Tucker Payment Processing · Standards self-assessment

OWASP Top 10 for LLM Applications 2026

How Tucker Payment Processing answers each of the ten risks · 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 or a certification. Tucker Payment Processing is a fictional company and processes no real payments, but its AI is real: Tucker, the support agent, and the AI SOC Squirrel, the security analyst, both run on Google Gemini. Every claim below names the code, configuration or automated test that backs it, and anyone can watch the defenses work live in Cipher's Lair.

The 2026 edition of the list, released by the OWASP GenAI Security Project in August 2026, is built on one idea: the model will be fooled, so the job is to limit what a fooled model can do. That is the design principle of this site. Tucker is never trusted to decide anything that matters; Python code decides, and the AI can only read.

The ten risks

LLM01:2026 Prompt Injection — MET

Three layers. First, every message is normalized and stripped of hidden control characters, then checked against known injection phrasing and blocked before it reaches the model. Second, and more important, a successful injection gains nothing: Tucker's tools are read-only, and whose account can be read is decided by Python from the verified session, never by the model. Third, indirect injection is handled the same way: when the AI SOC Squirrel reads attack logs, every line is shortened, defanged and labeled as untrusted data, and the actions it can take are checked by code. Tucker accepts text only, so image and audio injection do not apply.

Evidence: `app/guardrails.py`, `app/tools.py`, `app/cipher/soc.py`, `tests/test_guardrails.py`, `tests/test_cipher_soc.py`; Cipher attacks "Order Tucker to ignore his rules" and "Plant instructions in the logs for the AI SOC".

LLM02:2026 Sensitive Information Disclosure — MET

The best protection is data the AI can never see. The database holds no full card numbers (only the last four digits and a random vault token), and the tools return only the verified customer's own records, minimized: last four digits, no ZIP code or email echoed back. Every reply then passes an output filter that masks anything shaped like a card number (Luhn-checked), a Social Security number or an API key. Visitor IP addresses are never sent to the model, and Gemini's paid service does not train on these chats.

Evidence: `db/schema.sql`, `app/tools.py`, `app/guardrails.py`, `tests/test_guardrails.py` (masking, last-4 only, key scrubbing, another customer's transaction refused); Cipher attack "Ask for every customer's full card number".

LLM03:2026 Excessive Agency — MET

Tucker has ten tools and every one is a read-only lookup: no refunds, no money movement, no card changes, no order edits. Each page's chat gets only the tools it needs (the store chat cannot reach account tools), a reply may use at most four tool rounds, and the database login behind the chat can only read company data, so even a code mistake could not let the AI change anything. The AI SOC Squirrel has two tools: one reads this attack run's events, the other only proposes an action from a fixed menu of three, which Python accepts or rejects. Blocking the attacker is done by code whether the AI answers or not.

Evidence: `app/tools.py`, `app/chat.py`, `db/roles.sql`, `app/cipher/soc.py`, `tests/test_page_chats.py`, `tests/test_cipher_soc.py`, `tests/test_guardrails.py` (database refuses writes); Cipher attack "Get Tucker to issue a refund".

LLM04:2026 Supply Chain

MET

Every Python dependency is pinned to an exact version. GitHub Dependabot raises alerts for vulnerable or malicious packages and opens fix pull requests (one was reviewed and merged on November 24, 2026), CodeQL scans every change and secret scanning blocks committed keys. The pages load no third-party scripts, fonts or trackers, except Cloudflare's human check on Cipher's Lair. The model is Google's hosted Gemini, reached through Google's official SDK, with the model version fixed in one setting.

Evidence: `requirements.txt`, `app/config.py`, `app/llm.py`, `deploy/apache-hardening.conf` (Content-Security-Policy); Cipher attacks "Check for third-party scripts to poison" and "Check for unpinned dependencies".

LLM05:2026 Data and Model Poisoning

MET

The site trains and fine-tunes nothing, so there is no training data to poison. The only data that shapes Tucker's answers is the policy library (four Markdown documents) used for retrieval. Those live in the code repository under the same change control as the code, and an automated test fails if the policy pages visitors read ever differ from what Tucker is given.

Evidence: `policies/`, `app/policies.py`, `deploy/build_policy_pages.py`, `tests/test_policy_pages.py`.

LLM06:2026 Unbounded Consumption

MET

Layered limits, all in one settings file: 500 characters per message; per-minute, per-session and per-day limits per visitor and per network (an IPv6 visitor counts as their whole /64 network); a fair share so one network cannot use up the day's AI budget (200 questions, tightening to 50 once half the day's budget is spent); request bodies over 16 KB refused unread; and money caps of $0.75 a day and $8 a month in the app, with Google's own $9 hard cap behind them. When a cap is hit Tucker "takes a nap" instead of failing, and the owner is alerted.

Evidence: `app/config.py`, `app/limits.py`, `app/spend.py`, `app/alerts.py`, `deploy/apache-hardening.conf`, `tests/test_limits.py`, `tests/test_guardrails.py` (spend cap); Cipher attacks "Send a novel to run up the AI bill", "Flood the chat with requests" and "Send an enormous request to exhaust memory".

LLM07:2026 Misinformation

PARTLY MET

Tucker is told to use his tools for every fact and never invent account data, numbers or policy rules. Account answers come from database lookups, and policy answers come from retrieval over the policy documents, with the source policy named. The model runs at a low temperature (0.3). What a real company would add: an evaluation set of known questions run on every change to measure answer accuracy, and a visible way to reach a human. Those are not built here.

Evidence: `app/chat.py` (instructions), `app/policies.py`, `app/llm.py`.

LLM08:2026 Hidden Context Exposure

MET

Tucker's instructions are public: they are in the open-source repository. So leaking them gives an attacker nothing, and that is deliberate. No secret, key, customer data or security rule lives in the prompt. Authorization, limits and masking are Python code the model cannot see or change, and the tool descriptions expose no account numbers or internal IDs. Requests to reveal the hidden prompt are also blocked at the input check.

Evidence: `app/chat.py`, `app/tools.py`, `app/guardrails.py`; Cipher attack "Trick Tucker into revealing his hidden instructions".

LLM09:2026 Vector and Embedding Weaknesses

NOT APPLICABLE

By design there is no vector database and there are no embeddings. Retrieval is simple, auditable keyword scoring (TF-IDF) over four public policy documents, and that library holds no customer data, so nothing can leak between users through it.

Evidence: `app/policies.py`.

LLM10:2026 Improper Output Handling — MET

The AI's output is treated as untrusted. The chat inserts every reply as plain text, never as HTML, and the Content-Security-Policy forbids inline scripts besides. The AI SOC Squirrel must answer in a small JSON format, which Python validates and strips of markup before anything is shown, and its report PDF is built from those checked fields. Model output never reaches a database query, a shell or a URL.

Evidence: `site/assets/chat.js`, `app/cipher/soc.py`, `deploy/apache-hardening.conf`, `tests/test_cipher_soc.py`; Cipher attacks "Make Tucker print a card number" and "Cross-site scripting through the chat".

Cipher uses the 2026 numbering

Cipher's Lair and the AI SOC Squirrel's incident reports label every AI attack with the 2026 numbers above. They were updated from the 2025 edition on November 24, 2026, three months after the new list was released. Each AI attack is also tagged with its MITRE ATLAS technique (for example AML.T0051.000, direct prompt injection), the adversary-technique catalog security operations teams use.

The web application list too

Cipher's 50 attacks also cover the OWASP Top 10:2025 for web applications: broken access control (7 attacks), security misconfiguration (9), supply chain (2), cryptographic failures (5), injection (6), insecure design (4), authentication failures (3), software and data integrity (2), logging and alerting (1) and exceptional conditions (4), plus 7 aimed at the AI. Each run is checked by the real defenses, counted by code, and written up by the AI SOC Squirrel in a downloadable incident report.

Evidence: `app/cipher/attacks.py`, `app/cipher/detect.py`, `app/cipher/runner.py`.

Beyond the demo

A production deployment would add an AI evaluation suite run on every release, a software bill of materials (SBOM) for each build, red-team exercises by people other than the builder, and a human escalation path from the chat.

Source: https://github.com/mcunningham-infosec/tucker-payment-processing/blob/main/docs/standards/owasp-llm-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.