OWASP TOP 10 FOR AGENTIC APPLICATIONS · 2026 · FACILITATOR GUIDE

Insecure Inter-Agent Comms

Multi-agent systems pass messages between planners and executors. On an open channel, those messages can be read, changed, or forged.

ASI07:2026 | SEVERITY: HIGH | RISK 07 OF 10

WHAT THIS DOCUMENT IS

The companion to the Insecure Inter-Agent Communication 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

Multi-agent systems reinvent network security, usually badly. An internal message bus with no authentication is a remote-code-execution primitive with extra steps: whoever reaches it can issue orders that executors obey.

> **The risk in one sentence.** Manipulation of the messages exchanged between agents, planners, and executors — interception and modification, injection of malicious instructions, or forging messages that appear to come from a trusted agent.

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

- Explain why unauthenticated agent channels are dangerous.
- Recognise interception, injection, and forgery of messages.
- Apply message signing, encryption, and identity verification.
- Test whether a forged message can drive an action.

02 · Background — why this risk exists

SLIDES 4–6

When teams split work across planner and executor agents, the messages between them become a control plane. Yet these channels are frequently plaintext, unauthenticated, and unauthorised, on the reasoning that they are internal. The receiving agent then trusts a self-declared 'from' field — a claim, not evidence. Replay makes it worse: a legitimately authorised message captured once can often be resent indefinitely.

Why it matters now

Agent-to-agent protocols are young and were designed for capability first. Deployments regularly expose tool servers and message buses without authentication, and default configurations frequently bind broadly rather than to localhost. The security model many teams assume — 'it's behind the firewall' — is the same assumption that made lateral movement easy for decades.

What changes when the system is agentic

Classic LLM or application	Agentic system
A single service validates its own inputs and speaks to one trusted backend.	Many agents exchange messages over shared channels; if identity and integrity aren't enforced, any participant can forge or alter another's orders.

03 · Explaining it to a non-technical audience

SLIDE 5 — THE ANALOGY

Orders shouted across a noisy floor. On a busy factory floor, workers act on shouted orders without checking who shouted. A stranger yells "the boss says shut down line 3" — and it happens, because nobody verifies the **voice**, only the words.

In the analogy	Maps to	Why it works
THE SHOUT	**Unsigned message**	A command taken at face value with no proof of sender.
THE STRANGER	**Injected/forged traffic**	Anyone on the channel can insert or alter orders.
THE FIX	**Verify the voice**	Sign and encrypt messages; verify sender identity, not just the content.

> **PRESENTER TIP**
> The noisy-factory-floor analogy makes the 'from field' problem obvious. Ask: how would the worker verify the voice? That is exactly what a signature does.

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 · OBSERVE	Attacker gains access to the agent channel.
02 · READ	Unencrypted messages reveal tasks and data.
03 · ALTER	A message in flight is modified.
04 · FORGE	A message is spoofed as a trusted agent.
05 · EXECUTE	The receiving agent acts on the tampered order.

05 · Attack vector 1 — Interception & modification

SLIDE 8

HOW IT WORKS Messages travel unencrypted or unverified, so an attacker on the path can read them and change their contents before delivery.

What it looks like in practice:

- Plaintext task messages.
- No integrity check on payloads.
- Man-in-the-middle on the bus.

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

```
planner → executor: {do:"summarize"}
← attacker modifies in flight →
executor receives: {do:"delete all"}
✗ tampering undetected
```

06 · Attack vector 2 — Injected instructions

SLIDE 9

HOW IT WORKS An attacker inserts new messages into the channel, adding instructions the legitimate agents never sent.

What it looks like in practice:

- Extra tasks pushed to the queue.
- Injected 'system' directives.
- No sender authentication.

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

```
queue ▸ [ summarize, translate ]
attacker ▸ push { do:"export_db" }
executor ▸ runs export_db
✗ injected instruction executed
```

07 · Attack vector 3 — Forged trusted-agent messages

SLIDE 10

HOW IT WORKS A message is spoofed to look like it came from a trusted agent, so the receiver grants it authority it shouldn't have.

What it looks like in practice:

- Spoofed 'from: planner' field.
- Replayed old authorised messages.
- Identity asserted, never verified.

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

```
msg{ from:"planner", do:"drop users" }
executor ▸ trusts 'from' field
executor ▸ drop table users
✗ forged authority obeyed
```

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 **1,862 MCP servers exposed on the internet with no authentication**
>
> 2025-07-17 · SECURITY RESEARCH (INTERNET-WIDE EXPOSURE STUDY)
>
> Knostic scanned for MCP protocol markers and identified 1,862 internet-exposed MCP servers. Of 119 they manually verified, all 119 granted access to internal tool listings with no authentication whatsoever — meaning any external party could enumerate an AI system's full capability inventory and invoke registered tools before any security control applied. This is the 'it's internal so it's safe' assumption failing at scale.
>
> **Source:** Knostic
>
> https://www.knostic.ai/blog/mapping-mcp-servers-study

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

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

Anthropic's MCP Inspector proxy accepted unauthenticated requests to launch MCP commands over stdio (CWE-306). Any website a developer visited could reach the locally bound API, including via DNS rebinding to defeat same-origin protections, and execute code on the developer's machine. CVSS 9.4, fixed in 0.14.1.

Source: GitHub Advisory Database; analysis by Oligo Security

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

DISCLOSED VULNERABILITY **mcp-remote — a malicious server executing commands on its client**

2025-07-09 · CVE-2025-6514 · VENDOR SECURITY RESEARCH

mcp-remote passed a server-supplied authorization_endpoint URL to open() without validation during OAuth setup. A malicious or compromised MCP server could return a crafted value and execute arbitrary OS commands on the client machine — the trust relationship inverted, with the remote peer attacking the local agent. CVSS 9.6, affects 0.0.5–0.1.15, fixed in 0.1.16.

Source: Or Peles, JFrog Security Research

https://research.jfrog.com/vulnerabilities/mcp-remote-command-injection-rce-jfsa-2025-001290844/

RESEARCH PAPER **Prompt Infection — injections that self-replicate between agents**

2024-10 · ARXIV:2410.07283 · PEER-REVIEWED RESEARCH PAPER

Lee and Tiwari demonstrate malicious prompts that propagate across interconnected agents much like a computer virus: each agent's output is itself an injection aimed at the next. They show the infection persists even when agents restrict what they share, and propose LLM tagging as a partial defence. ESORICS 2025 Workshops.

Source: Donghyun Lee and Mo Tiwari

https://arxiv.org/abs/2410.07283

09 · Worked scenario — The forged planner order

SLIDE 11

Illustrative. Constructed to show the mechanism end to end.

Planner and executor agents communicate over an unauthenticated internal bus. An attacker injects a forged 'planner' message telling the executor to delete records.

1. **SETUP — Open bus.** Agents exchange messages with no signing or encryption.
2. **FORGE — Spoofed sender.** The attacker crafts a message with from:planner.
3. **TRUST — Face-value authority.** The executor trusts the 'from' field and acts.
4. **IMPACT — Destructive command.** Records are deleted on a forged order.

WHY THIS MATTERS

Inter-agent messaging is often treated as 'internal and safe,' so it skips the authentication and integrity controls external APIs get — exactly the gap an attacker needs.

10 · Common misconceptions

EXPECT THESE AS QUESTIONS

What people say	What to say back
"It is behind the firewall."	Perimeter security fails to one compromised host or one SSRF. The bus should authenticate every message regardless.
"We validate the message schema."	Schema validation proves the shape is right, not that the sender is who they claim or that the content was not altered.
"Nobody knows our internal endpoints."	Obscurity is not authentication, and internal topology is routinely discoverable.

11 · Containment — the three layers

SLIDE 13

Apply the controls you would demand of any external API. Sign every inter-agent message and verify the signature before acting on it; authenticate both ends with mutual TLS so identity is proven rather than asserted. Include a nonce or timestamp and reject stale or duplicate messages, which closes replay. Authorise per command — being a valid sender does not mean being allowed to issue that particular instruction. Then alert on unsigned, mismatched, or malformed messages, because those are unambiguous attack signals rather than noise.

LAYER 01 — AUTHENTICATE SENDERS

- Sign every message; verify the signature on receipt.
- Use mutual TLS between agents.
- Trust identity that's proven, not asserted.

LAYER 02 — PROTECT IN TRANSIT

- Encrypt agent-to-agent channels.
- Add integrity checks so tampering is detected.
- Use nonces/timestamps to stop replay.

LAYER 03 — VALIDATE & SCOPE

- Validate message schema and authority per action.
- Scope what each agent may command.
- Log and alert on unsigned or invalid messages.

12 · For developers — secure code patterns

SLIDE 14

VULNERABLE

```
# trust the 'from' field
if msg.from == "planner":
    execute(msg.action)
# no signature, no encryption
```

HARDENED

```
chan = mtls_channel(peer_certs)
if not verify_sig(msg, keys[msg.from]):
    drop(); return
if fresh(msg.nonce) and authz(msg):
    execute(msg.action)
```

Messages are **signed and verified**, channels are **encrypted (mTLS)**, replay is blocked with **nonces**, and each command is **authorised per sender**.

Where the controls live

Point	Control
SIGN	**Message signatures.** Every inter-agent message is signed; receivers verify before acting.
ENCRYPT	**Secure channels.** Use mTLS/encryption so traffic can't be read or altered in flight.
FRESH	**Anti-replay.** Nonces and timestamps prevent replaying old authorised messages.
AUTHZ	**Per-command authority.** Validate schema and check the sender is allowed to issue that action.

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
UNSIGNED	**Missing signature.** A message accepted without a valid signature.
MISMATCH	**Sender mismatch.** The claimed sender doesn't match the verified identity.
SCHEMA	**Malformed order.** Messages that don't match the expected task schema.
REPLAY	**Duplicate nonce.** The same authorised message seen more than once.

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	**Intercept.** Read messages on the channel. Pass = traffic is encrypted and unreadable.
PROBE 2	**Modify.** Alter a message in flight. Pass = the integrity check rejects it.
PROBE 3	**Forge.** Spoof a trusted sender. Pass = signature verification fails and it's dropped.
PROBE 4	**Replay.** Resend an old authorised message. Pass = the nonce/timestamp blocks it.

Ship-ready checklist

Control	Done when
Signed messages.	Every inter-agent message is signed and verified before action.
Encrypted channels.	Agent-to-agent traffic uses mTLS/encryption.
Anti-replay.	Nonces or timestamps prevent message replay.
Sender authorisation.	Each command is checked against what that sender may do.
Alerting.	Unsigned, mismatched, or malformed messages are logged and alerted.

16 · Knowledge check and discussion

SLIDE 20

Q1. Is an internal agent bus safe to leave unauthenticated?

No. Anyone who reaches it can read, alter, or forge messages. Sign, encrypt, and verify sender identity.

Q2. What's wrong with trusting a message's 'from' field?

It's unverified. A sender can be spoofed. Verify a cryptographic signature, not a self-declared name.

Q3. Which control stops replay of an old authorised order?

Nonces or timestamps that make each message fresh and single-use.

Take it back to your own systems

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

1. Are our agent-to-agent messages signed and encrypted, or trusted because they are internal?
2. If someone reached our message bus, what is the most damaging order they could issue?
3. Could an old authorised message be replayed against us today?

17 · Glossary

Term	Meaning
Message bus	Shared channel agents use to exchange tasks and results.
mTLS	Mutual TLS — both agents authenticate with certificates.
Integrity check	Verifying a message wasn't altered in transit.
Replay attack	Re-sending a valid message to trigger an action again.
Nonce	A one-time value that makes each message unique.

18 · Key takeaways

SLIDE 22

1. **Internal channels aren't automatically safe.** Anyone on them can read, alter, or forge.
2. **Verify the sender, not the label.** Signatures beat self-declared 'from' fields.
3. **Encrypt and anti-replay.** mTLS plus nonces close the gap.
4. **Authorise each command.** Scope what every agent may order.

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
1,862 MCP servers exposed on the internet with no authentication	https://www.knostic.ai/blog/mapping-mcp-servers-study
MCP Inspector — missing authentication on a critical function	https://github.com/advisories/GHSA-7f8r-222p-6f5g
mcp-remote — a malicious server executing commands on its client	https://research.jfrog.com/vulnerabilities/mcp-remote-command-injection-rce-jfsa-2025-001290844/
Prompt Infection — injections that self-replicate between agents	https://arxiv.org/abs/2410.07283

Citation. OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications (2026)*, risk ASI07:2026 — Insecure Inter-Agent Communication.