

```

cve:none { }--:=                                sync.c() await=-
risk-- --+===+-=                                pat3.fix(pr)-:=
deploy() fn() =>                                guard() --+====-
</> :: [ok]--=:                                { } sync.c()=-
rngr.scan(repo) if(vuln){patch()} { } // guard() guard() sbom+-
risk-- git push risk-- --+===+ [ok] fn() // sbom level++ deploy()
rngr.scan(repo) pat3.fix(pr) && -> </> sbom => && ++ --+===+ :=
policy.enforce() sbom cve:none // 0x26D07C rngr.scan(repo) sbom+
ver8.triage()---                                level++ deploy()
-> fn() 0x26D07C                                run() {##} =====
=> [ok] --+===+=:                                policy.enforce()
</> 0x26D07C+:+                                npm audit ->-=:+
[ok] 0x26D07C--++                                ::=====-
</> //at //at //                                0x26D07C
--+===+ npm audit                                sync.c()
git push level++                                => {##}-
=> pat3.fix(pr):                                await ::
pat3.fix(pr)---                                //:::-:=
{##} npm audit=-                                { }--=-
{##} // run()--                                { } </>-
&& sbom :: npm audit risk-- === :: npm audit fn() // cve:none {##} {##} fn()--=
cve:none //at cve:none await --+===+ :: ver8.triage() :: level++ :: :: npm audit+
harness(agents) level++ cve:none --+===+ pat3.fix(pr) run() npm audit cve:none #!
sbom --+===+ === 0x26D07C === --+===+ </> </> sync.c() {##} => 0x26D07C && fn():=
cve:none => sync.c() {##} harness(agents) ver8.triage() run() ::
guard() npm audit { } {##} deploy() { } #! && deploy() // fn()--
-> sync.c() if(vuln){patch()} cve:none 0x26D07C //at fn() //===-
//at git push harness(agents) => git push => level++ -> //---===
&& level++ =>--:                                sync.c() //at--:
level++ ++ fn()=-                                :: guard()==+=
#! cve:none { }:-                                level++ </>+---
++ ++ sync.c()=-                                sbom && => :: //
=== => :: { } ::                                sync.c() run()+=
deploy() :: sbom                                guard() ++-----
await run()--+=-                                policy.enforce()
ver8.triage()=-                                pat3.fix(pr)-:=

```

The Applied AI Playbook for AppSec:

Four Levels for Architecting Security into the Agentic SDLC

#{}#
{##}
#{}#

Download Now

Executive summary

In IDC’s July 2026 survey, 79.6% of AppSec leaders said frontier AI code security tools were making them rethink their dedicated AppSec tooling. Meanwhile, their security engineers are already building their own agents, and most of those experiments work. The hard part is turning one engineer’s agent on one repository into a program that runs across every repository and pull request, at a cost someone can defend.

This guide gives AppSec leaders a map for that journey. It describes four levels of AI maturity across two tracks:

**Secure:**
how you govern the code your developers’ agents write.

**Defend:**
how your own team uses AI for security work.

Level	What it looks like	Build or buy
1 Governed generation	Approved coding agents, with security guidance built in as code is written	Mostly buy
2 Agentic security tools	Purpose-built agents find, triage, and fix vulnerabilities; people approve the fixes	Mostly buy
3 Custom security skills	Your team builds skills that encode its own process, with evals for each one	Both
4 Orchestrated security agents	Specialized agents work as a system, each with autonomy it has earned	Both

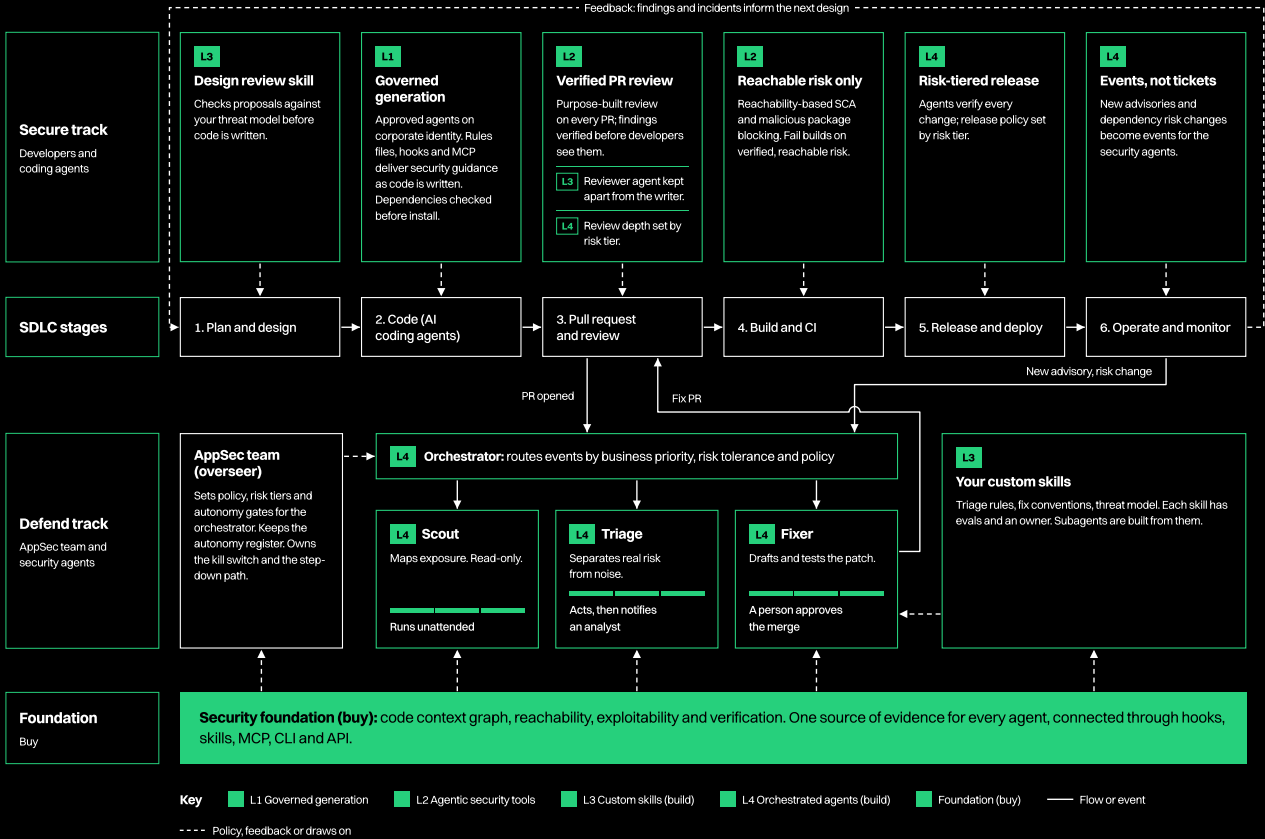
Every level comes with the evidence that proves you’ve reached it, a cost metric, the trap to avoid, and a gate to clear before moving up. The guide also gives you a five-question test for build versus buy. The short answer: build what’s unique to your business, and buy the security foundation underneath it. That matters more than it sounds. Endor Labs research found that agents re-deriving security context use 12x more tokens than agents that can look it up.



Level 4 is the destination, and the route is gated on purpose. Autonomy you haven’t earned is the quickest way to lose the trust you need to keep going.

The AI-powered SDLC: where security works at each level

Developers' coding agents write the code (Secure track). The AppSec team's agents act on events (Defend track). Both run on one security foundation.



Why AppSec needs a map through the agentic forest

In [IDC's July 2026 DevSecOps and Application Security Survey](#), 79.6% of respondents said code security capabilities from frontier AI companies were causing at least a moderate reassessment of their dedicated AppSec tools. Nearly half (47.9%) expected to reduce, consolidate, or replace at least one of those tools. Only 18.5% reported little or no impact.

At the same time, your security engineers are probably already building agentic tools to assist with security tasks (whether they admit it or not): an agent that reviews pull requests, a skill that triages a reachability finding, a script that drafts dependency upgrades overnight. Most of these start as an experiment on one repository, and most of them work (eventually).

The trouble starts when one engineer's experiment has to become the security program. An agent that works for one person, against one repository, on one task is a very different thing from a system that has to work across every repository and pull request in the enterprise, every day, at a financial cost someone has to defend in a budget review. Readers with the right number of battle scars will appreciate the effort it takes to turn a quick scripted tool into something robust.

Scale changes the economics (or tokenomics, if you must). Prototyping burns a few tokens and probably stays within an engineer's quota, so it's effectively invisible. Running agents continuously across large codebases, however, means paying for repeated context gathering, model evaluation, patch verification, and the people who keep it all running. [Industry benchmarks put review and verification at about 59.4% of token spend, with code generation closer to 8%.](#) [Endor Labs research](#) found that agents re-deriving security context (reachability, exploitability, remediation paths) used 12x more tokens than agents that could look that context up using an API.

So build versus buy has become an enterprise architecture question. What should your team build? Which specialized capabilities should your agents call? And what belongs in the harness around the model?

Most AppSec teams don't lack ambition. Nor do they lack 'encouragement' to innovate. They lack a map. This guide lays out four levels of AI maturity across two tracks: how you secure the code your developers' agents write, and how your own team builds with AI. It gives you a way to place your program, what reaching each level requires, and an honest account of what's worth building versus bringing in.

The destination for most is Level 4: a coordinated system of security agents your team builds and runs, guided by your own priorities, policies, and risk tolerance. The route there has a few sets of ‘you must be this tall to ride’ barriers on purpose. Autonomy you haven't earned is the fastest way to lose the trust you need to keep going. No one wants to be the ill-fated sorcerer's apprentice, even if it makes for a good story. The good news is that this isn't completely unmapped territory, we are already helping some of our most advanced customer deploy and operate at level 4.

The model at a glance

The model has two tracks because almost every program we talk to is uneven. A team can be building its own triage skills while developers' coding agents still run on personal accounts. Having only a single score and a single track would keep things simple, but hide this potential division between developers and security team maturity. This design decision sparked some spirited ‘debate’ within the team. Are two tracks is it overly complex? Does this model promote a divide between application security and engineering? Your feedback on the model is both appreciated and will decide who gets to wear the ‘I told you so’ t-shirt at the next project meeting.

The Secure track covers the code your developers' AI agents write, and the agents themselves. The Defend track covers AI doing AppSec work: finding, triaging, and fixing. The names follow NIST's draft Cyber AI Profile (IR 8596), which separates securing AI systems from AI-enabled cyber defense.

Level	Secure track	Defend track	Your team's role	Build or buy
1 Governed generation	Approved agents; security guidance delivered as code is written	Ad hoc AI help; people do the work. Users generate prompts for analysis	Operator	Mostly buy
2 Agentic security tools	Purpose-built review on every PR, verified before developers see it	Bought agents triage and fix; people approve	Approver	Mostly buy
3 Custom security skills	Your policies run as skills; reviewer kept apart from writer	Custom skills call bought security context, each with evals	Builder	Both
4 Orchestrated security agents	Agents verify every change; review set by risk tier	Specialized subagents work as a system; autonomy gated per agent	Overseer	Both



A note on 'buy': Much as a commercial organization like Endor Labs enjoys the exchange of money for products and services, we're going to define 'buy' as 'obtain from a vendor'. Endor Labs makes available several tools that are provided at no direct cost, but do require you to be an Endor Labs customer to be of any use. Other vendors, no doubt, follow a similar model. So buy doesn't necessarily mean spend in every context in this document.

The levels describe capability and ownership. Autonomy is a separate setting you grant per agent and per action type, inside any level. A triage agent that only reads code can run unattended long before an agent that merges patches should. If you already use an autonomy scale, such as the [AWS Agentic AI Security Scoping Matrix](#), the appendix maps it to these levels.



Score each track separately. A gap of more than one level between them is a warning sign. Either developers' agents are moving faster than your controls, or your team is building on ground it doesn't govern.

Activity: Where are you today?

Take a break from reading (we do our best, but this isn't Proust, after all) with a 'fun' activity. Below are eight questions, one per level on each track. Answer yes only if someone on your team could produce the evidence within a working day. Self-reported maturity tends to bring out the optimists in even the most cynical leader, and the gaps it hides are exactly the ones that hurt later.

◆ Secure track

Level	Can you say yes?	Show it with
1	Every coding agent in use runs under a corporate account, and agents get security guidance as they write, through rules files, hooks, or an MCP server	An inventory of agents and MCP servers with owners; the share of AI-authored PRs scanned before merge
2	Every PR, including agent-written ones, gets a purpose-built security review, and findings are verified before developers see them	Precision of findings reaching developers; fix acceptance rate
3	Your own security policies run as skills or hooks inside developers' agents, and the agent that reviews code is separate from the one that writes it	A skills registry with owners; the share of violations caught as code is written
4	Agents verify every change across all repositories, and the human review a change needs is set by its risk tier	Share of changes verified by agents; security escapes tracked per risk tier

◆ Defend track

Level	Can you say yes?	Show it with
1	Your team has a written stance on which AI tools may see which security data	The policy and the approved tool list
2	Purpose-built agentic tools triage or fix findings, and you measure how often they're right	Agent-analyst agreement rate; share of fixes merged without rework
3	At least one skill your team built runs in production on a security task, with an eval suite and a named owner	Eval pass rates; tokens and minutes per task across repositories
4	Specialized agents act on events without a person starting each task, and each agent's autonomy is set and recorded per action type	An autonomy register with promotion history; kill-switch test results



Scoring

Your level on each track is the highest level where you can answer yes to that question and every question below it. A yes at Level 3 with a no at Level 2 puts you at Level 1, with a head start.

Be strict. A "no" is a to-do item you can plan around. An overconfident "yes" becomes a gap you discover during an incident. Optimism makes for good PowerPoints but poor operations.

For a more detailed result, including what to do next on each track, take the Maturity Quiz. It takes about five minutes.

1 Level 1: Governed generation

Most organizations have achieved Level 1, at least on paper. Developers use AI coding agents every day. What decides whether you're really at Level 1 is whether the company can see those agents, control them, and embed security guidance into them as code is written.

What it looks like

On the *Secure* track, developers use approved coding agents through enterprise accounts, so access and data handling follow company policy rather than personal sign-ups. You keep an inventory of the agents, models, MCP servers, and skills in use, with an owner for each. You can't govern what you can't list, and the list is usually longer than anyone expects.

Security guidance reaches agents at generation time. Rules files (CLAUDE.md, Cursor rules, Copilot instructions) carry your secure-coding expectations; the [OpenSSF guide to security-focused AI code assistant instructions](#) is a good starting point. Hooks and MCP servers go further, checking a dependency before the agent installs it or flagging an insecure pattern before it lands in a commit. Every AI-authored pull request is scanned before merge.

This matters because working code and secure code are different bars. As of now, the [Agent Security League leaderboard](#) shows the top agent-and-model combination reaches 87.2% functional correctness but only 37.4% security correctness, a gap that holds across all 29 benchmarked combinations. Passing tests isn't the same as being secure.

87.2%

Functional correctness

The top agent-and-model combination

37.4%

Security correctness

A gap that holds across all 29 benchmarked combinations

On the *Defend* track, Level 1 is individual and informal. Security engineers ask a chat assistant to explain a CVE, draft a detection rule, or summarize a finding. That's fine. The work here is a written stance on which AI tools may see which data: source code, vulnerability details, customer information.

How to measure it

Track	Evidence	Cost metric
Secure	Inventory of agents and MCP servers with owners; share of agent sessions under corporate identity; share of AI-authored PRs scanned before merge	Share of AI-written code covered by security checks
Defend	A written AI data-handling stance and approved tool list	Not measured yet

Build or buy?

Mostly buy

Mostly buy. Enterprise agent plans and the security context delivered through hooks and MCP are products, and they change weekly. Building your own hook layer across three or four coding agents is possible, but you'd be maintaining integrations against tools that ship faster than you can track them.



What you build is *policy*: your rules files, your approved tool list, and the decision about which checks run where.



The paper governance trap

The policy exists, the approved tool list exists, and half the developers are running agents on personal accounts with no hooks installed. At Level 1, what's enforced is what counts. Check identity and coverage numbers, not policy documents.

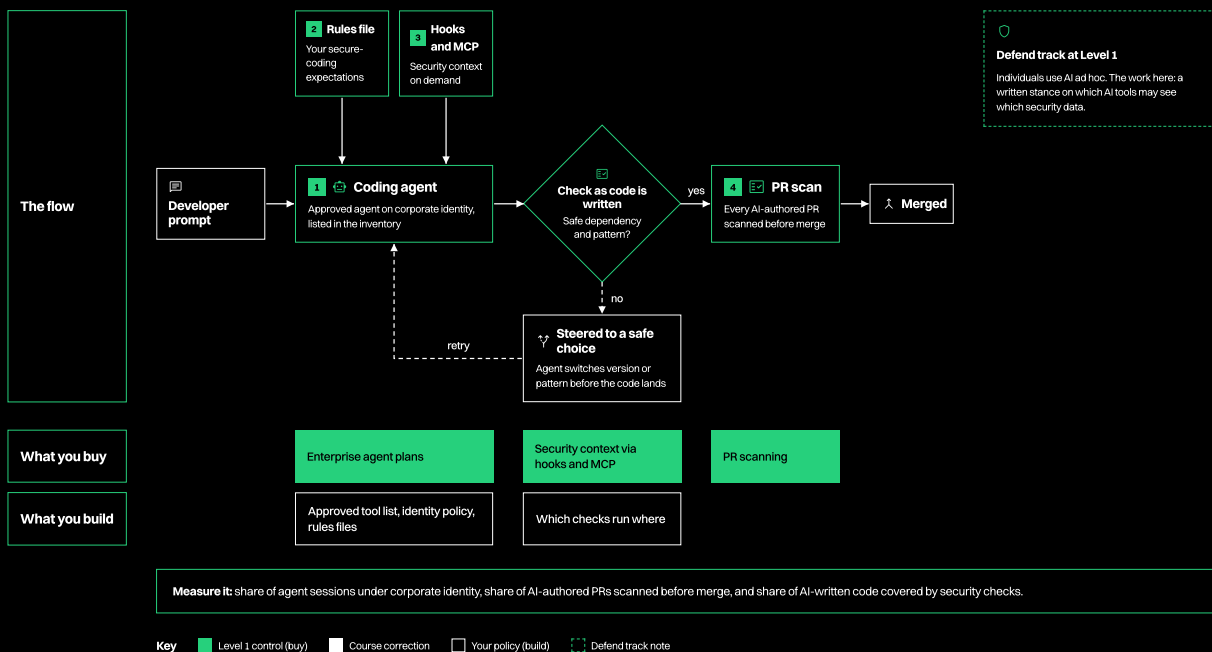


Ready for Level 2 when

- ☐ The agent inventory is complete, and every entry has an owner
- ☐ Corporate identity is enforced for coding agent use
- ☐ AI-authored PRs are scanned before merge
- ☐ You have baseline numbers for backlog size, mean time to remediate (MTTR), and false-positive rate

Level 1: security guidance reaches agents while they write

Approved agents on corporate identity, guided by rules files, hooks and MCP, with every AI-authored PR scanned before merge.



That last item matters more than it looks. Without a baseline, you can't tell whether the tools you bring in at Level 2 are helping.

2 Level 2: Agentic security tools

At Level 2, the team brings in purpose-built agents to find, investigate, triage, and fix vulnerabilities. People still approve what those agents propose. At this stage in your journey, that's still the right call.

What it looks like

On the Secure track, every pull request gets a security review built for the job: static analysis that reasons over data flow rather than matching patterns, software composition analysis (SCA) that checks whether vulnerable functions are actually reachable, and checks on the packages agents pull in, all explained in easy-to-consume language to make human review as easy as possible. The differences from Level 1 scanning is verification, and context. Findings arrive with evidence that the issue is real, reachable, and exploitable, before a developer ever sees them. Context ensures you're not rebuilding an LLM's 'understanding' of a problem each time, and also helps find logic and architecture flaws.

That difference decides whether developers act on findings. Developers who get verified findings fix them. Developers who get noise learn to ignore the tool, and as agents produce more code, the noise only gets louder.

Speed matters too, because review has to fit inside the PR loop at agent volume. In [Endor Labs testing](#), Claude Code Security took 18 to 20 minutes on average to review a PR, against under a minute for Endor Labs. In an eight-project benchmark, Endor Labs AI SAST found 2.6x as many real vulnerabilities as Claude Opus 4.7, including 63 that no other tool detected.

18 to 20 min Claude Code Security Minutes on average to review a PR	<1 min Endor Labs Under a minute to review a PR	2.6x Real vulnerabilities found Including 63 that no other tool detected
--	---	---

On the Defend track, bought agents work the backlog: assessing exploitability, triaging findings, and drafting fixes or dependency upgrades. An analyst approves each fix. Endor Labs has developed a series of open-source agents that tackle common security tasks and can use the context and information provided by the AURI platform to triage and remediate discovered flaws.

How to measure it

Track	Evidence	Cost metric
Secure	Precision of findings reaching developers; fix acceptance rate; MTTR on AI-authored code	Cost per verified finding
Defend	Agent-analyst agreement rate on triage; share of fixes merged without rework	Cost per verified finding



Cost per verified finding is the number to start tracking now: licenses, tokens, and human review time, divided by the findings that turned out to be real. It's the figure you'll compare against when you decide what to build at Level 3.

Build or buy

Mostly buy

Mostly buy. Detection, verification, and fix generation require vulnerability intelligence, language coverage, and program analysis that change faster than an internal team can keep up with. Analysts expect the same pattern: Gartner reportedly predicts that through 2027, at least 40% of organizations will rely on their application security testing vendor as the default source of AI autoremediation. As technology evolves, what we think of as Application Security Testing will transform into Agentic application security testing. And, no this isn't a simple case of AI-washing (although surely instances of that are proliferating) Agentic Security Testing (AST) uses AI harness and model reasoning to identify complex logic vulnerabilities that traditional testing tools miss. Agentic AST (sometimes referred to as "AI SAST") helps cybersecurity leaders identify application exposures and secure homegrown code.



What you build is the plumbing around the tools: routing findings to code owners, merge policy, ticketing integration, and the asset criticality data that tells an agent which services matter most.



The trap: unverified findings at AI speed

Many agentic security options exist in the marketplace right now. Gartner warns of "agent washing" and estimates only about 130 of the thousands of vendors claiming agentic AI are real. Success will rely not on their ability to generate findings, but on generating only valid findings that need addressing. Ask every vendor for the evidence behind a finding, and measure precision yourself before you trust it.

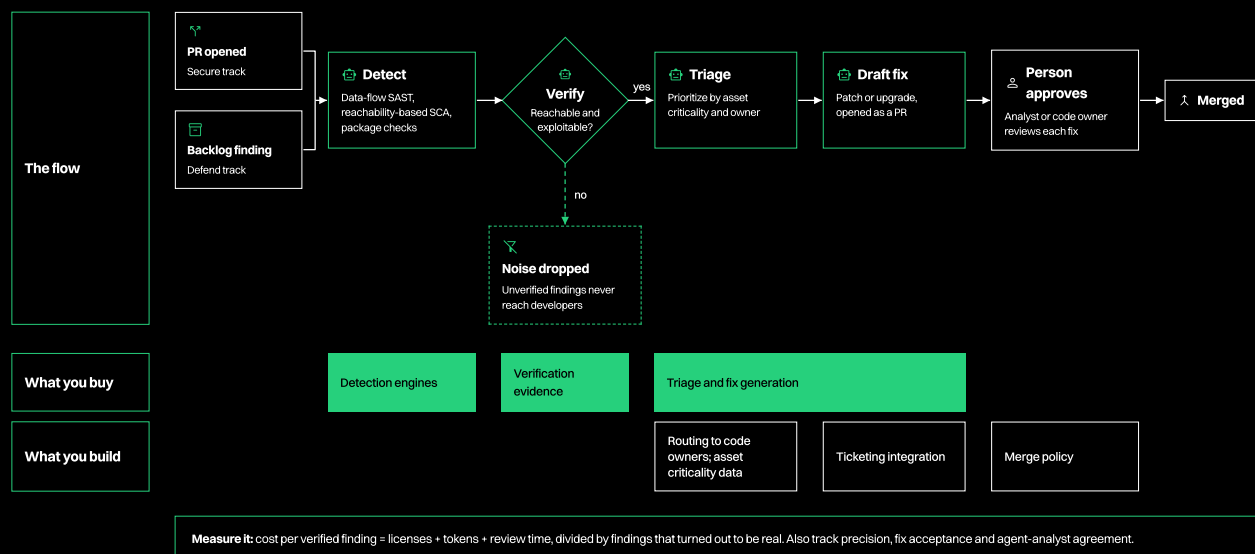


Ready for Level 3 when

- ☐ Finding precision is measured, and your **teams trusts it**
- ☐ Security context (reachability, exploitability, dependency data) is available to your own remediation agents through an API or MCP server
- ☐ Agents you build will get a scoped identity, logged tool calls, and human approval for anything irreversible
- ☐ Dedicated headcount to build and maintain skills (spare time doesn't count)

Level 2: bought agents do the work, people approve it

Purpose-built tools find, verify, triage and fix. Only verified findings reach developers, and a person approves every fix.



Key ■ Bought agentic tool ■ Human approval Your plumbing (build) Filtered out

Ready for Level 3 when precision is trusted, security context is available to your own agents through an API or MCP server, and an engineer has dedicated time to build skills.

3 Level 3: Custom security skills

Level 3 is where your team starts building things. A skill is a small, specific capability that runs inside an agent harness such as Claude Code or Codex and encodes how your organization does one security task. Bought tools know about vulnerabilities. Skills know about you: your triage rules, your fix conventions, your threat model, and which services keep the CFO up at night. Skills can leverage the agents you've built or bought at level 2 to do the heavy lifting, but they provide the additional organizational (or even project) context.

What it looks like

On the Defend track, a triage skill takes a finding, pulls reachability and asset criticality (probably using an agent from level 2), applies your rules, and writes a recommendation an analyst can accept in seconds. A fix skill drafts a patch (again possibly using remediation advice from an agent) that follows your conventions and runs your tests. A design review skill checks a proposal against your threat model before anyone writes code.

On the Secure track, your policies move into developers' harnesses as skills and hooks: your approved authentication patterns, your rules for handling secrets, your dependency standards. One principle carries over from good engineering: keep the agent that reviews code separate from the agent that writes it. As practitioner Bassim Eledath puts it, "don't let the same model grade its own exam."

Every skill needs an eval suite: a set of known cases with expected outcomes, run every time the skill, the prompt, or the underlying model changes. Without evals you can't tell improvement from drift, and you can't make the case for more autonomy at Level 4.

The decision that sets your costs

The most expensive design choice at Level 3 is what a skill has to work out for itself versus what it can look up. A skill that asks the model to determine reachability from scratch has to explore the codebase by brute force: searching files, reading them, and following data flow one step at a time. That's slow, it's expensive, and it produces noisier results. This is where leveraging agentic tools from your security vendor can slash token costs.

Endor Labs ran 34 security prompts across 13 scopes. Agents that could read precomputed security evidence used 6.6 million tokens and 133 minutes. The same model working alone used 79.5 million tokens and 370 minutes. Cost also became predictable: without evidence, token spend swung 4x between projects; with it, the swing narrowed to 1.8x, because a lookup costs the same on 10 files or 10,000.

6.6M

Tokens with precomputed evidence

Agents that could read
precomputed security evidence

79.5M

Tokens, same model working alone

34 security prompts across 13 scopes

133 min

With precomputed evidence

Minutes to complete the same prompts

370 min

Same model working alone

Minutes to complete the same prompts

How to measure it

Track	Evidence	Cost metric
Secure	Skills registry with owners; share of policy violations caught as code is written rather than at review	Tokens per review, and its variance across repositories
Defend	Eval suite per skill with pass rates; tokens and minutes per task across repositories	Tokens per task, and its variance across repositories

Build or buy Both

Both. Build the skills, prompts, evals, and the policy they encode, because nobody else can. Buy the security context and verification those skills call, through an MCP server or API, because re-deriving it on every question is where the 12x comes from.

The [MIT NANDA study of enterprise AI](#) found that pilots stall mainly because of a learning gap: tools that don't retain feedback or adapt to how the organization works. Skills are how you close that gap on top of tools you've bought. It also makes it easier to maintain a core set of agents, with skills providing an exception-and-customization layer for non-standard projects.



The trap: the one-engineer demo

A skill that works well on one repository, on one engineer's laptop, with no evals and no owner. It gets demoed, praised, and quietly abandoned when its author changes teams. Anything you build at this level needs its own small SDLC: version control, tests, an owner, and a place in the on-call rotation.



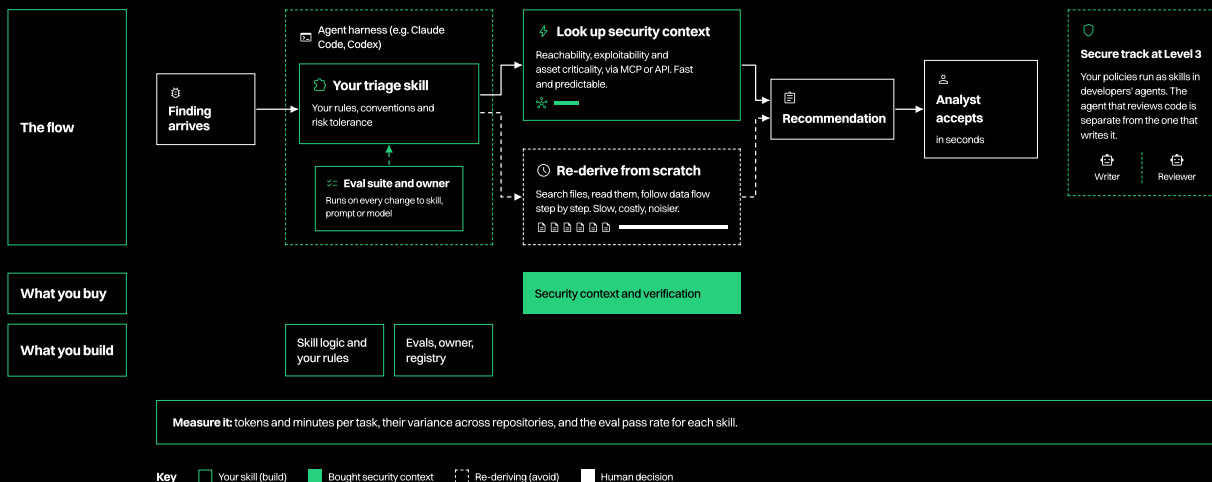
Ready for Level 4 when

- ☐ Each production skill has an eval record and an accuracy threshold it must meet
- ☐ Agreement between agents and analysts is tracked separately for each action type
- ☐ Cost per task is predictable across repositories
- ☐ You have a kill switch, and a way to step an agent's autonomy back down automatically
- ☐ Orchestration has an owner whose job it is

The shift to Level 4 also changes what oversight looks like. Anthropic's [research on Claude Code usage](#) found that experienced users let the agent run without approval more often (over 40% of sessions, up from about 20% for new users) but also interrupt it more. Oversight moves from approving each step to watching and stepping in when it matters. Your skills' eval records are what make that move safe.

Level 3: your skill, bought context

Your team builds skills that encode its own rules. The expensive decision is whether a skill looks security context up or works it out from scratch.



4 Level 4: Orchestrated security agents

At Level 4, specialized agents work as a system. An orchestrator receives events, such as a new pull request, a new advisory, or a change in a dependency's risk, and routes the work to subagents that each do one job well. A scout maps exposure. A triage agent separates real risk from noise. A fixer drafts the patch, runs the tests, and opens the PR. The system follows your business priorities, risk tolerance, and policies, and your team oversees it rather than driving every step.

What it looks like

On the Defend track, nobody starts any task manually. Work arrives as events, and agents act on it within limits you've set. You set those limits separately for each agent and each action type. An agent that only reads code and queries security context can run unattended. An agent that changes code, configuration, or installs tools stays behind human approval until it earns more, matching the caps in the Cloud Security Alliance's [autonomy levels framework](#).

Autonomy is earned through promotion gates: time at the current level, accuracy above a threshold on its evals and in production, and a clean incident record. It's lost automatically after an incident. And no agent can raise its own autonomy.

An autonomy register makes this concrete. The agent names and thresholds below are *illustrative*; set your own from your Level 3 eval data, and choose a functional breakdown that works for you.

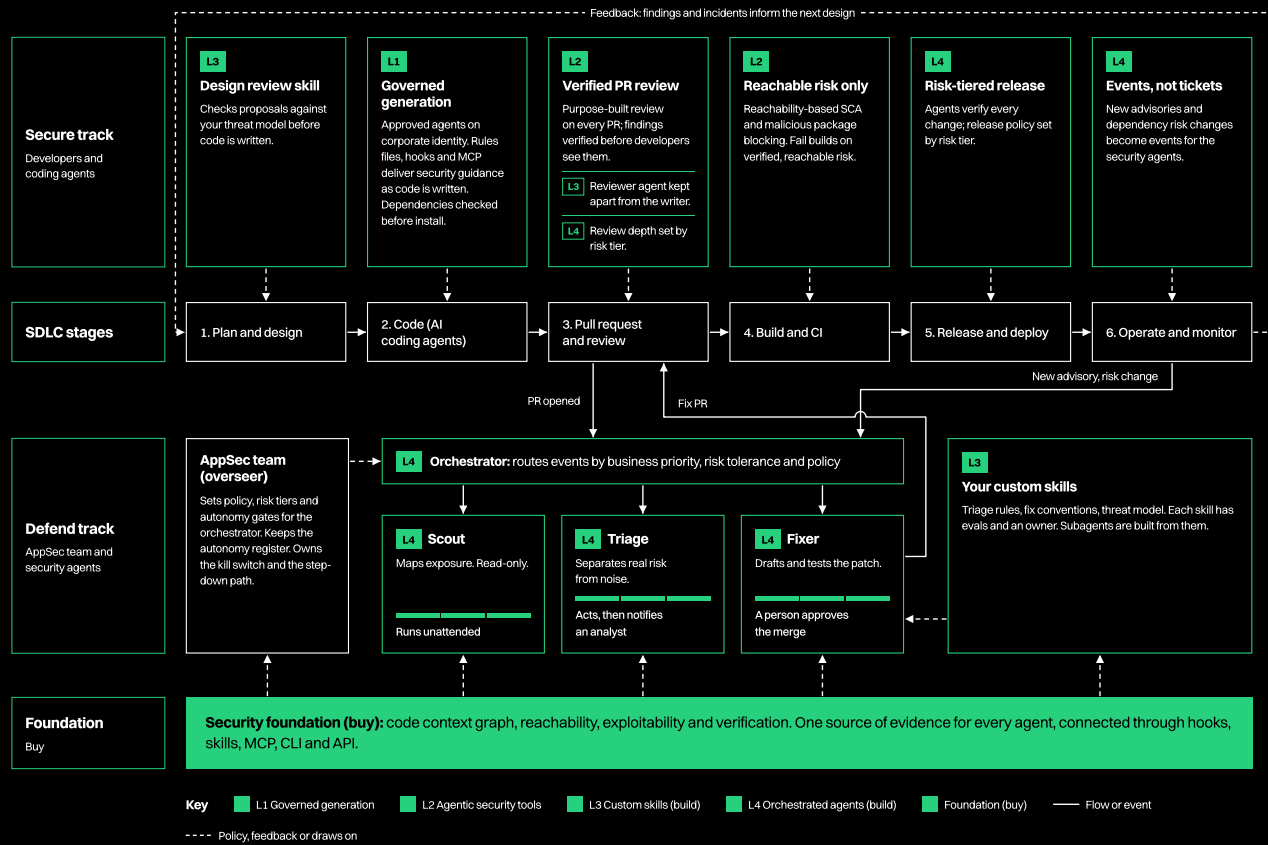
Agent	Action	Current autonomy	To earn the next step
Scout	Read code; query reachability and dependency data	 Runs unattended	Not needed
Triage	Close a finding as unreachable	 Acts, then notifies an analyst	Agreement with analysts above your threshold for 4 straight weeks
Fixer	Open a fix PR	 Acts; a person approves the merge	Merged-without-rework rate above your threshold
Fixer	Merge a patch-level dependency upgrade on low-criticality services	 Acts within policy, logged	Zero rollbacks over the trial period, plus sign-off

On the Secure track, agents verify every agent-written change across all repositories, and the human review a change gets depends on its risk tier. Some engineering teams already run what Dan Shapiro calls a "[dark factory](#)", where nobody reviews AI-written code line by line, and the system has to prove it works instead. If your developers are heading there, security verification has to be part of that proof.

Two engineering requirements come with Level 4. The first is graceful degradation: when something looks wrong, the system automatically drops autonomy rather than failing open, as the [AWS Agentic AI Security Scoping Matrix](#) recommends. The second is protection against what the Five Eyes agencies' agentic AI guidance calls structural risk, where one compromised agent poisons the others. Give each agent its own scoped identity, authenticate messages between agents, and log every agent's reasoning and tool calls.

The AI-powered SDLC: where security works at each level

Developers' coding agents write the code (Secure track). The AppSec team's agents act on events (Defend track). Both run on one security foundation.



How to measure it

Track	Evidence	Cost metric
Secure	Share of changes verified by agents; security escapes tracked per risk tier	Cost per verified change
Defend	Autonomy register with promotion and demotion history; kill-switch and step-down test results	Cost per remediated risk, including the people who maintain the system

Build or buy Both

Both, with a clear line. Build the orchestration, the risk model, the autonomy gates, and the workflows, because those are your business. Buy the deterministic foundation every agent calls, either directly or, more likely, through sub-agents you already have in place, with code context, reachability, and verification that must be identical across thousands of repositories. Maintaining call graphs across dozens of languages is a lot of work that doesn't set your program apart.



Automation complexity is a big risk

An agent mesh nobody can debug, with no evals and no way to step down. Gartner predicts that over 40% of agentic AI projects will be canceled by the end of 2027 because of escalating costs, unclear business value, or inadequate risk controls. We can't promise that some of the security projects won't be in that 40%. Two things that will lessen the pain: 1) Keep an honest track of the automations and their value vs. complexity. Take a flexible approach to what you push to level 4. Remember, it took 17 years from the [start of the Agile+CI/CD movement](#) to get to where we are today, with plenty of lessons along the way. Don't expect not to learn any during this revolution.

When to stay at Level 3

Maybe level 4 is the ultimate goal for your program. But not everything needs to get there. Individual workflows can stop short of it. Rare, high-impact changes, such as authentication logic or cryptography, can keep a human approver indefinitely without holding the program back. Your program is at Level 4 when your high-volume workflows run as a governed system. The rare, high-impact ones can still have a person in the loop.

Build, buy, or both? (Hint: it's probably going to be both)

The short version: build what's unique to your business, and buy the security foundation underneath it. From Level 3 up, almost everything is "both": your skills and orchestration running on context and verification you've bought.

The longer version is five questions. Run any capability through them before you commit engineering time or budget.

1

Is it unique to your business? You can't buy your policies, risk tolerance, fix conventions, and workflows because no vendor knows them. Build these.

2

Does something you already own cover part of it? Extend before you build or buy anything new. Forrester recommends the same control-first approach for agentic AI security purchasing: find the gap, check existing tools for partial coverage, and only then look at new vendors.

3

Does it change faster than you can track? Vulnerability and malware intelligence, language and framework coverage, and model behavior all move weekly. If keeping up would take a dedicated team, buy it.

4

Does it need to give the same answer at every scale? Call graphs, reachability, and dependency resolution should be computed once and looked up, not re-derived by a model on every question. Re-deriving costs 12x the tokens and makes spend swing 4x between projects.

5

Who maintains it a year from now? The token bill is the cheap part. Evals, integrations, identity and audit logging, on-call, and the tool's own SDLC cost more. If nobody can own it, buy it.

Applied to each level, the answers look like this.

Level	Mostly	What you build	What you buy
1 Governed generation	Buy	Rules files, approved tool lists, policy	Enterprise agent plans; security context delivered by hook or MCP
2 Agentic security tools	Buy	Routing, merge policy, integrations, asset criticality data	Agentic review, triage, and fix tools
3 Custom security skills	Both	Skills, prompts, evals, encoded policy	The security context and verification skills call
4 Orchestrated security agents	Both	Orchestration, risk model, autonomy gates, workflows	The deterministic foundation and specialized engines every agent calls

The evidence leans toward buying the foundation. In MIT NANDA's study of enterprise AI initiatives, bought and customized tools reached deployment about twice as often as internal builds (roughly 67% versus 33%). The same study had too little data to measure hybrid approaches at all. That's the gap this model is built for: bought foundations, with skills and orchestration that make them yours.



There's an honest counterpoint. Buying alone leaves you with tools that don't know your environment and don't learn from your decisions. If you stop at Level 2, you'll get the vendor's idea of your risk. Levels 3 and 4 are where your idea takes over.

The trade-offs

Every level up buys you something and costs you something. These are the costs we see teams underestimate most often.

Not every workflow needs Level 4

The program goal is Level 4. That doesn't mean every workflow should get there. Low-volume, high-impact changes may never generate enough data to justify autonomy, and that's fine. Spend your orchestration effort where volume is high, and mistakes are recoverable, such as dependency upgrades and triage of reachable findings.

Technical debt and maintenance

A working prototype is where the cost starts. Every skill and agent you build needs evals, version control, an owner, integration upkeep when the harness or model changes, and a place in someone's on-call rotation. Budget for that before you start, or the one-engineer demo will become the one-engineer outage.

Lock-in

Agentic workflows tend to hardwire prompts, tools, and guardrails to one model or one harness, raising switching costs over time. Keep skills small, keep their evals independent of any one model, and put security context behind a standard interface such as MCP. Then you can swap the model and rerun the evals to see what changed.

Costs scale with codebase size unless context is precomputed

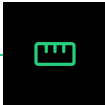
A skill that re-derives security context on every question gets slower and more expensive as repositories grow, and its cost becomes hard to predict. A skill that looks context up costs about the same on a small service as on a monolith. This is the trade-off that most often turns a promising Level 3 pilot into a budget conversation.

Your team's skills can erode

When agents do all the triage, junior analysts stop learning how to triage. Plan for it. Rotate people through reviewing agent decisions, use disagreements between agents and analysts as training material, and keep humans responsible for the work that agents shouldn't own, such as threat modeling and detection design.

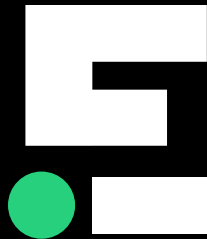
Your first 90 days

Wherever you start, the first 90 days look similar. You establish where you really are, close the gaps that make the next step unsafe, and prove one thing at the next level before scaling it.

Days 1 to 30: ▶	Days 31 to 60: ▶	Days 61 to 90: ▶
establish the baseline ◆ Run the self-assessment for both tracks, with evidence. Note the gap between them. ◆ Build or finish the inventory of coding agents, models, MCP servers, and skills, with owners. ◆ Enforce corporate identity for coding agents, and embed security guidance through rules files and hooks. ◆ Record baseline numbers: backlog size, MTTR, false-positive rate, and what you currently spend per verified finding.  Day 30	measure one bought capability, pick one thing to build ◆ Put your main detection and triage tooling under measurement: precision, fix acceptance, and cost per verified finding. ◆ Confirm that security context is available to agents through an API or MCP server. If it isn't, that's your first buying decision. ◆ Choose one workflow to build a skill for. Pick something high-volume, well understood, and recoverable when it goes wrong. Triage of reachable SCA findings is a common first choice. ◆ Write the eval suite before the skill.  Day 60	ship one skill and decide what scales ◆ Ship the skill with an owner, an eval record, and a cost-per-task figure measured across several repositories. ◆ Compare its cost per verified finding with your Level 2 baseline. ◆ Start an autonomy register, even if every entry still reads "human approves." ◆ Decide what the next skill is, and what you'll buy rather than build to support it.  Day 90



At day 90 you should be able to show three things: where you stand on each track, one capability you built with its evals and cost, and the numbers behind your next build-or-buy decision.



Where Endor Labs fits

Endor Labs builds the foundation layer in this model: the security context and verification your agents call so they reason over facts instead of hunting for them. Build what's unique to your business. Don't rebuild the security foundation underneath it.

AURI by Endor Labs, the security harness for agentic software development, builds a deterministic model of how your applications behave across first-party code, open source dependencies, and container images, grounded in function-level data. Your agents read that evidence instead of reconstructing it. AURI connects wherever agents work, through hooks, skills, MCP, CLI, and API, so the same foundation serves developers' coding agents and the security agents your team builds.

Here's how that maps to the levels.

Level	What Endor Labs provides
1 Governed generation	Coding Agent Governance inventories agents, models, MCP servers, and skills; enforces policy through native hooks; blocks malicious packages at install time; and embeds AURI in MCP-compatible agents so code is checked as it's written
2 Agentic security tools	AI SAST that follows data flow, full-stack reachability for SCA, and agentic remediation, with evidence attached to every finding
3 Custom security skills	The code context graph, Code Search, and Code Navigation, exposed to your skills so they look up reachability and exploitability instead of re-deriving it
4 Orchestrated security agents	The same deterministic foundation under every subagent, so results stay consistent and costs stay predictable from one repository to thousands

The evidence behind it:

- ◆ **12x fewer tokens:** across 34 security prompts in 13 scopes, agents with Endor Labs evidence used 6.6 million tokens versus 79.5 million for the same model alone.
- ◆ **2.6x more real vulnerabilities** found than Claude Opus 4.7 in an eight-project benchmark, including 63 that no other tool detected.
- ◆ **PR reviews in under a minute**, compared with 18 to 20 minutes on average for Claude Code Security.
- ◆ **Predictable cost:** token spend varied 1.8x between projects with Endor Labs evidence, versus 4x without.
- ◆ **A real zero-day:** Endor Labs AI SAST found a previously unknown vulnerability in Anthropic's buffa library (CVE-2026-55407) through data-flow analysis.
- ◆ **92% less SCA noise** with function-level reachability, 6x faster CVE remediation, 83% fewer blocked PRs, and 10x fewer security tickets.

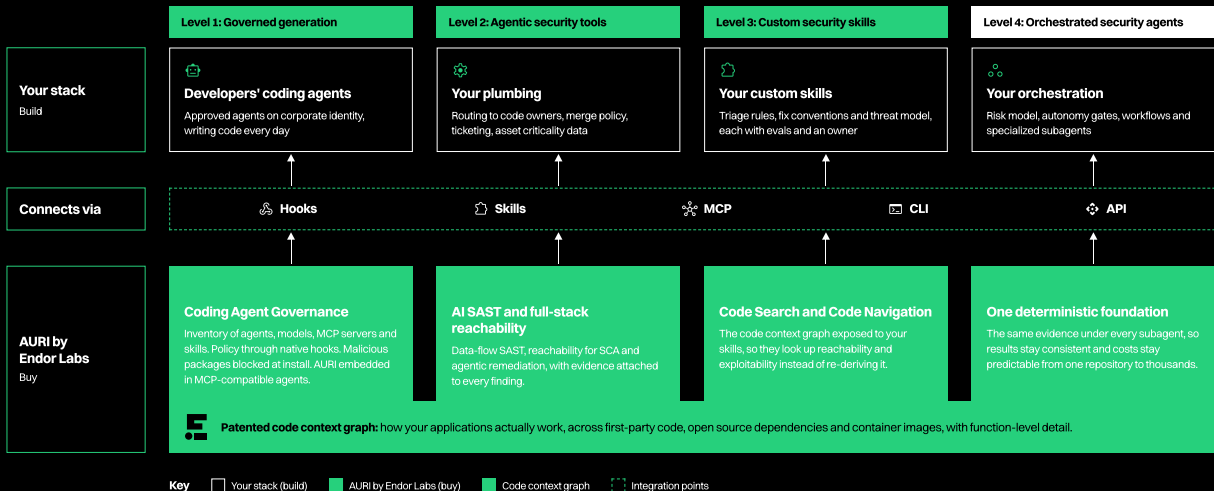


Next steps

Use the [Agentic Remediation Token ROI Calculator](#) to see what unaided agents cost on your own repositories. Or join the Build, Buy, or Both? webinar on November 12 for a live walkthrough of the four levels, with time for your questions.

Your stack, our foundation

Build what's unique to your business. Don't rebuild the security foundation underneath it.



Appendix: how the four levels map to other frameworks

Our levels track capability and ownership. Most published frameworks track a single agent's autonomy. This table shows the typical autonomy you'd expect for AppSec agents at each of our levels, so you can translate between them. Treat it as a rough guide: at Levels 3 and 4, autonomy varies by agent and action type.

Our level	AWS Agentic AI Security Scoping Matrix	CSA autonomy levels (v2.0)	Developer side (Dan Shapiro's five levels)
1 Governed generation	Scope 1, no agency (AppSec use of AI)	L0, AI advises only	Levels 1 to 2: coding intern, junior developer
2 Agentic security tools	Scope 2, prescribed agency	L1 to L2, approval per action or per plan	Level 3: developer, with a human reviewing all code
3 Custom security skills	Scopes 2 to 3, per skill	L2 to L3, conditional autonomy for proven skills	Level 4: engineering team, specs in and agents working
4 Orchestrated security agents	Scopes 3 to 4, set per agent	Mostly L3; L4 only for read-only agents	Level 5: dark factory, where verification replaces review

Sources: [AWS](#), [Cloud Security Alliance](#) (published as unofficial research), [Dan Shapiro via Simon Willison](#).