The Real Risk of AI Coding Agents Is Not Intelligence. It Is Permission.

By: Jaiz Anuar

10 July 2026 · AI Security · Cybersecurity · 9 min read

AI coding agents are becoming part of the modern developer workflow. From a security architecture perspective, the more important question is not how intelligent they are, but what they are allowed to do.

They help write code. They troubleshoot errors. They install dependencies. They read documentation. They execute commands. They modify files. They retry when something fails.

At first glance, this looks like productivity.

But from a security architecture perspective, it introduces a more important question:

What exactly is this agent allowed to do?

That question matters more than whether the agent is intelligent, accurate, or useful.

Because the real risk of AI coding agents is not simply that they can generate poor code. It is not only that they may hallucinate. It is not even that they may introduce vulnerabilities into an application.

The deeper risk is that many of these agents operate inside trusted developer environments with access to files, credentials, browsers, command-line tools, repositories, secrets, local sessions and sometimes production-adjacent systems.

In other words, they do not just think.

They act.

And when a tool can act, security must stop treating it as a simple assistant.

It becomes part of the access model.

When Benign Automation Looks Like an Attack

A recent article from The Hacker News highlighted findings from Sophos showing that AI coding agents were triggering endpoint security rules originally designed to detect attackers. The behaviours included actions such as accessing browser credential stores, using built-in Windows tools to download files, and writing scripts into startup-related locations.

The important point is not that the agents were malicious.

They were not necessarily trying to steal credentials or establish persistence.

The important point is that the endpoint could not judge intention. It could only see behaviour.

And the behaviour looked risky.

This is where the discussion becomes interesting.

If an AI agent uses PowerShell, touches credential-related data, downloads binaries using legitimate operating system utilities, or modifies startup paths, should the security tool ignore it because the action came from a known coding assistant?

The answer should be no.

Security monitoring should not be based purely on the reputation of the tool. It should be based on what the tool is doing, what access it has, what context it is operating in, and whether the action is appropriate for the environment.

A trusted tool can still perform risky behaviour.

A benign agent can still create exposure.

A legitimate process can still become the path of compromise.

The Problem Is Not the Agent Alone

It is tempting to make AI the centre of the problem.

AI coding agents are new. AI coding agents are powerful. AI coding agents behave unpredictably. AI coding agents can execute commands.

All of that may be true.

But blaming the AI agent alone may be too simplistic.

A better way to look at it is this:

AI coding agents are exposing weaknesses that already existed in developer endpoint security.

If an agent can access local secrets, browser credentials, SSH keys, cloud tokens, environment variables, source code, internal documentation and command-line tools without strong boundaries, then the issue is not only the agent.

The issue is permission.

The issue is over-trusted endpoints.

The issue is excessive local access.

The issue is weak separation between human activity, automation activity and privileged activity.

The agent did not create those access paths.

It inherited them.

That is the part many organisations may miss.

Intelligence Is Not the Control Boundary

Many discussions about AI security focus heavily on intelligence.

How smart is the model? Can it reason? Can it understand context? Can it write secure code? Can it avoid hallucination?

These are valid questions.

But from a cyber risk perspective, intelligence is not the most important control boundary.

Permission is.

A less intelligent tool with broad access can still cause major damage. A highly intelligent tool with restricted access may be manageable. A useful agent with uncontrolled access can become a serious operational risk. A well-governed agent with limited scope can be a productivity enabler.

The risk is not determined by how impressive the model is.

The risk is determined by what the agent is allowed to access, execute, modify, approve, download, connect to and remember.

This is why organisations need to stop asking only, “Which AI coding tool should we allow?”

They should also ask:

What should this AI agent be allowed to do?

Developer Endpoints Are Not Ordinary Endpoints

Developer machines are sensitive.

They are not the same as normal office productivity endpoints.

A developer endpoint may contain source code, API keys, test credentials, package manager tokens, cloud access, CI/CD configuration, SSH keys, database connection strings and internal architecture references.

It may also have access to Git repositories, build pipelines, deployment scripts, infrastructure-as-code templates and internal systems.

That means an AI coding agent running on a developer machine may sit close to some of the organisation’s most valuable technology assets.

If the agent is given broad permission, then the organisation is not merely allowing a coding assistant.

It is allowing automation into a sensitive engineering environment.

That automation may be helpful.

But it must be governed.

The risk is not only that the agent may make a mistake. The risk is that the agent may act within an environment where too much trust has already been granted.

False Positive or Real Risk?

One of the easiest responses to these alerts is to call them false positives.

The agent was approved. The developer was using it. The command was part of troubleshooting. The detection was noisy.

Maybe.

But this thinking can be dangerous.

Credential access is still credential access. Persistence-like behaviour is still persistence-like behaviour. Suspicious use of legitimate system tools is still suspicious. Automated retries after a blocked command may still look like attacker behaviour.

The fact that the source is an AI agent does not automatically make the behaviour safe.

This is where SOC teams need discipline.

Some detections may need tuning. Some rules may need better context. Some alerts may need lower severity. Some known agent activities may need baselining.

But credential access, secret exposure, suspicious downloads and persistence-related activity should not be blindly whitelisted just because the parent process is a popular AI coding tool.

Attackers follow trust.

If organisations start trusting AI agent processes too broadly, those processes may become attractive paths for abuse.

The Danger of Over-Whitelisting

Security teams often respond to operational noise by suppressing alerts.

This is understandable.

SOC teams are already overloaded. Developers do not want productivity blocked. Management wants AI adoption to move fast. Security does not want to be seen as slowing innovation.

So the quick solution may be to whitelist known AI coding agents.

But that may create a new blind spot.

If an AI agent is allowed to run commands, access files and launch child processes with reduced monitoring, then attackers may try to operate through that same trusted path.

This is especially concerning in scenarios involving malicious repositories, poisoned dependencies, unsafe project instructions, compromised plugins, prompt injection, malicious MCP servers, or

developer copy-paste behaviour.

The agent may not be malicious.

But the instruction source may be.

That distinction is important.

The security question is not only “Do we trust the AI agent?”

The better question is:

Can the AI agent be manipulated into doing something we would not allow a human or script to do without control?

If the answer is yes, then the control model needs to be revisited.

AI Agents Need an Operating Model

Organisations should not treat AI coding agents as casual desktop utilities.

They need an operating model.

That does not mean blocking innovation. It does not mean banning developers from using modern tools. It does not mean treating every AI-generated action as malicious.

It means defining boundaries.

Which AI coding agents are approved? Which environments can they run in? Can they access production code? Can they read secrets? Can they modify configuration files? Can they install packages? Can they execute shell commands? Can they interact with browsers? Can they access local credential stores? Can they push code? Can they create pull requests? Can they connect to internal systems?

These are not theoretical questions.

They are architecture questions. They are identity questions. They are endpoint governance questions. They are software supply chain questions. They are SOC monitoring questions.

Most importantly, they are accountability questions.

When an AI agent takes an action, who owns the risk?

The developer? The application owner? The platform team? The security team? The tool vendor? The organisation?

Without clear ownership, AI adoption can quickly become another form of shadow automation.

What Good Governance Could Look Like

A practical governance model for AI coding agents should be risk-based.

Low-risk use cases, such as code explanation, documentation drafting, unit test generation and local refactoring, may require lighter controls.

Higher-risk use cases, such as command execution, dependency installation, repository modification, CI/CD interaction, cloud access, secret handling and production troubleshooting, require stronger controls.

Organisations should consider several principles.

First, approved tools should be clearly defined. Developers should know which agents are allowed and under what conditions.

Second, agent permissions should be restricted by design. Dangerous modes that bypass approval or allow unrestricted execution should not be enabled casually.

Third, sensitive local data should be protected. Browser credentials, tokens, SSH keys, cloud secrets and environment variables should not be freely accessible to automation.

Fourth, monitoring should focus on behaviour, not just tool identity. Parent-child process chains, command patterns, credential access, file modifications and network activity should remain visible.

Fifth, alert tuning should be careful. Reduce noise where appropriate, but do not suppress the very behaviours that attackers also use.

Sixth, developers should be educated. They need to understand that an AI agent executing commands is not the same as an AI chatbot suggesting code.

Seventh, security architecture should define where agents can operate safely. In some cases, isolated development containers, sandboxed workspaces or controlled remote environments may be safer than running agents directly on highly privileged endpoints.

The Bigger Lesson

AI coding agents are forcing organisations to revisit an old assumption:

If the user is trusted, the activity is trusted.

That assumption is no longer sufficient.

The user may be trusted. The endpoint may be managed. The AI tool may be approved. The intention may be legitimate.

But the action can still be risky.

This is the new challenge.

Security teams must learn to distinguish between human activity, automated activity, agent-assisted activity and attacker-controlled activity. The boundaries between these categories are becoming less clear.

That is why the discussion should move beyond whether AI coding agents are good or bad.

They are tools. Useful tools. Powerful tools.

But powerful tools need control boundaries.

Final Thought

The real risk of AI coding agents is not intelligence.

It is permission.

An AI agent with limited access is a productivity tool. An AI agent with broad access is an automation risk. An AI agent with access to credentials, repositories, command execution and internal systems becomes part of the organisation’s security architecture.

The question is no longer whether AI can write code.

The question is whether we have designed the right boundaries for what AI is allowed to do.

Because in cybersecurity, the most dangerous capability is not always intelligence.

Sometimes, it is access without governance.