AI Agent Pre-Deployment Security Checklist

Five control families to verify before any agentic system reaches production.

> Agentic systems fail differently from traditional software. Their behavior is probabilistic, their attack surface includes the tools you hand them, and their failures are often silent. Run this list before you ship. Each control is a yes or no. If you cannot check it, you have found work to do.

1 Probabilistic Behavior Testing

You cannot unit-test a distribution with a single pass. Verify the agent behaves within tolerance across repeated, adversarial, and drifted conditions.

☐ Re-test schedule defined A fixed cadence (weekly at minimum, plus on every model or prompt change) is documented and has a named owner.

WHY *Model and dependency updates silently shift behavior. Untested drift is undetected risk.*

☐ Adversarial test suite runs in CI PyRIT or an equivalent tool executes a maintained attack-prompt corpus on every build.

WHY *Manual red-teaming does not scale and is not repeatable. CI makes regressions immediately visible.*

☐ Confidence and refusal thresholds set Minimum confidence and refusal rate thresholds are configured and asserted in automated tests.

WHY *An agent that never refuses or always refuses is misconfigured; both failure modes must be bounded.*

☐ Non-determinism bounded Temperature, top-p, and seed policy are documented and locked for production; deviations require a change record.

WHY *Undocumented sampling parameters make behavior unreproducible and failures undiagnosable.*

☐ Behavioral baselines captured Output distributions and key behavioral metrics are recorded; significant deviations trigger alerts.

WHY *You cannot detect drift without a baseline to compare against.*

☐ Golden-path and known-bad prompts both covered The test suite includes representative happy-path inputs and a curated set of known-attack inputs.

WHY *Testing only the happy path catches regressions but misses the adversarial failure modes that matter most.*

☐ Output schema validated automatically Every agent response is validated against a declared schema or format contract before downstream consumption.

WHY *Silent schema drift causes cascading failures in dependent systems and is rarely caught without enforcement.*

☐ Regression gate enforced The build fails if attack pass-rate or behavioral metrics worsen compared to the baseline.

WHY *A gate that does not block the build is a suggestion, not a control.*

2 Training Data and Supply Chain

The data and weights your agent relies on are part of its attack surface. Verify provenance, integrity, and access controls before the model touches production.

☐ Corpus provenance documented Sources, licenses, collection dates, and responsible parties for all training and fine-tuning data are recorded.

WHY *Undocumented data makes compliance, legal, and security reviews impossible to complete honestly.*

☐ RAG and index sources inventoried Every data source feeding retrieval-augmented generation is listed, access-controlled, and reviewed.

WHY *An unlisted or over-permissioned index is an injection vector and a data-leakage risk.*

☐ Poisoning resistance considered Any user-contributed or repository-sourced content fed into training or indexing has been reviewed for poisoning risk.

WHY *Adversarial training-time inputs can embed persistent backdoors that survive fine-tuning.*

☐ Model and dependency versions pinned All model weights, libraries, and infrastructure dependencies are version-pinned and a software bill of materials is recorded.

WHY *Floating versions allow silent upgrades that change behavior and introduce unreviewed code.*

☐ Third-party model weights integrity-verified Checksums or signatures for downloaded weights are verified before use.

WHY *A compromised weight file is indistinguishable from a legitimate one without an integrity check.*

☐ Sensitive data excluded from corpora PII, credentials, and confidential material are filtered or excluded from training datasets and indexed content before ingestion.

WHY *Data included at training or indexing time can surface in model outputs under adversarial or unexpected prompts.*

☐ Update approval path defined The process for approving a model or data version bump, including who signs off and what tests must pass, is written down.

WHY *Ad hoc updates bypass the controls that protect production and leave no audit trail.*

☐ Indexed content treated as untrusted Code repositories, wikis, and other external corpora are treated as untrusted input and never granted instruction-level authority.

WHY *Trusted-by-default indexed content is the primary vector for indirect prompt injection attacks.*

3 Agent Tool Controls

Every tool you give an agent expands its blast radius. Minimize permissions, validate inputs, and audit every call.

☐ Tool allowlist verified The set of tools available to the agent is explicit, minimal, and reviewed; no implicit or ambient tool access exists.

WHY *Tools not on the allowlist cannot be called; implicit access creates undocumented capabilities that are difficult to audit.*

☐ Every tool parameter schema-validated Tool inputs are validated against a strict schema before execution; malformed inputs are rejected, not coerced.

WHY *An agent can be tricked into constructing malicious tool inputs; schema validation is a hard gate.*

☐ Credentials scoped per tool Each tool uses a dedicated, least-privilege credential; no broad tokens or shared secrets are used.

WHY *A single compromised credential should expose only one tool's scope, not the entire environment.*

☐ Destructive actions gated Irreversible or destructive tool calls require explicit confirmation or a human-in-the-loop step before execution.

WHY *An agent acting on a malicious prompt should not be able to cause irreversible harm without a human checkpoint.*

☐ Tool outputs validated before re-entry Outputs from tool calls are treated as untrusted and validated before they re-enter the agent's context window.

WHY *A tool returning attacker-controlled content can inject instructions directly into the agent's reasoning.*

☐ Rate and quota limits per tool Each tool has enforced rate and quota limits to bound the blast radius of a runaway or compromised agent.

WHY *Without limits, a single misconfigured call can exhaust resources or trigger mass data exfiltration.*

☐ Secrets never in model context API keys, passwords, and tokens are injected at the infrastructure layer and never appear in prompts, responses, or logs.

WHY *Anything in the context window can be extracted by a sufficiently crafted prompt or logged inadvertently.*

☐ Tool call audit trail captured Every tool invocation is logged with its full input parameters and outcome in a tamper-resistant store.

WHY *Without a complete audit trail, incident investigation and compliance review are impossible.*

4 Prompt Injection Defense Layers

Assume injection will be attempted. Defense must hold even if the model is fully compromised. Architecture, not model judgment, is the control.

☐ Input sanitization and structural separation applied System instructions and user-supplied data are structurally separated and the model is never asked to treat data as instructions.

WHY *Models that cannot distinguish instructions from data are trivially injectable; structure is the first defense.*

☐ Untrusted content clearly delimited Retrieved content from web pages, documents, or repositories is wrapped in explicit delimiters and the model is instructed never to follow instructions within them.

WHY *Indirect injection attacks embed instructions in retrieved content; delimiting is the primary mitigation.*

☐ Output validated against policy before action Agent outputs are validated against a declared schema and policy before any action or tool call is executed.

WHY *A compromised model may produce plausible but malicious outputs; validation is the last gate before action.*

☐ Behavioral monitoring for anomalous tool use Tool-use sequences are monitored for anomalies (unexpected chaining, unusual targets, high frequency) with alerting.

WHY *Injection-driven behavior often manifests as atypical tool-use patterns that are detectable at the orchestration layer.*

☐ Privilege boundaries hold under full model compromise Authorization checks live outside the model; the system does not rely on the model refusing a request to enforce access control.

WHY *If the model is compromised, it will not refuse. The surrounding system must enforce all boundaries independently.*

☐ Spotlighting applied to retrieved content Retrieval-augmented content is marked with a spotlighting technique (e.g., XML tags, special tokens) so the model can distinguish it from authoritative context.

WHY *Without spotlighting, retrieved content and system instructions are indistinguishable to the model.*

☐ Egress controls active The agent cannot send data to arbitrary external destinations; outbound tool calls are restricted to a known allowlist of endpoints.

WHY *Exfiltration via injection requires an outbound channel; controlling egress removes the most direct exfiltration path.*

☐ Injection test cases in the CI adversarial suite The automated test suite includes prompt-injection payloads covering direct, indirect, and multi-turn attack patterns.

WHY *Injection defenses that are not tested regularly regress silently as prompts, models, and retrieval sources change.*

5 Pre-Ship Sign-Off

This section is not about code. It is about accountability. Someone must own the decision to ship, with full knowledge of the residual risks.

☐ Audit logging verified end to end Logs capture prompts, tool calls, and outcomes in a tamper-resistant store; retention, access, and alerting have been tested with a real event.

WHY *Logs that have not been tested may be missing data or misconfigured; verify before production, not during an incident.*

☐ Scope-limited credentials confirmed in production Least-privilege credentials are in place in the production environment, not just in staging or developer configs.

WHY *Credentials that were scoped correctly in testing are sometimes replaced with broad tokens for convenience at deploy time.*

☐ Monitoring and alerting active and tested All monitoring rules and alert channels have been verified with a real fired alert in the production environment.

WHY *An alert that has never fired has never been proven to work; paper monitoring is not monitoring.*

☐ Rollback and kill-switch rehearsed A procedure to disable or roll back the agent exists, is documented, and has been tested in a non-production drill.

WHY *The first time you need a kill switch should not be during a production incident.*

☐ Data-handling reviewed for applicable obligations Data flows have been reviewed against relevant regulatory obligations, including EU AI Act deployer duties where the system is in scope.

WHY *Regulatory obligations for high-risk AI systems include specific deployer duties that cannot be retrofitted after launch.*

☐ Incident runbook exists with a named owner A written runbook for agent-related incidents exists, names a specific owner, and has been reviewed in the last 90 days.

WHY *Runbooks without named owners are orphaned; when an incident occurs, ownership ambiguity delays response.*

☐ Residual risks documented and accepted Known risks that remain after controls are applied are written down and explicitly accepted by a named decision-maker.

WHY *Undocumented residual risk is not accepted risk; it is unknown risk, which is categorically worse.*

☐ Sign-off recorded A record exists showing the decision to ship, with date, version, approver name, and a reference to the risk acceptance document.

WHY *Without a recorded sign-off, there is no accountability and no basis for a post-incident review.*
