Tucker Payment Processing · Standards self-assessment

OWASP Top 10 for Agentic Applications 2026

How Tucker and the AI SOC Squirrel answer the risks of AI that uses tools and takes actions · Self-assessment · November 24, 2026

8 met **1 partly met** **1 not applicable**

About this document

This is a self-assessment written by Michael Cunningham, who designed and built the site. It is not an audit. The OWASP GenAI Security Project published this list in December 2025 for agentic AI: systems that plan, call tools and act, rather than only answer. Both AIs on the site are agents in that sense. Tucker, the support agent, decides which of his lookup tools to call. The AI SOC Squirrel, the security analyst, reads attack events and proposes responses.

The design answer to almost every risk on this list is the same: agency without authority. Each agent can choose what to look at, but nothing it chooses can change anything. Every action that matters is taken, or refused, by Python code that the agent cannot see or influence.

The ten risks

ASI01 Agent Goal Hijack — MET

Injected instructions can try to redirect an agent, but there is nothing to redirect it toward. Tucker's only possible actions are read-only lookups of the verified customer's own records, so a hijacked Tucker can at worst say something wrong. Known injection phrasing is blocked before it reaches him. The AI SOC Squirrel reads attacker-controlled log text, so every line is shortened, defanged and marked as untrusted data, and Cipher deliberately plants instructions in the logs on every run to prove the squirrel reports them instead of obeying them.

Evidence: `app/guardrails.py`, `app/tools.py`, `app/cipher/soc.py`, `tests/test_cipher_soc.py`; Cipher attack "Plant instructions in the logs for the AI SOC".

ASI02 Tool Misuse and Exploitation — MET

Tools are few, narrow and checked. Each chat page offers only the tools it needs, unknown tool names are refused, a reply may use at most four tool rounds, any argument a tool doesn't expect is dropped before it runs, and every database query is parameterized. No tool takes a customer number from the AI: account tools act only on the customer the session verified. The squirrel's action tool can only propose one of three fixed actions, and Python rejects anything off the menu.

Evidence: `app/tools.py`, `app/chat.py`, `tests/test_guardrails.py`, `tests/test_page_chats.py`, `tests/test_cipher_soc.py`; Cipher attack "Get Tucker to issue a refund".

ASI03 Identity and Privilege Abuse — MET

The agents hold no credentials. There is no API key, password or token in either AI's context that it could leak or reuse. Who a visitor is gets decided by Python's identity check, which locks after repeated failures and issues a new session ID on success. The database login behind the chat can only read, and it authenticates with the operating system's identity rather than a password, so even the application's own access can't be stolen and replayed.

Evidence: `app/session.py`, `app/tools.py`, `db/roles.sql`, `deploy/postgres-store-login.md`, `tests/test_limits.py`.

ASI04 Agentic Supply Chain Vulnerabilities — MET

No third-party plugins, tool servers or downloaded agents are used. Every tool is a short Python function in this repository, reviewed and tested like the rest of the code. The model comes from Google through its official SDK, the instructions are in version control, and Python dependencies are pinned and watched by Dependabot, with CodeQL and secret scanning on every change.

Evidence: `app/tools.py`, `app/llm.py`, `requirements.txt`.

ASI05 Unexpected Code Execution — NOT APPLICABLE

Neither agent can run code. There is no code interpreter, shell or file tool, and nothing either AI writes is ever executed, evaluated, or used to build a database query or a URL. Their output is shown only as plain text.

Evidence: `app/tools.py`, `site/assets/chat.js`.

ASI06 Memory and Context Poisoning — MET

There is no long-term memory to poison. Tucker's conversation memory is the last eight messages of one visitor's session, kept in server memory only and gone after 30 minutes of inactivity, so one visitor cannot plant anything another visitor's Tucker will read. The policy library he retrieves from is four documents under version control. The squirrel sees only the current attack run's events, never other visitors' traffic or earlier runs.

Evidence: `app/session.py`, `app/chat.py`, `app/policies.py`, `tests/test_cipher_soc.py`.

ASI07 Insecure Inter-Agent Communication — MET

The two agents never talk to each other. Each talks only to Python, and the squirrel's answers must come back as a small, fixed JSON format that Python validates field by field and strips of markup before anything is used or shown. There is no channel an attacker could spoof to pass instructions from one AI to the other.

Evidence: `app/cipher/soc.py`, `tests/test_cipher_soc.py`.

ASI08 Cascading Failures — MET

Every AI failure stops where it happens. If Gemini is down or over budget, Tucker apologizes or naps and the store keeps working. If the squirrel returns garbage, loops or says nothing, its report is built from plain facts by code, and the attacker is still blocked by code after 50 attacks. Spend caps, rate limits and the four-round tool limit stop one runaway conversation from draining what everyone else needs.

Evidence: `app/chat.py`, `app/spend.py`, `app/cipher/detect.py`, `app/cipher/soc.py`, `tests/test_guardrails.py`, `tests/test_cipher_soc.py`.

ASI09 Human-Agent Trust Exploitation — PARTLY MET

Tucker says he is an AI in every chat, is instructed never to ask for a full card number, CVV, password or Social Security number, and his replies are filtered for anything that looks like one. The Behind the Scenes panel shows people exactly what happened to each message, so trust is earned in the open. What a real deployment would add: a visible way to reach a human, and monitoring of conversations for manipulation, since a fooled model could still say something misleading.

Evidence: `app/chat.py`, `app/guardrails.py`, `site/index.html`.

ASI10 Rogue Agents — MET

A compromised or misbehaving agent has no way to act on its own: it runs only when a visitor asks something or an attack run wakes it, it cannot schedule itself, and every action is code's decision. Its activity is counted in the owner's nightly report and real-time alerts. And there are kill switches: Cipher's Lair turns off with one setting, the spend caps stop all AI use, and the model can be swapped or disabled in one place.

Evidence: `app/cipher/api.py`, `app/spend.py`, `app/config.py`, `app/report.py`, `app/alerts.py`.

The same attacks in MITRE ATLAS

Cipher tags each AI attack with its MITRE ATLAS technique as well as its OWASP category, so the live log and the incident report speak both languages: direct prompt injection (AML.T0051.000), indirect prompt injection through the logs (AML.T0051.001), system prompt extraction (AML.T0056), data leakage (AML.T0057), unauthorized tool use (AML.T0053) and cost harvesting (AML.T0034.001).

Evidence: `app/cipher/attacks.py`.

Beyond the demo

Agents with real authority, such as one that could issue refunds, would need human approval for each consequential action, per-action limits, an audit trail of every tool call, and regular red-team exercises against the agent itself. This site avoids those risks by not granting the authority at all, which is the strongest control when the business can live without it.

Source: https://github.com/mcunningham-infosec/tucker-payment-processing/blob/main/docs/standards/owasp-agentic-top-10.md · Live site: https://tuckerpaymentprocessing.com
Written by Michael Cunningham with AI assistance (Claude). A self-assessment, not an audit, certification or legal opinion. Tucker Payment Processing is a fictional company and processes no real payments.