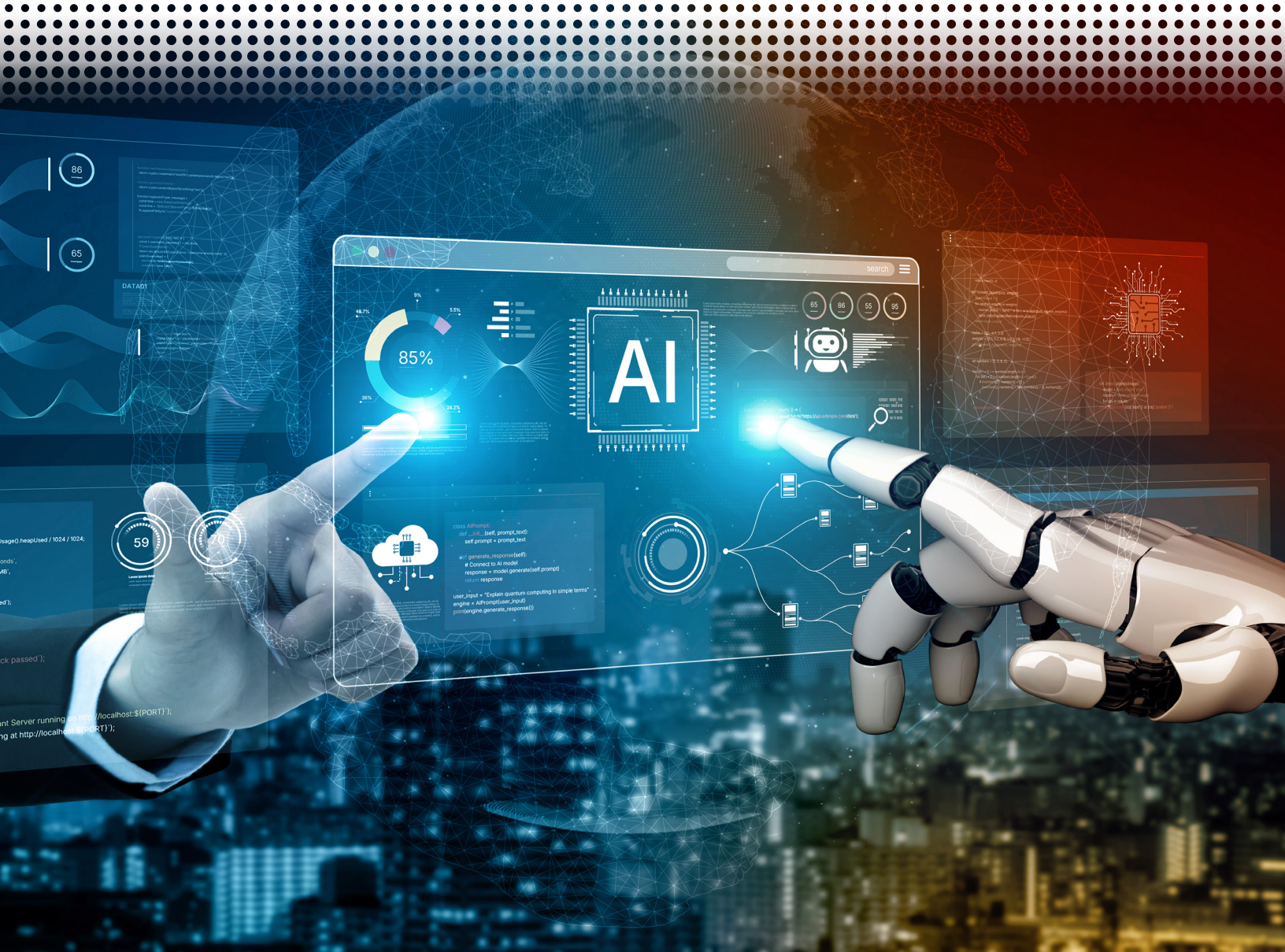
radware

Guide

How to Survive the OWASP Top Ten for Agentic Applications



Contents

Introduction

AI agents are redefining the way major industries operate. As they plan, decide and act across multiple steps and systems, cybercriminals are finding new ways to exploit their autonomy and trust. This guide to the OWASP Top Ten for Agentic Applications provides an overview of the risks posed by agentic AI use and the best ways to keep your organization secure.

ASI01: Agent Goal Hijack

AI agents can perform a series of tasks on their way to accomplishing an end goal. The downside? Savvy attackers find ways to manipulate final objectives, decision paths or multi-step plans. They exploit the way agents process natural language instructions and related content, and take advantage of the fact that the model can't reliably tell one from the other. This goes beyond a single "bad answer" to a prompt; it redirects the agent's planning and actions.

Risks:

Goal manipulation via indirect prompt injection, deceptive tool outputs and forged inter-agent messages, malicious artifacts driving action selection

How to Mitigate:

To prevent agent goal hijacking, organizations must apply security solutions that can map original agent and tool goals, detect goal divergence in real time and block any malicious attempt to hijack the agent's original goal. In addition, organizations must treat all natural language inputs as untrusted. They should route them through the same prompt-injection and content-filter safeguards used for LLM apps before they can influence goals, planning or tool calls. Enforce least privilege and require human approval or policy checks for high-impact actions. Lock and change control system prompts and reward/goal definitions. Validate user intent and agent intent at run time; pause/alert on unexpected goal shifts. Sanitize connected sources and continuously log/monitor goal state, tool use sequences, and deviations; red team for goal override scenarios.

Key Approaches:

Implement input hardening for all carrier channels (RAG, files, web, email), least privilege tools and human in the loop for high impact steps, locked system prompts and change control, runtime intent validation and anomaly detection

ASI02: Tool Misuse & Exploitation

There are several ways agents can be made to misuse legitimate tools such as APIs, shells, browser and MCP-provided tools. Causes include prompt injection, misalignment and unsafe delegation or ambiguous instruction. This can lead to data exfiltration, output manipulation or workflow hijacking. Tools covered in this risk category are legitimate but used in an unsafe or unintended way.

Risks:

Deletion of valuable data, manipulation of output, over-invoking costly APIs, exfiltration of information

How to Mitigate:

Apply the **principle of least privilege** and scope tools to minimal actions; gate sensitive calls with policy checks or human approval. Validate and **sanitize model output before tool invocation;** implement allow lists, schema validation, and strict argument contracts. Harden the **tool plane** by verifying tool identities, descriptors, routing and versions. Monitor for anomalous chaining, call bursts and cross domain pivots. Add rate limits, budgets and circuit breakers to prevent loop amplification and cost blow ups.

Key Approaches:

Implement fine-grained tool scoping and approvals for destructive ops, output tool validation (contracts, schemas, allowlists), tool identity integrity checks, rate limiting, budgets, and chain anomaly detection to tool validation (contracts, schemas, allow lists)

ASI03: Identity & Privilege Abuse

In the gap between user-centric identity systems and agentic design, attackers sometimes manipulate delegation chains, role inheritance, control flows and agent context to abuse agent identities, credentials and inherited permissions. This allows them to bypass controls, impersonate agents, move laterally and escalate privileges across APIs and services.

Risks:

Delegated privilege abuse, memory-based escalation, cross-agent trust exploitation, device-code phishing across agents, workflow authorization drift, forged agent persona, identity sharing

How to Mitigate:

Use **short lived, scoped credentials** with independent **agent identities** (not user tokens). Enforce role-based access control and attribute-based access control (RBAC/ABAC), just-in-time (JIT) expiring secrets, and least privilege per tool/integration. Isolate identities per agent and per environment. **Log/audit** every privileged action and continuously monitor for anomalous permission usage.

Key Approaches:

Separate agent vs. user identities; implement short-lived tokens, rotation and vaulting; use fine-grained RBAC/ABAC and JIT access with privileged action logging and anomaly detection

ASI04: Agentic Supply Chain Vulnerabilities

Attackers can target the agentic supply chain through third-party agents, tools and related artefacts that are malicious or compromised. These attacks can occur through static or dynamically sourced components, which are chosen by agentic ecosystems at runtime and create a live supply chain with cascading vulnerabilities across agents.

Risks:

Compromised tools/connectors, malicious updates, tainted models or dataset artifacts in the pipeline, unvetted MCP servers or tool catalogs

How to Mitigate:

The goal is to ensure you know which components your agents utilize, whether they're internal or third-party options. Apply AI security solutions that help you map those components at all times and in real-time, and allow you to control usage. Apply **software-supply-chain-discipline** to agent stacks: signed artifacts, version pinning, and integrity checks on tools and descriptors; vet third party model/data sources and MCP servers; sandbox and monitor third-party components; remove/disable unused tools.

Key Approaches:

Ensure SBOM/AIBOM and signature/provenance enforcement, MCP/tool catalog allow lists and integrity checks, controlled updates; version pinning; rollback plans, sandbox and monitor external components

ASI05: Unexpected Code Execution (RCE)

When agentic systems, including vibe coding tools, create and execute code, it is often generated in real time. This code bypasses traditional controls and leaves an opening for attackers to strike through prompt injection, tool misuse or unsafe serialization, converting text into executable behavior. Automated code generation or outside attacks could delete production data, embed shell commands calling for data exfiltration, hidden back doors and more.

Risks:

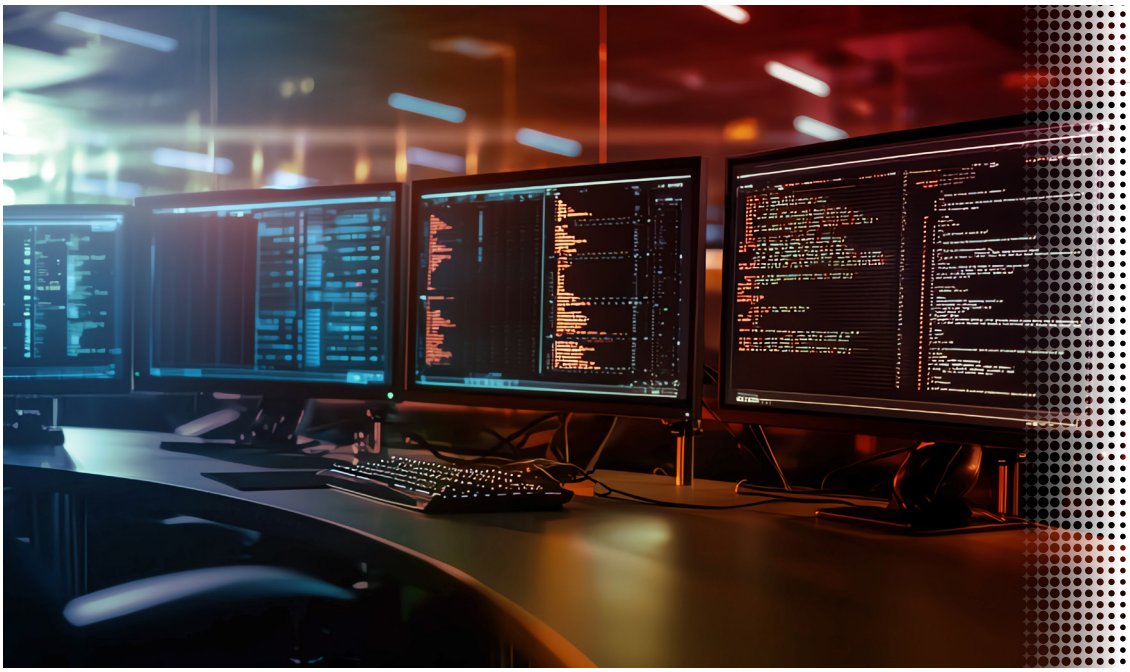
Prompt injection, code hallucination, shell command invocation, unsafe function calls

How to Mitigate:

Enforce **strict sandboxing**, non-root execution, read-only filesystems where possible; validate **arguments and outputs** before execution (schemas, allow lists); disallow high-risk ops by policy; egress controls for code exec sandboxes; monitor for **command anomalies** and block network exfil from exec contexts.

Key Approaches:

Ensure strong sandboxing and least privilege for exec tools, output to exec validation (contracts/allowlists), egress controls and read-only FS; telemetry on commands to exec validation (contracts/allow lists)



ASI06: Memory & Context Poisoning

Agentic systems access stored data like conversation history to preserve continuity across tasks and reasoning. Adversaries corrupt **session context** or **long term agent memory** (vector stores, notes, summaries) so future reasoning and actions are steered toward harmful outcomes—distinct from one-off prompt manipulation.

Risks:

Poisoned memory entries altering future plans, corrupt RAG/context stores influencing tool calls, persistent “drift” via recurring seeded artifacts

How to Mitigate:

The most efficient way is to implement agentic AI protection solutions that monitor agent memory at all times to prevent memory poisoning and to prevent malicious instructions that infiltrated into Agents’ memory. Apply **sanitization and validation** to anything persisted to memory; gate what gets written with policies and filters. Use **provenance tags**, TTLs, and confidence scores; detect and quarantine suspicious embeddings/entries; snapshot and **audit memory changes**; add **human review** for high impact updates; cross check with trusted sources before acting.

Key Approaches:

Implement memory write policies and sanitization, provenance/confidence tagging, memory audit trails and snapshot/rollback, cross source validation before actioning memory.

ASI07: Insecure Inter Agent Communication

Agents are constantly making contact with one another. They exchange messages, delegate tasks and pass tool outputs. Without authenticated channels, schemas, and content validation, forged or malformed messages can drive unsafe actions or lateralize risk across agent swarms.

Risks:

Spoofed peer messages/instructions, schema less payloads carrying prompt carriers, unauthenticated/unencrypted channels

How to Mitigate:

Require mutual auth, integrity protection, and strict schemas for inter-agent payloads; validate intent and scope on receipt; quarantine free-text fields from directly steering tools; implement allow-listed message types and rate limits; observe conversation graphs for anomalies (loops/fan out).

Key Approaches:

Enact mTLS/authN and integrity checks; typed contracts/schemas; reject freeform instructions, message allowlists; rate limit coordination paths, graph/sequence anomaly detection across agents form instructions lists; rate limit coordination paths

ASI08: Cascading Failures

In an AI-first world, one incident can bypass protections and spread across autonomous agents into system-wide mayhem. Small errors (a poisoned context, a mis scoped tool, a bad delegate) **amplify** through multi step plans or multi agent workflows, triggering runaway costs, outages, or compounding safety breaks. This risk focuses on the propagation and amplification of faults rather than just the original incident.

Risks:

Planner loops and uncontrolled retries, chain of tools amplification and cross domain pivots, hidden dependencies across agents/services

How to Mitigate:

Engineer **circuit breakers**, budgets, and **step limits**; add *plan validation* and **fail-safe** defaults; implement saga/rollback patterns for side effecting actions; design **blast-radius isolation** between agents/domains; make failure modes observable (cost, latency, error budgets) and auto halt on anomaly.

Key Approaches:

Implement rate limits, budgets, circuit breakers; max steps; plan validation and safe fallbacks, transactional rollbacks / compensations, blast radius isolation between agents/services limits, budgets, circuit breakers; max step radius isolation between agents/services

ASI09: Human Agent Trust Exploitation

Agents can act as a bad influence, abusing trust established with a human user through natural language fluency. Attackers can exploit this to influence user decisions, steal important data or direct outcomes for malicious purposes. They manipulate agent outputs, status messages, and summaries—nudging approvals, bypassing policy, or laundering malicious decisions through “AI said so” dynamics.

Risks:

Social engineering via authoritative agent messaging, rubber stamping of risky actions, misleading rationales or hallucinated justifications

How to Mitigate:

Mandate **human-in-the-loop** for high risk actions with clear, verifiable **explanations** (evidence links, provenance) and **differential UI** cues for confidence/uncertainty. Audit approvals; train users on prompt-borne threats; establish **kills witches** and require secondary approvals for irreversible operations. **-in-the-loop** risk actions with clear, verifiable borne threats; establish **-switches**

Key Approaches:

Implement structured human approvals and evidence-backed rationales, confidence/uncertainty surfacing; EDU for operators, kill switches and secondary approvals for destructive ops, audit trails for all human agent decision points switches agent decision points

ASI10: Rogue Agents

Rogue agents stray from their original purpose, acting harmfully or deceptively within their environment. Autonomous misalignment or emergent behavior leads an agent to act **outside intended bounds** without an active attacker in the loop.

Risks:

Misaligned objectives/reward hacking, unbounded autonomy, persistence of harmful behaviors across runs

How to Mitigate:

Constrain **agency by design** (least agency principle): minimize autonomy to what the use case requires; define **hard guardrails**, scopes, and **kill-switches**; ensure **observability** into agent rationale, plan, and tool calls; use **progress-gated** workflows with checks at decision points; regularly red team for misalignment and verify rollback/containment. (Mirrors OWASP's emphasis on least agency and strong observability for agent operations.)

Key Approaches:

Implement dedicated agentic AI protection solutions that monitor agents and tools and know agents goals, roles and privileges. It should be able to identify goal / role and privileges divergence at real-time and be able to report and block those actions. Use methods such as least agency + explicit scopes/guardrails, explainability/traceability of plans, rationale, tool calls, progress gates and kill switches, scheduled red teaming for misalignment/containment

Harness your AI agents and protect against their risks.

Discover Radware Agentic AI Protection

RWI6310 | 03.09.2026