

Homework — Annotate an Agent Workflow for Injection Surfaces

devs + security + operators · ~25 minutes · architecture analysis

Below is the lifecycle of a perfectly ordinary support agent. It does what users actually ask for: it reads a link, searches for context, writes a draft, and posts it to a ticket. Nothing in this workflow is unusual, and nothing in it is safe. Your job is to walk every step and make the injection surfaces explicit before an attacker does it for you.

For each of the seven steps, write two short notes: (1) the **injection surface** introduced or compounded at that step, and (2) one **concrete mitigation** you would apply at that step. "Concrete" means it names a mechanism — an isolation session, a sanitiser, an allowlist, an approval gate — not "be careful" or "validate input."

Do the full annotation before opening [answer.html](#). The exercise only works if you have committed to a surface and a mitigation for each step on your own first.

The agent workflow

A user opens a support ticket containing a link. The agent:

1. Receives the user request (the ticket text plus the link).
2. Fetches the linked URL.
3. Summarizes the page.
4. Calls a search tool with the summary as the query.
5. Reads the matching search results.
6. Writes a draft response.
7. Posts the draft to the ticket and notifies a human reviewer.

For each step, write your annotation in this shape:

"Step N. Surface: <what untrusted bytes enter the context, and how>. Mitigation: <one concrete mechanism that contains the blast at this step>."

The whole exercise is an excuse to practice spotting **data/control-plane conflation**. At every step, ask: which bytes in the model's context were authored by the operator, and which were authored by whoever wrote the page, the search result, or the ticket? The model cannot tell them apart. You have to.

Self-check

- ✓ You wrote an annotation for every one of the seven steps — none skipped, even the ones that "look fine."
- ✓ Each annotation names the surface in terms of *which untrusted bytes* enter the context at that step, not just "the input."
- ✓ Each mitigation is concrete — it names a mechanism (sub-session with no tools, allowlist, sanitiser, length cap, structured notification, human approval) — not a generic "validate" or "be careful."
- ✓ At least one of your mitigations is a **human-in-the-loop** gate on a side-effect step, and at least one is an **isolation** step where untrusted content is processed in a session with no tools.
- ✓ You can state, in one sentence, the worst thing that happens if every mitigation you proposed fails. If you can't state it, you have not finished the exercise.