AI SECURITY ENGINEERING HANDBOOK · 2026

Chapter 02 · Architecture and Trust Boundaries

Standalone study module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Study module	Single chapter	Learners

CHAPTER 02

Architecture and Trust Boundaries

HANDBOOK STUDY COMPANION: STUDY FRAME

Use this chapter to build vocabulary, judgment, and role-readiness. Pair it with the Field Guide when you need applied actions, checklists, and control execution.

STUDY FOCUS

STUDY FOCUS	WHY IT MATTERS
How to read AI architecture maps, identify trust zones, classify components, and distinguish data, authority, and evidence flows.	Teams cannot reason about AI risk until they know where trust changes and which boundary enforces the decision.

Study Outcomes

- Map model, app, retrieval, tool, identity, provider, and telemetry boundaries.
- Explain how AI trust boundaries differ from ordinary application diagrams.
- Identify which evidence belongs to each boundary.

DOMAIN MAPPING

RELATED AIPSA DOMAINS	APPLIED NEXT STEP	WORKBENCH INSTRUMENTS	RELATED SERVICES
LLM Application Security, Secure AI Architecture Design	LLM application security	Threat Canvas	AI Product Security Assessment

CERTIFICATION AND ASSESSMENT BOUNDARY

This chapter supports training, diagnostic preparation, scorecards, interviews, and role-readiness evaluation. It does not guarantee credential outcomes.

The most expensive AI security mistakes are architectural. They are expensive not because they are technically complex but because they are discovered late, after the design has shipped to production, after customers are using the system, and after changing the architecture requires re-engineering that the team never budgeted for. A team that asks "where does this design place trust?" before building will almost always produce a more secure system than one that patches controls onto a finished product.

A team that asks "where does this design place trust?" before building will almost always produce a more secure system than one that patches controls onto a finished product.

HANDBOOK

The product security surface extends well beyond the model. Prompts and context assembly, retrieval pipelines, tool and API integrations, authorization and identity controls, and logging infrastructure all sit within the product boundary — each a distinct attack surface that requires independent controls rather than a single perimeter.

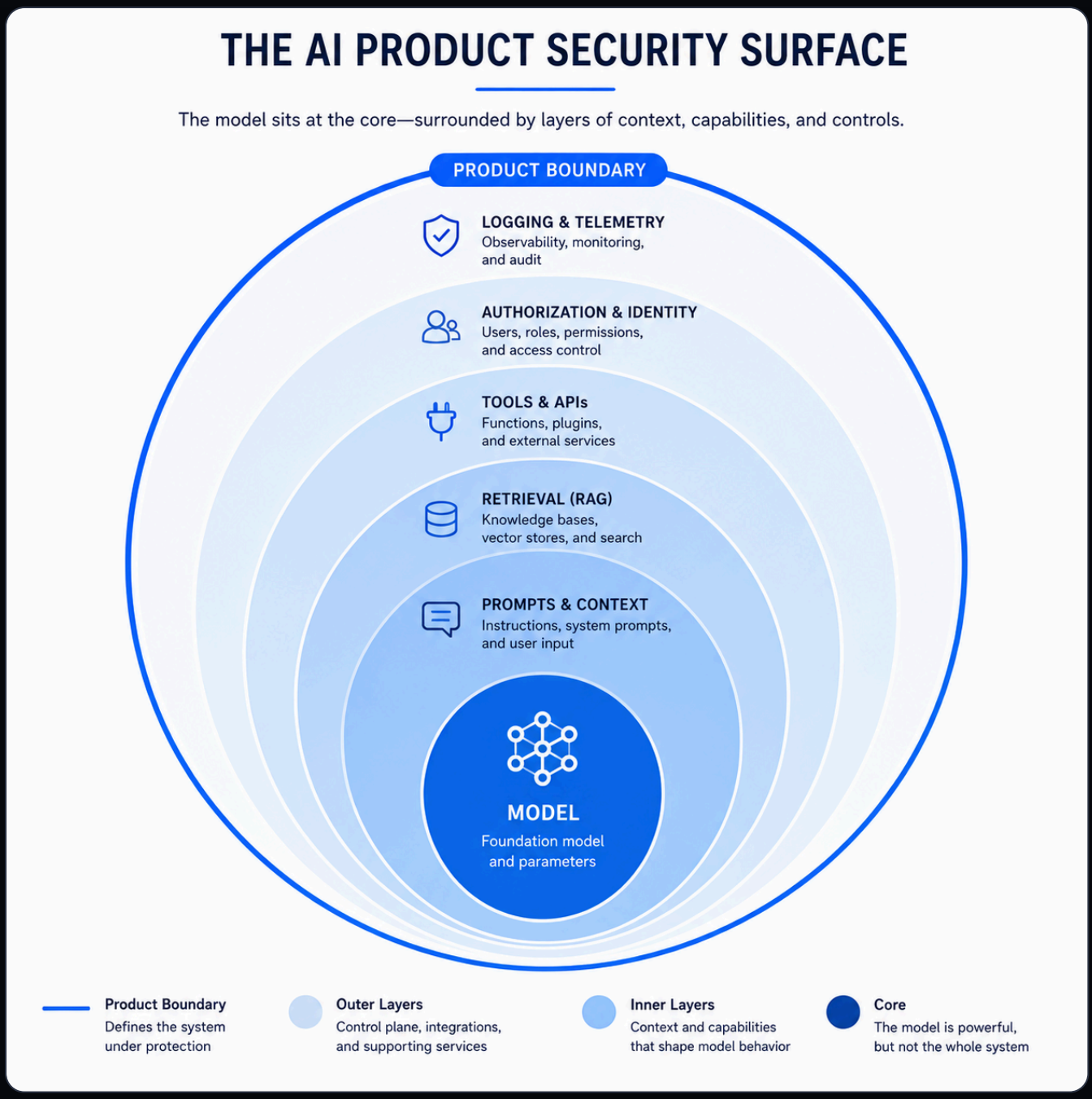

FIGURE 1: FIGURE 4: CONCENTRIC RING DIAGRAM OF THE AI PRODUCT SECURITY SURFACE — FROM THE MODEL CORE OUTWARD THROUGH PROMPTS, RETRIEVAL, TOOLS, AUTHORIZATION, AND LOGGING TO THE OUTER PRODUCT BOUNDARY

CORE CONCEPTS

CONTEXT TRUST TIERS

Every segment entering the model's context should have an explicit trust level and a defined allowed influence on system behavior. System instructions define the application contract. Developer instructions define task boundaries. User input scopes the user's request. Retrieved documents provide evidence. Tool outputs report external state. Conversation history provides session continuity. The architecture must enforce these semantics — retrieved content cannot override policy, tool outputs cannot grant new authority — through structural controls, not through model instruction alone.

DATA PLANE AUTHORIZATION

Authorization must be enforced before data enters the model context. Any design that retrieves first and filters later has already crossed the authorization boundary. Once a privileged document enters the context window, the model has processed it — regardless of what the generated answer shows. The data plane design must enforce user identity, tenant, role, document classification, and purpose before retrieval results are assembled into context. Output filtering operates as a secondary check, not a primary authorization control.

INDEPENDENT DEFENSE LAYERS

Defense-in-depth for AI systems requires layers that do not all fail for the same reason. A system prompt, output filter, and model self-critique may all be compromised by the same adversarial context because they all pass through the same model reasoning. Independent layers operate through different mechanisms: retrieval authorization enforces access before context assembly; runtime tool policy enforces permissions before execution; schema validation operates on structured output; approval gates involve deterministic human decisions; release gates operate before deployment. The architecture should assume the model can be wrong and still prevent unacceptable outcomes.

FALLBACK PATH SECURITY INVARIANTS

AI systems degrade, fail over, switch providers, serve cached answers, or fall back to simpler workflows under error conditions. Each fallback path must preserve the security properties of the primary path: authorization checks, logging coverage, rate limits, approval requirements, and data classification enforcement. A fallback that bypasses retrieval filters or skips tool policy is not resilience — it is an alternate attack surface that receives less scrutiny than the primary path. Security invariants must be specified as explicit requirements for every fallback route.

AGENT BLAST RADIUS AS A DESIGN CONSTRAINT

Blast radius — the maximum damage a single tool call or action chain can cause — is an architecture problem, not a prompt engineering problem. The credential scope, resource boundaries, and approval thresholds that limit blast radius must be designed at the architecture layer before any tool is connected. Attempting to constrain blast radius through prompt instructions or approval

dialogs after the integration is built is retrofit security. Tool permission boundaries, credential scope, resource type and quantity limits, and reversibility requirements must be specified during design, not added as hardening.

THE PRACTITIONER'S CHALLENGE

The political challenge is timing. Architecture security review is most valuable before decisions are made, but security most often arrives after the prototype is built and the team has momentum behind the existing design. The practitioner must calibrate between redesign recommendations that would require significant re-engineering and control additions that reduce risk while preserving the existing structure. The discipline is identifying which trust decisions are structural — they require architecture change — and which are tactical — they can be addressed through controls added to the existing design.

The structural challenge is that AI system architecture review requires combining security expertise with genuine understanding of how LLM applications, RAG pipelines, and agent orchestration work. A security reviewer who treats all AI systems as web applications will miss context authority failures, retrieval authorization gaps, and agent action chain risks. A reviewer who focuses only on AI-specific failure modes will miss identity management, secrets handling, API security, and logging gaps. Effective architecture review for AI systems requires both layers.

The technical challenge is that AI systems are non-deterministic and context-dependent. Security properties that hold in normal conditions may fail under adversarial context. A design that correctly enforces retrieval authorization on standard queries may fail when an adversarially crafted query is designed to produce unexpected filter behavior. Architecture review must evaluate security properties under adverse conditions, not only under expected inputs. This requires adversarial reasoning about how each design decision behaves when inputs are hostile.

HOW TO APPROACH IT

- Start with a trust model document before reviewing any code. The trust model names each component in the architecture, assigns it a trust level, and defines what decisions it is allowed to make on its own. The model component makes generation decisions, not authorization decisions. The retrieval component enforces data plane authorization, and cannot be bypassed by model output. The tool layer enforces credential-level permissions that cannot be exceeded by model instruction. Writing the trust model first makes it possible to evaluate the implementation against stated security properties rather than against what the team intended.

- Review context assembly as a first-class security surface. Trace how every context segment enters the model's context window: system instructions from where, developer instructions from where, user input with what sanitization, retrieved content with what authorization checks, tool outputs with what provenance labeling, conversation history with what trust level. The review should identify every point where a lower-trust segment might influence model behavior as if it were a higher-trust segment.

- Evaluate data plane authorization independently of output filtering. The question is not "does the model avoid revealing unauthorized data?" but "does unauthorized data enter the context window?" These are evaluated differently. Data plane authorization is tested by attempting unauthorized retrieval requests and verifying that the retrieval layer rejects them before results are returned. Output filtering is tested by sending potentially unsafe queries and verifying that the output is handled correctly. Both tests are needed; they are not substitutes.

- Assess agent blast radius at the design stage. For each tool the agent can call, define the resource class, the credential scope required, the maximum action volume per session, the approval requirements, the reversibility classification, and the logging requirements. Then trace the maximum-blast-radius action chain the agent can execute across its full tool set. If that chain can cause harm that the organization is not prepared to accept, redesign the permission boundaries before the integration is built.

- Review fallback paths with the same security requirements as primary paths. List every condition that routes traffic to a fallback: provider unavailability, rate limiting, error conditions, latency thresholds, and degraded mode configurations. For each fallback path, verify that authorization, logging, rate limiting, approval requirements, and data classification enforcement are preserved. Document the security invariants that must hold across all paths and make them explicit in the architecture.

OUTPUTS AND DELIVERABLES

- The foundation artifacts are the **AI system trust model**, **context trust-tier specification**, and **data plane authorization design**. The trust model names each component, its trust level, and the decisions it can make independently. The context trust-tier specification defines the authority of system instructions, developer instructions, user input, retrieved content, tool outputs, and conversation history — and the structural controls that enforce those authority distinctions. The data plane authorization design specifies which filters must be applied before retrieval results enter context, what happens when authorization metadata is missing, and how the system fails closed.

- The agent and composition artifacts are the **agent permission matrix**, **blast-radius analysis**, and **multi-model trust chain specification**. The permission matrix lists every tool with its permission class, credential scope, resource limits, approval requirements, reversibility classification, and audit requirements. The blast-radius analysis documents the maximum-harm action chain for the current tool set and the design choices that constrain it. The multi-model trust chain specification defines, for each model composition, what trust level is assigned to each model's output as it enters the next model's context.

- The review artifacts are the **architecture security review checklist**, **fallback security invariants document**, and **architecture decision record template**. The review checklist gives security teams a consistent evaluation framework for AI system designs. The fallback invariants document specifies which security properties must hold across all routing paths, including degraded mode. The ADR template captures security-relevant design decisions: what was chosen, what was considered, what security properties were preserved, and what residual risks were accepted.

COMMON FAILURE MODES

- **Model-Enforced Authorization:** The design asks the model to honor authorization boundaries rather than enforcing them at the retrieval or data access layer. Works fine in demo conditions; fails under adversarial context or model behavioral variation. Fix: enforce authorization before context assembly, treat model behavior as one layer of defense, not the primary one.

- **Prompt-Security Architecture:** Every security property is expressed in system prompt language — "do not reveal," "do not call," "always require approval." This creates a design that is one well-crafted adversarial input away from failing. Fix: express security properties as deterministic controls outside the model's reasoning path: retrieval filters, credential scope, runtime policy, schema validation.

- **Fallback Blind Spot:** The primary path has strong security properties; the fallback path was designed for reliability without reviewing its security invariants. Under stress or degraded conditions, the fallback path has weaker authorization, less logging, or different tool permissions. Fix: specify security invariants as requirements for all paths in the architecture, not just the primary path.

- **Blast Radius Added Retroactively:** Tools are integrated with broad credentials for ease of development; blast radius constraints are added as prompts, approvals, and monitoring after an incident signals the risk. At that point, the credential scope still allows the broad action — only the application layer is constrained. Fix: design credential scope, resource limits, and approval placement as architecture requirements before tool integration begins.

IMPLEMENTATION CHECKLIST

- [ ] Write a trust model document naming each component's trust level and decision authority before implementation begins.
- [ ] Specify context trust tiers for every segment entering the model's context window.
- [ ] Verify that data plane authorization is enforced before retrieval results enter context, not after generation.
- [ ] Design agent blast radius constraints at the credential and resource boundary layer, not at the prompt layer.
- [ ] Specify fallback security invariants and verify they hold under each fallback condition.
- [ ] Evaluate independent defense layers to confirm they have different failure modes.
- [ ] Produce an architecture decision record for each security-relevant design choice.
- [ ] Conduct adversarial architecture review: evaluate security properties under hostile input conditions, not only expected inputs.

RELATED READING

- Handbook chapters: Chapter 3 (Threat Modeling) for applying threat analysis to the architecture; Chapter 4 (Prompt Injection) and Chapter 5 (RAG Authorization) for the specific failure modes these architectural decisions address; Chapter 6 (Agentic Permissions) for blast radius and tool permission design.
- Field Guide: Secure AI Architecture Design for trust-boundary checks, fallback control review, and evidence paths.