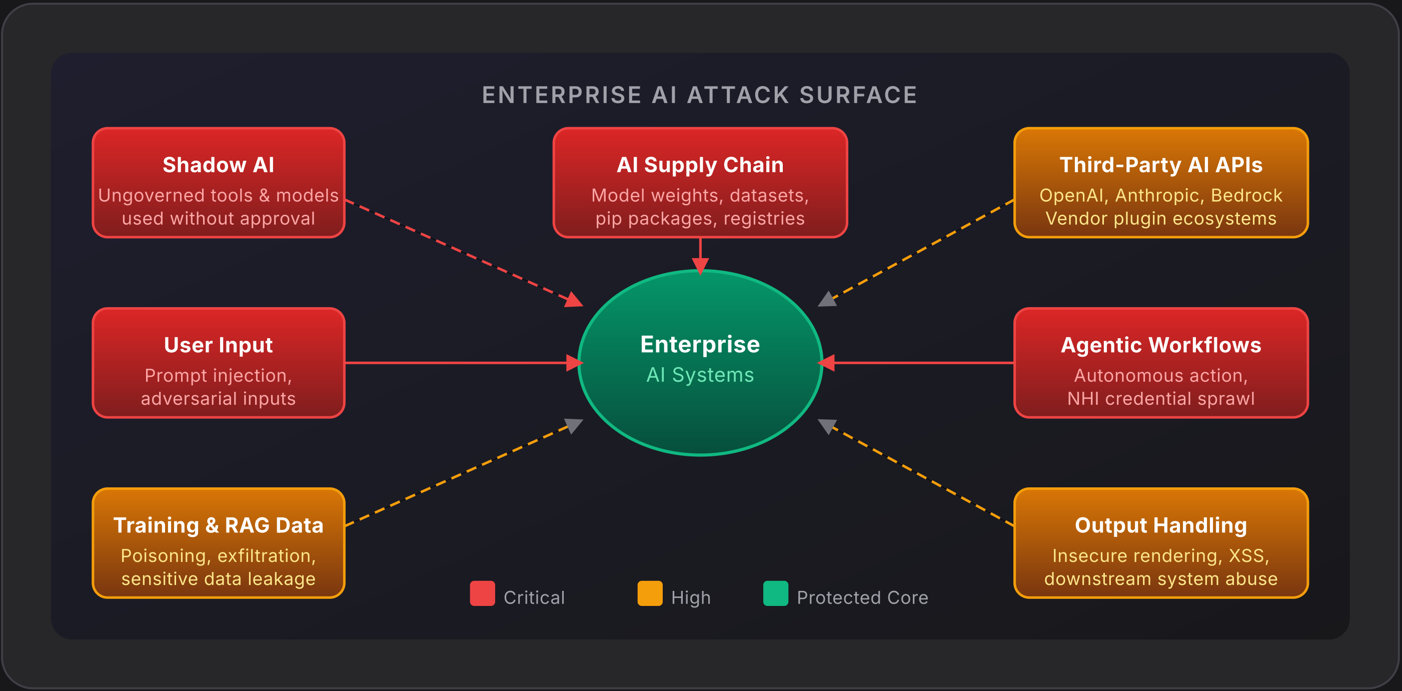
23 FEB 2026 • 22 MIN READ

Securing Enterprise AI Adoption & Agentic Workflows: The CISO Playbook

Every enterprise is adopting AI. Most are doing it faster than their security programs can keep up. This is not a future problem — it is the defining AppSec challenge of right now. When employees use unauthorized AI tools, when DevOps ships LLM-powered microservices without threat models, and when autonomous agents get production credentials, your attack surface grows in ways your existing controls were never designed to see.


ENTERPRISE AI ATTACK SURFACE
Shadow AI
Ungoverned tools & models used without approval
AI Supply Chain
Model weights, datasets, pip packages, registries
Third-Party AI APIs
OpenAI, Anthropic, Bedrock Vendor plugin ecosystems
User Input
Prompt injection, adversarial inputs
Enterprise AI Systems
Agentic Workflows
Autonomous action, NHI credential sprawl
Training & RAG Data
Poisoning, exfiltration, sensitive data leakage
Output Handling
Insecure rendering, XSS, downstream system abuse
🟥 Critical
🟧 High
🟩 Protected Core

Enterprise AI attack surface: seven distinct threat zones converge on AI systems that were never designed to be this interconnected. Most enterprises are currently undefended across at least three of them.

⚠ The Problem Isn't AI — It's Ungoverned AI

The boardroom wants AI adoption. Business units are already using it — with or without security's blessing. A recent wave of enterprise surveys consistently finds the same pattern: a significant fraction of employees are using AI tools that IT has never approved, processing data that security has never classified, in workflows that compliance has never reviewed. This is shadow AI, and it is the enterprise's most pressing ungoverned risk category today.

The security problem is not that AI is inherently dangerous. It is that the adoption outpaces governance, the threat models are missing, and the controls that worked for traditional software are poorly mapped to AI's actual failure modes. Firewalls don't stop prompt injection**OWASP LLM01:2025 — Prompt Injection**A vulnerability where user-supplied content manipulates an LLM's behavior or output in unintended ways, potentially bypassing safety controls, leaking confidential instructions, or causing unauthorized actions.Read official definition ↗. DLP doesn't catch model inversion**Model Inversion Attack (Fredrikson et al., 2015)**An attack that exploits a model's prediction confidence scores to reconstruct sensitive attributes of training data — including face images, patient records, and PII — through repeated targeted queries to the model's API.Read official definition ↗. Endpoint agents don't see what happens inside a third-party API call.

What "Shadow AI" Actually Looks Like in Enterprise

Shadow AI is not just employees using ChatGPT. It manifests across three layers that security teams systematically miss:

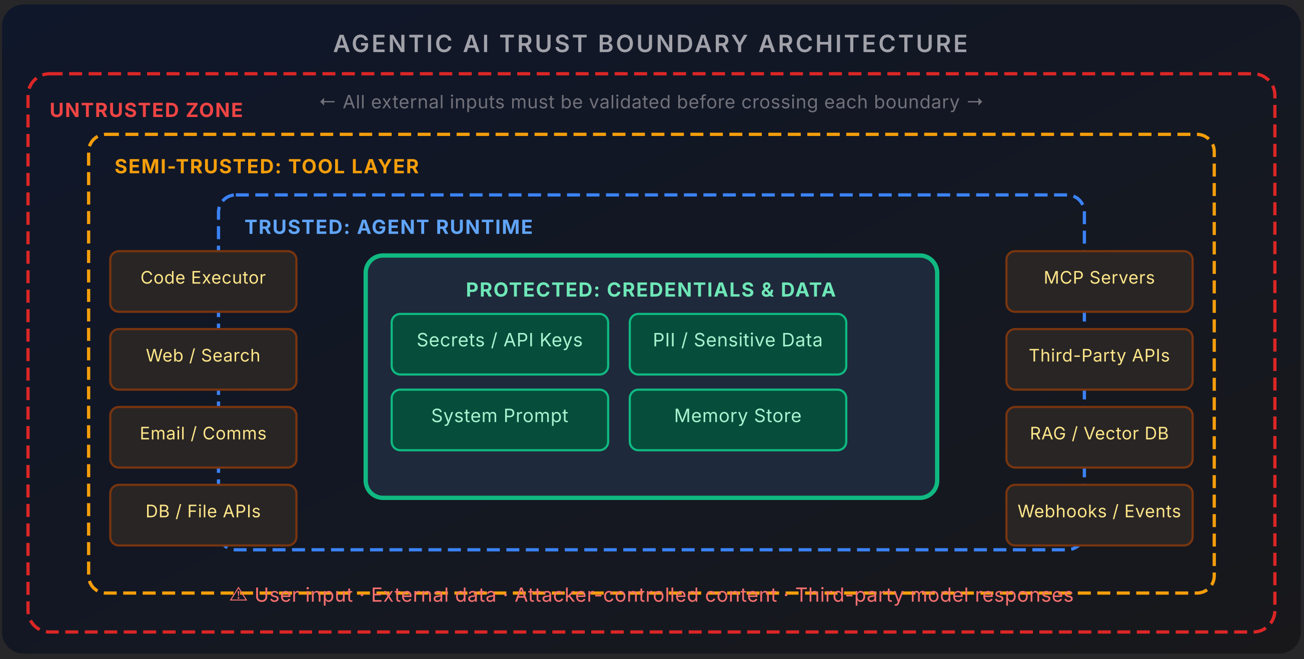
- **Consumer AI tool use:** Employees paste source code, customer data, internal documents, and PII into public AI assistants. The data leaves your perimeter. You never see it in your DLP logs because it's an HTTPS POST to a trusted CDN.

- **Developer-deployed AI:** Engineers integrate AI APIs directly into applications during development — often before a design review, sometimes before there's a data classification. The integration ships to production before AppSec has context.
- **Vendor AI features:** SaaS vendors are quietly enabling AI features in products your enterprise already uses. Salesforce Einstein, Microsoft Copilot, Slack AI, GitHub Copilot — these ingest your data to train or improve their models by default in some configurations. You may have consented to this in a terms-of-service update you didn't read.

Each layer requires a different detection and control strategy. Consumer tool use is primarily a policy and awareness problem, addressable with acceptable use policies, browser proxies, and DLP tuning. Developer-deployed AI requires security integration into CI/CD. Vendor AI requires contract review, data processing agreement (DPA) updates, and vendor risk reassessment.

🤖 Agentic AI: When the Attack Surface Walks Itself

Agentic AI systems — AI that plans, calls tools, takes actions, and persists state across multiple steps — represent a qualitative shift in enterprise risk. A standard LLM integration has a bounded blast radius: bad output, possible data disclosure, maybe a confused user. An agentic system with production credentials and tool access has an unbounded blast radius: data exfiltration, infrastructure changes, communications sent in your name, persistent backdoors in memory stores.



Trust boundary architecture for agentic systems. Each boundary crossing requires explicit validation. Credentials and sensitive data must never flow outward past the Protected zone boundary — yet most enterprise agent deployments have no such boundaries at all.

The trust boundary diagram above describes what security architects should design. What most enterprises have deployed looks nothing like it: agents with unrestricted internet access, credentials stored in plaintext environment variables, no validation of tool responses before passing them back to the LLM, and no audit log of what actions the agent actually took.

The Five Properties That Make Agentic Systems Uniquely Risky

1. **Autonomy without oversight:** Agents execute multi-step plans without human approval at each step. An attacker who influences step one influences all subsequent actions. By the time a human reviews anything, the damage may be irreversible.

2. **Credential accumulation:** Agents require credentials to do their jobs — API keys, OAuth tokens, database credentials, cloud IAM roles. These accumulate in environment variables, memory stores, and session state. A compromised agent is a compromised credential store.

3. **Instruction ambiguity:** Agents interpret natural language instructions. "Send the report to the team" is ambiguous. An attacker who can influence the agent's context (via poisoned RAG retrieval, malicious tool output, or injected system context) can redefine what "the team" means or what "the report" contains.

4. **Memory persistence:** Agents maintain state across sessions via vector databases and persistent memory stores. An attacker who successfully injects into agent memory doesn't just affect the current session — they affect all future sessions that retrieve that memory.

5. **Tool chaining amplification:** Individual tools may be low-risk. Chained together autonomously, they enable high-impact outcomes. Read-access to a vector store + ability to call a web API + email tool = data exfiltration channel. No single tool grants exfil capability, but the chain does.

Agentic Threat Modeling: STRIDE-LM Extensions

Standard STRIDE was designed for human-triggered software interactions. Agentic systems require four additional threat categories:

- **Planner Subversion (PS):** Manipulation of the agent's goal-setting or next-action decision process. Vectors: system prompt injection, poisoned tool descriptions, manipulated memory retrieval. Impact: attacker redirects agent's entire action sequence.
- **Tool Weaponization (TW):** Agent is induced to use legitimate tools in malicious ways. Vectors: crafted tool responses, indirect prompt injection**OWASP LLM01:2025 — Indirect Prompt Injection**A variant of prompt injection where malicious instructions are embedded in external content (documents, web pages, tool outputs) that the LLM retrieves and processes, causing it to act on attacker-controlled directives without the user's knowledge.Read official definition ↗ via retrieved documents. Impact: legitimate tool calls achieve attacker objectives (exfil via email tool, SSRF via search tool).

- **Persistence Implant (PI):** Attacker plants instructions or data in the agent's memory store that persist across sessions. Vectors: memory poisoning, RAG index poisoning. Impact: persistent backdoor that survives session termination.
- **Cascading Authorization (CA):** Agent uses its legitimate credentials to authorize subsequent actions that exceed intended scope. Vectors: over-privileged IAM roles, ambiguous authorization scope. Impact: privilege escalation through the agent as proxy.

Trust Architecture for AI Systems

The trust boundary diagram in the previous section describes the target state. Getting there requires deliberate architecture decisions: which components sit in which trust zone, what validation happens at each boundary crossing, and how credentials flow (or don't) between zones. Most enterprises deploy agentic systems with no trust zoning at all — every component implicitly trusted, credentials freely shared, tool outputs passed directly back to the LLM without inspection.

Zero-trust principles apply to AI systems just as they apply to network architecture: never trust, always verify, enforce least privilege at every boundary. The implementation is different because the "boundary" is often a function call or a prompt, not a network packet — but the discipline is identical.

The Four Trust Zones and Their Enforcement Requirements

Every component in an agentic system falls into one of four trust zones. Zone assignment determines what validation is required before the component can interact with components in other zones.

Zone	Components	Trust Level	Boundary Enforcement
Protected	Secrets, API keys, PII, system prompt, memory store	Unconditional	No outbound data flow. Credential broker only. Encryption at rest + in transit. Append-only access log.
Trusted	Agent runtime, orchestrator, planner, LLM inference endpoint	Conditional	Verify integrity at startup. Authenticate all outbound calls. Log all tool invocations with full parameters.
Semi-Trusted	Tool layer: APIs, MCP server**MCP Server (Model Context Protocol)**A server implementing Anthropic's Model Context Protocol that exposes tools, resources, and prompts to AI agents. MCP servers are a critical trust boundary — compromised or malicious MCP servers can weaponize agents against their operators.Read official definition ↗s, RAG/vector DB, code executor, webhooks	Verify per call	Validate all responses before passing to agent. JSON schema enforcement. Instruction-detection classifier on all retrieved content. Allowlist of approved tool endpoints.

Zone	Components	Trust Level	Boundary Enforcement
Untrusted	User input, external data, third-party model responses, attacker-controlled content	Never trust	Sanitize before any zone crossing. Run injection classifier. Enforce length and encoding limits. Never allow direct LLM context injection without validation.

OPA Policy Enforcement: Trust Zone Boundary Control

Open Policy Agent (OPA) with Rego lets you express trust zone enforcement as code that's reviewable, testable, and auditable. The following policy enforces the key invariants: no raw credentials in tool calls, no unvalidated untrusted content injected into agent context, and required logging for all Protected zone access.

trust_zones.rego — OPA policy enforcing AI system trust boundary invariants

```
package ai.trust_zones

import future.keywords.in

# —— Protected Zone: credentials must never appear in tool call parameters ——
deny[msg] {
    input.action == "tool_call"
    tool_param := input.parameters[_]
    is_credential_pattern(tool_param)
    msg := sprintf("VIOLATION: Credential pattern detected in tool call parameter '%v
}

is_credential_pattern(val) {
    val_str := sprintf("%v", [val])
    patterns := ["sk-", "api_key=", "secret=", "token=", "password=", "Bearer ", "AWS
    pattern := patterns[_]
    contains(lower(val_str), lower(pattern))
}

# —— Semi-Trusted Zone: all retrieved content must be scanned before injection ——
deny[msg] {
    input.action == "inject_retrieved_content"
    not input.metadata.injection_scan_passed == true
    msg := "VIOLATION: Retrieved content must pass injection classifier before LLM co
}

# —— Protected Zone: all access must be logged ——
deny[msg] {
    input.action in ["read_secret", "access_pii", "read_system_prompt", "read_memory'
    not input.metadata.audit_logged == true
    msg := sprintf("VIOLATION: Protected zone access '%v' must be audit-logged before
}

# —— Untrusted Zone: user input must be sanitized before any zone crossing ——
deny[msg] {
    input.source == "user_input"
    not input.metadata.sanitized == true
    input.action != "receive_raw_input"  # Raw receive is always allowed; subsequent
    msg := "VIOLATION: Unsanitized user input cannot cross trust zone boundary."
}

# —— Trusted Zone: all outbound tool calls must be to the approved allowlist ——
deny[msg] {
    input.action == "tool_call"
    not input.tool_endpoint in data.approved_tool_endpoints
    msg := sprintf("VIOLATION: Tool endpoint '%v' is not in the approved allowlist.",
}

# Policy is satisfied when no violations
allow {
    count(deny) == 0
}
```

Security Architecture Review: 12-Question Checklist

Before any agentic system touches production data, it must pass this architecture review. Each question maps to a specific trust zone enforcement requirement. A "No" answer on any question is a blocking finding — it must be resolved before deployment.

1 **Trust zone assignment:** Has every component in the system been explicitly assigned to a trust zone (Protected / Trusted / Semi-Trusted / Untrusted)?

2 **Credential isolation:** Are all secrets accessed through a credential broker (Vault, AWS Secrets Manager, GCP Secret Manager)? Are raw credentials absent from environment variables, logs, and agent context?

3 **Tool endpoint allowlist:** Is there a defined, reviewed allowlist of all external endpoints the agent is permitted to call? Is calling an unlisted endpoint blocked at the infrastructure layer?

4 **Tool response validation:** Are all tool responses validated against a JSON schema before being passed to the agent? Does validation include an instruction-detection classifier for retrieved content?

5 **Input sanitization:** Does all untrusted input (user messages, external documents, API responses) pass through a sanitization layer before any trust zone crossing?

6 **Memory store access control:** Is the vector/memory store on a separate access control boundary from the agent runtime? Are writes to the memory store authenticated and logged?

7 **Audit logging:** Is every Protected zone access (secret read, PII access, memory write) logged to an append-only, tamper-evident store with agent identity and timestamp?

8 **Human escalation path:** Is there a defined set of high-consequence actions that require human approval before execution? Is the escalation mechanism tested?

9 **Kill switch:** Is there a mechanism to immediately terminate all running agent instances and freeze their credential access? Has it been tested in the last 90 days?

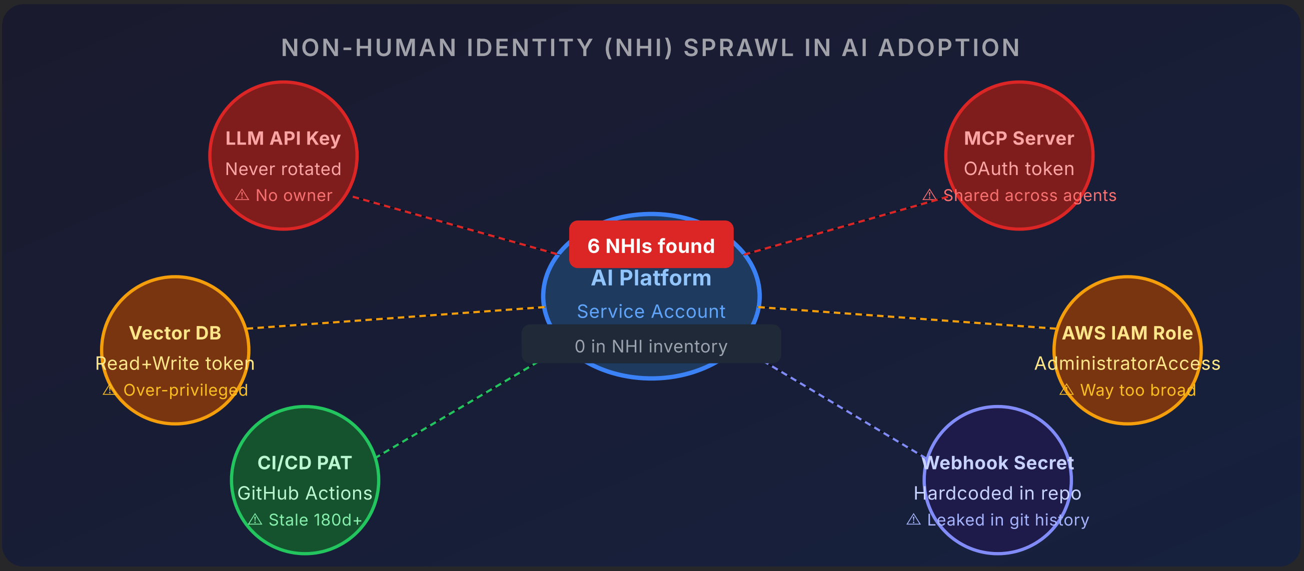
10 **Blast radius analysis:** Has a blast radius analysis been conducted for the worst-case compromise scenario (agent fully controlled by attacker)? Is the outcome acceptable?

11 **Threat model:** Has a threat model been completed using the STRIDE-LM extension? Are all identified threats either mitigated or accepted with documented rationale?

12 **Policy-as-code enforcement:** Are trust zone boundary invariants expressed as OPA/Rego policies that are tested in CI and enforced at runtime?

Non-Human Identity Sprawl: The Fastest-Growing Attack Surface

Non-Human Identities (NHIs) — API keys, service accounts, OAuth tokens, agent credentials — are growing faster than any other identity category in the enterprise. Every AI integration creates new NHIs. Every agentic deployment multiplies them. And unlike human identities, NHIs rarely have an owner, never expire by default, and never get offboarded when a project ends.



NHI sprawl in a typical AI-adopting enterprise. Six non-human identities orbit a single AI platform — none inventoried, most over-privileged, some compromised before anyone notices. This scenario is routine, not exceptional.

The NHI problem is structural. Traditional PAM (Privileged Access Management) tools were built for human accounts — they assume a named owner, a provisioning request, and an offboarding process. AI agent credentials don't arrive through ServiceNow. They get created in a Jupyter notebook at 11pm by a data scientist who needed the pipeline to work.

NHI Governance Framework for AI Deployments

An effective NHI program for AI environments requires five controls:

1. **Discovery and inventory:** You cannot govern what you cannot see. Deploy secrets scanning (Trufflehog, GitLeaks) across all code repos — including forks and PRs. Use cloud-provider tools (AWS IAM Access Analyzer, Azure Managed Identity audit) to find service accounts. Run periodic sweeps of environment variable stores (K8s secrets, SSM Parameter Store, GitHub Actions secrets) looking for API key patterns. Build a live inventory: who owns this credential, what does it have access to, when was it last used, when does it expire.

2. **Least privilege enforcement:** AI agent credentials should follow the same least-privilege principle as human accounts — only the permissions needed for the specific task, scoped to the minimum resource set. An agent that reads from a vector database should have read-only access to that specific index, not read/write to all indexes. An agent that calls an email API should be scoped to send, not read. Review existing agent credentials against this principle and remediate. This is almost always a finding.

3. **Automated rotation:** Credentials that never rotate become permanently valid attack primitives. Implement automated rotation for all AI agent credentials via your secrets management platform (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault). Short TTLs (1–24 hours) for highly privileged credentials. Rotation events should trigger alerts in case the credential was cached somewhere unexpected.

4. **Usage monitoring and anomaly detection:** Baseline the usage patterns for each AI agent credential — typical call volume, time of day, source IP range, API methods called. Alert on deviations: a credential that normally makes 100 API calls per hour suddenly making 10,000 is an incident. A credential that normally calls read-only endpoints suddenly calling write endpoints is an

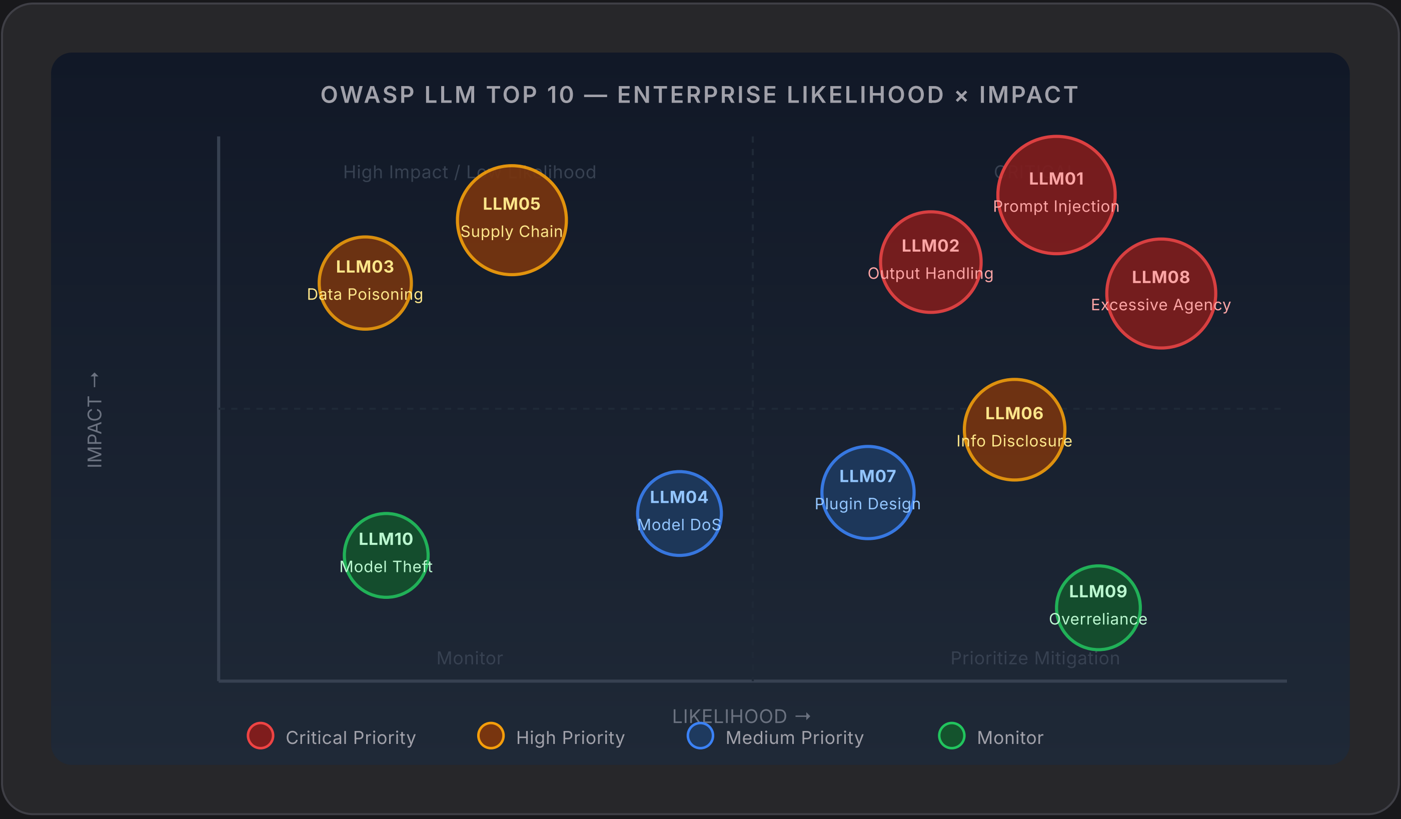
incident. This is achievable with CloudWatch, Azure Monitor, or a SIEM with the right correlation rules.

5 **Orphan detection and revocation:** When an AI project ends, its credentials don't automatically die. Implement a quarterly review process for all NHIs: confirm the owning team still exists, the project is still active, and the credential is still being used. Revoke anything that fails this review. Set expiry dates on all new credentials at creation — never create a non-expiring credential for an AI agent.

```
# Example: AWS IAM policy for a scoped AI agent (read-only RAG access)
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RAGVectorReadOnly",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::company-rag-index",
        "arn:aws:s3:::company-rag-index/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/ProjectID": "ai-agent-rag-v2"
        }
      }
    }
  ]
}
# Note: No s3:PutObject, no s3:DeleteObject, no cross-bucket access.
# The agent can read the RAG index. That is all.
```

OWASP LLM Top 10: Enterprise Risk Mapping

The OWASP Top 10 for Large Language Model Applications is the closest thing the industry has to a standardized AI vulnerability taxonomy. For enterprise AppSec teams, the challenge is not awareness of the list — it is mapping each item to your specific deployment context and prioritizing mitigations accordingly.



OWASP LLM Top 10 mapped by enterprise likelihood and impact. Prompt Injection (LLM01), Excessive Agency (LLM08), and Insecure Output Handling (LLM02) are the critical-priority cluster. Supply Chain risk (LLM05) carries catastrophic potential even at lower likelihood.

The matrix is useful as a prioritization tool, but it flattens nuance that matters operationally. LLM01 (Prompt Injection) and LLM08 (Excessive Agency**OWASP LLM06:2025 — Excessive Agency**A vulnerability where an LLM-based agent is granted more permissions, tool access, or autonomy than required for its intended function — amplifying the blast radius when the agent is compromised or manipulated.Read official definition ↗) are both critical — but they require fundamentally different mitigations. Prompt Injection is primarily an input validation and sandboxing problem. Excessive Agency is primarily an authorization and architecture problem.

Enterprise Mitigation Mapping: Critical Tier

LLM01 — Prompt Injection: Separate system instructions from user data at the architecture layer, not just in the prompt. Use structured input schemas that reject free-text in positions that influence agent behavior. Deploy a secondary LLM or rule-based filter as an "injection detection" layer on all inputs before they reach the primary model. For agentic systems, treat every retrieved document, tool response, and external API result as potentially adversarial — never embed them directly in the system prompt context.

LLM08 — Excessive Agency: This is an architecture problem, not a prompt engineering problem. Implement explicit authorization gates before any agent action that has real-world consequences (sends communication, modifies data, calls external APIs). Design agents with the minimum tool set required for their function. Enforce tool allowlists at the framework level, not via instructions to the model. Implement rate limits on all agent-accessible APIs. Require human-in-the-loop approval for actions above a configurable impact threshold.

LLM02 — Insecure Output Handling: Never render LLM output as HTML without sanitization. Treat all model output as untrusted input to downstream systems. If output is passed to a shell, SQL interpreter, or another model, apply the same injection protections you would for user-supplied data. Use output schemas to constrain model responses to structured formats, reducing the attack surface for instruction injection in outputs.

Enterprise Mitigation Mapping: High Priority Tier

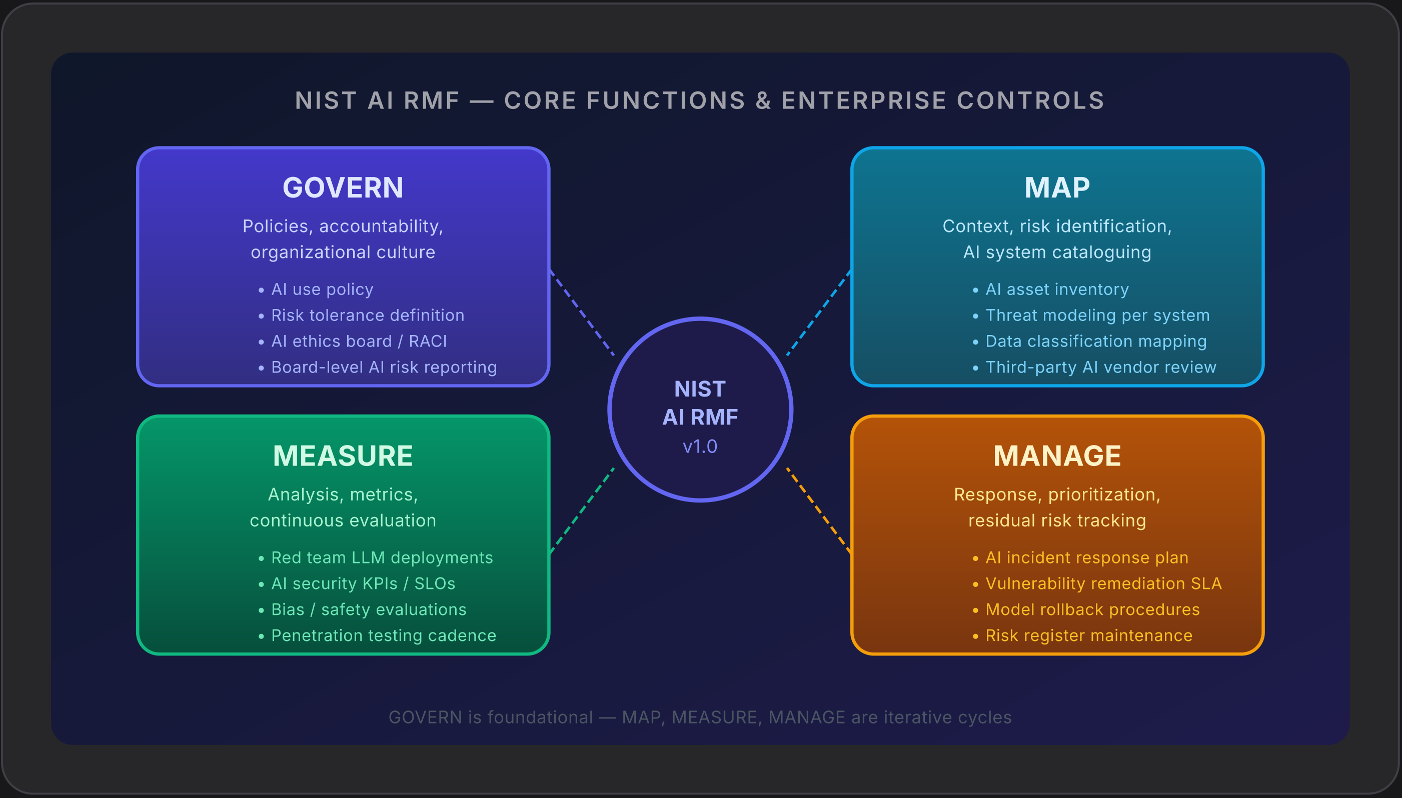
LLM05 — Supply Chain: Audit your AI supply chain: model weights (source, hash verification), training datasets, fine-tuning pipelines, and pip/npm dependencies in AI projects. Run dependency vulnerability scanning (pip-audit, Safety, Snyk) on all AI-related repos. Verify model provenance — use models from sources with published security disclosure policies and reproducible builds where available. Never load model checkpoints from untrusted sources.

LLM06 — Sensitive Information DisclosureOWASP LLM02:2025 — Sensitive Information DisclosureA vulnerability where an LLM inadvertently reveals confidential data — including system prompts, training data, PII, credentials, or proprietary business logic — through its responses.Read official definition ↗: Classify data before it enters any AI system. PII, financial data, and intellectual property should never appear in prompts to public AI APIs unless under a data processing agreement with appropriate controls. Implement output scanning — monitor model outputs for patterns that indicate training data memorization (PII, internal code patterns, credentials). This is a real phenomenon, not a theoretical one.

```python
# Example: Output scanning for PII in LLM responses
import re

PII_PATTERNS = {
    'ssn': r'\b\d{3}-\d{2}-\d{4}\b',
    'credit_card': r'\b(?:\d{4}[- ]?){3}\d{4}\b',
    'email': r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
    'api_key': r'(?:sk-|pk_|AIza)[A-Za-z0-9_-]{20,}',
}

def scan_output(text: str) -> list[str]:
    findings = []
    for label, pattern in PII_PATTERNS.items():
        if re.search(pattern, text):
            findings.append(label)
    return findings

# Use in production: block or redact responses with PII findings
output = model.generate(prompt)
findings = scan_output(output)
if findings:
    log_security_event('pii_in_output', findings=findings, truncated_output=output[:2
    output = '[Response redacted: contained sensitive content]'
```

NIST AI RMF: Building Your Governance Framework

The NIST AI Risk Management Framework (AI RMF 1.0) provides the most mature governance structure available for enterprise AI security programs. Unlike the OWASP list (which is tactical), the AI RMF is strategic — it tells you how to build and sustain a program, not just what to fix today. For CISOs needing to justify AI security investment to the board or align with regulatory requirements (EU AI Act, NIST CSF 2.0), the AI RMF is your language.


NIST AI RMF — CORE FUNCTIONS & ENTERPRISE CONTROLS
GOVERN
Policies, accountability, organizational culture
• AI use policy
• Risk tolerance definition
• AI ethics board / RACI
• Board-level AI risk reporting
MAP
Context, risk identification, AI system cataloguing
• AI asset inventory
• Threat modeling per system
• Data classification mapping
• Third-party AI vendor review
NIST AI RMF v1.0
MEASURE
Analysis, metrics, continuous evaluation
• Red team LLM deployments
• AI security KPIs / SLOs
• Bias / safety evaluations
• Penetration testing cadence
MANAGE
Response, prioritization, residual risk tracking
• AI incident response plan
• Vulnerability remediation SLA
• Model rollback procedures
• Risk register maintenance
GOVERN is foundational — MAP, MEASURE, MANAGE are iterative cycles

NIST AI RMF core functions with enterprise control mapping. GOVERN is foundational — without organizational policy and accountability structures, the other three functions produce reports that nobody acts on.

The most common AI RMF implementation failure is skipping GOVERN and jumping straight to MEASURE. Security teams want to run red team exercises and produce metrics. But if there is no AI use policy, no risk tolerance definition, and no accountable owner for AI risk at the executive level, the metrics go nowhere. GOVERN is the prerequisite.

Practical GOVERN Implementation for Enterprises

The GOVERN function requires three foundational artifacts before anything else:

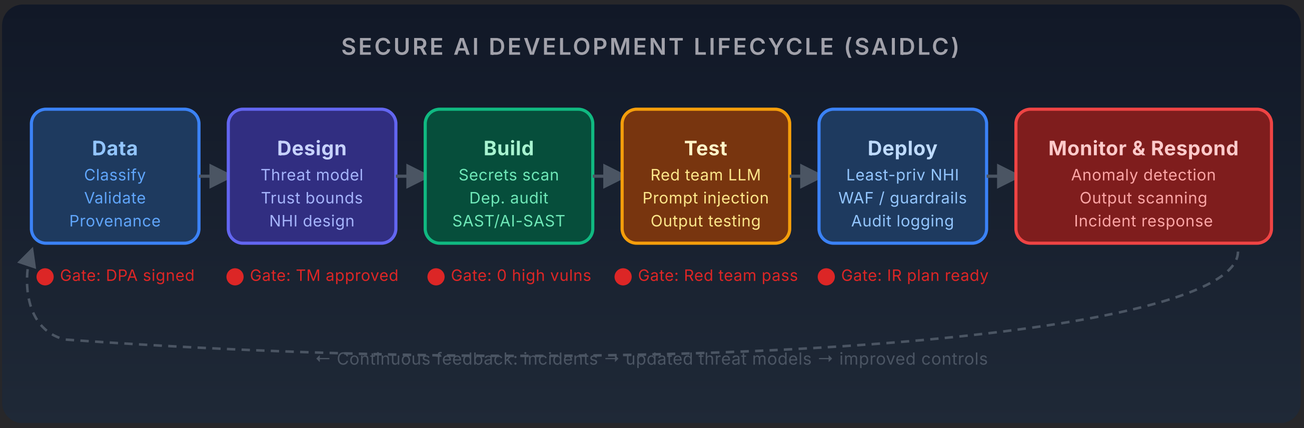
1. **AI Use Policy:** A written, board-approved policy that defines which AI tools are approved, which categories of data can be processed by AI systems, what approval process exists for new AI integrations, and what employees must do when they discover a potential AI security incident. This policy should be integrated into your acceptable use policy (AUP) and updated annually. Without this, every shadow AI situation becomes a grey area.

2. **AI Risk Tolerance Statement:** Explicit definition of which AI-related risks are acceptable, which require mitigation, and which are unacceptable regardless of business benefit. This should specifically address: probability of AI system compromise, probability of data exfiltration via AI, probability of AI-assisted fraud, and probability of regulatory violation via AI use. Quantified tolerances, not platitudes.

3. **AI Risk RACI:** Clear ownership. Who is accountable for AI security risk? (CISO, typically.) Who is responsible for implementing controls on AI systems? (AppSec team, typically.) Who must be consulted before deploying a new AI system? (Data privacy, legal, infosec.) Who must be informed? (Board AI committee.) Without a RACI, every AI security question becomes a meeting without a decision.

The Secure AI Development Lifecycle (SAIDLC)

Your existing SDLC was built for deterministic software. AI systems are probabilistic, data-dependent, and behavior-emergent — they fail in ways that no unit test catches and no static analyzer flags. The Secure AI Development Lifecycle extends your existing SDLC with AI-specific gates at every phase, from data acquisition through deployment and monitoring.



Secure AI Development Lifecycle. Each phase has a mandatory security gate. "Deploy" is only reached after data provenance, threat model approval, zero high-severity vulnerabilities, and red team sign-off. The feedback loop ensures incidents improve future threat models.

The gates are the critical element. Most enterprises have SDLC documentation with security phases but no actual enforcement. An AI system can skip the threat model because the sprint is ending. The SAIDLC gates must be enforced in your CI/CD pipeline — automated checks that block deployment when criteria are not met, not advisory warnings that developers click through.

AI-Specific SAST: What to Add to Your Pipeline

Standard SAST tools (Semgrep, SonarQube, Checkmarx) have limited coverage for AI-specific vulnerabilities. Augment them with these checks:

- **Secrets in prompts:** Detect hardcoded API keys, credentials, or PII in prompt template strings. Pattern: any string literal that contains a prompt instruction combined with a secret-looking pattern.
- **Unvalidated LLM output use:** Detect cases where the output of an LLM call is passed directly to shell execution, SQL queries, file writes, or HTML rendering without sanitization.
- **Over-privileged tool definitions:** Detect agent tool definitions that include dangerous capabilities (shell exec, arbitrary HTTP, file

system write) combined with instructions that could allow prompt injection to trigger them.

- **Insecure model loading:** Detect loading of ML model files (.pkl, .pt, .h5) from remote URLs or unverified sources without hash validation. Pickle deserialization is a known RCE vector.

```
# Semgrep rule: detect LLM output used in subprocess without sanitization
rules:
  - id: llm-output-in-subprocess
    patterns:
      - pattern: subprocess.run($CMD, ...)
      - pattern-either:
          - pattern-inside: |
              $RESP = $LLM.generate(...)
              ...
              subprocess.run($RESP, ...)
          - pattern-inside: |
              $RESP = $CHAIN.run(...)
              ...
              subprocess.run($RESP, ...)
    message: LLM output used directly in subprocess — potential command injection
    severity: ERROR
    languages: [python]
```

Red Team Methodology for Enterprise LLM Systems

A complete LLM red team engagement for an enterprise AI system should cover five attack categories in order of priority:

1. **Direct prompt injection:** Systematically test the system prompt's resilience. Can you extract it? Can you override it? Can you get the model to ignore its instructions via role-play, fictional framing, or encoding attacks (base64, leet speak, Unicode)?
2. **Indirect injection via data sources:** Can you inject instructions via the system's data sources — uploaded documents, retrieved web pages, database contents, email bodies? What happens when a retrieved document contains "IGNORE PREVIOUS INSTRUCTIONS"?
3. **Authorization bypass:** Can you access data or capabilities you should not? Can you get the system to act on behalf of another

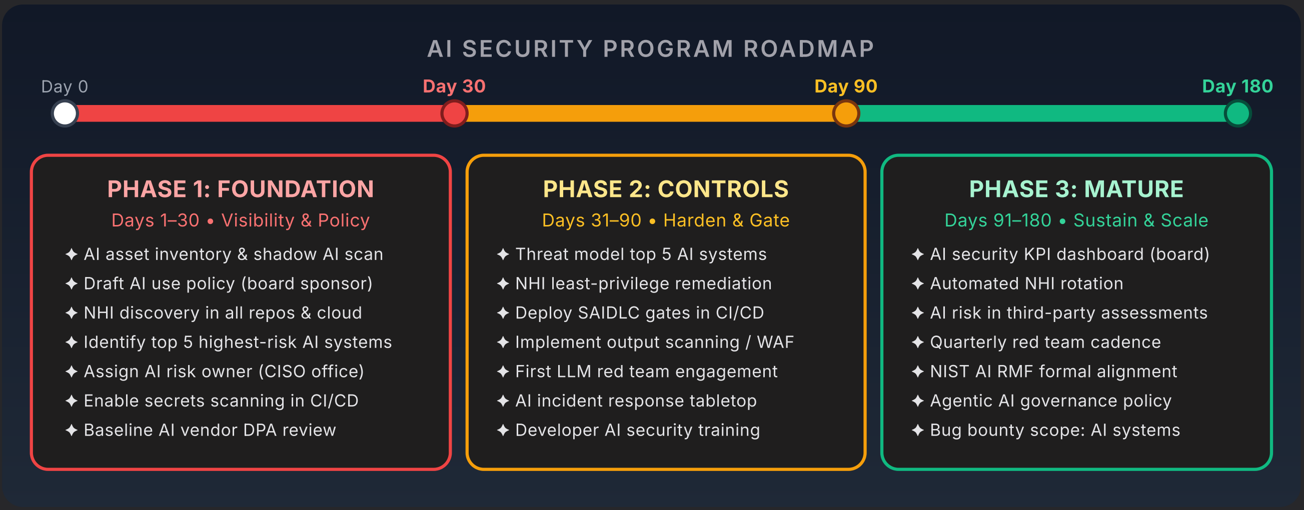
user? Can you escalate from viewer to editor via the AI interface?

4 **Information extraction:** Can you extract training data, other users' data, system configurations, or API keys from the model's responses? Try many-shot extraction, completion attacks, and membership inference**Membership Inference Attack (Shokri et al., 2017)**An attack that determines whether a specific data record was used to train a model by exploiting differences in the model's confidence outputs on training vs. non-training data — enabling privacy violations without direct data access.Read official definition ↗ probes.

5 **Agentic abuse (if applicable):** Can you trigger the agent to take actions beyond its intended scope? Can you exfiltrate data via a side channel (encoded in a search query, embedded in an output filename)? Can you persist instructions across sessions via memory poisoning?

Your 30/90/180-Day Roadmap

The most common mistake CISOs make with AI security is trying to solve everything at once. The threat surface is large and growing — attempting a comprehensive program from day one produces a large roadmap document that nobody executes. The roadmap below is sequenced by dependency and impact: each phase builds on the last, and each is achievable within the timeframe for a team of 2–4 security engineers.


AI SECURITY PROGRAM ROADMAP
Day 0
Day 30
Day 90
Day 180
PHASE 1: FOUNDATION
Days 1–30 • Visibility & Policy
✦ AI asset inventory & shadow AI scan
✦ Draft AI use policy (board sponsor)
✦ NHI discovery in all repos & cloud
✦ Identify top 5 highest-risk AI systems
✦ Assign AI risk owner (CISO office)
✦ Enable secrets scanning in CI/CD
✦ Baseline AI vendor DPA review
PHASE 2: CONTROLS
Days 31–90 • Harden & Gate
✦ Threat model top 5 AI systems
✦ NHI least-privilege remediation
✦ Deploy SAIDLC gates in CI/CD
✦ Implement output scanning / WAF
✦ First LLM red team engagement
✦ AI incident response tabletop
✦ Developer AI security training
PHASE 3: MATURE
Days 91–180 • Sustain & Scale
✦ AI security KPI dashboard (board)
✦ Automated NHI rotation
✦ AI risk in third-party assessments
✦ Quarterly red team cadence
✦ NIST AI RMF formal alignment
✦ Agentic AI governance policy
✦ Bug bounty scope: AI systems

30/90/180-day AI security roadmap. Phase 1 is about visibility — you cannot control what you cannot see. Phase 2 deploys controls where risk is highest. Phase 3 builds the sustaining program that doesn't require heroics to maintain.

Phase 1 Detail: The First 30 Days

The first 30 days are entirely about gaining visibility. Do not attempt to fix things yet. Your goal is to answer: What AI systems are running in our environment? Who owns them? What data do they process? What credentials do they hold?

Start with passive discovery before active remediation. Passive discovery is low-risk and high-return. Active remediation on systems you don't fully understand can break production and create adversarial relationships with engineering teams who will resist your future work. Build the map before you start removing roads.

AI asset inventory methodology:

1. Query your API gateway and cloud billing for calls to known AI endpoints (api.openai.com, api.anthropic.com, bedrock.us-east-1.amazonaws.com, *.azure-api.net/openai). This gives you active integrations.
2. Search code repositories (GitHub Advanced Security, GitLab, Bitbucket) for AI SDK imports (openai, anthropic, langchain,

llama-index, transformers). This gives you developer activity.

3. Survey business unit leaders with a simple questionnaire: "What AI tools does your team use? Which are approved by IT?" This surfaces shadow AI at the consumer tool level.

4. Review SaaS vendor contracts for AI feature addenda or data sharing amendments that were signed in the last 18 months. Many AI features were added via T&C updates, not new contracts.

Phase 2 Detail: The Critical 90-Day Controls

With visibility established, Phase 2 deploys controls where risk is highest. Prioritize based on your Phase 1 findings, but in most enterprises, the following sequence holds:

1. Threat model the top 5 AI systems first. Use STRIDE-LM. Document attack paths. Identify controls that are missing. This creates a defensible record that the system was analyzed, which matters for compliance and incident response. Schedule a workshop with the owning engineering team — do not threat model in isolation. They know the system. You know the attack techniques.

2. NHI least-privilege remediation on identified over-privileged credentials. Start with the worst offenders: any AI agent with cloud administrator credentials, any service account with access to all data stores, any API key with no expiry. These are your P0 remediations. Don't try to fix everything — fix the ones that, if compromised, would cause a breach.

3. SAIDLC gate deployment in CI/CD. Even a minimal gate set adds significant value: block secrets in code (Trufflehog pre-commit hook), require a threat model to exist before deployment to production (enforce via deployment pipeline check against threat model registry), and enable dependency audit on AI-related repos. These are pipeline changes, not code changes — they apply to all future deployments automatically.

Phase 3 Detail: Building the Sustaining Program

The Phase 3 objective is sustainability. Everything in Phases 1 and 2 was done by humans doing one-time work. Phase 3 automates the repetitive components and institutionalizes the program so it survives personnel changes.

The board-level AI security KPI dashboard is not optional for organizations with significant AI exposure. Boards cannot govern what they cannot see. Quarterly metrics should include: number of AI systems in inventory (trending), number of NHIs with expiry dates (trending toward 100%), number of AI security vulnerabilities found vs. remediated, and time-to-detect for AI security incidents. These four metrics give the board meaningful signal without operational noise.

🏁 The Bottom Line for CISOs

The enterprise AI adoption wave is not slowing down because security hasn't caught up. It is continuing regardless. The question is not whether your organization will deploy AI — it already has. The question is whether your security program is positioned to see it, govern it, and defend it.

The good news: this is solvable with existing security disciplines extended for AI's specific failure modes. You are not starting from scratch. Your threat modeling skills translate. Your identity governance skills translate. Your red team capability translates. What is new is the attack surface, the vocabulary (prompt injection, NHI sprawl, tool poisoning), and the urgency.

Start with visibility. Assign ownership. Gate deployments. The enterprises that do these three things in the next 90 days will have a meaningful security advantage over those that spend 90 days building a comprehensive framework document that nobody executes.

Get the Complete CISO Playbook (PDF)

The full guide includes: detailed NHI governance templates, SAIDLC gate implementation scripts, red team methodology for LLM systems, NIST AI RMF alignment worksheets, OWASP LLM Top 10 mitigation playbook, and 40+ implementation-ready policy templates. Everything your AppSec team needs to execute the roadmap.

Get the Full PDF — $27 →