CYBERSECURITY READINESS PODCAST SERIES

Episode 117

Surviving AI-Generated Code

with Nir Valtman, Chief Executive Officer and Founder, Arnica

Host	Dr. Dave Chatterjee, Duke University
Guest	Nir Valtman, Chief Executive Officer and Founder, Arnica
Topic Focus	Surviving AI-Generated Code — Secure-by-Default Rules for Coding Agents, Developer–Security Alignment, Workflow Design, Metrics, and Governance-Driven Adoption
Podcast Portal	https://www.cybersecurityreadinesspodcast.com/

Summary

In Episode 117 of the Cybersecurity Readiness Podcast Series, Dr. Dave Chatterjee is joined by Nir Valtman, Chief Executive Officer and Founder of Arnica, an application security platform built to secure AI-generated code as it is written rather than after the fact. After recent episodes on defending the security operations center, AI going up against AI, and agentic AI replacing brittle incident response playbooks, this conversation looks upstream at the code itself. A growing share of the code in organizations is generated by tools such as Copilot, Cursor, and Codex, and much of it lands in the code base without anyone sitting with it the way a developer once did. The question behind the episode, surviving AI-generated code, is whether that code can be trusted and, if not, what organizations should do about it. Valtman, a former CISO at Kabbage ahead of its acquisition by American Express, VP of Product and Data Security at Finastra, and head of application security at NCR, brings a builder's and a breaker's view of the problem.

Valtman reports that Arnica benchmarked the top six coding agents and the top six models and found that, when prompted to develop very simple features, roughly 35 percent of sessions included significant vulnerabilities: ship more code, and you ship more vulnerabilities. His answer is to control how code is written at the point of the prompt, by injecting secure-coding instructions into the rule files (such as agents.md) that steer coding agents, rather than adding friction at code review. According to Arnica's recent research, such rules leave about 98 percent of sessions free of vulnerabilities. He pairs this with private, blameless notifications delivered where developers already work, and reports that 78 percent of the issues Arnica flags never reach the code review stage. Two data points frame the accountability challenge: only 13 percent of recent open source contributions are attributed to a developer, and 82 percent of meaningful findings were authored by developers who are no longer at the company.

The episode's central lesson is that surviving AI-generated code means engineering trust rather than assuming it, and that the work is as much about people and process as about scanning technology. Viewed through Dr. Chatterjee's Commitment–Preparedness–Discipline (CPD) Framework, commitment means leadership that sets the tone and holds a named human accountable for the code, whoever or whatever wrote it; preparedness means an updated threat model, secure-by-default rules, and validated findings routed to the right owner at the right time; and discipline means feedback loops, outcome testing, and metrics that track adoption and developer impact rather than code generated. Valtman's advice to senior leaders is to integrate security into every development life cycle they run, to avoid applying the most expensive AI models to everything, and to remember that within a year organizations will be measured not on how many tokens they spend but on the value they get from them.

Discussion Highlights

From Breaking Systems to Securing Code: Nir Valtman's Path to Arnica

Dr. Chatterjee opens by noting his curiosity about someone who enjoyed breaking into systems as early as 13, and about how that hacker's habit of asking how something could be abused led to founding Arnica. Valtman says it is more a state of mind than a career plan: some people get satisfaction from building software and some from breaking it apart, and he belongs to

the second group. He enjoys understanding how a piece of technology is built and then using that knowledge either to solve a problem or to make it do something it was never expected to do. On the business side, though, he prefers to build rather than break.

At Finastra and the other companies where he worked, he was hired to be on the defending side, but he still liked to build. When a process had no tool to support it, he would write the code himself, whether a custom dashboard or a piece of software that could prevent a particular type of attack. His co-founder and CTO, he jokes, would prefer that he not touch the code. Before the main discussion, Dr. Chatterjee offers a primer: AI coding tools such as Copilot, Cursor, and Codex write code from plain-language instructions; developers merge that code through pull requests; application security teams scan it with tools such as static application security testing (SAST); and coding agents can be steered by rule files such as agents.md. The challenge is that code now arrives faster than anyone can review it.

Securing AI-Generated Code at the Point of the Prompt

Dr. Chatterjee acknowledges that AI will generate more code because it is fast and productive, but that, like any machine-driven work, it can make mistakes. Valtman explains that organizations need to evolve into a new agentic software development life cycle, and that the first piece is controlling how code is written at the point of the prompt. If a developer asks for a feature that lets users upload files to verify their faces, for instance, the agent can be mandated to write code that validates that the uploaded file contains nothing malicious. Catching the same issue later, at the pull request, is possible, but every additional comment at the code review stage slows development velocity.

The closest-to-the-agent approach is therefore to control the system prompt of the agents. It works differently for Copilot, Codex, Claude Code, and others, but injecting instructions on how to build securely prevents a significant number of issues from ever appearing in review. To illustrate the stakes, Valtman shares Arnica's benchmark of the top six coding agents and top six models: in roughly 35 percent of sessions in which they were prompted to build very simple features, the output included significant vulnerabilities. Shipping more code means shipping more vulnerabilities, which is why the rules matter.

> *The fact that you ship more code means you ship more vulnerabilities.*
> **— Nir Valtman**

Developers Versus Defenders: Removing Friction Instead of Adding It

Dr. Chatterjee draws on his research into the long-running conflict between developers and defenders, two groups driven by different incentives. Developers are pushed to ship quickly to reach the market, while defenders are expected to make the product as secure as possible before it goes out. He wonders why the two cannot sit on the same team, organized by project, so that development, testing, and security happen in the same iterative cycle, and he cautions that a "let velocity prevail, we will patch later" mindset runs contrary to what is essential. He sees Arnica's proactive approach to securing AI code as valuable for that reason.

Valtman says the vision of Arnica is building a world in which software development is unimpeded by risk, and that explains every feature the company builds. What he has learned over the years is that identifying vulnerabilities, bugs, or tech debt is the easiest part. The real need is a solution for a process, for a lack of trust, and for differing goals. To embed security in the developer life cycle, a vendor must first win the hearts of the developers and only then the wallets of the security organization. Dr. Chatterjee agrees that every implementation has people, process, and technology dimensions.

Accountability and Attribution: Who Asked the Agent to Write This Code?

Dr. Chatterjee introduces the CPD Framework as a lens for the discussion: commitment is leadership's resource-backed resolve to treat security as a strategic priority; preparedness is the architecture, processes, and people needed for proactive threat detection and remediation; and discipline is the day-to-day enforcement that turns policy into habit. In his words, the difference between documentation and actual practice is what the framework is designed to assess, diagnose, and resolve. Beginning with commitment, he stresses that leaders must set the tone, create the right incentives and culture for both developers and security staff, and make sure someone is accountable. Whether AI generates the code or a human and AI do so together, there has to be a human accountable, who is supported with the necessary tools and processes.

Valtman adds that, looking at recent open source contributions, only 13 percent of the code is attributed back to a developer. The vast majority is authored by agents but not traced to the person who asked for it, which makes attribution a real challenge. The fix is again in the rules: inject a requirement that tells the agent to state who asked it to write the code, and attribution comes by default. Such changes can be driven through workflows that developers adopt, without needing a management mandate. The only thing management needs to see is that, with the rule injected, 98 percent of sessions come out free of security vulnerabilities. That creates alignment with the security team and with developers, who want one thing from security: to be taken off their backs.

> *There has to be a human accountable.*
> **— Dr. Dave Chatterjee**

Why Secure-by-Default Is Not Universal: Custom Workflows, Friction, and Gates

Asked why such workflows are not already in place, Valtman says that in companies with about 100 developers or more, and increasingly as they grow, every engineering team has its own agentic software development life cycle. Teams deploy to production through different steps, some through vibe-coded apps and their own checks, so the bottleneck for a security review differs from team to team. Application security teams typically have no service-level agreement to respond, and so they often delay the outcome. The better position for security is to be involved at the points of least friction and even to have agents fix the code themselves, which requires a workflow fine-grained enough for every engineering team. He acknowledges that this is his plug, and Dr. Chatterjee welcomes it, since the need for checks and balances on mission-critical code is real.

Dr. Chatterjee raises the dilemma of approval workflows when the reviewer is not available. Valtman describes how some customers use hard gates and soft gates. A hard gate means someone from security must review and approve, or the issue must be fixed, before production; deployment is technically blocked. A soft gate lets a lower-risk change, such as a medium-severity finding in a medium-importance application, merge and go to production while the issue is tracked as a backlog item for later. The two can be mixed according to severity and business importance, so the absence of a reviewer does not stall everything.

Preparedness: Threat Models, Validated Findings, and the Right Developer at the Right Time

Dr. Chatterjee summarizes three preparedness best practices and invites Valtman to correct or expand them. First, update the threat model, because AI-generated code fails in ways older scanning tools were not built to see. Second, build detection and validation for those new failure patterns, drawing on the mindset of someone who sees a system and immediately asks how it could be broken, much like an in-house penetration tester. Third, deliver validated, prioritized findings to the right owner rather than a flat list, which depends on clear workflows and clear accountability.

Valtman agrees and adds a data point: 82 percent of the meaningful findings in Arnica's system were authored by developers who are no longer at the company. Generating a code fix is no longer the hard part, since an agent can tackle the entire backlog if enough tokens are thrown at it. The real problem is identifying the best next developer to review the fix and approve the merge. Flagging the last person who committed to a repository does not work as well as finding someone with the right domain expertise, and timing matters too: a developer is more likely to act when already working in that area of the code. The technology can automate the fix, but the process depends on the human who accepts the change.

Discipline: Feedback Loops, Governance Evidence, and Metrics That Matter

Dr. Chatterjee points to the inconsistency among humans, as some have the training and attention to spot a problem and others do not. Drawing on his auditing background, he describes running regular sample tests on critical applications to confirm that results match expectations, so that organizations do not simply trust the process, the people, or the technology but also evaluate the outcome. Valtman says this is what separates the highest-performing teams, which maintain a continuous feedback loop and learn from past behavior. In Arnica's case, if developers consistently dismiss certain findings as false positives, the system raises the pattern with the security team and asks whether it should learn from it for the next scan. One click reduces friction with developers. It is a high upfront investment, he says, but it pays off once enough data accumulates for an agent to use as context.

On explaining AI code governance to regulators and customers, Valtman calls the domain fairly new. Auditors today mainly look for the ability to scan within the development life cycle, typically at code review, which is what appears in SOC 2 reports. Controlling agents to write code that is secure by default, or requiring an agent to attest to the developer behind the code, is more advanced, and there is market education to be done; once you see it, he says, you cannot unsee it. For metrics, Dr. Chatterjee cites best practice that favors time to fix, alert volume, and compliance readiness over features shipped or code generated. Valtman focuses on developer impact and adoption: the percentage of repositories that carry the agentic rules; the version of those rules, which Arnica refreshes every three to six months by benchmarking the latest agents; how many notifications reach developers and how many of those issues still appear in code review; and the path from detection to production. Across roughly 5 million scans a month, 78 percent of flagged issues never reach code review, and only about 1 percent make it to production.

Tool Selection, Fewer Portals, and Blameless Notifications

Dr. Chatterjee notes from his preparation that some tools raise many false alarms, and developers become wary of them. Valtman explains that code quality scanners focus on accuracy: they find fewer things, but what they find is correct. Security

scanners are different, because the goal is to miss nothing, which comes at the price of some false positives. Security vendors should deliberately tune for that trade-off, as Arnica does internally, while keeping the noise low enough that developers still trust the results.

Asked whether juggling multiple coding platforms and portals makes developers' lives harder, Valtman says it is real: developers do not want to do security by default, and a security portal is just another system, like logging into the HR system. Developers like to automate their jobs, so rather than asking them to pull results, Arnica pushes the right finding to the right developer at the right time through Slack, Microsoft Teams, or Google Chat, privately, in what he calls a blameless and shameless way. A message in a channel that calls out a developer by name is unwelcome, whereas a private note that no one else sees, and that warns that code reviewers will see the issue unless it is fixed now, is something a developer is happy to act on.

> *We call it in a kind of blameless and shameless way.*
> **— Nir Valtman**

Final Words for Senior Leaders

Asked to speak directly to senior leadership, Valtman offers two takeaways. First, find a way to integrate security into the development life cycle, or the multiple life cycles an organization runs, because the dividends are large. Second, do not apply the best AI models to everything; that is a waste of time and money. Organizations should find a product that optimizes how they use their AI credits, because within a year they will be measured not on how many tokens they spend but on the value they get from them. The goal is to minimize and optimize token use for the maximum outcome.

> *In a year you will not be measured on how many tokens you spend, you will be measured on what is the value for those tokens.*
> **— Nir Valtman**

The CPD Framework Applied to Surviving AI-Generated Code

Dr. Chatterjee maps his Commitment–Preparedness–Discipline (CPD) Framework onto the practices discussed in the episode. The framework's value here is that it exposes the gap between documenting what should be done and confirming that it is actually being done, which matters when the author of the code may be an agent rather than a person.

CPD PILLAR	APPLICATION TO PASSWORDLESS ADOPTION
COMMITMENT	Leadership sets the tone and backs security as a strategic priority with resources. A human is held accountable for every flaw in code, whether AI-generated or co-authored, and is empowered with the right tools and processes. Incentives and culture are designed to win the hearts of developers and the cooperation of security teams, and leaders close the attribution gap, given that only 13 percent of recent open source contributions are traced to a developer.
PREPAREDNESS	Update the threat model for failure patterns that older scanners were not built to see. Inject secure-by-default rules into the agents' rule files and refresh them every three to six months by benchmarking the latest agents. Build detection and validation for new failure patterns, and route validated, prioritized findings to the developer with the right domain expertise at the right time. Design hard and soft gates by severity and business importance, and fit workflows to each engineering team.
DISCIPLINE	Turn policy into habit. Test outcomes regularly rather than trusting the process, the people, or the technology alone. Run feedback loops that learn from dismissed findings, notify developers privately where they already work, and track adoption and developer impact: rule coverage across repositories, rule version, issues resolved before code review, and time from detection to production.

Actionable Recommendations

RECOMMENDATION	DETAIL
Secure Code at the Point of the Prompt	Inject secure-coding requirements into the rule files and system prompts that steer coding agents, so vulnerabilities are prevented before code reaches review. Arnica's benchmark found significant vulnerabilities in roughly 35 percent of sessions on simple features, and its recent research indicates well-crafted rules can leave about 98 percent of sessions clean.
Assign Human Accountability and Close the Attribution Gap	Name a human accountable for AI-generated code. Require agents to state who asked them to write the code, since only 13 percent of recent open source contributions are attributed to a developer, and support the accountable person with tools and process.
Win the Developers First	Choose approaches that remove friction rather than add it. Developers want security off their backs; align them with the security team through shared workflows instead of top-down mandates.
Tailor Workflows and Use Hard and Soft Gates	Recognize that each engineering team has its own agentic life cycle. Place security where it creates the least friction, let agents fix code where possible, and mix hard gates for critical findings with soft gates for lower-severity issues so a missing reviewer does not stall delivery.
Route Findings to the Right Person at the Right Time	With 82 percent of findings authored by developers who have left, do not default to the last committer. Identify a developer with the right domain expertise and deliver the finding when they are already working in that part of the code.
Update the Threat Model and Test Outcomes	Account for the ways AI-generated code fails that older scanning tools miss, build detection and validation for those patterns with an attacker's mindset, and periodically sample the results on critical applications to confirm they match expectations.
Build Feedback Loops and Choose Tools Carefully	Let the system learn from findings that developers repeatedly dismiss, with the security team approving what it learns. Understand the trade-off between accuracy-focused code quality scanners and security scanners that avoid missing anything, and tune for coverage without drowning developers in false positives.
Notify Privately Where Developers Work	Push findings through Slack, Teams, or Google Chat rather than another portal, and do so privately and without blame, so developers can fix issues before they surface in code review.
Measure Adoption and Developer Impact	Track the share of repositories with agentic rules, the rule version in use, the share of flagged issues resolved before code review, and time from detection to production, alongside time to fix, alert volume, and compliance readiness, rather than features shipped or code generated.
Prepare to Explain Governance and Spend AI Wisely	Be ready to show regulators and customers how AI-generated code is governed, beyond scanning at code review, including secure-by-default controls and attribution. Avoid applying the most expensive models to every task and optimize token use for value.
Apply the CPD Framework to Survive AI-Generated Code	Use Commitment to secure leadership sponsorship and accountability, Preparedness to equip workflows, rules, and people, and Discipline to verify through feedback loops and metrics that practice matches policy.

Time Stamps

TIME	CONTENT
0:02	Introduction — Dr. Chatterjee's background and credentials
0:49	Host framing: can we trust the code AI generates for us?; guest introduction

2:47	Nir Valtman's background and path to founding Arnica
5:00	Primer on AI coding tools, pull requests, SAST, and rule files
6:41	Securing code at the point of the prompt; the 35 percent benchmark finding
8:58	Developers versus defenders; Arnica's vision; winning the hearts of developers
12:38	Dr. Chatterjee introduces the CPD Framework
13:51	Commitment: leadership tone and human accountability
15:02	Attribution: 13 percent of code traced to a developer; rule files and workflows
16:57	Why secure-by-default workflows are not universal: custom agentic life cycles
19:19	Approval bottlenecks; hard gates and soft gates
20:49	Preparedness best practices: threat models, detection, validated findings
23:28	82 percent of findings from departed developers; the right developer at the right time
26:03	Human variation and audit-style outcome testing
27:19	Feedback loops and learning from dismissed findings
29:10	Explaining AI code governance to regulators and customers
30:49	Metrics: adoption, rule versions, and the 78 percent resolved before code review
34:43	False positives and scanner selection
36:41	Avoiding portal sprawl; private, blameless notifications
39:00	Final words for senior leaders
40:40	Special thanks and episode wrap-up

Memorable Quotes

The fact that you ship more code means you ship more vulnerabilities.
— Nir Valtman

Developers want one thing when it comes to security, and that is take them off their backs.
— Nir Valtman

You first need to win the hearts of the developers.
— Nir Valtman

There has to be a human accountable.
— Dr. Dave Chatterjee

It's not about documenting all the things that should be done. It's all about are they really being done?
— Dr. Dave Chatterjee

We call it in a kind of blameless and shameless way.
— Nir Valtman

> *There's market education there, and the moment that you see it, you cannot unsee it anymore.*
>
> **— Nir Valtman**

> *In a year you will not be measured on how many tokens you spend, you will be measured on what is the value for those tokens.*
>
> **— Nir Valtman**

Disclaimer

The information contained in this summary is for general guidance only. The discussants assume no responsibility or liability for any errors or omissions in the content of this podcast. The information is provided on an as-is basis with no guarantee of completeness, accuracy, usefulness, or timeliness. The opinions and recommendations expressed are those of the discussants and not of any organization.

Dr. Dave Chatterjee | dchatte.com | dchatte@gmail.com | linkedin.com/in/dchatte
© 2026 Dave Chatterjee. All intellectual property rights reserved. | cybersecurityreadinesspodcast.com