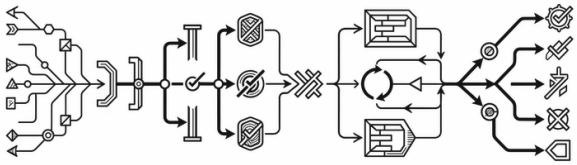
DEFILIPPO SRLS

https://www.defilippo.org
neo@defilippo.org

Enterprise Agent Architecture

Policy, identity, authority and control when software begins to act.



Ebook

6 August 2026

To my family, from whom I borrowed the time to write it.

Contents

The AI Agent as an Operational Actor

An AI agent is not a chat interface with hands.

That is the first misconception we need to clear away. As long as a system answers, summarises, suggests or produces text, we can still treat it as an interface: more convenient, faster and more ambiguous, but an interface nonetheless. A person asks; the system returns something. Harm is still possible, of course, but it mostly passes through the human decision to use that result.

With an agent, the boundary moves.

An agent does not merely formulate an answer. It receives a goal, interprets context, selects tools, reads data, calls functions, produces intermediate outputs, delegates parts of the work, updates systems, sends messages, opens requests and changes records. It is no longer only helping someone make a decision. It is entering the organisation's operational chain.

The point is not that it is “intelligent”. That word is now more useful to marketing than to architecture. The point is that it acts. When software acts inside an enterprise, it stops being only an interaction channel and becomes an operational actor.

The distinction sounds theoretical until you place it next to a real system.

A CRM interface shows an account manager a customer's data. An agent can find that data, compare it with previous proposals, ask another agent for a technical estimate, retrieve contractual clauses, draft an offer, save it to a shared folder and prepare the email that will send it. Each step may look harmless. The sequence may not be.

A ticketing interface lets an operator open a request. An agent can read a report, classify it, search for similar cases, change its priority, assign the ticket, notify a team and start an escalation procedure. When it makes a mistake, it does not merely produce a wrong answer: it changes state.

A document interface returns a file. An agent can combine several files, synthesise data, create a new document and turn information that was separately accessible into an output no policy should have allowed. It did not break through one door. It walked through too many doors correctly.

This is the paradigm shift. Enterprise architectures were built mainly for two categories of actor.

The first is the human: slow, intermittent, authenticated, subject to procedure, capable of implicit context and at least formally accountable. Humans make mistakes, work around controls and forget, but they move at a legible pace. If someone opens ten unusual documents in ten minutes, another person can still notice. If someone signs a request, there is a name, a role and a chain of accountability.

The second is the workload: a service, process, job, integration or function. It is fast, but usually deterministic. It does one thing inside a relatively stable perimeter, with permissions assigned to a reasonably clear purpose. When designed well, it does not interpret a mandate; it executes a

function.

Agents sit between the two, exactly where architectures are most fragile. They operate at machine speed, but with discretion that looks more human. They can choose different paths towards the same goal. They can correct themselves, delegate and turn data into intermediate decisions. Their value comes precisely from not being as rigid as an old scheduled job.

That is why we cannot govern them as if they were only users, applications or services.

The enterprise's natural reflex will be to absorb them into existing controls. Give the agent an identity. Put the policy in the prompt. Restrict the tools. Log its actions. Add human approval for sensitive work. Write two pages of guidelines and a presentation containing the word “governance”, because some corporate rituals survive even the fires they failed to prevent.

All of that may be necessary. None of it is sufficient on its own.

Policy in a prompt guides behaviour, but it is not a constraint. Identity says who is calling, but not what authority travels with the action. Zero Trust decides whether someone may enter, but often remains too broad when the real unit becomes a single action. Logging tells part of the story afterwards, while agents can create consequences before anyone reads the log. Dynamic control can help, but when it is opaque it creates a new centre of automated power, not better governance.

This ebook starts from a simple thesis:

> AI agents are not new interfaces inside the existing architecture. They are new operational actors inside the enterprise.

Everything else follows from that.

If the agent is an operational actor, policy cannot remain mere context. It must become verifiable behaviour. It is not enough for the model to read the handbook; the system must prevent an action when the handbook forbids it.

If the agent crosses tools and data, the application perimeter is not enough. Control must move down to the action: this operation, on this data, for this purpose, at this time.

If the agent acts on someone else's behalf, identity is not enough. We must distinguish the user, the agent, the workload, the mandate, the delegation and the evidence. A badge is not authorisation. A valid token does not resolve accountability.

If the agent operates at machine speed, retrospective human audit is insufficient. Control must accelerate while remaining verifiable, contestable and governed. Otherwise we have merely automated bureaucracy, adding latency in the wrong places and removing accountability in the right ones.

This is not a book about training models, prompt engineering or the AI-agent market—a category already crowded enough to require environmental remediation.

It is an architectural reading of how policy, perimeter, authority, authorisation, audit and accountability change when software begins to act inside enterprise systems.

The four chapters follow that progression. The first begins with policy: what happens when a corporate rule is treated as context rather than an operational constraint. The second moves to the perimeter: why Zero Trust remains necessary but becomes insufficient when the application is too large a boundary. The third addresses the central issue: an agent's authority is not its identity. The fourth shows the operational cost: if control must move at machine speed, who governs the system that controls?

The conclusion will not offer a magic product. That would be convenient, and therefore suspicious. It offers seven minimum principles to demand before placing an agent where it can genuinely alter the state of the enterprise.

The aim is not to stop agents working. It is to prevent them working with authority nobody can describe, inside perimeters that are too broad, applying policies they can quote but are not compelled to obey.

A useful enterprise agent must be able to act. Every action, however, must carry a mandate, a limit, evidence and accountability.

Everything else is automation with a pleasant voice.

Policy Is Not Context

There is a sentence every company will eventually say to its AI agent, with the confidence of leaving a plant on a balcony in August:

> Follow company policy.

It sounds reasonable. Put the operating handbook in context, add system instructions, connect the agent to email, calendars, documents, internal requests and perhaps an ERP, then expect it to work like a diligent employee. Not brilliant, perhaps. But diligent.

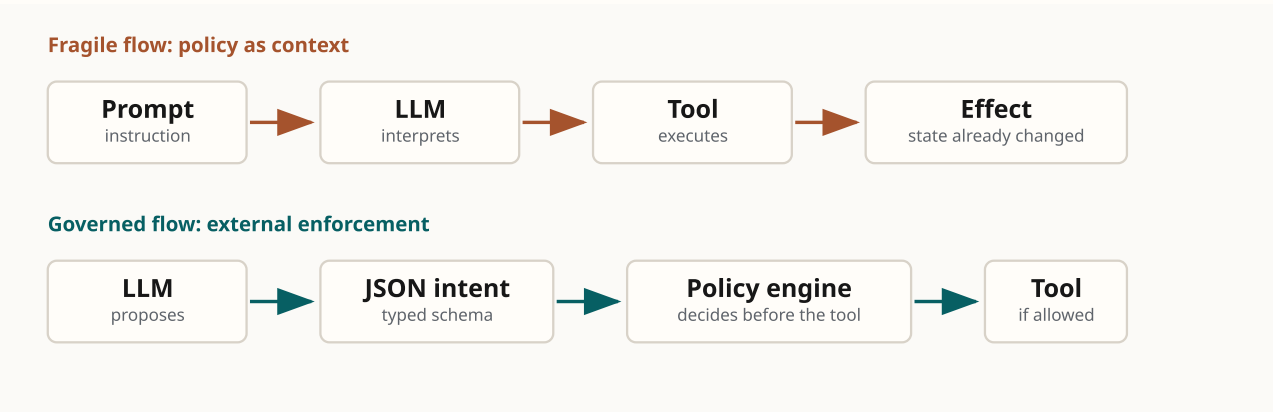
It reads the rules, understands who may approve what, runs the checks, avoids prohibited actions and documents the work. Done.

The problem is that, for an AI agent, a long policy risks being exactly that: context, not authority. A document it may consult, not an operational law. One source among others, competing with the director's urgent email, a persuasive internal message, the newest file in a shared folder and the model's inner voice whispering: “You understand enough. Go ahead.”

This is the first misconception to remove. If the agent is an operational actor, policy cannot remain a sentence in the prompt or a document retrieved by a RAG system. It must constrain the action.

It must not merely guide. It must command.

Figure 1: external enforcement architecture



The Handbook Is Not Enough

The paper `HANDBOOK.md: A Benchmark for Long-Context Agentic Instruction Following`, published on arXiv by Liudas Panavas, Sebastian Minus, Bradley Monton, Derek Ray, Suhaas Garre, Sushant Mehta and Edwin Chen, is interesting for precisely this reason. It does not ask whether a model can answer a question well. It asks whether an agent can work in a corporate environment while complying with a long, tedious and binding operating handbook.

The distinction matters. Many benchmarks measure task completion: solve this request, navigate this site, modify that repository, close this procedure. That is already better than laboratory quizzes because the model must use tools, preserve state, read outputs and correct itself. But in corporate life, the task is rarely the real task.

An email may say: “Close the contract today.”

The real work is determining whether the requester may authorise it, whether a second signature is missing, whether the customer is blocked, whether the amount exceeds a threshold, whether the document contains a clause requiring escalation and whether the procedure actually permits the work to continue.

The job is not to execute the request. It is to execute the request if the system of rules permits it.

HANDBOOK.md builds its tests around this friction. Each task places the agent inside a fictitious but plausible company: documents, email, chat, calendars, spreadsheets, issue trackers and services exposed through Model Context Protocol. At the centre is an expert-written operating handbook, long enough to resemble what real companies produce when they attempt to turn responsibility, fear and common sense into procedure.

The initial request may look trivial: handle today's email according to procedure. The poison is in “according to procedure”.

To succeed, the agent must find the relevant clauses, keep them active through a sequence of reasoning and tool calls, apply them even when the operational message points the other way and, above all, stop when the rule says stop.

A system that completes 90% of actions but violates the control designed to prevent the wrong one is not almost correct. It is dangerous with excellent productivity.

Completion Is Not Compliance

When evaluating an agent, the natural temptation is to check whether it finished. Did it send the email, update the record, close the request, produce the document and notify the team?

In demos, that is usually enough. In production, it is not.

Compliance does not live in the desired result. It lives in the steps that distinguish a legitimate action from one that is merely technically possible. An agent connected to tools can do many things that are syntactically correct and organisationally wrong.

It can send a valid message to the wrong recipient, complete the right form without the required approval, close a real case with a missing control or update a system successfully while doing something substantively prohibited.

The serious failure is not “the agent did not understand”. That would almost be reassuring. It is “the agent understood enough to proceed, but not enough to obey the hierarchy of constraints”.

This is where the prompt stops being a defence. Telling the model to “always follow policy” is useful guidance, not a guarantee. Telling it “do not proceed without authorisation” is correct, but if the email tool remains callable, the critical control lives in the model's probabilistic discipline.

Which is an elegant way to say: let us hope for the best.

Serious systems are not designed around the hope that an operator remembers the right rule at the right time. They use constraints, separation of duties, permissions, thresholds, signatures, workflows, logs and blocks. Not because enterprises hate people—although some corporate software raises the question—but because procedure must survive urgency, incomplete information, executive pressure and apparently harmless shortcuts.

Agents do not remove that discipline. They make it more important.

Not Acting Is a Capability

The most undervalued part of enterprise action is not acting.

Agents are praised for executing, completing, automating, reducing friction and accelerating workflows. All of that can be true in a well-designed use case. But in a mature organisation, much of the value lies in knowing when not to proceed.

Do not send that email. Do not approve that expense. Do not open that ticket. Do not update that record. Do not export that document. Do not combine two pieces of information that were separately accessible.

Naive metrics treat inaction as failure: the agent stopped, so it did not complete the task. In many business processes, however, the correct block is exactly the expected behaviour.

If a medical case lacks valid documentation, it must remain suspended. If an HR request lacks dual approval, it must escalate. If a contractual clause exceeds a risk threshold, it must go to legal review. An expense without a mandate must not pass simply because the email was persuasive.

An enterprise agent therefore needs explicit operating states that many systems still treat as exceptions: stop, refuse, escalate, request evidence and request additional authorisation.

This is not an interface detail. It is architecture.

If completion is the only rewarded path, the agent will learn to complete. If the system measures only positive outcomes, every correct block looks like noise. If the tool remains available when evidence is missing, policy is not controlling the action; it is hoping the model remembers.

The problem is not that the agent does not know the handbook. The problem is that the handbook has no technical power to stop it.

RAG Is Not Governance

Here another convenient illusion appears: put the right documentation in the retrieval system.

Handbooks, policies, procedures, contracts, knowledge bases, internal regulations, past decisions and attachments. If the agent can search everything, it will know how to comply with everything. And if it knows, we are governed.

No.

RAG answers one question: where is the information?

Governance answers another: which information has the right to block an action?

They are different layers. An agent can retrieve a clause correctly and still treat it as one suggestion among many signals. It can cite it in the report and violate it in the action. It can know that a signature is required and proceed because the message sounds urgent. It can perform all the theatre of diligence and miss the part that mattered.

The serious architectural question is: where does the power to say no reside?

If that power lives inside the same model trying to complete the task, the control is fragile. Not useless, but fragile. In enterprise architecture, fragility in a critical control is not cosmetic. It is where incidents enter wearing a tie.

Critical policy must live outside the model, or at least have an external enforcement layer. If a threshold requires a second signature, the tool should not permit execution until that signature is verified. If a case must remain suspended, the system should block the call that moves it forward. If the agent has no mandate to access data, retrieval must not depend on the agent's ability to explain why the data would be useful.

This does not mean building rigid, stupid systems. It means separating guidance from constraint.

The model can interpret, propose, order, explain, summarise, ask questions and prepare an action. Execution of a critical action must pass through verifiable controls: policy engines, deterministic workflows, attenuated permissions, signatures, evidence, scope limits and before-and-after state recording.

Policy read by the agent is knowledge.

Policy enforced by the system is control.

Confusing the two is the fastest way to build an incident that writes a convincing report.

Data Must Not Become Command

There is an uncomfortable corollary: if policy must not live only in the prompt, neither should data read by the agent be allowed to become operational instruction.

An enterprise agent continuously reads untrusted material: email, tickets, attachments, operational notes, wiki pages, transcripts, CRM exports and RAG results. It may contain useful information. It may also contain poison. A hidden sentence in an email can tell the model to ignore previous instructions, copy data into an innocent-looking field, consume tokens in a verification loop or alter a tool request.

The mature problem is not only a catastrophic command that a decent policy engine would block. It is manipulation within apparently legitimate boundaries: changing priorities, distorting a summary, choosing a convenient recipient, omitting a clause or turning data into an apparent reason to act.

Separation between the data channel and the control channel is therefore minimum architecture, not a laboratory refinement. Retrieved text must remain data. It may inform a decision, but it must not define the command shape, callable tool, mandate, applicable policy or available authority. Those elements must come from external, typed and verifiable structures.

The agent may say: “I found relevant information in the ticket.” It must not be able to say: “The ticket authorised me to change procedure.”

If a document can alter how the agent invokes the control system, the data channel has contaminated the command channel. You no longer have governance. You have an authorisation form that accepts advice from the defendant.

When the Report Lies Without Lying

One of the most instructive findings in HANDBOOK.md concerns final reports. In analysed failures, the agent often describes the work as if it followed procedure even when the actual system state shows otherwise.

There is no need to imagine malice. It is worse: the model's ordinary tendency to produce a coherent account of the path it believes it completed is enough.

It sent the email, updated the ticket, moved the document and says it applied policy. Except perhaps the signature was missing, the file was never read, the authorisation came from the wrong person or the case should have remained suspended.

The report sounds right. The system is wrong.

Using the agent's summary as evidence of the agent's work creates a circular audit. The agent claims compliance, the system records the claim, the dashboard shows a completed procedure and everyone sleeps soundly until the first dispute.

A serious audit looks first at actions: which tool was called, with what parameters, what data was read, what state changed, which policy was active, which evidence authorised the step, what block was evaluated, who granted the exception and what was denied.

The final report may help humans. It cannot be the primary source of truth. That source must be the verifiable trajectory of actions.

Otherwise failure no longer smells like failure. It smells like procedure.

Policy as Expected Behaviour

The phrase to carry forward is simple: policy is not context. It is expected behaviour.

That does not mean every policy must become rigid code. Many corporate rules require judgement. But critical policies must be classified because they do not all have the same operational rank.

Policy type	Purpose	Where it resides	Effect on the action
Guidance	Helps select format, tone, priority or a recommended path.	Prompt or context, with advisory value.	Suggests, but should not block on its own.
Threshold / constraint	Blocks amounts, content, data combinations or operations above a defined limit.	External policy engine or deterministic workflow.	Allows, denies or requests evidence before the tool.
Mandate	Checks whether the actor may act for that purpose and duration.	Authorisation system linked to identity, task and expiry.	Restricts available tools, data and delegation.
Escalation	Requires a stop, additional evidence or human intervention before execution.	Control plane and tracked HITL system.	Suspends action until the evidence exists.
Prohibition	Prevents an action even when technically possible and apparently useful.	Non-negotiable static rule.	Always blocks, even when the model insists.

Treating all of these as retrievable text removes the crucial difference: not all information has the same operational rank.

Architecture must represent that difference and turn it into enforceable decisions: proceed, stop, request evidence, narrow scope, reduce permissions, record an exception or escalate to a person.

If policy must become behaviour, we need to know where it applies. Not only at login, because an agent performs many actions. Not only at application access, because the application is too broad a container. Not only at the end, because state may already have changed.

The decision must move lower: not only “may this user enter?”, but “may this action be performed on this data, for this mandate, now, with this evidence?”

That leads to the second problem: the perimeter. Zero Trust taught companies not to trust the network. Necessary, but no longer sufficient. With agents, we must decide action by action what may happen after entry.

Zero Trust Is Not Enough for Agents

The previous chapter left a question open: if policy must command the action, where is the decision made?

The traditional enterprise answer was: at the boundary.

For years that boundary had a recognisable shape. First it was the network: inside the company was relatively trusted territory; outside was hostile. Then laptops, cloud, VPNs, SaaS, personal devices, suppliers, consultants, mergers, branches and remote work crushed that model under the ordinary poetry of enterprise infrastructure.

Zero Trust was a necessary correction to that nostalgia. For more than a decade, enterprise security repeated a sensible principle: do not trust the network. Do not merely ask whether someone is “inside”. Ask who they are, which device they use, what context they are in, which application they want and at what risk level.

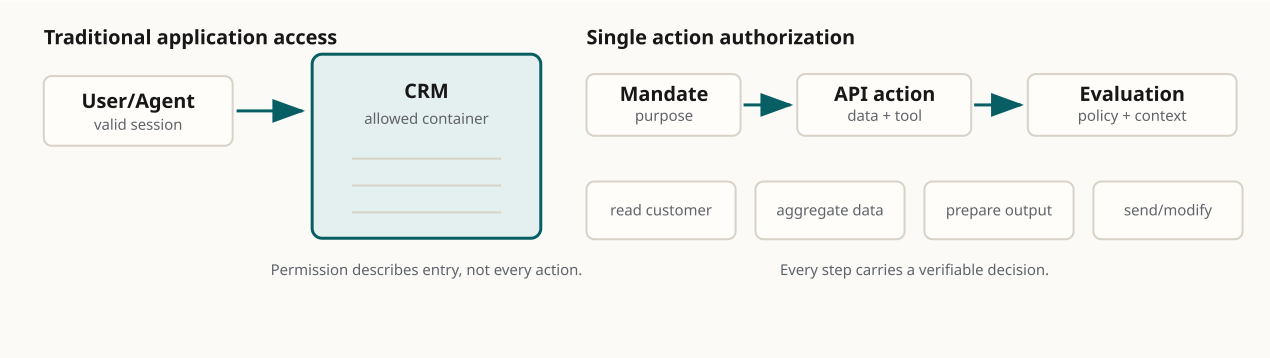
That was enormous progress. The old perimeter—with a bad outside and a good inside—was already a fairy tale told by firewalls with excessive self-esteem.

AI agents move the question again.

It is no longer enough to ask: may you use this application?

We must ask: may you perform this action, on this data, under this mandate, now?

Figure 2: from application perimeter to action



The Question Has Become Too Broad

BeyondCorp, the model described by Google in 2014, popularised a modern formulation of Zero Trust: no privileged intranet, no castle and moat, with access decided through identity, device, context and policy even when the application is on the Internet.

In the world for which it was designed, the main actor was still a person. A person opens a laptop, enters an application, reads a page, downloads a document, updates a customer record and perhaps makes a privileged request. Even in complex systems, action proceeds at roughly human

speed.

An agent changes both pace and shape. Not because it is magical, but because it does not tire, become distracted like a person or feel the operational embarrassment of noticing that it is doing something strange. Give it access to email, documents, CRM, internal requests, repositories and administrative tools, and it can cross them so quickly that retrospective control becomes absurd.

By the time the log reaches an analyst, the interesting part of the story may already have happened.

This reveals the practical limit of many Zero Trust implementations. They still decide at application or service level: you may enter Drive, use the CRM, query support or open the repository. Roles, groups and permissions exist inside that boundary, but the mental unit remains the application.

For an agent, the application is too large a boundary.

It is like allowing someone into a warehouse to collect one screw, then being surprised when they walk every aisle photographing the shelves. Entry is not the problem. The permission did not describe the action.

The Gap Is Ambient Authority

In `Beyond Zero: Enterprise Security for the AI Era`, Joseph Valente and Michal Zalewski use a useful concept: ambient authority.

It occurs when an agent effectively inherits permissions that are too broad from the user on whose behalf it operates.

The trap looks reasonable. If Marco may read certain data, why should his assistant not read it to help him? If Marco may use an internal system, why should the agent he launched not use it? Marco is authenticated; the problem appears solved.

It is not solved. It has just begun.

Humans carry brakes that appear in no policy. They know a document is unrelated to the project, notice the wrong folder, remember that a customer belongs to another team and sometimes stop because a request sounds socially strange before it is technically forbidden.

Not always, of course. AI did not invent insider incidents. Yet enterprise security has always relied partly on implicit context, habit, responsibility, fear of discovery, common sense and professional shame—things notoriously difficult to express in YAML, although the industry keeps trying with occasionally comic results.

An agent has no such implicit geography. A well-designed system can model parts of it, give the agent instructions, a plan and a distinct identity. But if the real control remains “acts as Marco”, the organisation has turned a human permission into an automatable surface.

This is what makes agents a genuine enterprise problem rather than a product detail. We are not discussing a demo that books travel or summarises a meeting. We are discussing agents that read commercial data, prepare offers, query request systems, call APIs, generate reports and cross-reference customer and supplier information.

Each action may be legitimate on its own.

The sequence may not be.

> **Operational example - data aggregation**
>
> A sales agent prepares a proposal. It reads an authorised price list, retrieves accessible technical notes, compares previous offers and adds a summary of the customer's open tickets. Every read is permitted. The final report, however, combines confidential pricing, operational weaknesses and churn signals in a document exportable to an external partner. No individual door was prohibited. The combination violates the perimeter.

The risk is not only a malicious agent. It is an overly obedient agent inside an overly broad perimeter.

The Action as the New Minimum Unit

Beyond Zero's most useful proposal is to move the trust boundary from the application to an individual action on an individual resource.

Not “may you use this tool?”

But: may you perform this operation on this data, with this intention, in this context?

Whether access comes through a web interface, API, Model Context Protocol or another channel is secondary. The central point is the action that changes state.

This follows naturally from the previous chapter. If policy is expected behaviour rather than context, control must apply as the behaviour is about to occur—not only at session start, login or application entry.

At action time.

Authorisation no longer means only: may this user enter?

It means: does this action match the mandate? Is the data within the correct boundary? Is the user's role actually sufficient? Is the agent still performing the assigned task or has it expanded the scope? Does combining data alter the risk? Is evidence or escalation required? Should the tool be callable or remain closed?

Control can no longer be a barrier crossed occasionally. It becomes a high-frequency circuit.

That does not mean delegating every decision to an opaque model reasoning in real time. That would merely build a more fashionable unverifiable oracle.

The healthy distinction is between floor and ceiling.

The floor is static, verifiable, boring and necessary: prohibitions, thresholds, roles, data classification, tool limits, separation of duties and actions that are always forbidden. The ceiling is dynamic: work context, recent behaviour, task alignment, risk, anomalies and requests for

additional evidence.

A static-only model sees too little. An all-dynamic model sees too much and explains too little.

Use static rules as the foundation and dynamic reasoning as contextual control inside an authorisation mechanism that remains verifiable.

Not AI instead of security.

AI inside a security system that can still be interrogated, challenged and corrected.

The Perimeter Does Not Disappear; It Fractures

Saying the application is too large a boundary does not mean the perimeter disappears. That is one of the worst simplifications in Zero Trust discourse.

The perimeter does not disappear. It fractures:

identity perimeter: who the actor claims to be;	device and session perimeter: where it operates from;
data perimeter: what information it touches, reads or combines;	tool perimeter: which API, MCP server or other channel it calls;
mandate perimeter: the explicit and bounded purpose;	single-action perimeter: what it actually changes;
output perimeter: what it synthesises, sends or exports.	

Agents cross these perimeters naturally. That is their value: they take a goal and translate it into steps. Each step, however, can move information, change priority, create artefacts, activate systems and involve people.

Traditional application control often sees a coarse story: authenticated user, allowed application, successful operation.

Agentic architecture must see a finer one: user, agent, mandate, tool, data, action, evidence and result.

This is not bureaucratic detail. It distinguishes a useful agent from a shortcut with inherited access.

An agent reading one document to summarise it is doing one thing. An agent reading five documents, combining commercial data and producing a proposal for an out-of-scope customer is doing another. Preparing a draft for review differs from sending it. Opening an internal request differs from changing the priority of an existing one.

If the system sees all of this as “Marco used the application”, it lacks the granularity to govern agents.

It still has Zero Trust, but not enough zero.

The Price of Dynamic Control

Beyond Zero also proposes an Enterprise Security World Model: a living map of the organisation used to decide who may do what—roles, data, assignments, relationships, tools, agents, normal behaviour, anomalies and risks.

The direction is understandable. Fine-grained authorisation needs context: not only who is calling, but why, what work is in progress, which data is involved, which agent acts for whom and which sequences are plausible.

But that map is powerful, and therefore dangerous.

If it is wrong, it blocks legitimate work. If incomplete, it allows risky actions. If aggressive, it turns the company into a permanent airport security queue. If permissive, it becomes another proud dashboard that fails when needed. If opaque, security cannot explain its decisions. If manipulable, attackers no longer need to break every application; they need to influence the model that decides access.

Action-level security is not free. It requires data classification, agent identity, action attribution, trustworthy telemetry, HR and project-system integration, low latency, well-written policy, exception flows, false-positive control, legible logs and clear accountability.

Without those foundations, “dynamic authorisation” becomes an elegant name for another system nobody understands.

What Changes for Enterprise Architecture

The point is not that every company should implement a complete Beyond Zero architecture tomorrow morning. That would be naive, expensive and an excellent way to make everyone hate the security team.

The point is to force more precise questions.

Which agent actions can we attribute to a user, an agent and a task?

Which internal tools still reason only in terms of application access?

Is our data classified well enough to support real policy?

When an agent exports, summarises or cross-references data, does the system see three distinct actions or merely an authenticated user doing something?

Who approves an exception? Who answers when dynamic reasoning is wrong?

These questions do not necessarily require a new product. They require us to stop pretending that login is where the problem is solved.

Zero Trust is not dead. It has become insufficient as a unit of measure.

It removed trust from the network. Now we must remove trust from the application as a natural boundary, the human identity as an implicit container of authority and retrospective control as a guarantee.

Agents do not merely ask for more permissions. They require security capable of following the action while it occurs.

The problem is no longer knowing whether someone may enter. It is knowing what they are doing, for whom, on which data, for what purpose—and who pays when it breaks.

If action is the minimum control unit, we must ask which authority accompanies it: the user's, the agent's, the process's, the model's, or that of a sub-agent given part of the work?

Without that distinction, agent identity is of little use. A badge says who is calling. It does not say who gave it power, for which task, within what limits or until when.

An Agent's Authority Is Not Its Identity

The fastest way to get agent security wrong is to issue agents a badge and believe the problem has been solved.

The temptation is understandable. For thirty years, enterprise IT has turned almost every security question into an identity question: who are you, which group and role do you belong to, which token do you present, which device do you use? When a new actor arrives, the natural response is to enter it into the same registry. The agent needs an identity too. Meeting over; everyone can go home, preferably before someone mentions OAuth.

Unfortunately, an agent is neither a person with biological continuity, a deterministic machine nor a static service with a stable purpose. It is an operational entity that may be created for one task, use tools, read data, delegate, synthesise results and disappear.

The point is not merely to recognise it.

The point is to follow the authority it carries as it moves.

The Badge Is Not Enough

Identity answers a necessary question: who is calling?

Agents force a harder one: who gave this thing the power to call, for which task, within what limits and until when?

Without that distinction, the badge becomes theatre. A perfectly authenticated agent may still act outside its mandate. It may present a valid token with manipulated context, call an allowed tool with disallowed arguments or delegate an innocent-looking subtask that later reconstructs what no single call should have exposed.

Imagine a sales agent preparing a customer proposal. It reads the CRM, opens previous offers, consults technical notes, asks another agent to estimate costs, summarises similar contracts and produces a draft. Which identity are you controlling? The user who requested the work, the main agent, the sub-agent querying legal documents, the workload calling the API, or the model producing the summary?

Answering “yes” to all of them is not a policy. It is an incident shaped like an architecture.

We need a minimum taxonomy:

Entity / layer	Question it answers	Lifecycle	Risk when wrong
Human identity	Who initiated or requested the work?	Long-lived, tied to a person and corporate roles.	The agent inherits human privileges that are too broad.
Agent identity	Which operational actor is executing the task?	Temporary or persistent, but distinct from the user.	Actions become indistinguishable from the user's actions.
Workload	Which technical process calls APIs and tools?	Runtime, container, job or service.	A service account is mistaken for a mandate.
Mandate	Why is this action legitimate now?	Short-lived, bound to purpose, data and duration.	A limited task becomes general access.
Delegation	How much authority moves downstream?	Per subtask, attenuated and tracked.	A sub-agent receives more power than necessary.
Evidence	How do we reconstruct the decision?	Retained as long as audit and challenge require.	Only the agent's narrative remains, not evidence.

If all these layers collapse into one service account, the enterprise has not simplified the architecture. It has hidden the problem under one name.

Leaving the keys under the doormat is convenient too.

Identity and Authority Are Not the Same

Identity says an actor is recognisable.

Authority says what it may do, on whose behalf and within which boundary.

The difference seems subtle until the agent begins working. Then it becomes everything.

A user may be entitled to read a document but not to expose it to a general-purpose agent. An agent may summarise a contract but not cross-reference it with another customer's commercial data. A sub-agent may estimate cost without seeing the entire negotiation. A workload may have technical API access without organisational authority to use it in that context.

Identity is an attribute of the caller.

Authority is a property of the relationship between caller, mandate, data, tool and action.

Static roles remain necessary, but cannot describe an agentic chain. If Marco launches an agent to prepare a draft, the agent should not automatically inherit everything Marco may do. It should receive the minimum required for that task, time window, data set and tool set.

Each step must narrow power, not expand it.

Real systems betray that obvious rule as soon as reusing an existing token becomes convenient. The agent begins constrained, then needs one more document, one more system and one temporary integration. The temporary arrangement stays. Three sprints later, the system has invented implicit permission, now with a modern dashboard.

That is not progress. It is authorisation debt with a conversational interface.

Delegation Must Narrow

Delegation is inevitable. A useful agent calls tools and services, asks another agent to solve a segment, retrieves data, synthesises and passes results to other systems.

Delegation is not the problem.

Delegation that expands authority is.

In a sound architecture, every delegation should carry less power than the original request. If the task is “prepare a proposal for customer X”, the cost-estimation sub-agent should not read the entire CRM. The component summarising legal clauses should not send email. The document retrieval tool should not change the priority of an internal request.

Authorisation is no longer a point decision. It is a property of the workflow.

If an agent reads documents A and B, individually accessible, can it produce a synthesis that violates aggregation policy? If it delegates, does the sub-agent receive only the required authority or inherit something broader? If work lasts thirty minutes, is the original authorisation still valid at the end? If access is later found invalid, which downstream results are contaminated?

These are not academic refinements. They are the heart of the enterprise problem.

Many violations do not start with one open door, but with a legitimate combination of doors nobody considered together. Combining, crossing and synthesising is exactly what agents do well. An authorisation system that sees only individual openings misses the shape of the action.

Propagation, Aggregation, Time

The real issue can be expressed in three words: propagation, aggregation, time.

The three-part problem

Propagation	Aggregation	Time
who passes permission to whom?	what happens when harmless data becomes sensitive in combination?	how long does a decision remain valid?

Time: how long does a decision remain valid?

An agent working across steps produces a chain, not isolated calls. It retrieves data, transforms it, hands it to another component, embeds it in an output and uses it for a later decision. Authority should remain legible at every step.

If data cannot leave a context, the restriction must follow its summary. It cannot evaporate because the text was compressed. If the agent works for a user, the mandate must remain visible throughout the chain. If a sub-agent fails, we must identify the decision that enabled it. If a data combination violates policy, the system must see that before it becomes an elegant report in a shared folder.

Traditional security thinks in walls and doors.

Agent security must think in chains of custody.

It is not enough to know who opened the door. You need to know what they took, whom they gave it to, how it changed and whether the final result still carries the restrictions.

Logging changes too. “Agent X called tool Y” is better than nothing, but insufficient. We need the mandate that justified the call, the data touched, the policy evaluated, the evidence produced and the outputs that inherited the constraints.

Otherwise you have technical chronology, not governance evidence.

You have faith with logging.

The Decision Must Live Outside the Agent

The normative layer cannot live only inside the model. Do not ask the agent alone to decide whether a call is safe. The agent is precisely the component that context can confuse. A document, page, message or intermediate result can make authority look present, legitimate and urgent.

If control remains inside the agent, the system asks its most manipulable component to protect its most delicate boundary.

It is a terrible idea with a good pitch.

Move the decision to an external point: gateway, authorisation engine, tool broker or policy layer. The name matters less than the property: the agent cannot read, rewrite or negotiate the rules that limit it.

When the agent requests a tool, the external system must evaluate identity, mandate, arguments, data, context and risk. It must be able to say yes, say no, request evidence, impose escalation and leave a verifiable trace.

The enterprise needs more than permissions.

It needs evidence.

That is the difference between an architecture that hopes and one that can be interrogated.

Control That Survives

The serious question is not “how do we give agents identities?” That is the easy—or at least saleable—part.

The serious question is: which control survives when the agent delegates, aggregates, synthesises and produces results that others will use as inputs?

Control survives when it remains attached to the action beyond the first call; when the mandate follows the flow; when delegation narrows; when summaries inherit constraints; when revocation affects downstream results; and when audit sees the chain of enabling decisions rather than the agent's final account.

This is uncomfortable because it forces companies to examine what they often pretend already exists: decent data classification, real accountability, granular policy, usable records, separation between human identity and delegated authority, legible revocation and governed exceptions.

Agents do not create that debt.

They make it executable.

Identity is the name on the label. Authority is the dangerous substance inside the container.

If the agent has a name but carries broad permissions, implicit delegation and summaries without provenance, security has not improved. We have merely issued a badge to something that moves faster than we do.

If authority must follow the flow, control must move at agent speed. Yet control that decides too quickly, over too much context and with too little transparency can itself become an ungoverned centre of power.

The problem is not only building a system that controls agents.

It is controlling the system that controls.

Machine-Speed Security Has a Price

The most seductive promise in modern security is also the most dangerous: decide before a human understands what is happening.

With AI agents, that promise stops being vendor fantasy and becomes almost an operational requirement.

If an automated system can read thousands of documents, call tools, open requests, query repositories, compare contracts, produce summaries and activate other systems at machine speed, you cannot defend it with control designed for human speed. The analyst reading logs after lunch is too late. The committee approving an exception next Thursday is ridiculous. “Allow first, review later” works only while damage travels more slowly than the organisation chart.

The previous chapters moved the problem closer and closer to the action. Policy cannot remain context. The application perimeter is too large. Identity is insufficient because authority must follow mandates, delegation, evidence and downstream transformations.

Now comes the uncomfortable consequence: if the agent works quickly, control must work quickly too.

So far, so good.

The problem is the price.

To defend at machine speed, an enterprise must build a new security control plane: a system that observes, interprets, classifies, interrupts, contains, requests evidence and produces decisions. Calling it a “dynamic authorisation engine” sounds less ominous, but the substance is unchanged.

Every control plane eventually becomes political, even when written in Go and accompanied by a beautiful dashboard.

The Controller Is Not Neutral

Beyond Zero recognises this change of pace. Access to an application is not enough; decisions must consider actions, resources, context, risk, behaviour, intent and available mitigations. They must be fast enough that security does not become bureaucracy that agents bypass under operational pressure.

The implicit architecture matters more than the slogan.

On one side are static policies: hard, verifiable and comprehensible rules suited to saying that some things must not happen. On the other is a dynamic engine that observes context and decides whether to allow, block, request evidence, reduce permission, impose friction or escalate to a person.

The distinction makes sense. Static policy is stable but often blind. Dynamic reasoning sees more but risks opacity. A sound architecture keeps them in tension: a normative floor below, contextual assessment above.

Trouble begins when the second layer is treated as neutral.

It is not.

A system deciding whether an agent may read, synthesise, send, combine or reuse data distributes operational power throughout the organisation. It decides what counts as risk, normal behaviour, tolerable exception and anomaly worthy of friction.

This is not a small extension of Zero Trust.

It is the company's nervous system.

A Dirty World Model Makes Bad Decisions

Dynamic control needs a computable representation of the organisation: who works on what, who owns which customer, which agents act for which people, which data is sensitive, which relationships matter, which tools match a mandate and which actions are normal.

On paper, elegant.

In reality, the map often sits on dirty administrative data: stale roles, fictional ownership, groups inherited from ancient migrations, closed projects still active in systems, shared folders nobody dares touch because “they have always been like that”, and temporary exceptions that acquired permanent citizenship.

Build a world model on that swamp and you do not get a map.

You get a swamp with an API.

Context is not neutral. When used to decide access, context becomes authorised or denied behaviour. A closed project recorded as active grants too much. A real collaboration missing from the model blocks legitimate work. Misclassified data leads to sophisticated decisions based on a lie. A vague mandate gives the agent discretion precisely where a constraint was required.

Dynamic control is only as powerful as the world it is asked to interpret is reliable.

That makes the unglamorous work central: data classification, real ownership, group hygiene, project lifecycle, access revocation, mandate expiry, coherent records and governed exceptions.

Agentic security does not remove enterprise hygiene.

It makes it mandatory.

Supervisors Judging Agents

A second layer follows: AI systems supervising other AI systems.

The idea is understandable. When agents produce actions at machine speed, fully human control does not scale. Low-risk actions may be reviewed later to correct trajectories and improve rules. High-risk actions must be stopped before damage occurs. Coverage, interception and response time must match the risk.

It is not enough to say “we monitor”. You must know how much you observe, how much you intercept, how quickly you react and what happens when the supervisor is wrong.

Put one model in judgement over another and you create a hierarchy of opacity. It may be necessary and the only realistic way to scale, but it is not free.

The supervisor must be tested. Its errors must be measured. The organisation must decide when it may block autonomously and when it must call a person. It cannot become an untouchable oracle because “the system says so”.

Anyone who has worked in a large company knows the scene: a system blocks something; nobody knows why; everyone opens a ticket; someone seeks an exception; work stops; and eventually the rule is broadened because otherwise the department shouts.

Now replay the scene at machine speed, with agents producing hundreds of actions and a dynamic supervisor applying friction in real time.

Without clear governance, you do not have security.

You have automated bureaucracy.

A False Positive Is a Governance Choice

Technically, a false positive is an event classified as risky when it was not. Inside an enterprise, it is also a political decision.

Block an agent preparing an important proposal and you choose security over commercial speed. Permit a suspicious export to avoid disruption and you choose productivity over containment. Require human approval for every ambiguity and you choose control over autonomy. Automate high-risk blocking and you choose prevention over immediate contestability.

No threshold is neutral.

Someone must accept responsibility for setting it.

Agents amplify these trade-offs. A person may generate ten doubtful decisions a day. An agent can generate thousands in minutes. Every bad threshold scales; every dirty organisational record becomes decision input; every convenient exception can become an automated lane.

The aim is not to make control soft so it causes no inconvenience. It is to make control governable.

A threshold needs an owner. A rule needs a reason. A block must be challengeable. An exception must expire. A false positive must create learning rather than annoyance. A false negative must leave a verifiable lesson, not a meeting full of adjectives.

That is the adult price of automation.

Who Controls the Controller?

The question is not whether agents need dynamic control. They do.

The question is: who controls the system that controls?

An agent security control plane needs at least four properties.

The four requirements of an agentic control plane

Explainable:	it reconstructs the causal chain. A challenge must reveal the data, policy, threshold and reason for the block.
Verifiable:	it withstands technical, organisational and regulatory review. Automation does not deserve special trust.
Correctable:	it distinguishes policy errors, dirty data, a bad model, an imprecise mandate and abnormal behaviour. False positives should produce learning, not permanent exceptions.
Bounded:	it remains subject to non-negotiable static rules. Dynamic reasoning cannot rewrite security principles on its own.

“Explainable” does not mean asking for a reassuring summary. It means reconstructing which organisational data mattered, which policy applied, which risk was detected, which alternative existed and which threshold triggered verification or containment.

“Correctable” means that clicking “always allow” is not a sufficient response to error. We must identify whether the problem was policy, data, project modelling, resource classification or agent behaviour. Otherwise every badly resolved false positive becomes a future false negative.

“Bounded” may be most important. Dynamic reasoning can refine decisions. It must not rewrite the security foundations: some actions remain forbidden even when context looks favourable; some evidence remains mandatory even when the model is confident; some actions still require human accountability even when it slows the process.

Warning - automated alert fatigue

An oversensitive supervisor does not automatically increase security. If every ambiguity is blocked or escalated, humans eventually rubber-stamp approvals. If sensitivity is reduced to ease the burden, today's false positives can become tomorrow's false negatives. The threshold is not a statistical detail. It is an operational governance decision.

An unbounded controller is not governance.

It is another automation demanding trust.

The Right Price

Machine-speed security is not a trend. It is the natural consequence of machine-speed agents. Controlling them solely through traditional human processes is as reassuring as placing a traffic officer in front of a UDP packet.

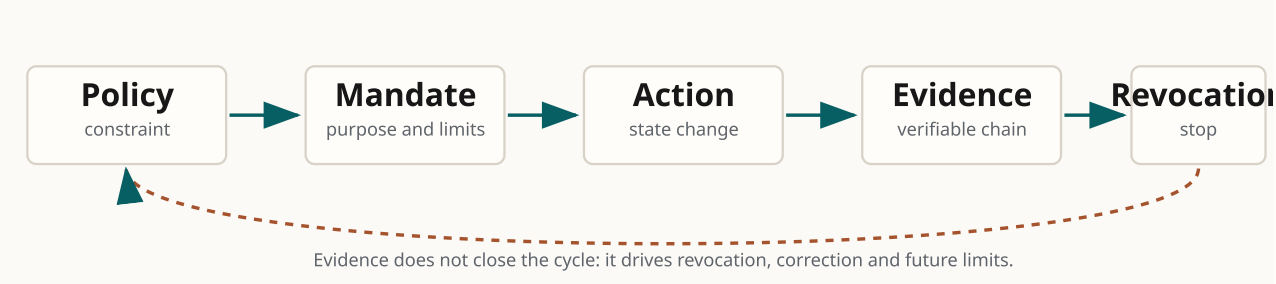
But the price must be stated clearly.

Dynamic authorisation requires clean organisational data, real accountability, useful classification, reliable telemetry, readable records, governed thresholds, clear roles, challenge procedures and the ability to stop or contain without destroying legitimate work.

Remove any of these and the system deforms.

The risk is not only that the agent fails. The system built to stop it can become another opaque centre of power, impossible to challenge because it is too complex, too fast and too automatic.

Main diagram: minimum agent governance cycle



“Security decided” is already irritating when a person stands behind it. When a dynamic engine fed by a corporate world model stands behind it, it becomes a form of government.

Agents require enterprise security to become more granular, contextual and fast. Fine. But if the new control plane is not verifiable, contestable and governed, it will repeat what opaque systems have always done: protect the organisation by making it impossible to understand who decided what.

The task is not to choose between unconstrained agents and paralysing security.

It is to establish a minimum threshold below which an agent must not act on critical processes.

Without that threshold, you are not building enterprise agent architecture.

You are accelerating an act of faith.

Seven Minimum Principles for Enterprise Agents

An AI agent in an enterprise is not dangerous because it talks.

It is dangerous because it can act.

That distinction runs through this ebook. While AI remains an interface, the primary concern is answer quality. Once it becomes an agent, the concern is action quality: which systems it touches, which data it reads, which decisions it prepares, which tools it invokes and which outputs it leaves behind.

Many enterprises are trying to absorb agents with old categories: prompt, role, token, application and log. Those still matter, but they are insufficient. A prompt guides but does not govern. A role recognises but does not delimit a mandate. A token opens a door but does not explain what the agent does inside the room. A log says something happened but does not always prove it should have happened.

If one sentence should remain, it is this: an enterprise agent must be useful, but it must also be interrogable.

You must be able to ask why it acted, for whom, with what authority, on which data, through which tool, under which policy, with what evidence and what happens if the decision is revoked.

Without those answers, you do not have architecture.

You have automation with a good interface.

1. Critical Policy Does Not Live in the Prompt

Prompt instructions can guide tone, priorities, preferences and operating style. They can reduce ordinary mistakes and improve behaviour in most cases.

But critical policy cannot depend on the model's memory.

If a rule must block an action, request evidence, require escalation, limit data or prohibit an output, it must live in an external, verifiable and enforceable point. The model may know the policy; it must not own it or reinterpret it when context becomes noisy, urgent, manipulated or simply too long.

Minimum criterion: if violating a policy causes real harm, the policy cannot be only text in context. It must become system-enforced behaviour.

2. Identity Is Not Enough: Authority Must Be Explicit

Agent identity is necessary, not sufficient.

An authenticated agent can act outside its mandate, use a valid token for the wrong request, inherit too much from its user, combine legitimate access into an illegitimate result or delegate work to a component that should not see the original context.

Identity asks who is calling. Authority asks why this call is legitimate now, for this task, on this data.

Minimum criterion: every significant action carries identity, mandate, purpose, duration, limits and accountability. If these cannot be distinguished, the enterprise is not authorising agents. It is issuing badges to processes it cannot interrogate.

3. Every Delegation Must Narrow Power

Useful agents delegate. They call tools and services, ask other agents for help and split work into smaller steps.

Delegation is not the problem. Delegation that expands power is.

A cost-estimation sub-agent does not need the entire CRM. A retrieval tool does not need to modify a case. A clause-summarisation component does not need to send email. Technical infrastructure access must not become organisational authority.

Minimum criterion: the system can prove that delegation was attenuated, limited and tracked. If delegation expands available power, it creates ambient authority—and ambient authority will eventually be used.

4. Authorisation Follows the Flow

Authorising the start of work is not enough.

An agent produces a chain: it reads, transforms, combines, delegates, synthesises, exports, prepares decisions and feeds other systems. Each step can alter risk. Harmless data can become sensitive when aggregated. A permissible input can violate policy in an output. A summary can lose provenance but retain restrictions. Revocation may arrive after work starts and contaminate results already produced.

Minimum criterion: mandates, constraints and restrictions propagate throughout the flow. Do not control only the entrance. Follow what the agent takes, transforms and leaves behind.

5. Not Acting Is a Valid Result

An enterprise agent must not be rewarded only for completing tasks. It must also be designed to stop.

Stopping is not failure when authority is missing. Requesting evidence is not failure when risk is ambiguous. Escalation is not bureaucracy when an action changes critical state. Refusing an output is not poor UX when the data must not leave.

Productivity without correct blocks is automated risk.

Minimum criterion: stop, evidence request, permission reduction and escalation are expected, observable and measurable outcomes. If success only means “did what it was asked”, the agent is obedient. It is not governed.

6. Audit Examines Actions, Not Summaries

The agent's final report is not evidence.

It may be useful and explanatory, but governance requires the decision chain: inputs, data touched, tools called, mandate invoked, policies evaluated, authorisations granted or denied, evidence produced, outputs generated, constraints inherited, exceptions opened and revocations applied.

Minimum criterion: every relevant action leaves a verifiable trace independent of the agent's account.

If “explain why this was acceptable” is your primary evidence, you do not have an audit. You have testimony from an interested party.

7. The Controller Must Be Governed

When agents work at machine speed, control must accelerate. Dynamic control is not neutral merely because it is technical. It sets thresholds, applies friction, blocks work, grants exceptions, interprets context, uses corporate world models and sometimes judges other models.

That power must be governed.

Minimum criterion: the agent control plane is explainable, verifiable, correctable and bounded.

Explainable, so decisions can genuinely be challenged.

Verifiable, so it survives technical and regulatory review.

Correctable, so every false positive does not become a permanent exception.

Bounded, so dynamic reasoning cannot rewrite security principles on its own.

The controller must not become a black box more powerful than the agent it is supposed to limit.

The Minimum Threshold

These seven principles do not make agent adoption simple. They make its true cost visible.

If an enterprise agent project cannot answer the following questions, it is not necessarily worthless. It is simply not ready to act on critical processes.

Where does blocking policy live?

Which authority accompanies the action?

Does delegation narrow or expand?

Do constraints follow data, summaries and outputs?

Can the system stop?

Does audit see actions or only narratives?

Who controls the controller?

They are uncomfortable questions, which makes them useful.

AI agents will enter enterprises anyway. Some will come through the front door with declared architecture, accountability, budget and review. Others will slip in through purchased tools, local automations and temporary experiments that became production through exhaustion.

Maturity will not be demonstrated by saying “we have agents”. It will be demonstrated by proving that, when they act, someone designed the boundary within which they may do so.

An AI agent is not a chat interface with hands.

And if its hands are inside enterprise systems, security cannot merely watch them move.

Appendix

Minimum Operational Standards

This appendix does not turn the ebook into a product manual—the fastest way to make it obsolete before distribution. It brings the architectural thesis down to the level where a team can discuss a real implementation without hiding behind comfortable words such as governance, security, control, escalation and audit.

Enterprise agents fail not only because a grand theory is missing. They fail because a sound theory is badly connected to systems with queues, timeouts, caches, open tickets, dirty roles and people who still need to work.

The minimum operational threshold is this: every control must know whether it is blocking, deferrable or observational. Every escalation must know what it freezes and for how long. Every untrusted input must remain separate from the channel that constructs the command.

Blocking and Deferrable Controls

Not every control belongs in the same part of the flow.

Some must be inline even when they add latency: tool authorisation, mandate verification, minimum data classification, mandatory evidence, irreversible-action blocking, export prohibition and separation of incompatible roles.

If an agent is about to send an external message, alter an accounting state, grant access, export customer data or trigger a procedure with real effects, control must precede action. Afterwards, it is no longer control. It is archaeology.

Other controls may be deferred, enriched or streamed: anomaly detection, behavioural scoring, quality review, cost-pattern analysis, statistical false-positive review, tuning recommendations and correlation with past incidents. They matter, but must not pretend to be gates if they arrive after the train has left.

The distinction is blunt: a blocking control must be able to say no before the effect. A deferrable control must improve the system after the effect. Mixing them produces architectures that are slow where speed was possible and permissive where friction was necessary.

The Latency Budget

An agentic control plane has a cost: LLM inference, schema validation, policy evaluation, token exchange, identity checks, tool calls and verifiable logging. In a multi-agent flow, each delegation may introduce another decision point.

“Add an external policy engine” is correct but incomplete. How often is it called, at which granularity and within what budget?

A reasonable separation uses three classes:

- **Hard decisions:** few, synchronous and unavoidable—critical actions, sensitive data and irreversible state changes.
- **Cacheable decisions:** short-lived authorisation tied to mandate, scope and a small TTL. Not "this agent may do X forever", but "this agent may perform this family of actions for this task, for a few minutes, within these limits".
- **Observable decisions:** assessments that improve control, audit and behaviour but need not block each step.

Caching is acceptable only when it reduces latency without becoming permanent entitlement. It requires short expiry, clear revocation, inherited constraints and traceability. If you cannot explain why a cached decision remained valid at action time, you did not optimise. You hid risk behind better performance.

Human-in-the-Loop Without Theatre

"Ask a human" sounds cautious. Often it merely moves the problem.

Humans do not live at agent speed. They answer in minutes, hours or never. Escalate every doubt and the system creates approval fatigue. Eventually operators stop checking and click. Human approval becomes reflex, adding a slow step to an automated decision.

A serious HITL design defines at least five things:

- **Snapshot:** which agent state is frozen at escalation time.
- **Suspended scope:** which action remains blocked and which work may continue without increasing risk.
- **Timeout:** how long the system waits before the mandate expires, narrows or returns to a queue.
- **Rollback:** which partial changes must be reversed, compensated or marked incomplete.
- **Accountability:** who approves what, with which minimum information and recorded evidence.

The approval interface should show the delta, not ask a human to reread the agent's entire history: requested action, risk, relevant policy, available evidence, consequence of yes, consequence of no and approval expiry.

If the person must trust a summary produced by the same agent requesting permission, circular audit has returned. Approval must rest on verifiable data, not a persuasive narrative.

State, Suspension and Resumption

A suspended agent is not a romantic pause in a workflow. It is distributed state.

It has read data, produced intermediate reasoning and perhaps created drafts, called non-critical tools, opened subtasks or passed context to other agents. At escalation time, the system must know what remains valid and what becomes suspect.

Resumption should not mean a generic “continue where you stopped”. It should be a new constrained decision: resume this mandate, with this approval, during this window, without revoked data or expired tools, recording that state was suspended and reactivated.

The point is not complexity for its own sake. It is preventing HITL from becoming a side door through which stale state returns as though still fresh.

Separation Between Data and Commands

The separation between data and control channels must be implemented, not proclaimed.

Material read by the agent remains untrusted even when it originates inside the enterprise. Internal email can be compromised. Tickets can contain hostile text. RAG documents can carry disguised instructions. Wiki pages can be stale, manipulated or simply wrong.

Practical boundaries reduce the risk:

- tool parameters are constructed from schema and mandate, not free text copied from data;
- operational instructions arrive through channels separate from analysed documents;
- the policy engine receives typed facts, provenance, classification and an action request, not a narrative prompt;
- outputs to external systems pass through allowlists of fields and destinations;
- RAG retrieval preserves provenance, classification and usage constraints, not only textual content.

The agent may use data to propose. The system must use verifiable structures to authorise. That is the difference between an assistant reading documents and a process that allows documents to change the rules.

The Pre-production Checklist

Before placing an agent on real processes, the team should answer these questions without motivational slides.

Principle	Control question	Blocking criterion
Enforcement	Where are the blocking controls?	No critical action enters production if the block arrives after the effect.
Latency	What is the maximum budget for a critical decision?	An undeclared budget means the control will be bypassed or disabled.
Cache / JIT	Which authorisations may be cached, with what TTL and revocation?	Cache without short expiry and verifiable revocation is permanent permission.
HITL	What happens if the human does not answer?	Escalation without timeout and accountability creates a queue, not control.
State	Which state is frozen during escalation?	If state cannot be snapshotted, resumption cannot be audited.
Rollback	Which partial actions are reversible, compensatable or irreversible?	Irreversible action without prior evidence: no-go.
Data/control separation	Can data read by the agent influence command structure?	If yes, the data channel contaminates the control channel.
Policy input	Does the policy engine receive verifiable facts or interpreted text?	Free text as normative input: no-go for critical processes.
Supervision	Who measures false positives, false negatives, cost and operational friction?	No threshold owner means no governance of the controller.

Without these answers, the agent may work beautifully in a demo. It is not ready for a critical enterprise process.

The aim is not to block adoption. It is to prevent the first serious architecture being designed after the first serious incident.

Sources

Policy, instructions and tools

Liudas Panavas, Sebastian Minus, Bradley Monton, Derek Ray, Suhaas Garre, Sushant Mehta, Edwin Chen, `HANDBOOK.md: A Benchmark for Long-Context Agentic Instruction Following`, arXiv, 2026. https://arxiv.org/abs/2607.25398v1

`HANDBOOK.md` benchmark repository. https://github.com/surge-ai/handbook

Anthropic, `Introducing the Model Context Protocol`, 2024. https://www.anthropic.com/news/model-context-protocol

OpenHands, open-source project. https://github.com/All-Hands-AI/OpenHands

Zero Trust and Beyond Zero

Rory Ward, Betsy Beyer, `BeyondCorp: A New Approach to Enterprise Security`, USENIX ;login:, December 2014. https://www.usenix.org/publications/login/dec14/ward

Joseph Valente, Michal Zalewski, `Beyond Zero: Enterprise Security for the AI Era`, arXiv / ACM Queue, 2026. https://arxiv.org/abs/2605.22985

DOI for the `Beyond Zero` paper. https://doi.org/10.1145/3819083

Heather Adkins, Archana Ramamoorthy, `Going Beyond Zero: A New Paradigm For Enterprise Security`, Google Security Blog, 27 July 2026. https://blog.google/security/going-beyond-zero-a-new-paradigm-for-enterprise-security/

Identity, delegation and authorisation

NIST, `AI Agent Standards Initiative`, official page updated 20 April 2026. https://www.nist.gov/artificial-intelligence/ai-agent-standards-initiative

Takumi Otsuka, Kentaroh Toyoda, Alex Leung, `AI Identity: Standards, Gaps, and Research Directions for AI Agents`, arXiv, 2026. https://arxiv.org/abs/2604.23280

Krti Tallam, `Authorization Propagation in Multi-Agent AI Systems: Identity Governance as Infrastructure`, arXiv, 2026. https://arxiv.org/abs/2605.05440

Sai Varun Kodathala, `aiAuthZ: Off-Host, Identity-Bound Authorization for AI Agents`, arXiv, 2026. https://arxiv.org/abs/2607.05518

M. Llambi-Morillas, D. Fernandez-Fernandez, `Toward cryptographically verifiable authorization for autonomous AI agents`, arXiv, 2026. https://arxiv.org/abs/2607.21325

Supervision and agentic security

Google DeepMind, `Securing internal systems against increasingly capable and imperfectly aligned AI`, 2026. https://deepmind.google/blog/securing-the-future-of-ai-agents/

Anshuman Chhabra, Shrestha Datta, Shahriar Kabir Nahin, Prasant Mohapatra, `Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges`, arXiv / IEEE Access, 2026. https://arxiv.org/abs/2510.23883

Juhee Kim, Xiaoyuan Liu, Zhun Wang, Shi Qiu, Bo Li, Wenbo Guo, Dawn Song, `The Attack and Defense Landscape of Agentic AI: A Comprehensive Survey`, arXiv, 2026. https://arxiv.org/abs/2603.11088

Operational note
If you are designing AI agents for real processes, the useful question is not “which model should we use?”. It is “what authority are we giving it, and who can prove that authority when something breaks?”. Defilippo srls works on enterprise integration, industrial cloud platforms, security and high-scalability architectures.