Tucker Payment Processing · Standards self-assessment

NIST AI Risk Management Framework (AI RMF 1.0)

How Tucker's risks are governed, mapped, measured and managed · Self-assessment · November 24, 2026

14 met **5 partly met**

About this document

This is a self-assessment written by Michael Cunningham, who designed and built the site. It is not an audit. Tucker Payment Processing is a fictional company with a real AI: Tucker, a customer-support agent, and the AI SOC Squirrel, a bounded security analyst, both running on Google Gemini. The NIST AI RMF (NIST AI 100-1, January 2023) is voluntary and organizes AI risk work into four functions: Govern, Map, Measure and Manage. Below, each of its 19 categories is mapped to what the site actually does, with the evidence. It also draws on NIST's Generative AI Profile (NIST AI 600-1, July 2024).

A one-person demonstration cannot show everything a large organization needs, such as committees, workforce programs and formal sign-offs. Where a category depends on that, it is marked Partly met, with what a real company would add.

Govern: a culture of risk management

GOVERN 1: Policies, processes and procedures are in place and followed — MET

The policies were written before the AI was built, and the AI follows them. The AI Assistant Use Policy (TPP-AI-01) sets what Tucker may and may not do, and the card security, refunds and data retention policies set the rules he explains. Tucker answers from these exact documents, and an automated test fails if the published policy pages drift from them. Every guardrail the policy promises is enforced in code and covered by a test.

Evidence: `policies/`, `app/policies.py`, `tests/test_policy_pages.py`, `tests/test_guardrails.py`.

GOVERN 2: Accountability is clear, and people are trained — PARTLY MET

One accountable owner, named on every page and in every source file: Michael Cunningham, who designed, built and runs the site and receives every alert and the nightly security report. In a real company this would be a RACI across the product owner, security, legal, compliance and an AI risk committee, with AI-specific training for each role.

Evidence: `README.md`, `app/report.py`, `app/alerts.py`.

GOVERN 3: Diverse perspectives and accessibility inform risk decisions — PARTLY MET

The site is designed for non-technical visitors: plain-language explanations, a guided tour and pages tested on phones and desktops. But all design decisions were made by one person. A real program would bring in reviewers from different backgrounds and test with users with disabilities.

GOVERN 4: The organization considers and communicates AI risk — MET

Risk is communicated openly. The Behind the Scenes panel shows every guardrail as it fires on each message, the About page explains each AI's limits, the code is public, and the security scorecard publishes independent scan results, including the ones that failed at first and how they were fixed.

Evidence: `site/about.html`, `docs/scans/`, `site/assets/chat.js`.

GOVERN 5: Feedback from people outside the team is collected and used — MET

Anyone can report a vulnerability privately through GitHub, as published in the security policy and in a standard security.txt file. Anyone can also test the defenses live in Cipher's Lair.

Evidence: `SECURITY.md`, `site/.well-known/security.txt`.

GOVERN 6: Third-party AI and software risks are addressed — MET

Third parties are known and bounded. Google Gemini (paid tier: chats are not used for training) receives no IP addresses and no full card data, and the spend caps protect against runaway use. Dependencies are pinned and watched by Dependabot, code by CodeQL and secret scanning. Cloudflare's human check is the only third-party script, and only on Cipher's Lair.

Evidence: `requirements.txt`, `app/llm.py`, `app/spend.py`, `deploy/apache-hardening.conf`.

Map: understand the context and the risks

MAP 1: The context is established and understood — MET

The purpose, users and setting are documented: a public demonstration of AI governance for a fictional payment company, used by anonymous visitors, holding only fictional data. The stakes are clear too. In a real payment company, the same chatbot would touch cardholder data, money and customer trust.

Evidence: `README.md`, `policies/ai-assistant-policy.md`.

MAP 2: The AI system is categorized — MET

Two AI systems, each categorized by what it can do. Tucker is a customer-facing conversational agent with read-only tools. The AI SOC Squirrel is an internal analyst that reads one attack run's events and proposes actions from a fixed menu. Neither makes decisions about people: no credit, eligibility or fraud decisions.

Evidence: `app/tools.py`, `app/cipher/soc.py`.

MAP 3: Capabilities, goals, benefits and costs are understood — MET

Each AI's capabilities are listed on the About page in "can" and "cannot" terms. Costs are measured per call and capped daily and monthly, and the fallback when the AI is unavailable or over budget is designed in: Tucker naps, and the SOC report falls back to plain facts from code.

Evidence: `site/about.html`, `app/spend.py`, `app/cipher/soc.py`, `tests/test_cipher_soc.py`.

MAP 4: Risks are mapped for every component, including third parties — MET

Each component's risks are mapped to a control: the model (prompt injection, disclosure, misinformation), the tools (excessive agency), the retrieval library (poisoning), the web layer (the OWASP web Top 10), the database (least privilege) and the suppliers (Gemini, Cloudflare, Python packages). The mapping is published as the OWASP write-up beside this one, and Cipher tests 50 of those risks live.

Evidence: `app/cipher/attacks.py`, `docs/standards/owasp-llm-top-10.md`.

MAP 5: Impacts on people are characterized — MET

The impacts that matter for a support chatbot are privacy (exposing someone's account or card), financial harm (a wrongful refund or charge) and deception (people not knowing they are talking to an AI). The design answers each: account data only after identity verification and only that customer's own, no money-moving tools at all, and Tucker identifies himself as an AI.

Evidence: `app/tools.py`, `app/session.py`, `app/chat.py`.

Measure: test and track the risks

MEASURE 1: Methods and metrics are chosen and used — MET

More than 70 automated tests check the guardrails with a scripted stand-in model, so they run without calling the real AI. In production, every guardrail event is counted and reported nightly: blocked injections, masked outputs, lockouts, naps and spend.

Evidence: `tests/`, `app/llm.py` (FakeLLM), `app/report.py`.

MEASURE 2: The system is evaluated for trustworthiness — PARTLY MET

Security, privacy, safety and transparency are tested in code and live: Cipher runs 50 attacks against the real site, and the results are checked by code rather than by the attacker's script. Independent public scanners graded the site's security A+ (96% on internet.nl). What a real program would add: a measured accuracy evaluation of Tucker's answers and a bias review, which a fictional-data demo can't meaningfully run.

Evidence: `app/cipher/detect.py`, `docs/scans/`.

MEASURE 3: Risks are tracked over time — MET

The owner's nightly report tracks visitors, guardrail events, Cipher runs, anything unusual and AI spend against the caps, and real-time alerts fire when a limit starts biting. Dated scan results are kept in the repository.

Evidence: `app/report.py`, `app/alerts.py`, `deploy/alert-policy.json`, `docs/scans/`.

MEASURE 4: Feedback on whether the measurements work is collected — PARTLY MET

What real use shows feeds back into the measurements. For example, a deploy restart that interrupted an attack run was caught, measured and fixed, with a test added. A real program would hold a regular review of whether its metrics still catch what matters.

Evidence: `deploy/update.sh`, `tests/test_cipher_soc.py`.

Manage: act on the risks

MANAGE 1: Risks are prioritized and responded to — MET

The highest risks (data disclosure and unauthorized actions) are removed by design rather than filtered: the AI simply has no power to act. Lower risks are reduced in layers. Findings are fixed quickly: on November 24, 2026 every gap the independent scanners found was fixed the same day.

Evidence: `app/tools.py`, `db/roles.sql`, `docs/scans/2026-11-24/results.json`.

MANAGE 2: Benefits are maximized and harms minimized, with fallbacks — MET

Every AI failure has a safe fallback. If Gemini fails, Tucker apologizes instead of erroring. If the budget is spent, he naps. If the SOC AI returns garbage or nothing at all, the report is built from plain facts by code, and the attacker is blocked by code regardless.

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

MANAGE 3: Third-party risks are managed — MET

The model provider is limited by contract tier (no training on the data), by data minimization (no IPs or full card numbers) and by spend caps. Software dependencies are updated through reviewed Dependabot pull requests, and the server installs operating system security updates automatically every day.

Evidence: `deploy/auto-maintenance.sh`, `requirements.txt`, `app/spend.py`.

MANAGE 4: Response, recovery and communication are planned and monitored — PARTLY MET

Detection and response for attacks are built in: code rules alert after 15 blocked attacks from one source and block the source for 24 hours after 50, and the AI SOC Squirrel writes an incident report. Demo data resets nightly to a known state. A real company would add a written AI incident response plan, customer notification steps and recovery drills.

Evidence: `app/cipher/detect.py`, `app/cipher/soc.py`, `db/nightly_reset.sql`.

The Generative AI Profile

NIST AI 600-1 lists twelve risks specific to generative AI. The ones that apply to a support chatbot are covered above: confabulation (MEASURE 2 and the OWASP misinformation entry), data privacy (MAP 5), information security (prompt injection and the OWASP list), human-AI configuration (Tucker always discloses he is an AI and cannot act on his own), and value chain and component integration (GOVERN 6, MANAGE 3). The others, such as chemical, biological, radiological and nuclear (CBRN) information, violent or obscene content, and intellectual property, are reduced by the

topic limit, which keeps Tucker to Tucker Payment Processing questions only.

Source: https://github.com/mcunningham-infosec/tucker-payment-processing/blob/main/docs/standards/nist-ai-rmf.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.