OWASP TOP 10 FOR AGENTIC APPLICATIONS · 2026 · FACILITATOR GUIDE

Identity & Privilege Abuse

When agents trust each other by default, one low-privilege agent can borrow another's power — and act as something it isn't.

ASI03:2026 | SEVERITY: CRITICAL | RISK 03 OF 10

WHAT THIS DOCUMENT IS

The companion to the Agent Identity & Privilege Abuse 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

This risk is where classic access control meets agents — and loses, because the oldest shortcut in distributed systems ('internal calls are trusted') is catastrophically wrong when the caller is a language model that a stranger's text can steer.

> **The risk in one sentence.** Delegated authority, ambiguous identity, or trust assumptions let an agent take actions it was never authorized to take — impersonating another agent, inheriting privilege through a chain, or slipping around role-based access control.

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

- Explain why implicit trust between agents is dangerous.
- Recognise impersonation and privilege-inheritance patterns.
- Apply per-agent identity, scoped tokens, and least privilege.
- Audit agent-to-agent and agent-to-tool calls.

02 · Background — why this risk exists

SLIDES 4–6

Agent architectures inherit two habits from microservices: shared service credentials and implicit trust on internal networks. Both were tolerable when callers were deterministic programs. They stop being tolerable when a caller's behaviour can be redirected by content it read. Add delegation — planner calls executor calls tool — and authority accumulates along the chain: each hop keeps the broadest token it was handed, so the final actor holds more privilege than any single step required.

Why it matters now

The GitHub MCP case is the canonical example and worth dwelling on. No exploit, no credential theft. The agent simply held one token valid across both the public repository it was reading and the private repositories it was then persuaded to read. The scope of the delegated authority, not any software defect, is what turned a prompt injection into a data breach.

What changes when the system is agentic

Classic LLM or application	Agentic system
A service with one API key fails in known ways, and you rotate the key.	Agents delegate to each other dynamically, so authority flows along paths you didn't design — and implicit trust turns a small foothold into admin.

03 · Explaining it to a non-technical audience

SLIDE 5 — THE ANALOGY

The visitor badge that opens the vault. A courier is handed a staff badge "just to get through the lobby." But the badge also opens the vault. No one checks who's really holding it — the door only checks that **a** valid badge was tapped.

In the analogy	Maps to	Why it works
THE BADGE	**Delegated authority**	Power is passed between agents without re-verifying who's really using it.
THE VAULT	**Over-broad scope**	The borrowed identity can reach far more than the task ever needed.
THE FIX	**Check the holder**	Give each agent its own scoped identity, and verify it on every call — no shared badges.

> PRESENTER TIP
> The visitor-badge analogy lands hardest if you tie it back to the GitHub case immediately: same badge for the lobby and the vault, and the door only checked that a valid badge was tapped.

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 · DELEGATE	A task is handed from one agent to another.
02 · ASSUME	The callee is trusted because the call is "internal."
03 · INHERIT	The caller's or a shared token's scope is reused.
04 · ACT	The agent acts with privilege it was never granted.
05 · EVADE	The action slips past role-based access controls.

05 · Attack vector 1 — Agent impersonation

SLIDE 8

HOW IT WORKS One agent presents itself as another — often a higher-privilege one — because identity is asserted, not verified.

What it looks like in practice:

- Spoofing an "admin" agent's name or role.
- Reusing a shared service identity.
- No cryptographic proof of who is calling.

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

```
summarizer ▸ call as {role:"admin_agent"}
system: name matches → grant admin scope
summarizer ▸ now holds DB-WRITE
✗ acting as an agent it is not
```

06 · Attack vector 2 — Implicit trust between agents

SLIDE 9

HOW IT WORKS Internal calls skip authorization because they're assumed safe. Anything that can reach the internal channel inherits that trust.

What it looks like in practice:

- "Internal caller → skip authz."
- Shared network = shared trust.
- No per-call identity verification.

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

```
agentB ▸ request → admin_endpoint
system: internal caller → skip authz
admin_endpoint ▸ executes as admin
✗ RBAC bypassed via trust
```

07 · Attack vector 3 — Privilege inheritance through chains

SLIDE 10

HOW IT WORKS A long delegation chain widens scope: each hop keeps the broadest token, so the final actor holds more than any single step needed.

What it looks like in practice:

- Broad token passed down every hop.
- No scope narrowing on delegation.
- Final agent acts with accumulated power.

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

```
A(read) → B(read+write) → C
C reuses B's write token
C ▸ delete_records()
✗ read-only origin, destructive end
```

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.

> RESEARCHER DEMONSTRATION **GitHub MCP — one token spanning public and private scope**
>
> 2025-05-26 · SECURITY RESEARCH / RESPONSIBLE DISCLOSURE
>
> The GitHub MCP attack worked because the agent held a single token valid for both the public repository it was reading and the private repositories it was then coerced into reading. Nothing was 'hacked': the agent simply used delegated authority far wider than the task required, which is exactly the privilege-scoping failure this module describes.
>
> **Source:** Invariant Labs
>
> https://invariantlabs.ai/blog/mcp-github-vulnerability

CONFIRMED INCIDENT **1,862 agent tool servers exposed with no authentication at all**

2025-07-17 · SECURITY RESEARCH (INTERNET-WIDE EXPOSURE STUDY)

Knostic identified 1,862 internet-exposed MCP servers. Of the 119 they manually verified, every single one served its internal tool listing without authentication — no identity required to enumerate an AI system's capabilities or invoke its registered tools. Identity was not weak here; it was absent. Use this when someone argues that agent infrastructure is protected because it sits inside the network.

Source: Knostic

https://www.knostic.ai/blog/mapping-mcp-servers-study

DISCLOSED VULNERABILITY **MCP Inspector — missing authentication for a critical function**

2025-06-13 · CVE-2025-49596 · VENDOR ADVISORY

Anthropic's MCP Inspector proxy accepted unauthenticated requests to launch MCP commands (CWE-306, Missing Authentication for Critical Function). Any website a developer visited could reach the locally bound API — DNS rebinding defeated same-origin protection — and execute code as that developer. A textbook case of a privileged capability exposed without an identity check because the caller was assumed to be local and therefore trusted. CVSS 9.4, fixed in 0.14.1.

Source: GitHub Advisory Database; analysis by Oligo Security

https://github.com/advisories/GHSA-7f8r-222p-6f5g

09 · Worked scenario — The summarizer that deleted the database

SLIDE 11

Illustrative. Constructed to show the mechanism end to end.

A low-privilege "summarizer" agent calls a high-privilege "admin" agent's endpoint. Internal calls are trusted, so authorization is skipped.

1. **SETUP — Trusted internals.** Agent-to-agent calls on the internal bus skip authz checks.
2. **REACH — Cross-agent call.** The summarizer invokes the admin agent's endpoint directly.
3. **INHERIT — Scope granted.** Because the call is "internal," it runs with admin scope.
4. **IMPACT — Destructive action.** It deletes records it should never have been able to see.

WHY THIS MATTERS

Identity abuse is critical because it converts a small, low-value foothold into high-value action. The attacker doesn't need to breach the admin agent — just to be trusted by it.

10 · Common misconceptions

EXPECT THESE AS QUESTIONS

What people say	What to say back
"It's an internal call, so authorisation is unnecessary."	Internal describes the network path, not the trustworthiness of the request. Authorise every call, internal or not.
"One service account is simpler to manage."	A shared identity erases attribution and makes least privilege impossible: every agent holding it can do everything any of them can do.
"The 'from' field tells us who is calling."	A self-declared identity is a claim. Only cryptographic proof — mTLS, a signed token — is evidence.

11 · Containment — the three layers

SLIDE 13

Give each agent its own verifiable identity and stop issuing shared service tokens. Make credentials task-scoped and short-lived, so a token stolen or misdirected is worth little and expires quickly. Narrow scope at each delegation hop rather than passing the broadest token down the chain. Authorise every call — including agent-to-agent — and log the verified identity behind each action so escalation is visible in an audit rather than inferred after an incident.

LAYER 01 — PER-AGENT IDENTITY

- Give every agent its own cryptographic identity.
- Verify it on every call — no shared service tokens.
- Prefer mTLS or signed, short-lived tokens.

LAYER 02 — LEAST PRIVILEGE

- Scope tokens to the specific task, not the agent's max.
- Narrow scope on each delegation hop.
- No implicit trust for "internal" callers.

LAYER 03 — AUTHORIZE & AUDIT

- Enforce authz on every agent-to-agent and agent-to-tool call.
- Log the real identity behind each action.
- Alert on scope escalation and cross-agent calls.

12 · For developers — secure code patterns

SLIDE 14

VULNERABLE

```
# internal = trusted
if call.is_internal:
    run(action)   # no authz, no identity
# any internal agent inherits full power
```

HARDENED

```
ident = verify_mtls(call)   # who, really
scope = token_for(task)     # least priv
if not authz(ident, action, scope):
    deny()
run(action, as=ident)
```

Each agent has a **verified identity**, holds a **least-privilege scoped token**, and is **authorized on every call** — internal or not.

Where the controls live

Point	Control
IDENTITY	**Verify the caller.** mTLS or signed tokens prove which agent is calling — assertions aren't enough.
SCOPE	**Least-privilege tokens.** Issue task-scoped, short-lived credentials; narrow scope on every hop.
AUTHZ	**Check every call.** No "internal → skip authz." Authorize agent-to-agent and agent-to-tool calls alike.
AUDIT	**Log real identity.** Record the verified actor behind each action and alert on escalation.

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
ESCALATION	**Scope jump.** An agent suddenly acts with higher privilege than its role allows.
CROSS-CALL	**Unexpected caller.** A low-privilege agent invoking a privileged endpoint.
SHARED-TOKEN	**Reused credential.** The same token seen across agents that should have distinct identities.
WRITE	**Unexpected mutation.** A read-only agent performing writes or deletes.

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	**Impersonate.** Call an endpoint claiming another agent's role. Pass = identity is verified and the call denied.
PROBE 2	**Internal trust.** Reach a privileged endpoint via the internal bus. Pass = authz still runs.
PROBE 3	**Inherit scope.** Chain calls to reuse a broad token. Pass = scope is narrowed per hop.
PROBE 4	**RBAC bypass.** Try a write from a read-only agent. Pass = blocked and alerted.

Ship-ready checklist

Control	Done when
Per-agent identity.	Every agent has a verifiable cryptographic identity, not a shared token.
Scoped tokens.	Credentials are least-privilege, task-scoped, and short-lived.
No implicit trust.	Internal calls are authorized like any other.
Scope narrowing.	Delegation narrows privilege at each hop.
Identity audit.	The real actor behind each action is logged and escalation is alerted.

16 · Knowledge check and discussion

SLIDE 20

Q1. An internal agent calls a privileged endpoint. Skip authorization?

No. "Internal" is not "authorized." Verify identity and check authz on every call, internal or not.

Q2. Why is a shared service token dangerous between agents?

It erases identity. Any agent holding it acts as all of them, so you can't scope or attribute actions.

Q3. What limits privilege inheritance through a chain?

Narrowing scope at each hop and issuing least-privilege, task-scoped tokens rather than reusing a broad one.

Take it back to your own systems

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

1. Do our agents share a service account? What could the least-trusted one do with it?
2. If a low-privilege agent called a privileged endpoint right now, would anything stop it?
3. Can we answer 'which agent did this' from logs, or only 'the platform did this'?

17 · Glossary

Term	Meaning
Delegated authority	One agent acting on behalf of another, using granted permissions.
Least privilege	Granting only the permissions a specific task requires, nothing more.
mTLS	Mutual TLS — both sides prove identity with certificates.
RBAC	Role-Based Access Control — permissions tied to roles.
Privilege escalation	Gaining higher access than intended, here via trust or inheritance.

18 · Key takeaways

SLIDE 22

1. **Implicit trust is the vulnerability.** "Internal" must never mean "authorized."
2. **Give every agent its own identity.** Verify it cryptographically on every call.
3. **Scope tokens to the task.** Least privilege, short-lived, narrowed each hop.
4. **Audit the real actor.** Log who truly acted and alert on escalation.

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
GitHub MCP — one token spanning public and private scope	https://invariantlabs.ai/blog/mcp-github-vulnerability
1,862 agent tool servers exposed with no authentication at all	https://www.knostic.ai/blog/mapping-mcp-servers-study
MCP Inspector — missing authentication for a critical function	https://github.com/advisories/GHSA-7f8r-222p-6f5g

Citation. OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications (2026)*, risk ASI03:2026 — Agent Identity & Privilege Abuse.