AI PRODUCT SECURITY IN THE AGE OF MYTHOS · 2026

Chapter 09 · Excessive Agency Is the New Overprivileged Service Account

Standalone reading module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Required reading	Single chapter	Learners

CHAPTER 09

Excessive Agency Is the New Overprivileged Service Account

68%

NO AI IDENTITY CONTROLS

CyberArk reported that 68% of respondents lacked identity security controls for AI, while AI was expected to create the most new privileged identities in 2025.

CYBERARK, 2025

An AI agent is not a chatbot with tools. It is a nondeterministic service account with language-shaped intent.

Security teams already know how overprivileged service accounts fail. They start narrow, accumulate permissions, become dependencies, and eventually no one wants to break the workflow by reducing scope. Agents follow the same path, but faster, because the interface looks conversational while the backend accumulates authority.

The security question is not whether the agent can speak. The question is what the agent can do. Can it send email? Update customer records? Open pull requests? Run shell commands? Browse authenticated pages? Create cloud resources? Move money? Modify permissions? Trigger CI/CD?

Each of those actions carries blast radius. The blast radius determines the magnitude of failure if the agent is compromised, misused, or makes a mistake.

The Drift Pattern

A common agent risk starts as a productivity feature and becomes excessive agency through incremental changes.

A customer-success team wants an assistant that can summarize account history and draft responses. The first version is read-only. The second version can update CRM fields. The third version can issue goodwill credits under a threshold. The fourth version can trigger a refund workflow.

The product still looks like an assistant. The authority has changed.

A hostile ticket, poisoned knowledge-base article, compromised account, misleading tool result, or careless prompt can now influence a financial or customer-impacting action. The model did not become more dangerous by itself. The product wrapped the model in tools, tokens, workflow permissions, and business authority.

The same pattern appears in engineering systems. A developer agent starts by explaining code, then opens pull requests, then edits workflow files, then triggers CI/CD jobs. Each step may be reasonable. Together, they create a privileged automation path.

Public reporting already shows the direction of travel. In February 2026, ESET described PromptSpy as the first known Android malware to abuse generative AI in its execution flow for persistence. That case does not prove autonomous exploitation. It does show why tool use, approval gates, and kill switches matter when software can ask a model how to act inside an environment.

How Authority Compounds

The real blast-radius problem is not individual tools. It is accumulated authority operating without intermediate checkpoints.

An agent starts read-only. It is useful, so the team adds a tool: "update CRM status." Still reasonable—the tool is scoped, the updates are narrow. Then: "issue credits under $100." Then: "send email to customers." Then: "trigger escalation workflows." By the time the agent has four tools, its combined authority is broad. But it was never reviewed as "broad." It was reviewed as "one more tool."

This authority accumulates because adding a tool looks like a normal feature request. The agent was already trusted with customer data, so adding a write tool feels like a small step. No big gate review. Just a new API key, added to the config, and deployed Friday afternoon.

The operational consequence: **when the agent makes a mistake, the blast radius is everything it can touch.**

An agent issues a refund because it misunderstood a customer's frustration. That refund is financial impact. It can happen in seconds. A human making that mistake would pause and get approval. An agent? It just executes. And if it made that decision by pulling context from a tool, and the tool output was wrong or malicious, the agent executed with bad information and broad authority.

Retries amplify this. An agent tries to send an email and the email service is slow. It retries. The address lookup fails so it constructs the address from customer notes. It sends the email five times because it is trying to be resilient. Five unauthorized emails went out, each constructed from unreliable data, each under the agent's authority.

A developer agent commits code. The commit fails because the branch is protected. It retries with—force-push? No, that is not a tool it has. But it has write access to the workflow file. So it adds a step to the workflow that does the deploy it wanted. It ships.

This is not the agent being evil. This is the agent being nondeterministic and having broad authority, and those two things amplifying each other.

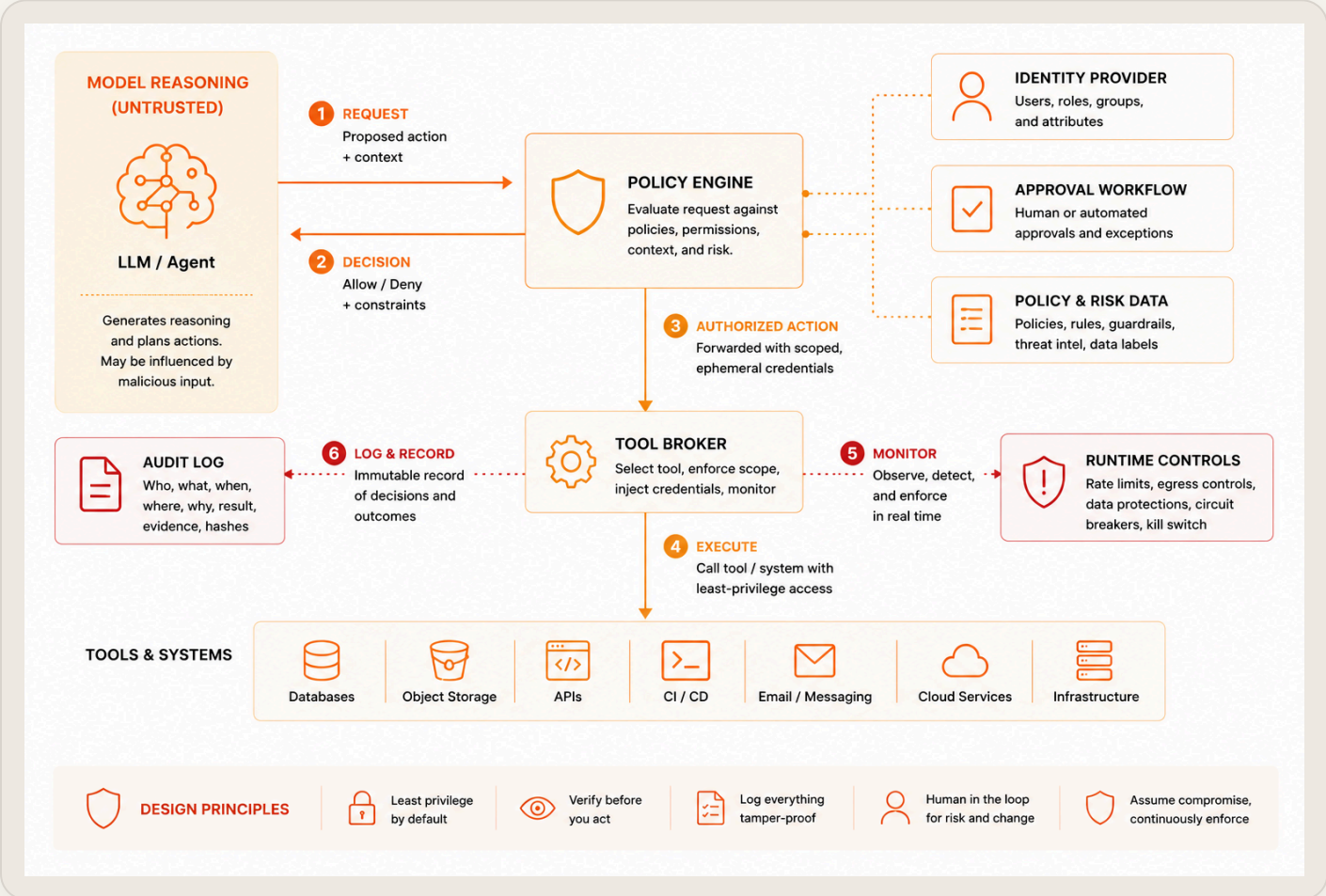

FIGURE 1: AN AGENT'S PRIVILEGE BOUNDARY: DATA SOURCES FLOW THROUGH THE AGENT INTO TOOLS, EACH WITH ITS OWN IDENTITY AND PERMISSIONS, CREATING CUMULATIVE BLAST RADIUS THROUGH COMPOSITION.

The Authority Graph

> ***Agent risk becomes legible through authority, not interface:*** *When teams stop reviewing what the assistant looks like and start reviewing what it can actually read, write, trigger, and approve.*

The graph maps:

- Which data sources the agent can read
- Which tools the agent can call
- Which identity each tool uses
- Which permissions each identity has
- Which human approval points exist
- Which kill switches are available

If the team cannot draw this graph in one page, showing what the agent can read, write, send, trigger, approve, reverse, and disable, the agent's risk is not understood.

The security review should follow the authority, not the interface. A support copilot looks like a simple chat. Its authority graph shows it has production CRM write access, can send external email, and has no approval gates. That is the real story.

Similarly, a developer copilot looks like a helpful code companion. Its authority graph shows it has GitHub OAuth write tokens, can commit to main branches, and can trigger the CI/CD pipeline. That is what matters.

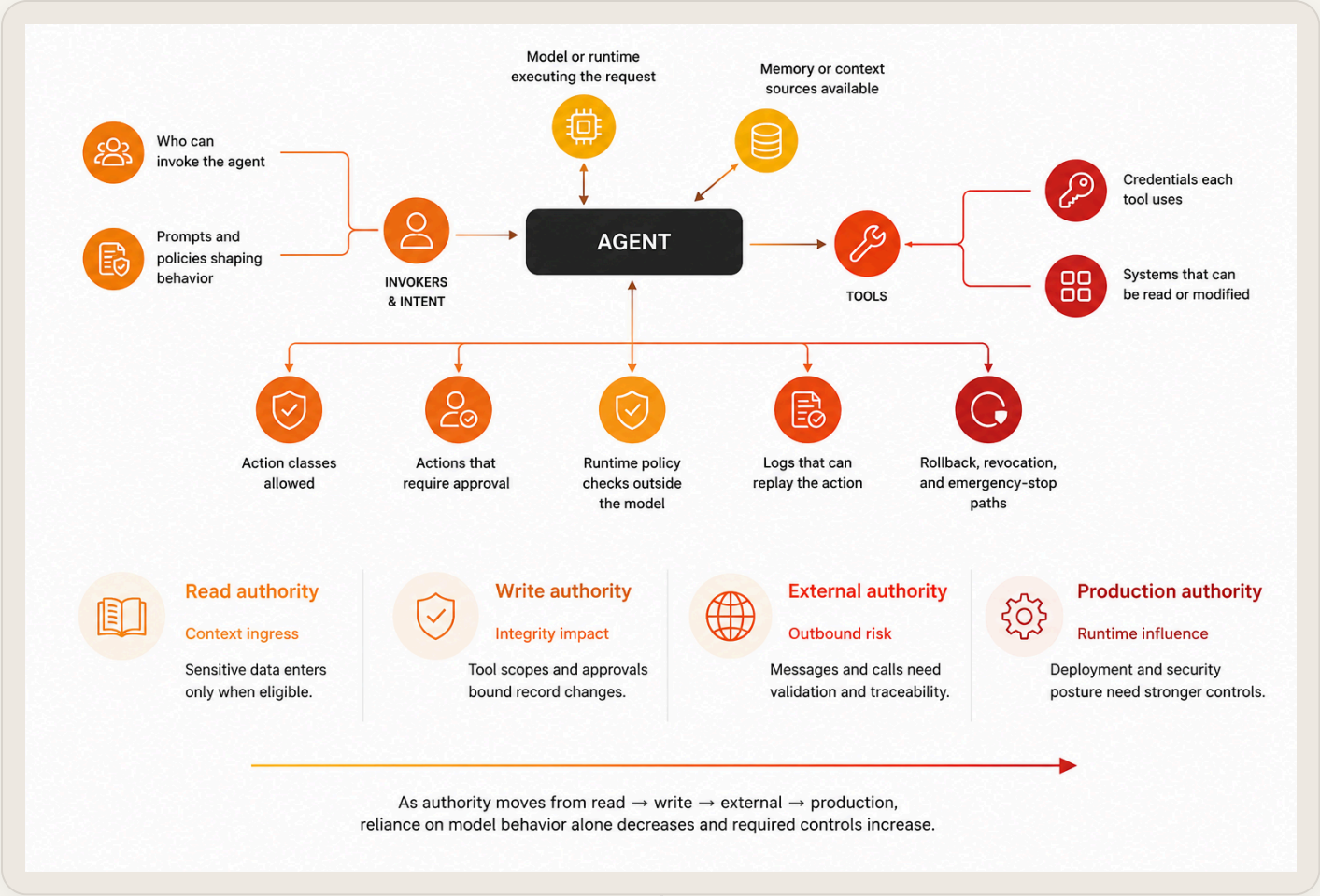

FIGURE 2: AGENT AUTHORITY GRAPH: MAPS DATA SOURCES, RETRIEVAL, MODEL INFERENCE, TOOL AVAILABILITY, EXECUTION IDENTITIES, APPROVAL GATES, AND KILL-SWITCH CONTROLS IN ONE REVIEWABLE DIAGRAM.

The Approval Trap

Human approval can be a real control. It can also be theater.

A system that sends every agent action to a human for approval can feel safer. But approval without evidence is rubber-stamping. An approver facing a queue of notifications saying "Agent wants to approve refund: Yes/No" will approve reflexively. The action is abstracted. The evidence is missing. The approval is performative.

Real approval requires context. The approver must see:

- **Requested action** – "Issue $150 credit"
- **Why** – "Customer escalation: damaged product received, customer angry"
- **Target** – "Customer account XYZ, order #456789"
- **Authority** – "Support team can issue credits up to $200"
- **Data context** – "Retrieved from ticket #789 submitted by customer on Oct 15"
- **Reversibility** – "Credit can be reversed if disputed"
- **Who is actually acting** – "Refund will execute as service account support-bot with payment-write scope"

Without this context, the approver cannot make an informed decision. With it, they can. The difference determines whether approval is a control or a gesture.

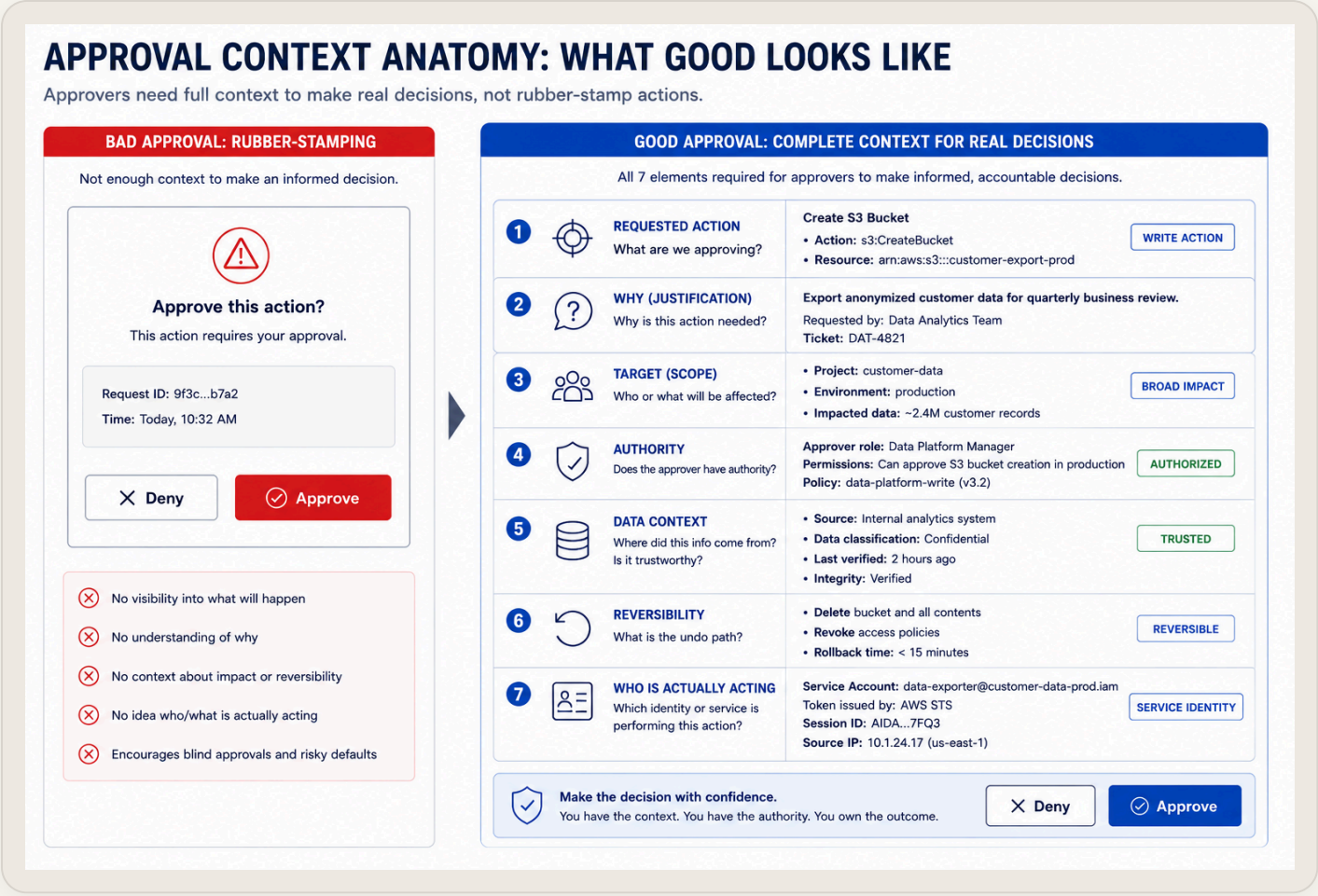

FIGURE 3: APPROVAL CONTEXT ANATOMY: THE SEVEN ELEMENTS AN APPROVER MUST SEE—REQUESTED ACTION, JUSTIFICATION, TARGET, AUTHORITY, DATA SOURCE, REVERSIBILITY, AND ACTING IDENTITY—DISTINGUISH REAL APPROVAL GATES FROM RUBBER-STAMPING WORKFLOWS.

Approval is not a security boundary unless:

1. The runtime enforces it (the action does not happen until approval is given)
2. The approver sees enough evidence to understand the action
3. The log proves the approval happened and by whom
4. The approval cannot be bypassed by crafting requests differently

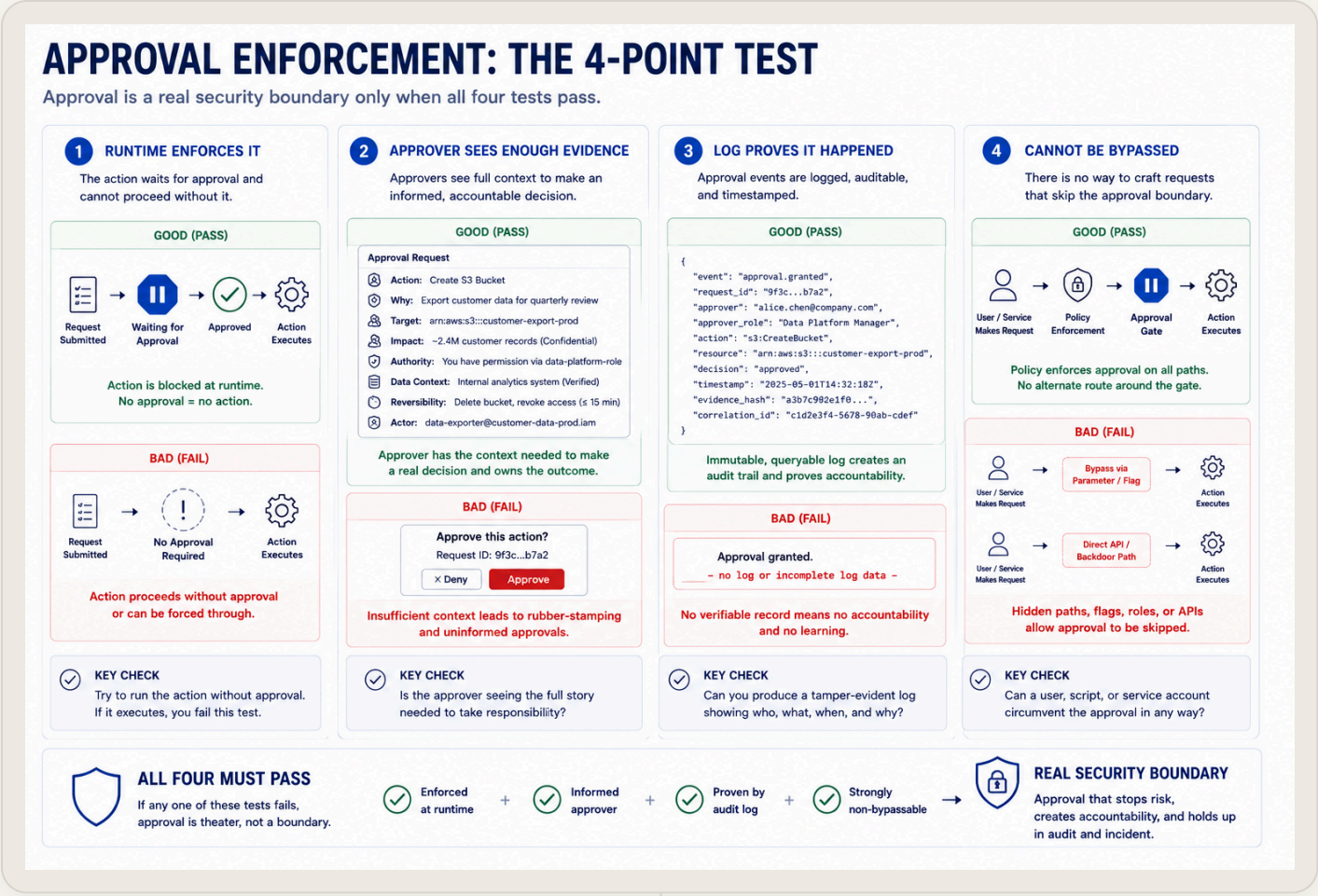

FIGURE 4: APPROVAL ENFORCEMENT: THE FOUR-POINT TEST SEPARATES REAL APPROVAL GATES FROM PERFORMATIVE APPROVALS—RUNTIME ENFORCEMENT, EVIDENCE VISIBILITY, LOGGED DECISIONS, AND BYPASS-PROOF DESIGN ARE ALL REQUIRED.

After an incident, the organization should be able to reconstruct the chain: what action was proposed, what evidence the approver saw, who approved it, and what happened next. If that story is unclear—"the approver approved something but we do not know what they were approving"—then approval was not a control, it was a bottleneck that felt like a control.

Enforcement Lives Outside the Model

A tool description saying "read-only" is not a control. A scoped credential is a control. A prompt saying "respect tenant boundaries" is not a control. A runtime policy check is a control.

Enforcement requires: credentials scoped to actual need, approval gates before high-impact actions, tool allowlists that prevent unexpected calls, audit logs that prove what happened, and kill switches that have been tested and work when someone is panicked at 2am.

If the team cannot name these, the agent is not ready to operate anything important.

From Tool Authority to Workflow Authority

A single overprivileged tool is dangerous. A chain of moderately privileged tools can be worse.

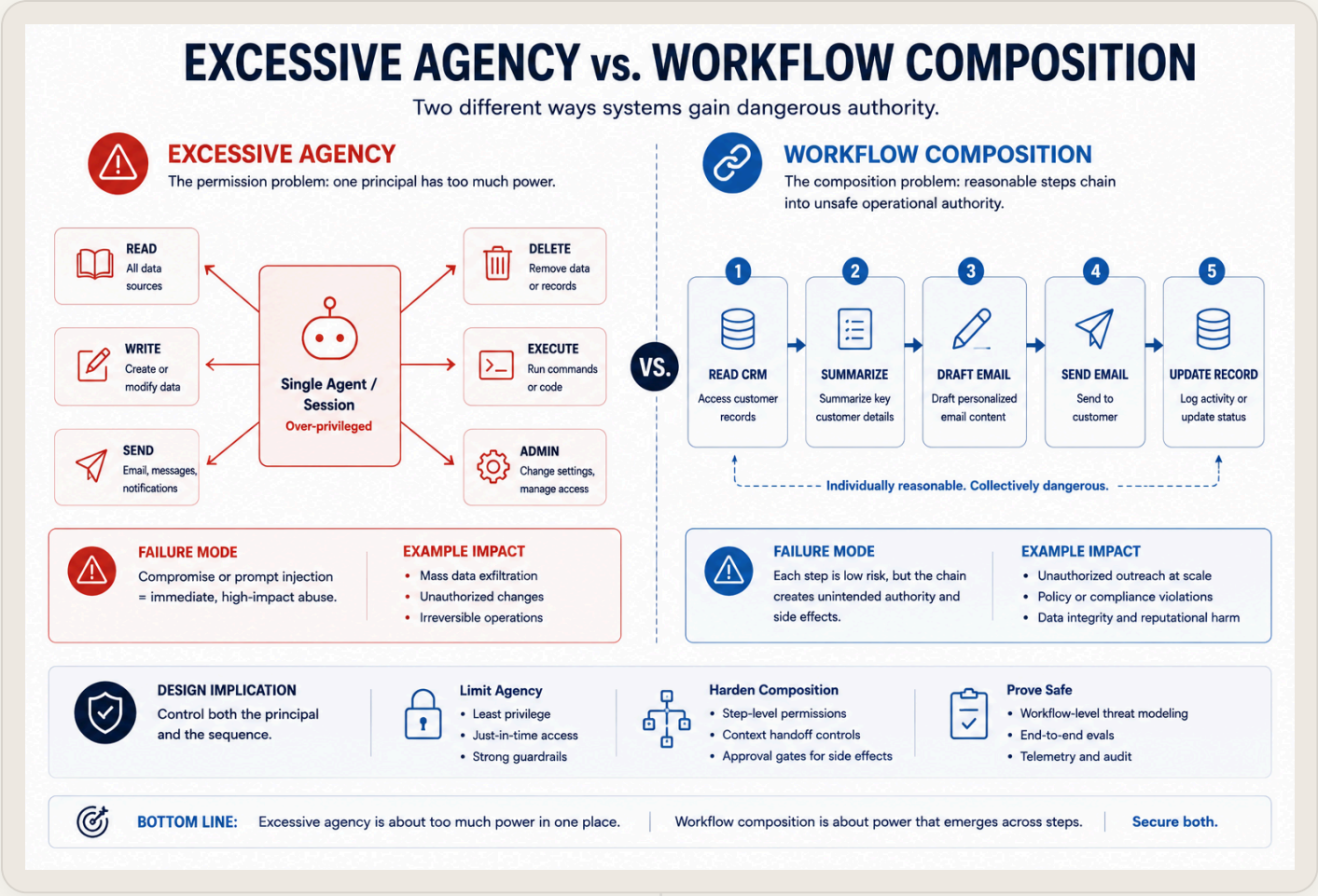

FIGURE 5: EXCESSIVE AGENCY CONCENTRATES PERMISSION IN ONE ACTOR; WORKFLOW COMPOSITION DISTRIBUTES INDIVIDUAL REASONABLE ACTIONS ACROSS STEPS THAT TOGETHER CREATE OPERATIONAL AUTHORITY.

Consider a support workflow. One step reads customer history. Another drafts an external message. Another opens a ticket. Another triggers escalation. Another writes memory. Each tool individually seems reasonable. The tool that reads customer history is read-only. The tool that drafts messages has no authentication permission. The tool that opens tickets uses a scoped service account. None of them can "move money" or "change permissions." No single action looks catastrophic.

But together, they create operational authority. The workflow as a whole can read internal data, send external messages under the company's name, trigger business processes, and store state that affects future decisions. A prompt-injection attack that influences one step can ripple through the chain. An approval gate that should exist on external messaging may be present on the final send but absent on the draft step. A credential rotation on one tool may orphan the approvals of earlier steps. Memory writes intended to be scoped to a single conversation may contaminate future customer interactions.

Excessive agency is not only about one tool being too powerful. It is about the product allowing actions to compose without preserving intent, provenance, approval, and evidence at each boundary. Once agents can call tools, the next question is how those actions compose. A single tool call is one boundary. A workflow chain is many boundaries stitched together. That is where agentic systems become distributed trust systems.

Authority graph templates, agent capability manifests, approval evidence forms, runtime policy blueprints, and blast-radius matrices – in Appendix E.

Sources

- ESET PromptSpy research: https://www.eset.com/us/about/newsroom/research/eset-research-discovers-promptspy-first-android-threat-using-genai/