

Automating Cybersecurity Recommendations for Remediation of Vulnerabilities within NASA’s Deep Space Network: An Agentic Approach

Blake Pennel

University of Kansas

Faculty Mentor: Mark D. Johnston, Jet Propulsion Laboratory / California Institute of Technology

Abstract

The emergence of Large Language Models (LLMs) has changed the cybersecurity landscape, giving adversaries a force multiplier for discovering and exploiting software vulnerabilities at unprecedented speed. Defenders of critical infrastructure must adopt comparable capabilities to keep pace. This project developed an agentic AI framework that assists in remediating software vulnerabilities within NASA’s Deep Space Network (DSN), the ground-based antenna network that communicates with spacecraft beyond Earth orbit. Unlike generic security tools, which rank vulnerabilities by a universal severity score, the framework fuses incoming vulnerability data with deep, DSN-specific operational context to produce tailored, actionable remediation recommendations. The system ingests context from internal documentation, configuration databases, and existing vulnerability sources through the Model Context Protocol (MCP); reasons over that context using a schema-constrained agent built on the Pydantic AI framework; and emits structured, validated JSON outputs describing each vulnerability’s operational classification, mission impact, and a proposed fix. The framework classifies remediation by *operational feasibility* rather than raw severity, reflecting the DSN’s overriding priority of continuous availability, and can identify multi-step exploit chains in which individually minor vulnerabilities combine into a critical compromise. All processing remains within approved computing boundaries, and a human reviews every recommendation through a purpose-built dashboard. The result is a working, end-to-end decision-support tool that reduces the cognitive burden on cybersecurity analysts while respecting the DSN’s unique operational constraints, and that is designed to improve automatically as underlying models advance.

1 Introduction

1.1 Background

The Deep Space Network (DSN) is NASA’s international array of large radio antennas that provides the communication link between Earth and spacecraft operating beyond the orbit of the Moon.¹ Operated by the Jet Propulsion Laboratory (JPL), it comprises three complexes spaced roughly evenly around the globe, in California, Spain, and Australia, so that a spacecraft is always in view of at least one site as the Earth rotates. Because a missed communication pass can mean lost science data or, in the worst case, a lost spacecraft, the DSN operates continuously, and its single most important operational property is *availability*.

Maintaining the security of the DSN’s software systems is therefore a delicate balance. Every computing system accumulates software vulnerabilities over time, and each vulnerability represents a potential entry

point for an attacker. The conventional response, patching the software and restarting or rebooting the affected machine, is often straightforward for ordinary enterprise systems. For a system whose defining requirement is uninterrupted operation, however, the same action can be operationally unacceptable: rebooting a mission critical assembly during an active tracking pass is not an option that a generic security tool can reason about.

1.2 Problem and Rationale

The urgency of this problem has increased sharply in recent years. The number of publicly disclosed software vulnerabilities (Common Vulnerabilities and Exposures, or CVEs) has grown substantially over the past decade (Figure 1),^{2,3} and the recent maturation of LLMs has begun to accelerate the discovery and exploitation of vulnerabilities by adversaries. In effect, offensive cybersecurity is being automated. A defensive posture that depends entirely on manual human effort cannot scale to meet an automated threat.

The specific difficulty for the DSN is not merely the *volume* of vulnerability alerts but the *context* required to act on them wisely. A typical vulnerability scanner reports findings ranked by the Common Vulnerability Scoring System (CVSS), a universal severity metric.⁴ This metric is deliberately context-free: it knows nothing about what a given machine does within the DSN, whether it can be patched during a scheduled maintenance window, or what would break if it were taken offline. Consequently, an analyst today must manually correlate each alert against internal documentation, configuration records, ticketing systems, and years of institutional knowledge before deciding on a course of action. This correlation work is slow, cognitively demanding, and precisely the kind of task at which a well-equipped LLM agent excels.

1.3 Approach and Main Conclusions

This project proposes and implements an *agentic* approach: an AI system that is given a set of tools and the autonomy to decide which tools to use in order to gather the context it needs, reason about a vulnerability, and produce a structured recommendation. The system deliberately keeps a human in the loop: it recommends, it does not act, preserving human authority over any change to operational systems.

The principal conclusion of this work is that such a system is not only feasible but effective. The framework reliably produces high-quality, DSN-specific remediation assessments; it grounds those assessments in authoritative internal documentation and cites its sources; it enforces structured, machine-readable output through automated validation; and it does so within approved data boundaries and at manageable cost. Because the framework is model-agnostic, its quality is expected to rise automatically as the underlying LLMs improve, a rare instance of a defensive tool that becomes a force multiplier over time. The remainder of this paper describes the system’s findings and capabilities (Section 2), the methods and architecture used to build it (Section 3), the challenges encountered and their resolutions (Section 4), and directions for future work.

2 Results

This section summarizes the capabilities of the completed system, which functions as an end-to-end pipeline from a raw vulnerability alert to a reviewed, structured remediation recommendation (Figure 2).

2.1 An End-to-End Remediation Pipeline

The system accepts vulnerability data drawn from existing internal sources, assembles the relevant operational context for the affected systems, reasons over the combined information using an AI agent, and

emits a structured recommendation for human review. Each stage is described in Section 3. In operation, the pipeline transforms an unprioritized list of scanner findings into a set of prioritized, contextualized, and actionable recommendations, each accompanied by the reasoning and source citations that justify it.

2.2 Context-Aware, DSN-Specific Assessments

The central result is that the agent reframes remediation *difficulty* in terms of operational feasibility rather than raw severity. A high-CVSS vulnerability whose fix is a routine, easily rolled-back update is classified as *easy*, since it fits a normal maintenance window; a modest-severity vulnerability whose fix demands a reboot of core services, a firmware change, or a performance-degrading patch is classified as *hard*. This inversion of severity-first logic, grounded in the agent’s consultation of internal documentation, is the essence of the DSN-specific value the tool provides.

Every assessment is emitted as a validated data structure containing, among other fields, the operational difficulty classification, a written rationale, an analysis of mission and system impact, a list of deployment considerations, and, where possible, the exact command or manual procedure to apply the fix. Assessments are required to cite the specific internal documents that informed them, increasing each recommendation’s auditability.

2.3 Enrichment with Authoritative Threat Data

Beyond the model’s reasoning, the system enriches each assessment with objective, external facts. Notably, it cross-references each vulnerability against the Cybersecurity and Infrastructure Security Agency’s (CISA) Known Exploited Vulnerabilities (KEV) catalog, an authoritative list of vulnerabilities known to be actively exploited in the wild.⁵ When a vulnerability matches, the catalog contributes several concrete data points that a generic severity score does not: the date the vulnerability was added to the catalog, a CISA-mandated remediation due date, a flag indicating whether the vulnerability has been used in known ransomware campaigns, the associated weakness types (CWE identifiers), and a short authoritative description. Because these are matters of public record rather than model judgment, they are injected as hard facts rather than left to the model to infer, both flagging vulnerabilities that warrant heightened attention and giving the agent a firm, externally grounded basis for prioritization.

2.4 Exploit Kill-Chain Analysis

Attackers rarely rely on a single vulnerability; they *chain* several together, using a minor foothold to reach a position from which a more serious compromise becomes possible.⁶ A conventional flat list of findings obscures this reality. The framework includes an experimental capability that reasons about how multiple vulnerabilities across interdependent systems can be combined into a multi-step exploit chain. For example, a low-severity foothold might lead to privilege escalation and then to lateral movement across systems (Figure 4). The result is rendered as an interactive attack-path diagram, converting a flat list into an attacker’s-eye view of systemic risk and helping analysts understand which combinations of otherwise minor issues deserve priority.

2.5 Interactive Dashboard and Human Oversight

All of the system’s outputs are surfaced through a web-based dashboard that serves as the point of human oversight. Through it, an analyst can launch analysis runs, review remediation recommendations grouped by operational difficulty, inspect the generated kill-chain diagrams, interact with the agent directly through a streaming chat interface, and monitor detailed operational metrics including token usage and cost. This

human-in-the-loop design is a deliberate safety property: the system is decision-support, and a person reviews and authorizes every recommendation.

2.6 Efficiency, Resilience, and Transparency

To be practical for real use, the system incorporates several engineering measures. Caching of repeated context queries reduces operational cost on repeated queries by an estimated 50–70%. Checkpoint-based processing allows a large analysis run to be interrupted and resumed with minimal loss, making it possible to process large vulnerability datasets reliably. The system adapts its request pacing in response to backend rate limits rather than failing outright. And it tracks token consumption and cost on a per-vulnerability basis, including what an equivalent run would have cost on a paid cloud backend, so that operational cost is fully transparent rather than a black box.

3 Methods

This section describes the technical architecture of the system (Figure 3). It is organized by component and is directed toward readers familiar with software systems and machine learning.

3.1 Overall Architecture

The system is organized into four cooperating parts: a core analysis engine written in Python that contains the agentic logic; a backend service (built with the FastAPI framework) that orchestrates analysis runs and streams results; a web dashboard (built with React and TypeScript) that provides the human-in-the-loop interface; and a set of context connectors implemented as MCP servers. Data flows from a vulnerability source, into the agent (which retrieves additional context on demand), out as validated JSON, and finally into the dashboard for human review.

3.2 The Agent and Structured Output

The agent is implemented using the Pydantic AI framework,⁷ which constrains an LLM to return output conforming to a strictly typed schema. Rather than producing free-form prose, the agent must populate a defined data structure (the remediation assessment) whose fields, types, and value constraints are declared in advance. When the model returns output that violates the schema, the validation error is fed back to the model, which then attempts to correct itself; the agent retries up to a fixed number of times. This “self correcting” loop is central to the system’s reliability, because it converts the LLM’s inherently probabilistic output into a dependable, machine-consumable contract that downstream components and other tools can rely on. Analysis inference is run at a temperature of zero to make the structured output as deterministic as possible.

The agent’s behavior is governed by a system prompt that instructs it to classify remediation by operational feasibility, to consult internal documentation before classifying, and to cite the specific documents it relies upon. Every assessment is additionally stamped with the versions of the prompt and schema used, the identifier of the model, and the backend on which it ran, providing full reproducibility and audit traceability.

3.3 Context Retrieval via the Model Context Protocol

Rather than injecting all potentially relevant information into the prompt in advance, which would be both expensive and quickly exceed the model’s context window, the system gives the agent a set of tools and lets

it retrieve only the context it needs, when it needs it. These tools are implemented as servers that speak the Model Context Protocol (MCP),⁸ an open standard for connecting LLMs to external data sources and tools. Each MCP server runs as a local process and connects only to internal resources, so that no sensitive data leaves approved boundaries.

The connected context sources include: an internal engineering documentation corpus that serves as the authoritative reference for DSN systems and interfaces (the primary and most important source); an internal wiki providing procedures and known issues; a database of domain-specific acronyms used to resolve system names; a configuration-item database describing deployed assets; and a document-processing service. In addition, the agent draws on existing vulnerability data compiled from internal scanning and reporting systems. This on-demand retrieval strategy is what allows each individual analysis to remain within the model's context budget.

3.4 Model Backends and Portability

Because the field of LLMs is advancing rapidly and because different data sensitivities call for different computing environments, the system is deliberately backend-agnostic. It presents a single, unified interface over three approved backends: a government-cloud managed service running a current-generation commercial model within a high-assurance compliance boundary;⁹ an internally hosted model service approved for controlled but non-cloud use; and a third internal option accessed through a local translation proxy. Switching between backends requires changing only a single configuration value, with no changes to application code. This design has two benefits: it allows analysis to be routed to the environment appropriate to the data's sensitivity, and it ensures that the tool will benefit automatically from future improvements in the underlying models.

3.5 Kill-Chain Analysis

The kill-chain capability is implemented as a second, specialized agent that takes a set of individual vulnerabilities and reasons about how they might be combined. It, too, consults internal documentation to understand system dependencies and data flows. Its output is a structured representation of one or more exploit chains, each an ordered sequence of vulnerability "nodes" with an associated attack narrative and overall severity, which the dashboard then renders as a diagram. The minimum chain length and the analysis's tool budget are configurable.

3.6 Efficiency and Resilience Mechanisms

Several mechanisms make the system robust for extended, real-world runs. A caching layer stores the results of repeated context queries, avoiding redundant work and reducing cost. A checkpoint mechanism persists progress after each assessment, so that an interrupted run can resume without repeating completed work and loses at most one item on failure. An adaptive pacing mechanism widens the delay between requests when the system detects backend rate-limiting and relaxes it again once pressure clears, honoring server-provided cooldown hints rather than repeatedly failing against the same limit. Finally, a token-tracking module records input, output, and cached-token counts and computes both actual and cloud-equivalent cost for every assessment and for each run in aggregate.

3.7 Additional and Experimental Capabilities

Several further capabilities were prototyped. The system supports multimodal (vision) analysis, allowing the agent to interpret diagrams and screenshots embedded in documentation directly, rather than relying

only on pre-generated textual descriptions of those images. An experimental vulnerability-simulation capability stands up a disposable, tightly sandboxed container and attempts to reproduce an exploit in order to assess its real feasibility, subject to strict host-safety guardrails (isolation, resource limits, and, by default, no network access). These capabilities are experimental and were not hardened for production use during the internship period.

4 Discussion

4.1 Interpretation of Results

The system demonstrates that an agentic LLM, properly constrained and equipped with the right context, can perform a task that previously required substantial manual analyst effort: translating a context-free vulnerability alert into a context-rich, operationally grounded remediation recommendation. The most important qualitative result is that the agent's recommendations reflect DSN-specific operational reasoning rather than generic severity ranking. This is possible only because the agent has access, through MCP, to the same authoritative internal documentation a human analyst would consult, and because the structured-output discipline forces its conclusions into a consistent, auditable form.

Two design decisions proved especially important. First, treating structured output as a contract, enforced by schema validation and self-correcting retries, was what elevated the tool from a promising demonstration to a dependable pipeline component. Second, on-demand context retrieval via MCP, rather than up-front context injection, was the key to operating within the model's context limits while still drawing on a large body of internal knowledge.

4.2 Challenges and Resolutions

Four principal challenges arose during development, each of which motivated a specific architectural response:

- **Model context limits.** The finite context window of the underlying models made it impossible to supply all relevant data at once. This was resolved through on-demand retrieval and checkpointed batching, keeping each analysis within budget.
- **Heterogeneous and messy data sources.** Internal data existed in many incompatible formats. Standardizing every source behind the common MCP interface, supported where necessary by bulk export and Python-based extract-transform-load (ETL) scripting, made these sources uniformly accessible to the agent.
- **Cost and rate limits.** Repeated queries and paid cloud inference introduced both monetary cost and throughput limits. Caching, adaptive pacing, and the ability to route controlled-but-non-cloud work to an internally hosted model backend together kept cost and throughput manageable.
- **Model reliability.** LLM output is inherently variable and occasionally malformed. Strict schema validation combined with self-correcting retries made the pipeline dependable despite this variability.

4.3 Relation to the Broader Effort and Significance

This work directly supports the operational cybersecurity mission of the DSN operations group, whose activities center on the discovery, remediation, evaluation, and compliance aspects of DSN security under an overriding priority of availability. By reducing the manual effort required to triage and contextualize vulnerabilities, the tool frees skilled analysts to focus on judgment and authorization rather than correlation.

More broadly, the project is a concrete instance of meeting AI-assisted offense with AI-assisted defense, an increasingly necessary posture for the defenders of critical infrastructure.

4.4 Future Research Directions

Several avenues remain for future work. The generated verification tests, which currently confirm a patch on paper, could be executed automatically within the existing sandbox to close the loop between recommendation and validation. Two research questions identified during the internship merit further study: whether structured vulnerability-test methodology data can measurably improve the agent's understanding of a vulnerability's real-world implications and operational impact, and whether the agent can reason about network security configurations, such as firewall rule sets, to make prioritization *network-aware*, accounting for actual reachability rather than assuming worst-case connectivity.¹⁰ Because the system's outputs are already structured JSON, deeper downstream automation (such as automatically filing remediation tickets) is a natural extension, deliberately gated behind human review. Finally, a systematic evaluation of recommendation quality against analyst ground truth would strengthen confidence in the tool's outputs.

Figures

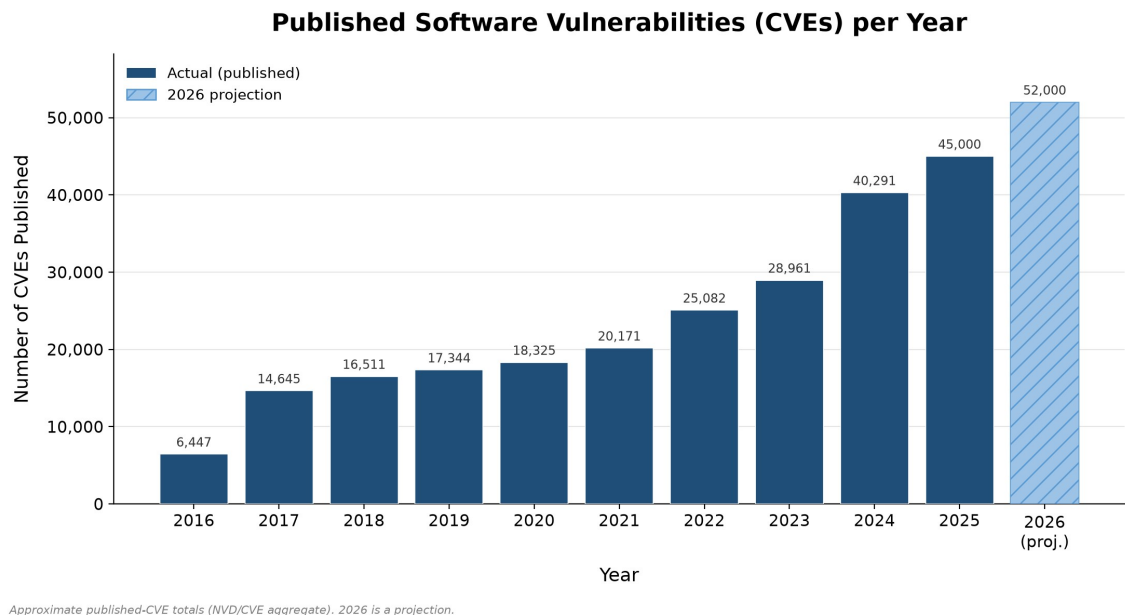


Figure 1: Growth of publicly disclosed software vulnerabilities (CVEs) per year over the past decade, with a projection for the current year. The steady increase, and the recent acceleration, motivate the need for scalable, AI-assisted defensive tooling.

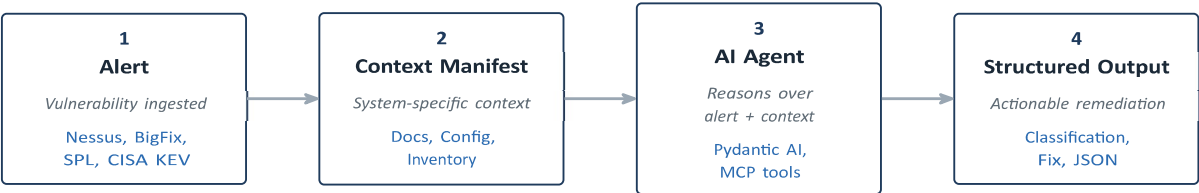


Figure 2: The four-step remediation pipeline. A vulnerability alert is ingested, relevant DSN-specific context is assembled into a manifest, the alert and context are reasoned over by the AI agent, and a structured JSON remediation output is produced. Every output is cited, typed, and auditable.

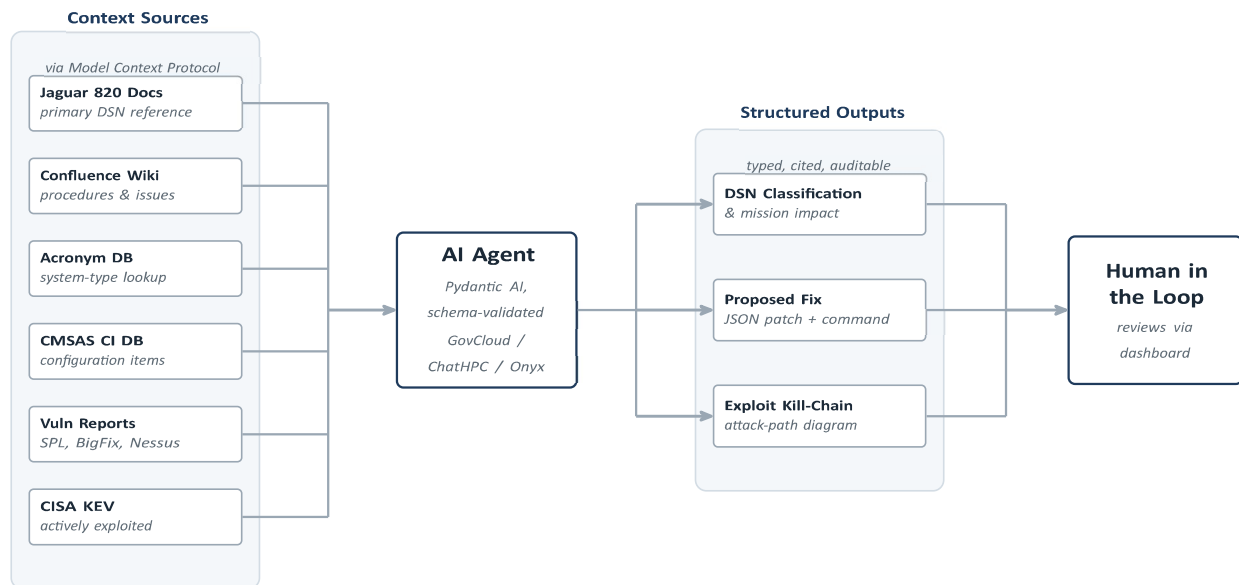


Figure 3: System architecture. Internal context sources are exposed to the AI agent through MCP; the agent, running on any of three interchangeable model backends, produces DSN-specific classifications, proposed fixes, and kill-chain analyses; all outputs converge on a human reviewer working through the dashboard.

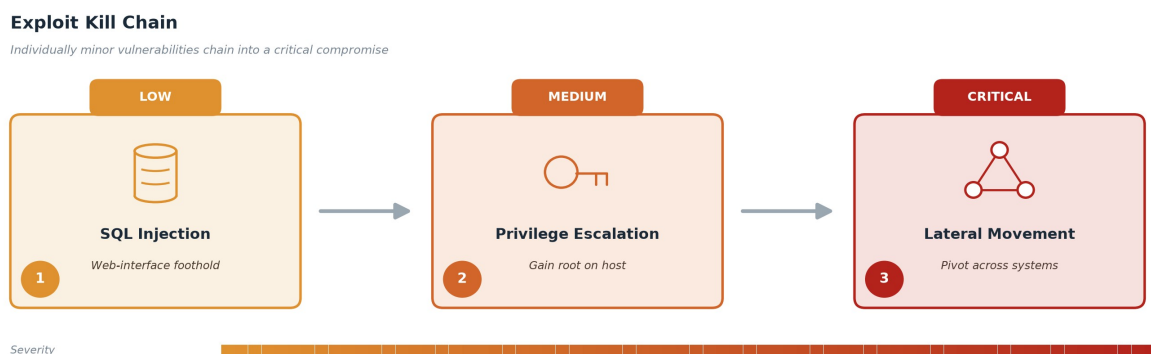


Figure 4: Example exploit kill chain. Individually minor vulnerabilities (a low-severity web foothold, a medium severity privilege escalation, and lateral movement across dependent systems) combine into a critical compromise. The agent identifies such chains and renders them as interactive diagrams.

References

1. NASA Jet Propulsion Laboratory. *Deep Space Network*. <https://www.jpl.nasa.gov/missions/dsn>
2. National Institute of Standards and Technology. *National Vulnerability Database (NVD)*. U.S. Department of Commerce. <https://nvd.nist.gov>
3. The MITRE Corporation. *Common Vulnerabilities and Exposures (CVE) Program*. <https://www.cve.org>

4. Forum of Incident Response and Security Teams (FIRST). *Common Vulnerability Scoring System (CVSS)*. <https://www.first.org/cvss/>
5. Cybersecurity and Infrastructure Security Agency. *Known Exploited Vulnerabilities Catalog*. <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>
6. Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (2011). *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains*. Lockheed Martin Corporation.
7. Pydantic. *Pydantic AI Documentation*. <https://ai.pydantic.dev>
8. Anthropic. *Model Context Protocol (MCP)*. <https://modelcontextprotocol.io>
9. Anthropic. *Claude Opus 4.8*. Large language model accessed via Amazon Bedrock on AWS GovCloud (FedRAMP High boundary). <https://www.anthropic.com/claude>
10. The MITRE Corporation. *MITRE ATT&CK Framework*. <https://attack.mitre.org>
11. Anthropic. *Claude Code*. AI coding assistant (accessed via AWS GovCloud) used for source-code development and manuscript drafting and editing. <https://www.anthropic.com/claude-code>

Acknowledgments

I thank my faculty mentor, Mark D. Johnston, and my deputy mentors, Jeff Singer and Wesley Walker, all of the Jet Propulsion Laboratory / California Institute of Technology, for their guidance and support throughout this project. I also thank the members of DSN Operations Group 4023 for sharing their operational expertise and internal knowledge of DSN systems. This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA, and was supported by the NASA/JPL Summer Internship Program (JPLSIP).

Generative AI disclosure. Generative AI tools were used during this work in two capacities: (1) as the core technology under study, forming the reasoning engine of the framework described herein (see refs. 7–9); and (2) as an assistive tool during development, specifically Anthropic’s Claude (Claude Opus 4.8) accessed through the Claude Code command-line interface and served via AWS GovCloud, which supported source-code development and the drafting and editing of this report.¹¹

Appendix A: Remediation Assessment Output Schema (Principal Fields)

The agent’s primary output conforms to a strictly typed schema. The principal fields are summarized below; the schema is enforced at runtime through automated validation.

Field	Description
vuln_id	Vulnerability identifier
cve_list	Associated CVE identifiers (may be empty)
severity	Standard severity level (Critical/High/Medium/Low/Unknown)
difficulty	Operational-feasibility classification (easy / hard)
affected_systems_count	Number of affected systems
reason	Written rationale for the difficulty classification
impact_analysis	Analysis of affected systems and mission impact
considerations	List of deployment/remediation considerations

remediation_action_command	Exact command to apply the fix, or null if manual
verification	Generated tests to confirm the patch and continued service health
known_exploited, kev_*	CISA KEV enrichment fields
prompt_version, schema_version, model_id, backend	Reproducibility / audit stamps
token & cost fields	Per-assessment token usage and actual / cloud-equivalent cost

Appendix B: Configuration Parameters

The system's behavior is governed by a set of runtime configuration parameters, the most significant of which are summarized below. Experimental capabilities are disabled by default, so that the system runs in its core, production-oriented configuration unless a capability is explicitly enabled.

Parameter	Governs
Model backend	Which of the approved model backends is active for a run
Sample size	Number of vulnerabilities analyzed in a run
Retry limits	Model-call and schema-validation retry attempts
Timeout limits	Per-call and total-run wall-clock timeouts
Tool-call budget	Maximum context-retrieval (MCP) calls per vulnerability
Request pacing	Base delay between requests, adapted under rate limiting
Cache settings	Cache enablement and entry lifetime
Checkpoint interval	How frequently run progress is persisted for resumption
Experimental toggles	Enable/disable kill-chain analysis, vision, and vulnerability simulation (all off by default)