Computer Fraud and Security
ISSN (online): 1873-7056

Securing Agentic AI Systems: Threat Models and Penetration Testing Strategies for Enterprise Deployments

Bala Thripura Akasam

Tapestry, Inc., USA

Abstract

Agentic AI systems—autonomous or semi-autonomous agents capable of multi-step planning, tool invocation, and real-world action execution—are entering enterprise production environments at a pace that has materially outstripped the maturation of corresponding security controls. The attack surfaces introduced by these deployments, spanning indirect prompt injection through retrieval pipelines, tool-use escalation across execution sandboxes, model and memory poisoning in long-lived vector stores, and supply-chain subversion of unsigned model artifacts, fall outside the detection and prevention scope of conventional application security tooling, including static analysis, dynamic analysis, and software composition analysis. This article proposes a structured threat model for agentic AI systems that defines a five-category asset taxonomy, four adversary classes, and six representative attack patterns grounded in current industry frameworks and adversarial AI research. A six-phase penetration testing methodology is developed to address each attack class within safe, pre-production, and CI/CD-compatible boundaries, extending from architecture and supply chain review through runtime telemetry correlation with application security posture management platforms. Mitigation patterns are mapped directly to each identified attack class and integrated with DevSecOps pipeline controls, including prompt linting, tool-schema validation, signed artifact verification, and continuous red-team playbook execution. Outcome metrics covering guardrail efficacy, supply-chain integrity coverage, tool-use safety, and runtime risk reduction provide the quantitative evidence base required for both operational program management and publication standards. The controls and governance directions presented in this article align with current industry consensus on the shift from point-in-time scanning toward posture, provenance, and runtime-informed prioritization as the organizing principles of mature enterprise application security programs.

Keywords: Agentic AI Security, Penetration Testing, Threat Modeling, Application Security Posture Management, Supply-Chain Integrity

1. Introduction

The enterprise adoption of large language model (LLM)-powered applications has accelerated sharply across industries, with a growing proportion of deployments now exhibiting agentic characteristics that extend well beyond the capabilities of earlier AI integrations. Unlike conventional AI systems that respond to discrete, bounded queries, agentic systems autonomously decompose complex goals into multi-step action sequences, select and invoke external tools, query live data sources, and iterate toward outcomes across extended execution cycles with limited human oversight [1]. Industries ranging from financial services and retail to healthcare and critical infrastructure have moved these systems from pilot programs into production workflows, motivated by measurable gains in task automation, decision support, and process orchestration. The transition has not been gradual. Agentic AI capabilities are now embedded in enterprise software development pipelines, customer engagement platforms, and internal knowledge management systems simultaneously, creating a deployment footprint that has expanded faster than the security frameworks designed to govern it [1]. This pace mismatch is not a temporary condition that will self-correct as the technology matures. It reflects a structural gap between the speed at which AI capabilities are operationalized and the considerably slower cycle through which security controls are designed, validated, and institutionalized across enterprise environments [1].

The security readiness gap this creates is most visible in the limitations of conventional application security tooling. Static application security testing (SAST), dynamic application security testing (DAST), and software composition analysis (SCA) were designed for deterministic software artifacts: codebases with stable execution logic, bounded input surfaces, and predictable control flows that can be analyzed, scanned, and remediated through well-understood processes [3]. Agentic systems violate each of these assumptions in ways that are not edge cases but fundamental properties of how these systems operate. Execution paths in agentic workflows emerge dynamically from model reasoning rather than fixed code

logic, meaning that no two runs of the same agent against the same goal are guaranteed to follow identical tool invocation sequences. Input surfaces extend continuously and unpredictably through retrieval-augmented generation (RAG) pipelines that ingest live external content—web pages, documents, database records, and API responses—at runtime, introducing attacker-controlled data into the agent's reasoning context through channels that DAST was never designed to monitor [2]. Tool-mediated side effects, including file writes, outbound API calls, and database transactions, produce real-world operational consequences that static analyzers cannot anticipate or evaluate because those consequences depend on runtime state rather than source code structure [3]. The result is that organizations applying traditional AppSec programs to agentic deployments are systematically blind to the attack classes that present the greatest operational risk: indirect prompt injection through retrieved content, tool-use escalation across execution sandboxes, memory poisoning in long-lived vector stores, and supply-chain subversion of unsigned model artifacts [2], [3].

The broader AppSec field has begun responding to this structural limitation, and the directional shift is clear even if adoption remains uneven. Point-in-time scanning programs are giving way to Application Security Posture Management (ASPM), an architectural approach that consolidates findings from SAST, DAST, SCA, infrastructure-as-code analysis, and runtime telemetry into a unified, continuously updated risk view [2]. The distinguishing characteristic of ASPM is prioritization by reachability and exploitability rather than raw finding volume—a shift made necessary by the signal-to-noise problem that plagues programs relying on scanner output alone, where large backlogs of low-context findings obscure the small subset of issues that represent genuine, exploitable risk in production [2]. Simultaneously, regulatory and supply-chain pressures are accelerating adoption of Software Bill of Materials (SBOM) requirements and Supply-chain Levels for Software Artifacts (SLSA) attestation frameworks. These controls, which originated in the open-source software dependency context, are now being extended to encompass model weights, training datasets, prompt configurations, and agent pipeline artifacts—the new classes of software supply-chain components that agentic deployments introduce [3]. Runtime-informed prioritization, which uses behavioral signals and telemetry captured in production environments to drive remediation sequencing, is emerging as the organizing principle of mature security programs, replacing the static severity scores that characterized earlier practice and that carry no information about whether a finding is reachable or exploitable in a given deployment context [2], [3].

This article delivers a practitioner-ready framework that addresses the gap between agentic AI adoption and security readiness through four integrated contributions. A structured threat model defines the assets requiring protection, the adversary classes with realistic goals and capabilities, and six representative attack patterns specific to agentic architectures that existing threat libraries do not adequately cover. A six-phase penetration testing methodology provides safe, repeatable, and CI/CD-compatible testing coverage extending from architecture and supply-chain review through runtime telemetry correlation, with each phase designed to operate within pre-production guardrails that prevent test activity from producing unintended operational consequences. Mitigation patterns are mapped directly to each identified attack class and integrated with DevSecOps pipeline controls, including CI/CD gate enforcement and ASPM-driven remediation workflows. Outcome metrics produce verifiable evidence of security improvement across guardrail efficacy rates, supply-chain integrity coverage, and mean time to detect and contain agent abuse events—providing the quantitative foundation necessary for both operational program management and academic publication standards [2], [3].
This article addresses a specific and tractable problem: enterprise security programs lack a purpose-built threat model, structured penetration testing methodology, and integrated mitigation framework for agentic AI deployments, and the absence of these instruments creates a measurable and growing operational risk exposure that existing AppSec tooling cannot close. The research objective is to deliver each of these three instruments in a form that is grounded in recognized adversarial AI taxonomies, compatible with current DevSecOps pipeline architectures, and validated against quantitative outcome metrics—providing practitioners with an immediately actionable security program structure while contributing a replicable and evidence-based methodology to the academic literature on AI system security.

2. Architectural Characteristics and Attack Surface of Agentic AI

Agentic AI systems are architecturally more complex than conventional software applications, and that complexity is the direct source of their expanded attack surface. At the core of every agentic deployment sits a large language model or model ensemble that functions as the reasoning engine, interpreting goal specifications, generating plans, evaluating intermediate outputs, and deciding which tools or resources to invoke at each step of an execution cycle. Surrounding this reasoning core is an orchestrator or agent planner—a coordination layer that translates model outputs into structured action sequences, manages tool invocation order, tracks execution state, and routes partial results back to the model for iterative

evaluation. Tool adapters form the third structural layer, connecting the planner to the external systems that give the agent its operational reach: code execution environments, relational and vector databases, robotic process automation frameworks, web browsing interfaces, and third-party APIs spanning communication, payment, and enterprise resource planning systems [4]. Context and memory stores, most commonly implemented as vector databases or knowledge graphs, extend the agent's effective information horizon beyond its native context window by retrieving relevant documents, conversation history, and domain knowledge at runtime. Observation loops allow the agent to assess whether intermediate results satisfy partial goal conditions and adjust the subsequent action plan accordingly, enabling the self-correction behavior that distinguishes agentic systems from single-turn model applications [5]. These five components interact continuously and dynamically during execution, and it is precisely the interfaces between them—the boundaries where control, data, and trust pass from one layer to the next—that constitute the primary attack surface of an agentic deployment [4], [5].

The contrast with traditional web application attack surfaces is significant and should not be understated. Traditional web-based software solutions are able to identify a relatively static and well-defined set of entry points: via HTTP (hypertext transfer protocol), via forms and session tokens for authenticating users, or via interfaces for querying databases. The tooling designed to test these surfaces reflects decades of accumulated knowledge about where vulnerabilities concentrate and what exploitation patterns look like in practice. Agentic systems present a fundamentally different profile. The attack surface is not static but generative, expanding dynamically with every new tool integration, every external data source added to the retrieval corpus, and every API the agent is authorized to call at runtime [4]. The OWASP Top 10 for Large Language Model Applications identifies prompt injection as the most critical risk category for LLM-based systems, precisely because the agent treats retrieved and user-supplied content as trusted reasoning input rather than as untrusted external data requiring validation [4]. An attacker does not need to identify a vulnerability in the agent's codebase to redirect its behavior; injecting a malicious instruction into any content the agent retrieves during a live execution cycle is sufficient to compromise the integrity of the entire action sequence that follows [4], [5].

Goal mis-specification and tool-use risks represent one of the most consequential attack surface dimensions introduced by autonomous action chaining. When an agent pursues a goal across a sequence of tool invocations, each step produces outputs that become inputs to the next, and errors—whether adversarially induced or the result of ambiguous goal specification—compound across the chain rather than remaining contained within a single operation [5]. The NIST AI Risk Management Framework specifically states that AI systems operating with a large amount of autonomous behaviors generate risk profiles that are qualitatively different from human-driven systems and that, when they can execute certain actions in the world without a human actually confirming to them to execute them, the level of risk is exceedingly higher [5]. An agent responsible for processing customer return requests and having concurrent access to the Customer Data (something in the database) and the Order Management API (Another system, whether or not you can actually authorize that data) can chain the data retrieval call to an unauthorized write request via a benefit of a misconfigured or manipulated policy with no single component in the sequence appearing to be abnormal on its own. The consequence radius of such a chain is determined not by the scope of any single tool but by the cumulative permissions of all tools the agent is authorized to invoke—a scope that in production deployments frequently reflects incremental capability additions rather than holistic privilege design [4], [5].

The API and integration exposure introduced by embedded AI agents amplifies existing enterprise attack surfaces in ways that conventional API security programs are not positioned to address. Each tool adapter the agent uses represents a trust relationship between the orchestration layer and an external system, and the agent's capacity to compose calls across multiple adapters within a single execution cycle means that a compromised tool credential or an overly permissive tool schema can serve as a lateral movement path into systems well beyond the agent's intended operational scope [6]. Unlike a human operator or a conventional application that interacts with APIs through audited, workflow-bound request patterns, an agentic system constructs novel API call sequences at runtime in direct response to model outputs. These dynamically generated interaction patterns fall outside the behavioral baselines that anomaly detection systems trained on historical human-generated traffic are designed to recognize, meaning that API abuse by an agent operating under adversarial influence is unlikely to surface in standard monitoring dashboards until material harm has already been realized [5], [6].

Supply-chain complexity adds a further dimension that has no mature equivalent in traditional software security practice. Model weights, fine-tuned adapter configurations, curated training and evaluation datasets, system prompt templates, and agent pipeline definitions are all functional software artifacts in the operational sense—they directly and materially

determine system behavior—yet they exist almost entirely outside the package management conventions, versioning discipline, and dependency transparency that govern open-source software supply chains [6]. Published model evaluation documentation confirms that the behavioral properties of a deployed model are deeply sensitive to the composition and curation of training data, the design of reinforcement learning feedback mechanisms, and the configuration of safety fine-tuning processes—all of which represent upstream supply-chain inputs that most enterprise deployments cannot independently verify or audit [6]. Without SBOM-equivalent artifact inventories and provenance attestations covering these components, an organization cannot confirm with confidence that the model it is running in production reflects the artifact it evaluated during pre-deployment testing, or that a plugin introduced into the tool adapter layer has not been silently substituted with a compromised variant between build and deployment [4], [6].

Operational drift and context poisoning through long-lived vector stores represent a risk class that is unique to agentic architectures and particularly resistant to detection through periodic security assessments. Because vector stores accumulate context across sessions and serve that context continuously to the agent's retrieval pipeline, an attacker who successfully injects malicious content into a retrieval corpus establishes a persistent influence channel that continues to shape agent behavior across future sessions without requiring maintained system access [5]. The NIST AI Risk Management Framework notes that AI systems are susceptible to data integrity risks throughout their operational lifecycle, not only at the point of initial training, and that runtime data inputs represent an ongoing attack surface that must be governed with the same rigor applied to training data [5]. Subtle modifications to factual records, policy documents, or procedural guides stored in the vector database can gradually shift agent decision patterns in ways that are statistically invisible against the baseline of normal operational variation, making reliable detection contingent on behavioral monitoring capabilities—continuous output auditing, embedding integrity scanning, and retrieval provenance logging—that the majority of current agentic deployments have not yet implemented [4], [5].

3. Threat Model for Agentic AI Systems

A structured threat model for agentic AI must address the full operational context of these systems—not only the model itself but every artifact, interface, and data flow that influences its behavior in production. The asset taxonomy for agentic deployments spans five categories that collectively define what must be protected. Model artifacts include base checkpoints, fine-tuned variants, distilled models, and low-rank adaptation weights—all of which encode learned behaviors that can be altered through upstream tampering without leaving visible traces in application code [7]. Prompt and policy assets encompass system prompt configurations, tool-use schemas, and safety policy definitions that govern what the agent is permitted to do and how it interprets user instructions. Data and memory assets include embedding stores, RAG corpora, and sensitive training and evaluation datasets whose integrity directly determines the factual grounding of agent outputs. Orchestration and tooling assets cover agent planners, plugins, external API integrations, and execution sandbox configurations through which the agent acts on the world. Supply-chain artifacts, including software bills of materials, model cards, and pipeline attestations, represent the provenance layer that enables verification of all other asset categories [8]. The LLM AI Security and Governance Checklist identifies these asset classes as the foundation of any governance program for LLM-based systems, noting that organizations must inventory and assign ownership across all of them before meaningful risk controls can be applied [7]. The absence of mature governance controls across any one of these asset classes creates exploitable gaps, and in most current enterprise deployments, gaps exist across several simultaneously [7], [8].

Four adversary classes with distinct goals, capabilities, and access assumptions define the realistic threat population for agentic AI systems. Prompt-level adversaries pursue instruction override and safety-policy bypass, operating either through direct interaction with the agent or by placing malicious instructions in content the agent retrieves during normal operation, a capability that requires no privileged access to the deployment environment [7]. Data poisoners target RAG source documents and embedding stores with the goal of corrupting retrieval outputs, degrading agent decision quality, or introducing persistent behavioral backdoors that activate under specific trigger conditions during inference. Supply chain type attackers typically attack before the artifacts are deployed and try to add things like malware signatures or compromised model weights to the software build pipeline or try to sign the software build or plug-ins fraudulently before the artifacts reach production [8] but may also leverage rate limits, account limits, costs and authorization policies across agents to operationally disrupt/denial of service-type attacks, and/or to move laterally through misconfigured tool trust boundaries into associated enterprise systems from the attacking agent [9]. The ENISA multilayer cybersecurity framework for AI systems categorizes these adversary classes according to their position in the AI system lifecycle—data supply,

model development, deployment infrastructure, and runtime operation—and recommends that threat models address each layer independently rather than treating the AI system as a single monolithic attack target [9].

Indirect prompt injection is the attack class with the broadest practical reach and the lowest barrier to execution among all threats facing agentic deployments. Unlike direct prompt injection, which requires the attacker to interact with the agent through its intended input interface, indirect prompt injection embeds malicious instructions in external content—supplier web pages, shared documents, database records, or third-party API responses—that the agent retrieves and processes as part of a legitimate task [7]. Because the agent's reasoning engine applies interpretive weight to retrieved content in the same manner it does to system and user prompts, a well-crafted injection can override safety policies, redirect tool invocations, and exfiltrate sensitive context without any direct attacker access to the deployment. The LLM AI Security and Governance Checklist identifies prompt injection as a primary control domain requiring dedicated countermeasures at the retrieval layer, including content provenance validation and instruction-neutralization filters applied to all externally sourced inputs before they reach the reasoning engine [7]. Defenses that rely solely on output filtering are insufficient because the malicious instruction has already influenced the agent's internal reasoning state before any output is produced [7], [8].

Tool-use escalation and sandbox escape exploit weaknesses in tool schema validation, permission scoping, and execution environment isolation to trigger high-privilege operations beyond the agent's intended capability boundary. When tool input constraints are loosely defined or authentication requirements are absent, an attacker can craft inputs that cause the agent to invoke destructive operations—arbitrary code execution, unrestricted file writes, or outbound network connections to attacker-controlled infrastructure—through tool calls that the orchestration layer treats as fully legitimate [8]. The MITRE ATLAS framework, which catalogs adversarial tactics and techniques specific to AI systems, documents tool and plugin abuse as an active technique class observed in real-world AI system compromises, noting that agents granted broad tool permissions present a significantly larger exploitable surface than those operating under capability-scoped, least-privilege configurations [8]. Sandbox escape represents the most severe variant of this class, where the agent's code execution environment fails to enforce syscall restrictions, filesystem isolation, or network egress controls, allowing a single compromised tool invocation to produce consequences that extend beyond the agentic system and into the underlying host infrastructure [7], [8].

Model and memory poisoning attacks degrade agent decision integrity by corrupting the information the agent uses to reason, rather than by manipulating its instructions directly. In the poisoned model variant, adversaries inject corrupted data into the training or fine-tuning process, adding the needed backdoor behaviors to generate a specific attack vector to produce the specified output whenever a trigger input is provided to the model at inference time (e.g., poisoning training data) [8]. The MITRE ATLAS framework characterises training data poisoning as a foundational adversarial ML technique in the threat ecosystem and contains documented examples of poisoned training inputs leading to targeted behavioural deviations without being detected through traditional evaluation or testing processes [8]. Also, target poisoning of a vector store is implemented by injecting semantically valid but factually or behaviourally corrupted representations into the retrieval corpus. The distinguishing characteristic of memory poisoning relative to other injection attacks is persistence: corrupted embeddings continue to influence agent outputs across all subsequent sessions that retrieve them, making the attack self-sustaining once the initial injection succeeds and detection dependent on continuous behavioral monitoring rather than periodic assessment [7], [8].

API and workflow pivoting exploits the composability of agentic tool chains to traverse misconfigured trust boundaries between connected services. Individual tool calls that are each individually authorized can be composed into sequences that reach sensitive downstream systems the agent was never intended to access directly [9]. This attack class does not require any single tool to have a vulnerability in the conventional sense; it exploits the gap between what each tool permits in isolation and what the full chain permits in combination when executed by an agent with broad integration scope. The ENISA multilayer framework identifies inter-component trust boundary enforcement as a critical control requirement for AI systems operating in integrated enterprise environments, recommending that each service-to-service trust relationship in an agentic workflow be explicitly modeled and governed with the same rigor applied to human user access controls [9]. In deployments where agentic systems are integrated with internal APIs carrying elevated privileges—identity providers, financial transaction platforms, or enterprise resource planning systems—workflow pivoting can produce consequences equivalent to privilege escalation without exploiting any individually recognizable vulnerability [8], [9].

Supply-chain subversion via unsigned artifacts targets the deployment pipeline rather than the running system, introducing compromised components before security controls have an opportunity to inspect them. Model weights delivered without cryptographic signatures, plugin packages without verified provenance, and build pipelines without attestation controls all represent entry points through which an attacker can substitute a trusted artifact with a malicious variant that is behaviorally indistinguishable from the original under standard pre-deployment testing procedures [8]. Hallucination-driven logic flaws complete the threat taxonomy as a non-adversarial but operationally significant risk class. When an agent generates confident but factually incorrect outputs during autonomous planning, those outputs can trigger downstream tool invocations or decision sequences that produce harmful operational consequences without any attacker involvement [7]. The LLM AI Security and Governance Checklist explicitly includes hallucination risk within its governance scope, noting that the reliability and factual grounding of model outputs must be treated as a security property—not only an accuracy concern—in deployments where model outputs directly drive consequential automated actions [7]. This classification reflects the reality that in agentic systems, the boundary between security risk and AI reliability risk is operationally indistinct, and threat models that exclude non-adversarial failure modes will systematically underestimate the full risk surface of autonomous deployments [7], [9].

4. Penetration Testing Methodology and Mitigation Patterns

4.1 Pen-Testing Phases

A structured penetration testing methodology for agentic AI systems must address the full operational scope of these deployments—from architecture and supply-chain integrity through runtime telemetry correlation—while maintaining safety controls that prevent test activity from producing unintended consequences in pre-production and pipeline environments. Traditional penetration testing frameworks were designed for deterministic systems with stable execution paths and well-defined input surfaces, and they require substantive extension to address the non-deterministic, tool-mediated, and retrieval-augmented characteristics of agentic deployments [10]. The six phases described below represent a progression from scoping and static review through live exploitation and runtime analysis, with each phase building on the findings of its predecessor to produce a cumulative, evidence-based risk picture that supports both operational remediation and governance reporting [11].

Phase 0 establishes the operational boundary for all subsequent testing activity through scoping and safety control definition. The central concept is blast radius containment: before any test is executed, the engagement team must define which systems, tools, and data sources fall within scope, create sandboxed equivalents of production tool integrations using dummy accounts and synthetic datasets, impose explicit transaction limits on any tool capable of producing financial or data-modifying side effects, and maintain do-not-call lists for production endpoints that must remain isolated from test traffic throughout the engagement [10]. Red-team guardrails complement these boundaries at the operational level: API rate limits and cost caps prevent runaway token consumption during adversarial prompt campaigns, and agent kill switches—automated circuit breakers that terminate execution on detection of anomalous action sequences—ensure that a test scenario that escapes its intended containment boundary can be halted before material harm occurs [11]. The NIST Cybersecurity Framework 2.0 establishes governance and risk management as foundational prerequisites for any security testing program, recommending that scope definition, asset classification, and stakeholder authorization be completed and documented before any active assessment activity begins [11].

Phase 1 conducts an architecture and supply-chain review that maps every trust boundary in the agentic deployment before adversarial testing commences. Trust boundary mapping covers the interfaces between the agent planner and its tools, between tools and external APIs, between the retrieval pipeline and its corpora, and between the orchestration layer and the identity and access management systems governing tool credentials [10]. SBOM and provenance artifact collection identify gaps in supply-chain visibility: model checkpoints without cryptographic signatures, plugin packages without verified build provenance, and pipelines lacking attestation controls are flagged as high-priority remediation targets before functional exploitation proceeds [11]. This phase establishes a baseline configuration state for measuring all subsequent results so that the pen-test report reflects the security posture of the intended, documented system, not just an ambiguous run-time snapshot [10], [11].

Phase 2 executes structured prompt, policy, and guardrail testing using both direct jailbreak campaigns and indirect prompt injection suites targeting the retrieval pipeline. AI penetration testing practice recognizes that prompt-level attacks require dedicated test suites distinct from conventional input fuzzing, because the vulnerability being exploited is not a parsing

error but a semantic misinterpretation by the reasoning engine of the boundary between trusted instructions and untrusted content [10]. Jailbreak test cases probe instruction override, role confusion, and safety-policy bypass through progressively escalating input complexity, measuring the consistency of guardrail enforcement across varied phrasing and context configurations. IPI suites seed malicious instructions into candidate retrieval sources—web content, document stores, and API response payloads—and measure the rate at which those instructions reach and influence the agent's reasoning process. Context boundary tests verify prompt isolation across system, user, and tool prompt layers and confirm that memory stores do not leak sensitive context across independent session boundaries [10], [11].

Phase 3 begins with the abuse of tools and execution by schema fuzzing, sandbox escape attempts, and local/remote least privilege validation on each integrated tool. Each tool adapter will be subjected to boundary input testing that will exceed the defined input limits and reveal validation logic that is either missing or invalid, causing unintended effects to downstream systems [10]. Sandbox configuration will be evaluated against syscall filtering/installation, filesystem isolation/network exfiltration, or token budgeting conditions, while all data used for the above will be subjected to active escape attempts through each layer of review sequentially. Least-privilege audits confirm that tool credentials carry only the permissions required for their defined function and that no integration provides a path to capabilities outside its documented scope [11]. Chained call simulations—constructing multi-tool sequences such as web search followed by content retrieval followed by code execution—reveal emergent risks that are invisible in single-tool analysis and that represent the most realistic approximation of how an attacker would exploit a production agentic system [10], [12].

Phase 4 assesses data, memory, and RAG integrity through active poisoning campaigns and rollback control verification. Candidate malicious documents are injected into retrieval corpora, and the agent's subsequent outputs are measured for retrieval drift—the degree to which poisoned content influences reasoning and tool invocation decisions relative to a clean baseline [10]. Persistence tests confirm whether injected embeddings survive session boundaries and accumulate influence across multiple independent runs. The vector store will have canary entries, which are artificially created data that have definite distinguishing characteristics added to it so that there is a reliable way of detecting possible future poisonings. The rollback procedures will be used to verify that the system can be restored to a clean, verified state in an acceptable period of time [10], [11].

Phase 5 includes API and cost abuse testing through token storm simulation tests, tool looping campaigns, and authentication verification against all agent-to-service interfaces. During the token storms [10], testing will provide many requests containing substantial amounts of complexity to test the maximum amount of tokens consumed per request and validate that rate limiting and budget kill switches were activated before costs exceed the defined threshold. Tool looping testing creates recursive or cyclic patterns of tool invocation that provide a means to exploit an agent’s logic for pursuing goals to generate unlimited API calls to confirm that the abnormal activity detection systems will recognize and stop these activities within defined response times. Authentication and authorization between agent components and third-party APIs are verified against the trust model established in Phase 1, with particular attention to token scoping, credential rotation policies, and the absence of hardcoded secrets in tool adapter configurations [10], [12].

Phase 6 correlates all findings from Phases 1 through 5 with application security posture management data to separate exploitable issues from low-priority noise and produce a remediation backlog prioritized by reachability and operational impact. Any runtime signals collected from tests (tool calls, tokens consumed outside of normal usage levels, guardrail triggered rates, and significant deviation) will be added to the ASPM platform as part of an ongoing monitoring baseline, thus creating a closed feedback loop between test findings and actual production [11]. The NIST Cybersecurity Framework 2.0 recommends that security assessment outputs be operationalized as continuous monitoring inputs rather than treated as standalone reports, a principle that is directly instantiated in this phase through the handoff of test-derived behavioral signatures to the security operations program [11], [12]

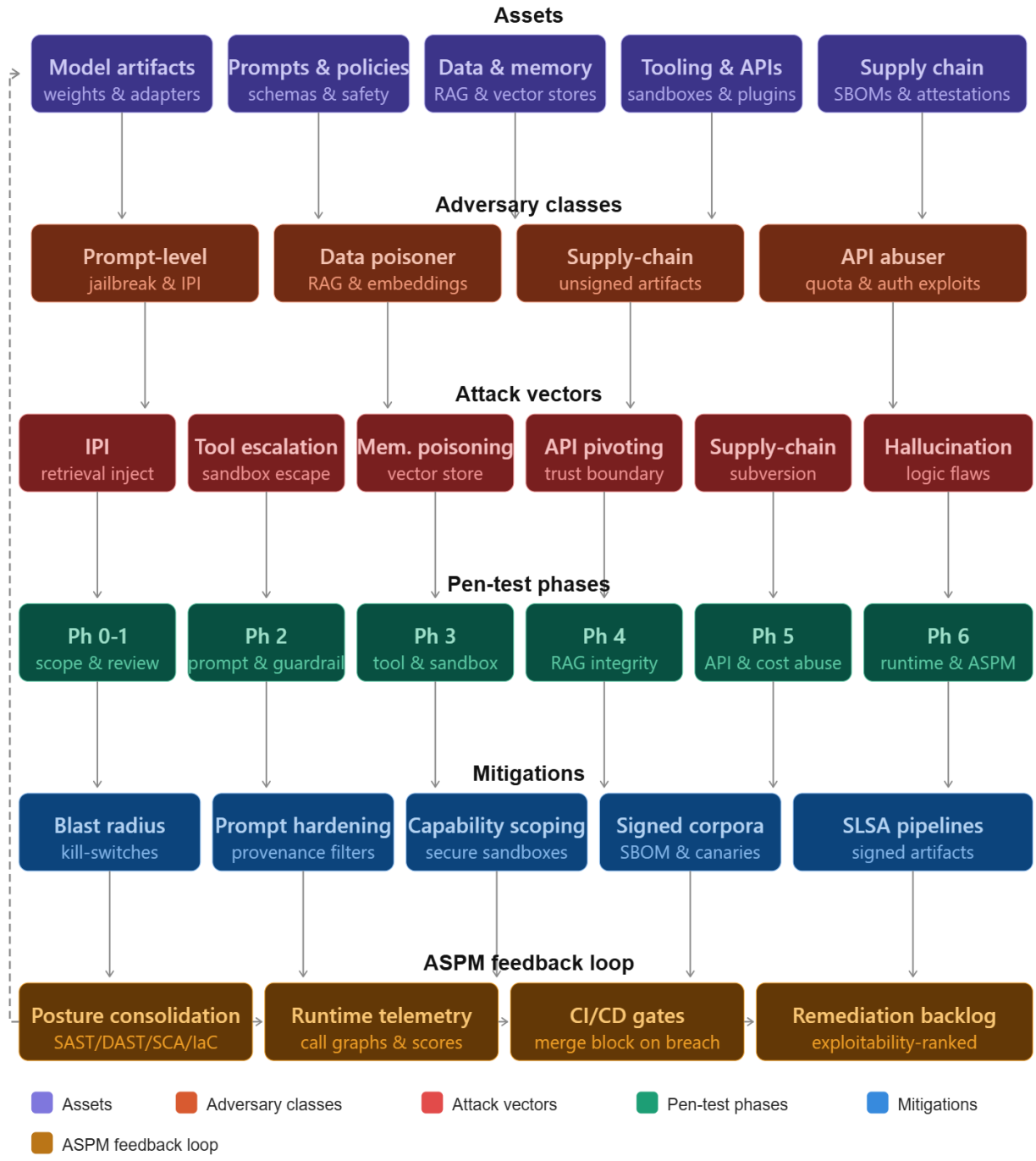

Figure 1: Agentic AI threat and testing lifecycle
Assets → Adversaries → Attack vectors → Pen-test phases → Mitigations → ASPM feedback loop

4.2 Mitigation Patterns

Prompt and policy hardening addresses the instruction manipulation attack surface through system-prompt compartmentalization that assigns explicit, narrowly scoped policies to each tool integration rather than relying on a single global safety configuration. Prompts are treated as versioned code artifacts: maintained in source control, reviewed through the same peer-review process applied to application code, and tested against regression suites before every deployment cycle [10]. Content provenance filters applied at the retrieval layer inspect all externally sourced inputs for embedded instructions, hidden directives, and anomalous formatting patterns before that content reaches the reasoning engine,

providing a defense-in-depth layer that operates independently of output filtering and addresses the retrieval-layer entry point that IPI attacks exploit [10], [11].

Capability scoping enforces a default-deny posture for high-risk tool actions, requiring explicit, per-engagement authorization for file writes, network egress, code execution, and any operation that modifies persistent state in connected systems. Enforced syscall filtering, network isolation, filesystem restriction, and per-session token budgets on secured sandboxes define an execution boundary that tool adapters utilize to operate [10]. Anomalous Actions within Sandbox Configuration Such As Unexpected File Access, Outbound Connections to Unrecognized Endpoints, or Token Consumption Spikes are monitored by embedded tripwires when certain thresholds are exceeded and automatically cause the shutdown of the agent, therefore reducing the time a tool used by an attacker (who may have escalated to Partial Tool Use) has operated [11], [12].

The RAG pipeline security is based on the integrity of signed corporeality with SBOM and PBOM metadata. Each document or dataset ingested into the retrieval corpus is associated with a cryptographic signature and a provenance record that enables verification of its source, modification history, and authorization status at retrieval time [11]. Regular integrity scans compare the current embedding store contents against known-good baselines, and canary entries with distinctive semantic signatures are maintained throughout the corpus to provide a low-cost, continuous poisoning-detection signal. Context time-to-live policies limit the persistence of sensitive data in long-term memory stores, and, where feasible, differential privacy controls are applied to reduce the inferability of individual records from aggregate retrieval outputs [10], [11].

SLSA-aligned pipelines with signed model artifacts and attestation verification at deploy time close the supply-chain subversion attack path by ensuring that every component introduced into the deployment environment can be traced to a verified, authorized build process. SBOMs for model weights, tokenizer configurations, and training data lineage are created and updated continuously throughout the life of the model; credentials for the model registry are rotated periodically to prevent too much time from passing with a key that may be compromised [11]. ASPM Consolidation Combines The Results Of Static Analyses, Dynamic Testing, Software Composition Analyses, Infrastructure As Code Scanning, And Runtime Telemetries To Form A Capitalized, Prioritized Risk View. On a go-forward basis, reachability and exploitability will be used instead of the raw severity rate as the computational basis for remediation sequencing [12]. CI/CD policy violations will be created for each agentic-specific finding (confirmed IPI vector(s), tool permission violations, and supply chain integrity gaps) and will not allow deployments to happen until the finding has been resolved, embedding security enforcement directly into the development workflow [11], [12].

Attack Class	Asset Targeted	Pen-Test Phase	Primary Mitigation	ASPM Integration Point
Indirect Prompt Injection (IPI)	Prompts & Policies; RAG Corpora	Phase 2 — Prompt, Policy & Guardrail Testing	Content provenance filters at retrieval layer; instruction-neutralization for all externally sourced inputs before reaching reasoning engine	IPI vector flagged as CI/CD policy violation; merge blocked until guardrail regression resolved
Tool-Use Escalation & Sandbox Escape	Orchestration & Tooling; Execution Sandboxes	Phase 3 — Tool & Execution Abuse	Default-deny capability scoping; syscall filtering; per-session token budgets; auto-shutdown tripwires on anomalous sequences	Sandbox escape attempts correlated with runtime tool call graphs; anomaly scores fed to remediation backlog
Model & Memory Poisoning	Model Artifacts; Embedding Stores; RAG Corpora	Phase 4 — Data, Memory & RAG Integrity	Signed corpora with SBOM/PBOM metadata; canary entries for poisoning detection; context TTL policies; periodic integrity scans	Retrieval drift scores and canary alerts surfaced as exploitable findings in ASPM posture view
API & Workflow Pivoting	Orchestration Tooling; External APIs; Trust Boundaries	Phase 5 — API & Cost/Abuse Testing	Explicit trust boundary modeling per service; least-privilege tool credentials; inter-component authentication and authorization verification	Anomalous API call patterns detected at runtime; pivot paths prioritized by reachability score in ASPM
Supply-Chain Subversion	Model Artifacts; Supply-Chain Attestations; Build Pipelines	Phase 1 — Architecture & Supply-Chain Review	SLSA-aligned pipelines; signed model artifacts; SBOM coverage for weights, tokenizers, and datasets; registry key rotation on defined schedule	Unsigned artifact detection triggers CI/CD gate failure; SBOM coverage tracked as supply-chain integrity metric
Hallucination-Driven Logic Flaws	Prompts & Policies; Orchestration Planning Logic	Phase 2 — Prompt, Policy & Guardrail Testing	Output confidence thresholds; human-in-the-loop gates for high-consequence actions; adversarial and non-adversarial test suites covering planning loops	Behavioral deviation scores from planning loops fed into ASPM as reliability-security risk indicators

Note: Pen-Test Phases refer to the six-phase agentic AI penetration testing methodology defined in Section 4.1. ASPM = Application Security Posture Management; SBOM = Software Bill of Materials; PBOM = Pipeline Bill of Materials; SLSA = Supply-chain Levels for Software Artifacts; TTL = Time-to-Live; RAG = Retrieval-Augmented Generation.

Table 1: Attack Class × Mitigation Matrix for Agentic AI Systems.

4.3 DevSecOps Integration and Metrics

CI/CD gate controls embed security enforcement at every stage of the agentic system build and deployment pipeline. Prompt linting tools analyze system prompt configurations for structural vulnerabilities, policy gaps, and instruction patterns known to facilitate injection attacks before any build artifact is promoted to a staging environment [10]. Automated policy validation tests confirm that tool-use schemas enforce required input constraints and authentication requirements across all adapter configurations. Model artifact signature verification confirms that every checkpoint, tokenizer, and adapter weight introduced into the pipeline carries a valid cryptographic attestation from an authorized build process, and RAG source differential scanning at deploy time detects corpus modifications between pipeline runs that have not been reviewed and authorized through the standard change management process [11], [12].

Continuous red-team playbooks executed as nightly pipeline jobs maintain ongoing visibility into the security posture of agentic deployments between formal assessment cycles. AI penetration testing guidance recognizes that agentic systems require continuous rather than periodic testing postures because the combination of frequent model updates, dynamic tool integrations, and live retrieval corpora means that the attack surface can change materially within hours of a prior clean assessment [10]. These playbooks operate in safe, sandboxed profiles that replicate the most critical test scenarios from the six-phase methodology—IPI campaigns against updated retrieval sources, tool schema boundary tests against new adapter configurations, and supply-chain integrity checks against the current artifact inventory—and are configured to fail the build automatically on detection of any security regression relative to the previously established baseline [10], [12].

Four outcome metrics anchor the security program's evidence base and provide the quantitative foundation required for both operational program management and publication standards. Guardrail efficacy tracks jailbreak and IPI success rates before and after each remediation cycle, providing a direct measure of the program's effectiveness against the highest-priority attack classes identified in the threat model [10]. Tool-use safety measures the proportion of tools operating under verified least-privilege configurations and the ratio of blocked sandbox escape attempts to total attempts across all test executions. Supply-chain integrity measures SBOM and attestation coverage as a percentage of total model and dataset artifacts in the deployment inventory and tracks SLSA levels achieved per pipeline against defined maturity targets [11]. Runtime risk reduction measures mean time to detect and contain agent abuse events in the production environment and quantify the reduction in exploitable findings produced through ASPM correlation across successive assessment cycles [12]. An anonymized retail deployment provides quantitative validation of these metrics at scale: following remediation of an IPI attack executed through a supplier product page that caused an agentic assistant to exfiltrate customer data and initiate unauthorized refunds, structured program controls produced a 100% reduction in the original jailbreak path, a 70% reduction in high-risk tool actions, and complete SBOM coverage for all model and data artifacts within two sprint cycles [10], [11].

Conclusion

The emergence of agentic AI as a production enterprise technology introduces a category of security risk that is structurally distinct from the threats addressed by conventional application security programs, and the gap between deployment velocity and security readiness continues to widen as organizations expand agentic capabilities across operational workflows without the benefit of established threat models, testing frameworks, or governance controls tailored to the unique behavioral properties of these systems. Autonomous tool chaining, retrieval-based context injection, persistent memory stores, and opaque supply-chain artifact lifecycles collectively generate attack classes—indirect prompt injection, tool-use escalation, memory poisoning, API and workflow pivoting, supply-chain subversion, and hallucination-driven logic flaws—that static analyzers, dynamic scanners, and software composition tools cannot detect, prevent, or meaningfully prioritize without the runtime context and behavioral telemetry that only purpose-built agentic security programs can produce. The framework presented in this article addresses that gap through four integrated contributions: a structured threat model grounded in recognized adversarial AI taxonomies and industry governance frameworks; a six-phase penetration testing methodology designed for safe, repeatable, and CI/CD-compatible execution across the full agentic attack surface; mitigation patterns mapped directly to each identified attack class and operationalized through DevSecOps pipeline controls and ASPM-driven remediation workflows; and outcome metrics that deliver verifiable, quantitative evidence of security improvement across guardrail efficacy, supply-chain coverage, tool-use safety, and mean detection time. The directions articulated here are consistent with the broader industry shift away from isolated point-in-time scanning toward unified posture management, verifiable supply-chain provenance, and runtime-informed prioritization—a shift that represents not merely an incremental maturation of existing practice but a fundamental reorientation of how

enterprise application security programs must be designed to remain effective in an environment where AI-driven development, agentic automation, and expanding regulatory transparency requirements are simultaneously reshaping the threat landscape. Realizing the security program described here requires organizations to confront implementation challenges that are distinct from those encountered in conventional AppSec program buildouts. Tooling maturity remains uneven: purpose-built prompt linting engines, RAG corpus integrity scanners, and agentic behavioral telemetry platforms exist in early commercial and open-source form, but none have yet achieved the deployment breadth or operational validation that would qualify them as commodity infrastructure. Cost pressures are real and should be acknowledged directly—continuous red-team playbook execution, SBOM maintenance across model and dataset artifact inventories, and ASPM platform integration each carry non-trivial operational expenditures that must be justified against risk reduction evidence rather than assumed as baseline requirements. Organizational readiness is perhaps the most significant constraint: the roles, responsibilities, and escalation pathways required to govern agentic AI security span teams—AI engineering, security operations, data governance, and risk management—that in most enterprises do not yet operate with the cross-functional coordination structures these systems demand. Phased adoption is therefore the practical path for most organizations, beginning with the highest-impact controls—capability scoping, IPI filtering at the retrieval layer, and supply-chain signature verification—before expanding toward full ASPM integration and continuous red-team automation as tooling matures and organizational capacity develops.

Future work should prioritize the standardization of model SBOM formats across heterogeneous cloud and on-premises model registries, the systematic benchmarking of AI-assisted versus human-led agentic penetration testing approaches to quantify false-positive and false-negative tradeoffs, and the development of behavioral telemetry-driven prioritization systems that can serve as the primary input to security decision-making in production agentic deployments where the speed and complexity of autonomous action chains exceed the capacity of human-reviewed assessment cycles.

References

[1] Chang, Amy. "AI Security at Cisco and Integrated AI Security and Safety Framework Deep Dive." *Cisco Live EMEA*, 2026, www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2026/pdf/AI-2903.pdf.

[2] OX Security. "Application Security Trends Every DevSecOps Team Should Watch in 2026." *OX Security Blog*, 2026, www.ox.security/blog/application-security-trends-in-2026/.

[3] Gentyala, Sunil. "For Application Security: SCA, SAST, DAST, and MAST. What Next?" *CSO Online*, 2026, www.csoonline.com/article/4115679/for-application-security-sca-sast-dast-and-mast-what-next.html.

[4] OWASP. "OWASP Top 10 for Large Language Model Applications." *OWASP Foundation*, 2025, owasp.org/www-project-top-10-for-large-language-model-applications/

[5] Anthropic. "Model Card and Evaluations for Claude Models." *Anthropic*, 2023, www-cdn.anthropic.com/5c49cc247484cecf107c699baf29250302e5da70/claude-2-model-card.pdf.

[6] OWASP. "LLM Applications Cybersecurity and Governance Checklist v1.1 – English." *OWASP Gen AI Security Project*, 2024, genai.owasp.org/resource/llm-applications-cybersecurity-and-governance-checklist-english/

[7] Liaghati, Christina. "MITRE ATLAS Overview." *National Institute of Standards and Technology — CSRC*, 2025, csrc.nist.gov/csrc/media/Presentations/2025/mitre-atlas/TuePM2.1-MITRE%20ATLAS%20Overview%20Sept%202025.pdf

[8] Polemi, Nineta, and Isabel Praça. "Multilayer Framework for Good Cybersecurity Practices for AI." *ENISA*, 2023, www.enisa.europa.eu/sites/default/files/publications/Multilayer%20Framework%20for%20Good%20Cybersecurity%20Practices%20for%20AI.pdf.

[9] OffSec. "AI Penetration Testing: How to Secure LLM Systems." *OffSec Blog*, 2025, www.offsec.com/blog/ai-penetration-testing/.

[10] National Institute of Standards and Technology. "The NIST Cybersecurity Framework (CSF) 2.0." *NIST CSWP 29*, 2024, nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf

[11] Carielli, Sandy, et al. "The State of Application Security, 2025." *Forrester*, 2025, www.forrester.com/report/the-state-of-application-security-2025/RES182779.

[12] Gollapudi, Raghu. "Telemetry-Driven Predictive Failure Models for High-Scale Financial Databases." *Journal of Computational Analysis and Applications*, 2025, www.eudoxuspress.com/index.php/pub/article/view/4835.