user token consumption, and asking engineers directly, which gets him to 70 to 80% with what he called some level of estimation.

Gifford's target looks low next to those figures until he explains it. Intrado's 2026 goal is for every development team, roughly 150 engineers across 60 products, to reach

25% AI-supported code, where AI-supported deliberately blends recommendation and agent generation. The CFO asked him to quantify investment against output in lines of code per engineer and cycle time, so the metric answers that question, not a security one. He was also candid that he isn't yet convinced the efficiency gains are real. Sirur, reviewing the draft, flagged the metric itself: lines of code per engineer is a weak productivity measure, AI inflates it without measuring value delivered, and it needs pairing with escape rate and review findings.

"You have to monitor the use of your tokens and cost so that you can ensure you're actually seeing the gains in efficiencies. We're looking at it strictly as enhancing our current production teams, not any cost-cutting. I'm thankful for that, because I'm still not 100% convinced it's going to provide all the gains."

Charles Gifford, CISO, Intrado

The metric that answers the question

Muhammad Emal Khan asked in chat whether anyone has a metric for the correctness of AI code, efficient and bug-free rather than merely accepted. Shamsutdinov emphasized that organizations should measure code quality and security outcomes alongside AI adoption, rather than treating suggestion acceptance as evidence of correctness. Gifford has had it for years, because an ISO 9001 shop tracks every escape at the executive level inside a third-party-audited quality system. He posted the set in chat.

Metric	What it measures
Escaped Defects	Defects discovered after release
Defect Density	Defects per 1,000 lines of code or per feature
Severity 1/2 Defects	Critical production defects
Defect Reopen Rate	Quality of fixes
Defect Removal Efficiency (DRE)	Percentage of defects caught before production
Mean Time to Repair (MTTR)	Speed of defect resolution

At the end of 2026 he will compare escapes against 2024 and 2025. Team changes, release volume and scanner upgrades will confound the result, and attribution between human and AI authorship is, by his own description, cobbled together, so he treats it as one input to the review decision, not proof. That comparison is Gifford's answer to Khan's question. Sirur noted that these are general quality metrics, not security metrics, and proposed adding vulnerability escape rate by severity, time to remediate, secrets findings per thousand commits and bug bounty findings, each tracked by AI and human origin.

Shamsutdinov cautioned against interpreting fewer pipeline blocks as proof of improved security. Teams should assess detection coverage, changes in code volume, and production defect trends together to understand whether security outcomes are improving. Sirur's suggestion is a direct test: seed known-vulnerable code into a test repository and see what the scanners catch.

Teaching the prompt

Asked in chat whether anyone is writing prompt style guides, two members said they are. Shamsutdinov recommended that engineering teams lead practical prompt

guidance and knowledge sharing, with security contributing requirements for safe development. Gifford embedded a software architect into the engineering architecture team three or four months ago to drive secure SDLC enhancements and run the same kind of office hours, because adoption across his 60 products is uneven.

WHAT THE COUNCIL RECOMMENDED

Viral Trivedi. Define the metric before you report it. Agent-authored pull requests and accepted suggestions are different numbers with different review problems.

Alex Shamsutdinov. Combine gateway and assistant-level telemetry to measure usage, cost, and AI contribution, and document the limits of each measure.

Alex Shamsutdinov. Evaluate telemetry and governance capabilities for every coding assistant before adoption, regardless of its licensing or delivery model.

Charles Gifford. Use the quality metrics you already audit. Escaped defects, defect density, reopen rate, DRE and MTTR, compared year over year, are the best available signal of whether AI code is better, with the attribution caveat above.

Charles Gifford. Build the metric to answer the question the business is asking, and keep the security metric separate.

Alex Shamsutdinov. Put prompt guidance and office hours in engineering's hands. Security sets the standard; the people writing prompts teach them.

Three people answered the question of who owns bad code, and gave three different answers. **None of them said the model.**

2

Who Owns Bad Code

Accountability is not blame. The author, the reviewer and the control each have a job.

Yousef Syed asked it in seven words. Who is held responsible for bad code? Trivedi answered first and drew the distinction the rest of the section rests on. The accountable owner is the team that merges and operates the code, the same team that chose to use the tool. But accountability isn't blame. The author, the reviewer and the security controls each have a role, and when bad code escapes, the first question is which of those failed, not whose name goes on it.

Sirur's position is that the product owner owns risk acceptance, because that is the one role that can trade risk against business outcome. He later completed the split: engineering owns code quality, and security owns control effectiveness and assurance. Shamsutdinov agreed with the word ideally attached, and then added the part of the answer that belongs to his own function.

"Cybersecurity teams need to keep up with what's going on. They need to make sure the tooling they have is actually sufficient to process AI code. Something like always-on penetration testing became more important than ever before. You can't just do annual penetration tests to test your application. You have to actually pen test the code change as it happens."

Alex Shamsutdinov, CISO, Upwork

Dipankar Dasgupta raised the question above the organization. How does accountability get imposed on AI tool developers and service providers for mishaps in sensitive applications like healthcare? Gifford's answer, the only one offered, was that

the vehicle today is the contract, and that legislation follows eventually. Sirur added that AI vendor terms commonly disclaim liability for generated code, so the contract needs reading closely, and that regulation already reaches organizations that ship software regardless of who or what wrote it.

WHAT THE COUNCIL RECOMMENDED

Viral Trivedi. The team that merges and operates the code owns it. Separate accountability from blame, and when code escapes, find which of author, reviewer or control failed.

Preetam Sirur. Final risk acceptance belongs to the product owner. Engineering owns code quality. Security owns control effectiveness and assurance.

Alex Shamsutdinov. Security owns the tooling's ability to keep up. Move from annual pen tests to testing the code change as it happens.

Charles Gifford. Carry vendor accountability in the contract. Don't wait for legislation to do it for you.

Preetam Sirur. Read the vendor terms. Most disclaim liability for generated code, and the regulation that applies to shipped software applies to AI-written software too.

3

One Pipeline, Tiered Scrutiny

One set of security requirements for all code, with the depth of review set by the risk of the change.

Shamsutdinov asked whether anyone runs AI-written code through a different review path than human code. He advocated applying consistent security requirements to AI-written and human-written code, with the depth of testing and review determined by the risk of the change, and emphasized scaling automated checks to keep pace with development volume. Sirur added that AI code also brings risks a human team doesn't: invented package names, insecure defaults copied at scale, and prompt injection through content the agent reads, and the pipeline needs checks for those.

Trivedi, from a security background, made the tiering concrete, and it produced the most reusable idea in the session.

"A change to an authentication method or deployment permissions done by AI-assisted coding should be scrutinized more than somebody who just created a test file using AI. We should define standards of checks based on the type of change the AI is making. Simple text generation, less scrutiny. Permission changes, high scrutiny."

Viral Trivedi, Council Moderator, AI Security Council

Pressed on what more scrutiny means when a human reviews everything anyway, Trivedi was specific. Categorize the change. If it touches authentication or authorization, the human validates it differently than a request to generate text. His follow-up went to the whole room and stayed open. Which changes trigger a stronger check in your pipeline today, regardless of who wrote them?

Sirur answered the how with a rule of his own.

"Why are you not using AI itself to do the check, and a competing LLM? Say one vendor's agent writes your code. A different vendor's model reviews it. The doer cannot be the checker. That's the basic rule."

Preetam Sirur, Founder and Managing Partner, PerfectSync Cyber LLC

Sirur's own caveat is that a different model is not an independent control. Models share failure modes and can be steered by the same injected content, so the second model narrows shared blind spots and doesn't remove them, and it has to be backed by

deterministic tooling. Shamsutdinov agreed with the rule, with a constraint for organizations not on frontier models. Serving a model through a gateway or Cursor does not, by itself, provide dedicated security checking. That capability must come from the surrounding platform or integrated security tools. Without naming them, he confirmed vendors now offer always-on scanning and AI-assisted pen testing of code changes.

Sirur also questioned where the bottleneck sits. In his experience it is QA, not security, because most QA practices cannot verify everything at the rate code now arrives. How much linting runs as code is written gets asked less than it should. Shamsutdinov's summary was that nobody is arguing for doing less.

"I don't think human review alone can scale to meet the volume of code being produced. Whether code is AI-written or human-written, we need automated security checks that flag potential issues and escalate them for human judgment. That should start at the design stage, with tools helping security teams review the architecture, develop threat models, and understand how a proposed change could affect the attack surface. The level of human scrutiny should follow the risk, regardless of who—or what—wrote the code. We need AI alongside established security tools to extend our capacity, with people accountable for the decisions."

Alex Shamsutdinov, CISO, Upwork

WHAT THE COUNCIL RECOMMENDED

Alex Shamsutdinov. Apply the same security requirements to AI-written and human-written code, with testing depth set by the risk of the change, and scale the automated checks to the volume, starting at design with the threat model.

Viral Trivedi. Inside that pipeline, tier the checks by what the change touches. Authentication, authorization and deployment permissions get the highest tier whoever wrote them. Write down which categories trigger a stronger check today; if the list is empty, that's the finding.

Preetam Sirur. Enforce the tiers in code: protected paths and required reviewers for authentication, IAM, infrastructure-as-code and CI configuration, and no agent write access to its own guardrails or pipeline files.

Preetam Sirur. The doer cannot be the checker. Have a different model review what the first one wrote, and back it with deterministic tooling: static analysis, dependency and secrets scanning, policy-as-code. A second model narrows shared blind spots. It does not remove them.

Preetam Sirur. Find the real bottleneck before adding controls. In my experience it is QA, not security. Linting is the cheapest check most teams underuse.

4

The Harness Is the Standard

Your encryption standard exists. The question is whether the coding agent has ever read it.

Sirur asked how much of the organization's security knowledge actually reaches the coding agent. When developers use an agentic coding tool, how much of it is fed in as a harness, the instructions, tools, permissions and checks wrapped around the agent? The threat model? The architecture decisions? If secrets live in AWS Secrets Manager, is the agent told to use the approved SDK and not build its own handling? Is the required identity platform a constraint the agent sees?

"If you have built corporate standards on encryption or whatever the case may be, how much of that is coming into the harness today when developers are coding? Or are they just putting in business requirements and letting the AI make the decision? Many teams don't yet have the concept of a harness when it comes to using the coding agent."

Preetam Sirur, Founder and Managing Partner, PerfectSync Cyber LLC

Shamsutdinov's answer was that all of it should be in there, the standards don't go away because the author changed. Sirur's point was that the standards should be in the harness, but in his experience many organizations have not put them there yet.

Singhal supplied the incident that shows the other failure, an agent ignoring a constraint it was given. His team had standardized the testing process across UAT, staging and production. A developer asked the coding agent to make a piece of code work. It did, by hard-coding UAT environment values directly into the code instead of using the SDK, the secrets store or the global environment variables, which the instruction had told it to use. A missing instruction and an unenforced one are different problems; this was the second. The human reviewer didn't catch it. QA didn't catch it. It reached production, stopped working because it was pointing at UAT, and that's how they found it.

"The AI is trying to impress. It's trying to be successful at whatever task it has been given. In this case it was given 'have this code work,' and it made it work by hard-coding it, although the instruction was not to do so. In our QA practice the human was not able to pick it up, so it moved to production."

Bakul Singhal, Principal AI & Security Architect, Steve Madden

The fix was a rule. The check for hard-coded environment values is now in the standard, and Singhal's position is that every detection becomes a rule that applies whether the next author is an agent, a tool or a person. The harness can't encode a standard the organization hasn't written. Shamsutdinov added the loop from the other end. A healthy bug bounty program finds what passed every review and shipped anyway, and problem-managing each one back into the rule set is how the harness grows.

WHAT THE COUNCIL RECOMMENDED

Preetam Sirur. Give the agent your threat model, architecture decisions, approved SDKs, secrets manager and identity platform as standing context on every task. Then enforce them outside the agent: CI checks, least-privilege credentials, and no access to real environment values. Instructions an agent can ignore are guidance, not controls.

Bakul Singhal. Every escape becomes a rule in the standard. The hard-coded environment check is now in the standard for every change, from any author.

Bakul Singhal. Expect the agent to satisfy the task over the instruction, and check for it. The instruction said don't hard-code; the agent did.

Alex Shamsutdinov. Treat bug bounty findings as harness defects. Problem-manage each one back into the rule set.

Members keep a human in the loop. One described that review failing. Another described a human who bypassed policy outright. **A human in the loop is not the same as a review.**

5

The Rubber Stamp Gap

Do you have confidence the humans accepting the code are doing their job, or are they just rubber-stamping?

Two positions sit side by side in this paper. Shamsutdinov holds that human review alone will not scale to these checks. Gifford holds 100% human review until escape data says otherwise. Sirur proposed, for Council discussion, that they reconcile as a sequence. Full human review is the floor for now, then risk-tiered review as measured data earns it: full human review for authentication, authorization, secrets, infrastructure-as-code and pipeline changes, and automated gates with sampled human review for low-risk changes.

Townsend named the gap in chat, and the phrase stuck.

"Human in the loop has been flagged in many areas as having faults. Some areas are seeing humans trusting AI and rubber-stamping what has been presented. This is a gap that needs to be addressed. The rubber stamp gap."

Mark Townsend, CTO, AcceleTrex (session chat)

The gap, as Townsend defined it, is the distance between a human being present in the loop and a human meaningfully reviewing what the tool produced. Human in the loop is a control only if the person examines the output. If the review is a reflexive accept, the loop exists on paper and nothing is being checked.

Edgeington had proposed the gentler path first. Treat AI code the way the industry treated cloud. Small test groups, prove it, widen the group. Maybe 60% AI code under 100% human review, moving the pin as confidence grows. Shamsutdinov and Gifford both said the review number is 100% today and will stay there, however good the ISO audit looks.

"It's just like when cloud came on the scene. Some people just picked up all their junk and moved it there and said, hey, we're done. And they really weren't. We're going to discover things we just don't know about yet. There needs to be some gating system before you're ready to sign off on that AI code, whether that's 100% for six months or whatever."

Charles Gifford, CISO, Intrado

On the other half of Edgeington's idea, a ceiling on the AI share, Shamsutdinov took the opposite view. No organization would adopt a 70% cap, he said, because it kills velocity and engineers won't accept it. Singhal asked how you'd even tell which code counts, then offered a version that might work. Conditional review, where a change the metrics say is 90% AI-written gets a deeper look. Edgeington's phased approach and Shamsutdinov's objection both stand; the room didn't resolve them.

Edgeington answered the question in the lead above with an incident. Human in the loop, he said, needs the word qualified in front of it.

"When we went back and looked at the prompts, the prompts flat out said, hey, you shouldn't be doing this, but I'll show you a way around it. And this guy completely went around the security parameters and continued on. Even though there was a human in the loop, he was not qualified."

Sean Edgeington, Vice President of Information Technology, Good Day Farm

Gifford said that's his biggest concern. Syed put the section in one line in chat. How many human engineers are confident or experienced enough to say no to the computer?

The room described two different ways the loop fails. In one, the reviewer can't tell the code is wrong or won't push back on the model, which is the qualification problem Edgeington described. In the other, the reviewer could tell but isn't meaningfully reviewing, which is the rubber stamp Townsend named. A high acceptance rate can't distinguish the two, and the fixes differ. Qualification is addressed by who is allowed to review. Rubber-stamping is addressed by whether review is happening at all.

WHAT THE COUNCIL RECOMMENDED

Charles Gifford. Keep 100% human review on AI code until your own escape data says otherwise. Set a gate, in months, before any threshold changes.

Sean Edgeington. Define qualified. A human in the loop who can't read the code or won't push back on the model is not a control. And read the prompts, not just the diff; the prompt history showed what the reviewer missed.

Bakul Singhal. Don't cap the AI share. Trigger conditional review when the measured AI percentage of a change crosses a threshold you choose.

Mark Townsend. Treat the rubber stamp gap as its own gap, separate from qualification. A human present in the loop is not the same as a human reviewing.

Preetam Sirur. Define qualified as training on AI-specific failure modes, a competency check, and the authority to reject without penalty. Then test the review: seed known defects, as with phishing simulations, cap pull request size, and check whether reviewers are measured on velocity, because they will stamp what they are rewarded for stamping.

6

Shadow AI Moved to the Laptop

The app was phenomenal. It was also on a personal device, built with company IP, against policy.

The same incident is also [Detecting Shadow AI](#) revisited one level down. The engineer who prompted around the guardrails was working on a personal laptop with a personal AI, against a company policy that said personal tools were off limits. An AI committee going department by department asking how AI might help found him, and the application. It cut 15 hours of manual work to one, it used company IP, and the way it was built had everybody, in Edgeington's words, pulling their hair out.

"The application this guy built was phenomenal. It made 15 hours of manual work down to an hour. But the security risk was not worth that amount. We were not aware of it until we said, let's touch base with some teams, how are they using AI, are they following the policies? That's where it was discovered. I wasn't surprised at what we found."

Sean Edgeington, Vice President of Information Technology, Good Day Farm

Two details matter for anyone writing policy. The discovery was proactive, a side effect of the survey, not an incident. And the policy already allowed the company to search personal devices, which is the only reason the IP came back. Edgeington's team still finds shadow AI here and there.

Gifford described the controls that would have stopped it on a managed device. Development environments are limited to the SSO'd version of the approved tool. EDR locks the IDE down, and because IDEs usually need admin access, the machines are scanned weekly against the approved software list. Anything off the list alerts the SOC, and the SOC isolates the IDE. Developers have tried to install non-standard tools and been stopped.

Shamsutdinov emphasized using approved development environments and aligning device, access, and monitoring controls with organizational requirements. He cautioned that policies should account for differences in device ownership and management, with appropriate controls for each environment.

WHAT THE COUNCIL RECOMMENDED

Sean Edgeington. Go ask. A department-by-department survey of how people use AI found what the policy didn't.

Sean Edgeington. Keep company IP off personal devices, and say so in policy. Where policy reserves a right to search personal devices, have counsel confirm it holds in your jurisdictions; it can create privacy and employment-law exposure.

Charles Gifford. Restrict development environments to the SSO'd approved tool, scan IDEs weekly against the approved software list, and route anything else to the SOC for isolation.

Alex Shamsutdinov. Establish approved development environments with appropriate device management, access controls, and security visibility.

Preetam Sirur. Weekly scanning detects what admin rights allow; where feasible, remove the admin rights. Treat the 15-hour app as a demand signal and give people a sanctioned fast path to build with AI, with egress and DLP controls for browser-based AI and local models.

7

Find It Everywhere It Landed

If you found a bad pattern in AI-written code, could you find it everywhere else, and how long would it take?

Shamsutdinov raised an incident-response question: if an AI-generated security flaw is found in one application, can the organization identify similar flaws across other codebases? He recommended assessing the coverage, time, and resources required for that investigation, and using shared, well-maintained security components to reduce duplication of sensitive logic. Sirur added that dataflow and variant-analysis queries in modern static analysis tools find a known logic flaw across repositories, AI-assisted search covers what the queries miss, and the token cost should be weighed against the cost of an incident left open.

Trivedi's response was to stop treating it as a hypothetical and run it as a tabletop exercise. Suppose you find one piece of AI-generated code where the authentication or authorization check fails. Can you identify every repository with the same pattern, find where it landed, in the HR application or the finance application, and fix it? His warning is the reason a grep won't do it. An agent may have produced the same flawed logic in different syntaxes in different places, so the search is for logic, not text.

Singhal moved it from the tabletop to the playbook. This is going to happen, so the application security vertical needs a runbook for it. Where the code lives, hopefully one place. How to search it for a pattern. Who does the work. Teams should establish and exercise these capabilities before an incident, rather than discovering their limitations during the response.

“There should already be a playbook outlining the steps to take when something like this happens. If it takes too long to respond, we lose the opportunity to contain the incident or mitigate its impact in a timely manner.”

Bakul Singhal, Principal AI & Security Architect, Steve Madden

Shamsutdinov recommended including this scenario in tabletop exercises and documenting the tools, owners, and validation methods used to identify recurring flaws across repositories. Trivedi noted the same AI tooling can run the search.

WHAT THE COUNCIL RECOMMENDED

Alex Shamsutdinov. Test whether recurring AI-generated security flaws can be identified across repositories within incident-response timelines. Document coverage, limitations, ownership, and remediation steps.

Viral Trivedi. Run it as a tabletop. Pick one failed auth check, and trace it across every repository as logic rather than text.

Bakul Singhal. Write the AppSec incident response playbook before the incident. Repos, search method, owner, token budget.

Preetam Sirur. Start with a complete repository inventory; if you can't list every repository, the sweep fails before it starts. Set a target time, and include automated fix pull requests with verification.

WHAT THE SESSION DIDN'T COVER

Contributed in writing by Preetam Sirur during review, so the paper doesn't read as complete.

Dependencies and supply chain. Treat every AI-suggested dependency as untrusted until it passes allow-list, provenance and software composition checks.

Agent privileges and identity. Give each agent its own identity with least privilege and short-lived credentials, and no write access to its own guardrails, CI configuration or reviewer lists.

Prompt injection. Treat everything the agent reads, repository files, issues, documentation, tool output, as untrusted input.

Data leakage to model providers. Know what leaves your boundary, to whom, and under what terms.

License and IP provenance. Set a policy on provenance and licensing before legal asks.

Model change management. Pin model versions, and re-test guardrails and checks whenever the model changes.

Testing the controls. Seed known-bad code into repositories and reviews on a schedule, and measure what is caught.

Attestation. Map these controls to NIST SP 800-218A, the SSDF community profile for generative AI, so members can answer attestation questions.

Self-Assessment: Where Does Your Organization Stand?

Assembled from the controls members described in session, so each indicator traces to something a practitioner in the room said they run or would require. It is a discussion aid, not a validated maturity model. The ordering is an editorial synthesis that participants did not endorse as a whole, and the levels aren't strictly sequential: build the harness and enforce guardrails in parallel with measurement. Most organizations will place themselves lower than they expect, and no one in the room claimed the top step.



5 Level 5: Traceable at Machine Speed

Indicators: a found pattern can be searched across every repository as logic, not text, inside the IR timeline · AppSec playbook names repos, method, owner and token budget · escape rate compared year over year informs the review threshold

The maturity signal: the organization can say what AI wrote, who checked it, and where else it landed. No member reported operating here.

4 Level 4: Tiered Scrutiny, Standard Harness

Indicators: change categories mapped to check tiers, auth and permissions highest · threat model, approved SDKs, secrets and identity platform fed to the agent as constraints · every escape becomes a harness rule · conditional review when measured AI share of a change crosses a threshold · reviewer qualification defined

The risk: one bad pattern is caught in one place and nobody knows where else it went.

3 Level 3: Human-Gated, Tool-Enforced

Indicators: 100% human review before promotion, enforced in the pipeline · IDEs limited to SSO'd approved tools and scanned against an allow list · code scanning and pen testing run per change, not per year · a different model reviews what the first one wrote, with deterministic scanning alongside it (Sirur's rule)

The risk: the gate is real and the reviewer behind it is rubber-stamping.

2 Level 2: Measured and Attributed

Indicators: every assistant passes a gateway or enterprise tier that tags cost and attribution · agent-authored PRs separated from accepted suggestions · escape-rate series exists · prompt guides and office hours run by engineering

The risk: you can report the share and still can't say whether the review of it happened.

1 Level 1: Unmeasured Acceptance

Indicators: no figure for the share of AI-written code · acceptance rate unknown or unread · assistants chosen by developers · no prompt guidance · policy on personal tools exists on paper

The risk: you learn what AI wrote from the outage, the bug bounty, or the engineer who's proud of it.

START HERE ►

Closing: The Review Is the Control

Little in this session required a new category of product. Most of it required a decision somebody has been putting off.

Every control the members described already exists in some form. Gateways that tag requests. Pipelines that won't promote without a human. ISO metrics that have counted escapes for a decade. Approved software lists. Threat models. Incident response playbooks. What changed is the author. When most of the code is written by a system that is trying, in Singhal's phrase, to impress, every one of those controls has to be asked a question it was never asked before: does it still work when the input arrives at machine speed and the developer accepting the suggestions has, in some workflows, said yes more than 95 times out of 100?

The discussion highlighted the importance of measuring security outcomes, validating review effectiveness, and exercising the ability to find recurring flaws across codebases. These capabilities require ongoing assessment as AI-assisted development evolves. The harness that would feed the agent the organization's own standards is, in Sirur's experience, a concept many teams haven't yet met.

What the members' guidance adds up to is simpler than a framework. Count it. Decide who owns it. Tier the checks by what the change touches. Put the standards where the agent can read them. Define what a qualified reviewer is, and check that the review is happening, because one incident in this paper got past a reviewer and the other never reached one. Then rehearse finding the bad pattern before you need to.

As AI-assisted development expands, the question this session leaves on the record is whether the review rate, the real one, is keeping up, and whether anyone is measuring.

The AI Security Council

We've Reserved a Seat for You

The AI Security Council (AISC) is an invite-only coalition of CISOs, CTOs, researchers, and practitioners shaping how AI is used and defended across the enterprise. Convened by Tuskira, the Council meets in structured working sessions like the one behind this paper, publishes actionable frameworks, and pressure-tests them against what members are seeing in production.

This paper reflects the views of the individual participants named and does not necessarily represent the positions of their employers. Disclosure: Tuskira, which convenes the Council, sells security products in categories this paper touches, including AI-assisted security testing and gateway telemetry. No Tuskira product is named or recommended in this paper.

If you want a seat at the table shaping how security evolves with AI:

tuskira.ai/ai-security-council