preprint

2025-12-8

A.I.G Technical Report

Tencent Zhuque Lab Team

https://github.com/Tencent/AI-Infra-Guard

Abstract

In this work, we present the design and implementation of A.I.G (AI-Infra-Guard), a comprehensive AI security risk self-examination platform. We will detail the platform's architecture, features, and how it integrates with the MCP Server and Jailbreak Evaluation.

A.I.G integrates capabilities such as AI infra vulnerability scan, MCP Server risk scan, and Jailbreak Evaluation, aiming to provide users with the most comprehensive, intelligent, and user-friendly solution for AI security risk self-examination. The platform couples static fingerprinting with agent-driven MCP inspections, exposes a modern web dashboard, and ships a fully documented API surface, so security teams can orchestrate scans through UI, CLI, or automation with equal ease. Its AI infra module already recognizes over 30 AI framework components and nearly 400 known CVEs, while the MCP Server scanner—powered by autonomous agents—covers nine major risk categories ranging from authentication bypass to insecure tool exposure. Jailbreak Evaluation leverages curated datasets and jailbreak toolchains to benchmark large-model guardrails across multiple providers.

Under the hood, AI-Infra-Guard relies on an extensible plugin framework. Fingerprint and vulnerability rules reside in declarative YAML packages, MCP scan heuristics, and prompt-security behaviors. Go-based services coordinate task execution, surface results through the websocket gateway, and emit Swagger-documented APIs for downstream systems. This layered design keeps rule authoring approachable while allowing enterprises to integrate Docker-based deployments, customize MCP workflows, and continuously expand jailbreak coverage without modifying the core engine.

1 Introduction

Large language model platforms have rapidly evolved from isolated model prototypes to integrated, full-stack production ecosystems. The rise of Model Context Protocol servers [1] and associated toolchains [2] exemplifies this evolution: enterprises increasingly integrate MCP-compliant agents, tool registries, and governance layers into their AI infrastructure [3] to enable LLM workflows that reliably call vetted tools, exchange structured context, and emit auditable traces. LLM systems now span heterogeneous runtimes—including cloud-native inference clusters, on-prem GPU fabrics, and edge accelerators—embracing declarative manifests, plugin catalogs, and CI-integrated deployment recipes that must keep pace with weekly model and prompt refresh cycles.

However, the very connective tissue and complexity that empower these MCP ecosystems simultaneously and significantly enlarge the attack surface. Specifically, tool exposure and capability routing introduce cross-tenant pivot opportunities; mis-scoped credentials enable autonomous agents to escalate from harmless retrieval to arbitrary code execution; insecure prompt bridges can leak sensitive context windows; and inadequate supply-chain vetting allows malicious tool packages or YAML fingerprints to poison the environment. Operational realities—including rapid model iteration, sprawling policy matrices, and opaque third-party dependencies—further compound these risks, making it difficult for platform owners to maintain continuous assurance over core security pillars: authentication, authorization, data handling, and jailbreak resistance. This systemic difficulty highlights a critical deficiency in current approaches: security validation efforts remain fragmented, often focusing on isolated components or one-off red teaming exercises, insufficient for the continuous, holistic assurance required in an MCP-compliant ecosystem.

Therefore, the operational necessity for continuous, holistic, and automated security validation has catalyzed the emergence of integrated AI Red Teaming platforms. To address this gap, we introduce AI-Infra-Guard, an open-source platform engineered to deliver comprehensive security visibility across the entire AI stack. AI-Infra-Guard specifically addresses the critical need for coordinated orchestration and continuous assurance, a requirement that remains insufficiently met by isolated academic research and fragmented commercial tools.

AI-Infra-Guard incorporates three principal capabilities, establishing it as a comprehensive AI Security Posture Management solution that uniquely integrates infrastructure, protocol, and model-level validation to unify visibility, risk assessment, and active defense management at scale:

- **AI Infra Scan**: Discovers known CVE vulnerabilities spanning heterogeneous AI framework components (such as Ollama and ComfyUI) used throughout training and deployment pipelines, thereby mitigating supply chain and infrastructure risks.
- **MCP Server Scan**: Employs AI agents to intelligently identify protocol-specific threats affecting MCP Servers—including tool poisoning, code vulnerabilities, and data exfiltration risks—empowering MCP developers to secure plugins before public release.
- **Jailbreak Evaluation**: Rapidly benchmarks prompt security by leveraging curated and diverse datasets to assess and compare large language model robustness against multi-hop adversarial prompts.

We present AI-Infra-Guard, the first open-source, integrated platform for continuous AI Security Posture Management. Our work contributes three novel, coordinated validation capabilities: (1) heterogeneous AI Infra Scan; (2) MCP Server Scan for protocol-level security; and (3) a dynamic Jailbreak Evaluation framework. By operationalizing continuous assurance, AI-Infra-Guard provides a systematic and extensible defense mechanism crucial for securing modern, full-stack LLM ecosystems.

2 Related Work

The domain of AI security has witnessed a proliferation of academic and industrial efforts, each focusing on specific facets such as model vulnerability detection, protocol risk analysis, or adversarial robustness evaluation. However, these advances have remained isolated, lacking a unified operational framework that comprehensively spans the diverse layers of modern AI infrastructure. A.I.G distinguishes itself by integrating select approaches and findings from these existing bodies of work, consolidating them within a cohesive, all-in-one testing and validation architecture. This design not only amplifies coverage across AI infrastructure, MCP protocol, and model-level defenses but also fosters interoperability and extensibility, previously unrealized in fragmented toolsets.

In this section, we provide a detailed survey of the specific methodologies, detection paradigms, and benchmark suites that A.I.G adopts or extends from prior art. We analyze how these elements are harmonized—through unified interfaces and orchestrated workflows—into the A.I.G platform, thereby enabling seamless, end-to-end security assessment in real-world scenarios.

2.1 AI Infra Scan

Prior taxonomies and broad surveys have systematized threats against machine-learning systems and characterized the space of ML-specific vulnerabilities, providing a conceptual foundation for subsequent empirical and tooling work. Kawamoto et al. [4] organize ML threats, assets, and mitigations—highlighting how traditional software security concerns (e.g., dependency vulnerabilities, insecure runtime environments) intersect with ML-specific attack surfaces such as data-poisoning, model-integrity, and prompt/interaction vectors. These taxonomies clarify why holistic, cross-layer defenses are necessary for modern ML deployments.

A growing body of recent empirical and conceptual work frames ML models and model hubs as a supply-chain problem. Several contemporary studies document concrete supply-chain risks for model distribution (e.g., replacement or tampering of published models), and propose provenance and artifact-signing mechanisms to improve integrity and transparency. Complementary empirical analyses of LLM repositories and their third-party dependencies demonstrate that vulnerable package dependencies are pervasive in ML projects and can materially increase exposure for deployed models and agents. These findings motivate targeted scanning of the heterogeneous libraries and tooling that populate training and inference pipelines. [5]

2.2 MCP Server Scan

With the rapid adoption of the Model Context Protocol (MCP) as a unified interface for tool-augmented LLM workflows, recent studies have systematically examined the security aspects of MCP-based systems. Guo et al. [6] present a comprehensive analysis of MCP security, identifying protocol-level considerations in tool invocation, context transmission, authentication, and capability routing. Their taxonomy clarifies the risks associated with enabling LLMs to dynamically discover and execute external tools, highlighting the importance of protocol-aware security validation.

The operational complexity of MCP-driven agent systems has led to research on "AgentOps" infrastructures. Wang et al. [7] survey agent operation pipelines—including monitoring, debugging, workflow orchestration, and safety enforcement—and outline approaches for integrating infrastructure scanning, tool governance, and security assessment across the MCP stack.

To support systematic testing, Yang et al. [8] introduce MCPSecBench, a benchmark and experimental platform for evaluating MCP implementations under adversarial scenarios, providing threat models, structured scenarios, and attack vectors. AI-Infra-Guard complements this approach by embedding protocol-level tests within operational security workflows.

LLM-driven red-teaming of MCP environments has also been explored. He et al. [9] propose a pipeline leveraging MCP tools to identify vulnerabilities and observe behaviors in interactive testing scenarios.

Recent studies analyze threats from malicious or compromised MCP servers. Zhao et al. [10] define attack categories such as capability escalation, cross-tool pivoting, sensitive-context exposure, and unauthorized execution, illustrating the role of MCP servers in the security landscape. Wang et al. [11] introduce MCPGuard, an automated framework for detecting misconfigurations and injection vectors in server implementations, contributing to MCP security management.

2.3 Jailbreak Evaluation

Recent open-source frameworks such as Garak [12] and DeepTeam [13] provide systematic platforms for LLM security evaluation through adversarial testing, prompt injection, and jailbreak detection. Garak offers a model-agnostic vulnerability scanner that generates adversarial probes and evaluates model outputs for failures including jailbreaks, hallucinations, data leakage, and harmful content. DeepTeam supports multi-turn and context-sensitive red-teaming, enabling structured attacks on complete LLM systems, including chatbots and agentic pipelines. Both frameworks provide automated, adversary-style evaluation for uncovering model-level vulnerabilities. The initial implementation of the Jailbreak Evaluation module in AI-Infra-Guard was inspired by DeepTeam, serving as a foundational reference for the design and development of adversarial assessment workflows.

Building on this baseline, in recent years, a range of innovative jailbreak operators have also emerged, further enriching the repertoire of adversarial techniques for systematic guardrail probing. A.I.G progressively integrated and extended a series of jailbreak operators to address the nuanced requirements of large-scale, multiprovider model benchmarking. These enhancements enable the framework to systematically probe model guardrails across diverse LLM providers, adversarial tactic classes, and deployment settings.

Recent research has explored diverse methodologies for evaluating and probing large language models. Strata-Sword [14] proposes a hierarchical benchmark that organizes jailbreak-style prompts according to their reasoning complexity, covering 15 categories and incorporating both English and Chinese linguistic patterns such as acrostics and riddles. By analyzing a broad set of LLMs under this taxonomy, the project provides a structured perspective on how models behave when confronted with prompts requiring different levels of reasoning and cultural–linguistic understanding.

In parallel, StegoRedTeam [15] introduces a red-teaming framework based on linguistic steganography, in which hidden instructions are embedded within benign text and later decoded through guided role-based prompting. This methodology offers a complementary viewpoint on LLM behavior by examining how models process covertly encoded information and structured decoding tasks. Together, these efforts contribute to a growing body of work that investigates LLM behavior under varied prompting conditions, spanning complexity-driven, multilingual, and steganography-based evaluation settings.

3 The A.I.G Framework

In this section, we will detail the architecture and implementation of the A.I.G framework.

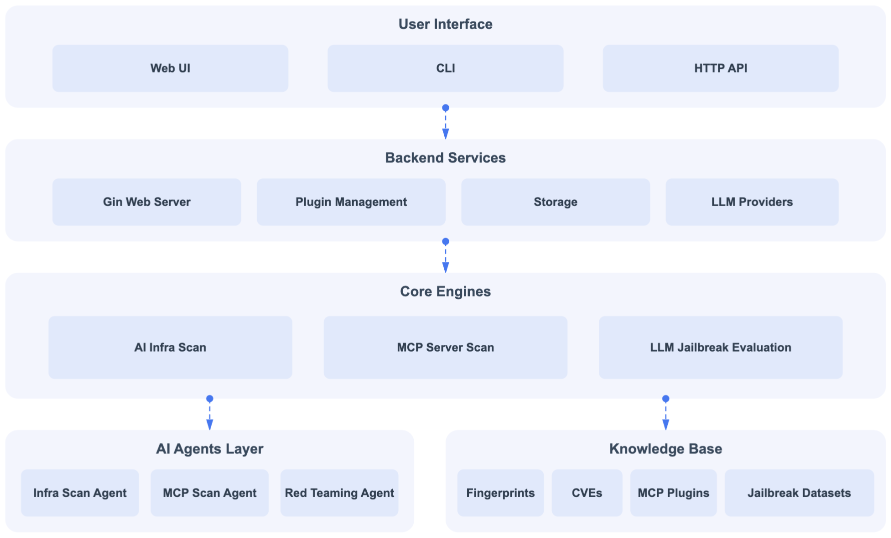

Figure 1: A.I.G Overall Architecture

As shown in Figure 1, A.I.G follows a vertically layered architecture that couples human-facing interfaces with a service substrate, specialized engines, and a shared intelligence plane. The user interface layer offers a web console, a command-line interface, and a programmatic API, allowing practitioners to submit scans, stream telemetry, and orchestrate remediation workflows through consistent semantics. These requests converge in the backend service layer where task scheduling, plugin governance, credentialed storage, and model brokering are jointly managed. The service layer maintains long-running agents, enforces identity and rate limits, and normalizes access to external LLM providers so that heterogeneous scans can share the same operational guarantees.

On top of this substrate reside the core engines that operationalize the three pillars of A.I.G. The AI Infra Scan engine performs reconnaissance-style probing of exposed AI services using fingerprint-driven enumeration and vulnerability templates to correlate infrastructure misconfigurations with known CVE knowledge. The MCP Server Scan engine interrogates Model Context Protocol deployments, dynamically loading plugins, replaying multi-stage agent behaviors, and vetting server-side execution paths for capability abuse. The Jailbreak Evaluation engine conducts prompt-based adversarial testing across multiple LLM providers, synthesizing attack operators, multi-turn probes, and red-team metrics to characterize guardrail robustness. These engines expose their capabilities through an AI agents layer that coordinates task lifecycles, accumulates evidence, and fuses cross-engine findings before feeding them back to the user.

Finally, the knowledge base consolidates reusable intelligence—AI fingerprints, vulnerability corpora, MCP manifests, and jailbreak datasets—and exposes versioned APIs that both drive scans and store their outputs. The bidirectional flow between engines and knowledge assets enables continual refinement: newly surfaced evidence enriches the corpus, while curated priors guide future scans. This architecture ensures that extending A.I.G with new scanners or datasets requires only localized changes within a layer, preserving global consistency in governance, telemetry, and LLM orchestration.

3.1 AI Infra Scan

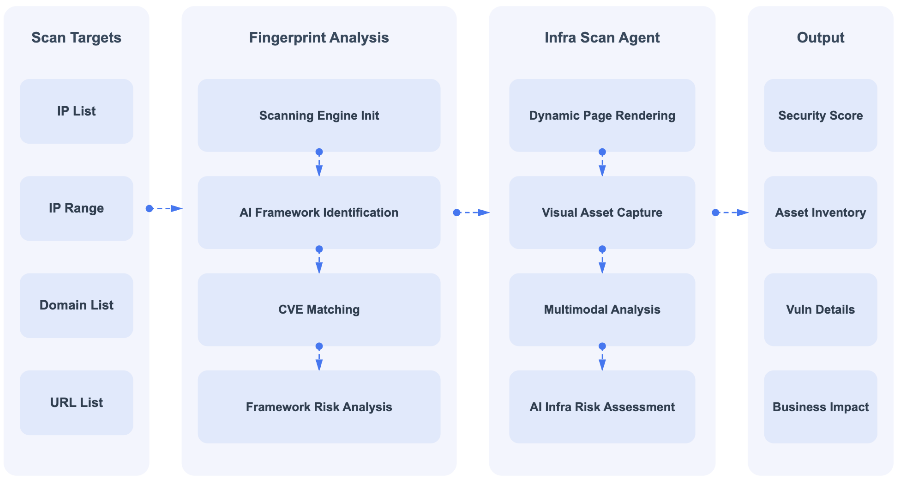

Figure 2: A.I.G AI Infra Scan Architecture

As depicted in Figure 2, the AI Infra Scan pipeline transforms heterogeneous target inventories—IP ranges, domains, and URLs—into structured security insights through a sequence of fingerprinting, multimodal inspection, and risk synthesis stages. The fingerprint analysis module initializes the scanning engine with concurrency-aware schedulers, performs AI framework identification by correlating HTTP, TLS, and asset signatures with our fingerprint corpus, and aligns observations with CVE knowledge to reason about framework-specific weaknesses. This stage narrows the hypothesis space before any intrusive probing takes place, thereby preserving operational efficiency while maintaining safety constraints for production systems.

The infra scan agent layer subsequently replays the surviving hypotheses through dynamic page rendering, visual asset capture, and multimodal content understanding. By coupling browser automation with OCR and vision-language reasoning, the agent can detect security disclosures that manifest only within rendered UI flows—such as leaked API credentials, exposed debug consoles, or misaligned access controls that appear after authentication bypass. The agent fuses these perceptual signals with network-layer telemetry to produce AI-infra risk assessments that quantify both technical exploitability and potential business impact. The final output consolidates normalized security scores, detailed vulnerability narratives, and asset inventories, enabling operators to prioritize remediation while feeding newly confirmed evidence back into the knowledge base for continual refinement.

3.2 MCP Server Scan

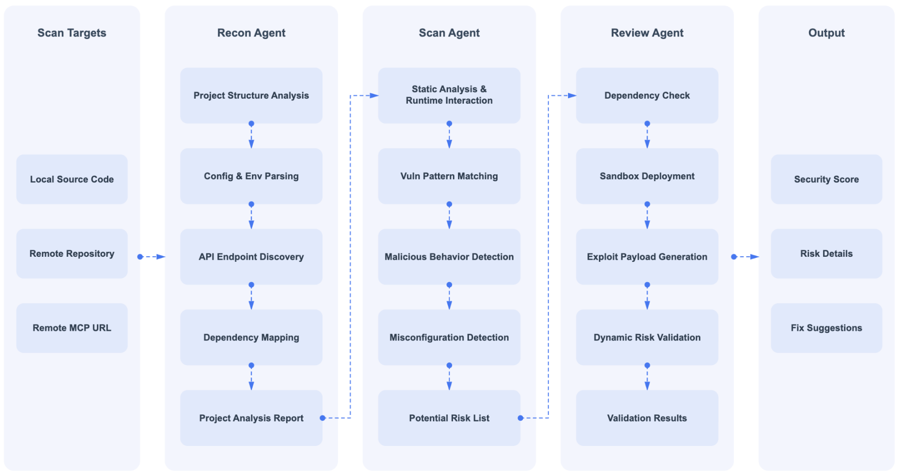

Figure 3: A.I.G MCP Server Scan Architecture

Figure 3 illustrates how the MCP Server Scan pipeline reasons over source artifacts and remote endpoints to expose protocol-level weaknesses before they are weaponized. The process begins by enumerating scan targets that may include local MCP toolchains, mirrored repositories, or remote MCP server URLs. A recon agent performs project-structure dissection, configuration parsing, and API endpoint discovery; this produces a dependency map and a machine-readable project dossier that captures trust boundaries, invocation flows, and capability descriptions. Recording the dependency graph at this stage is critical for later attribution, since compromised plugins often inherit vulnerable packages or mis-specified scopes from upstream modules.

The scan agent ingests the recon dossier and orchestrates both static analysis and constrained runtime interaction. It pattern-matches risky MCP primitives—such as unrestricted tool invocation, unsafe environment loading, or missing authentication hooks—while concurrently executing sandboxed calls to observe state transitions. This dual perspective enables the agent to flag malicious behaviors (e.g., arbitrary command execution) and configuration errors (e.g., permissive capability routing) in a single pass, ultimately emitting a prioritized risk list. The review agent then deepens due diligence through dependency audits, sandbox redeployment, and exploit-payload simulation. By validating suspected issues in isolated environments, it eliminates false positives and quantifies exploitability with concrete traces. The resulting deliverables combine security scores, richly documented risk narratives, and targeted remediation guidance, which in turn update the knowledge base so that future scans inherit the newly verified heuristics.

3.3 Jailbreak Evaluation

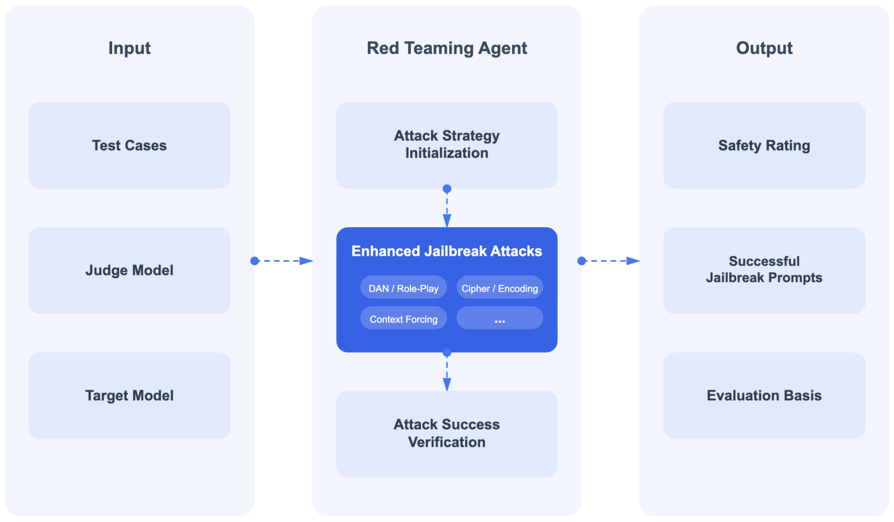

Figure 4: A.I.G Jailbreak Evaluation Architecture

The jailbreak evaluation pipeline (Figure 4) formalizes automated red-teaming as a closed-loop experiment that continually adapts attack strategies while enforcing measurable safety criteria. Inputs combine curated test cases, the target model under examination, and a judge model that encodes policy-specific rubrics. A red-teaming agent initializes the attack space by selecting tactic families—role-play, context forcing, cipher/encoding, or hybrid prompt chains—guided by priors accumulated in the knowledge base. During execution, the agent composes diversified prompts, reasons over model responses, and mutates the attack plan whenever the judge signals partial compliance, thereby emulating adaptive adversaries rather than static benchmarks.

Attack success verification is intentionally decoupled from prompt generation to prevent reward hacking. Each candidate jailbreak is replayed under controlled conditions, scored by the judge model, and escalated to ensemble verification if the content suggests high-risk leakage or capability abuse. The pipeline produces three classes of outputs. Safety ratings quantify residual risk for each model-policy pair, successful jailbreak prompts provide reproducible evidence for mitigation teams, and evaluation bases log the decision traces that justify every rating. This structure supports longitudinal benchmarking across providers, highlights tactic transferability, and feeds newly validated attack operators back into the shared dataset so that subsequent campaigns start from a progressively stronger baseline.

4 Limitations

Owing to the agent-based architecture underpinning our detection system—as well as the inherent stochasticity of large language models—A.I.G cannot currently guarantee full reproducibility of vulnerability detection results. Furthermore, while our framework integrates multiple models and rule-based methods to assess vulnerabilities across both models and supporting infrastructure, the continual evolution of underlying model architectures, training datasets, and advanced reinforcement learning techniques leads to shifts in model outputs and even downstream infrastructural changes. As a result, AI security remains intrinsically dynamic, and A.I.G must continuously adapt its methodologies and capabilities to keep pace with these advances.

5 Conclusion

AI-Infra-Guard provides a open-source, comprehensive, intelligent, and user-friendly solution for AI security risk self-examination. It integrates capabilities such as AI infra vulnerability scan, MCP Server risk scan, and Jailbreak Evaluation, aiming to provide users with the most comprehensive, intelligent, and user-friendly solution for AI security risk self-examination. Although A.I.G cannot guarantee hundred percent vulnerability detection results, it can continuously adapt its methodologies and capabilities to keep pace with the advances of AI security.

References

[1] Model Context Protocol Architecture. https://modelcontextprotocol.io/docs/learn/architecture

[2] Awesome MCP Servers. https://mcpservers.org/

[3] LLM Security for Enterprises: Risks and Best Practices. https://www.microsoft.com/en-us/research/publication/llm-security-for-enterprises-risks-and-best-practices/

[4] Kawamoto, Yusuke et al. Threats, Vulnerabilities, and Controls of Machine Learning Based Systems: A Survey and Taxonomy https://arxiv.org/abs/2301.07474

[5] Liu, Shuhan et al. An Empirical Study of Vulnerable Package Dependencies in LLM Repositories https://arxiv.org/abs/2508.21417

[6] Yongjian Guo,Puzhuo Liu et al. Systematic Analysis of MCP Security. https://arxiv.org/abs/2508.12538

[7] Zexin Wang, Jingjing Li et al. A Survey on AgentOps: Categorization, Challenges, and Future Directions. https://arxiv.org/abs/2508.02121

[8] Yixuan Yang, Daoyuan Wu et al. MCPSecBench: A Systematic Security Benchmark and Playground for Testing Model Context Protocols. https://arxiv.org/abs/2508.13220

[9] Ping He, Changjiang Li et al. Automatic Red Teaming LLM-based Agents with Model Context Protocol Tools. https://arxiv.org/abs/2509.21011

[10] Weibo Zhao, Jiahao Liu et al. When MCP Servers Attack: Taxonomy, Feasibility, and Mitigation. http://arxiv.org/abs/2509.24272v1

[11] Bin Wang, Zexin Liu et al. MCPGuard: Automatically Detecting Vulnerabilities in MCP Servers. http://arxiv.org/abs/2510.23673v1

[12] DeepTeam https://github.com/confident-ai/deepteam

[13] Derczynski, Leon et al. garak: A Framework for Security Probing Large Language Models. https://arxiv.org/abs/2406.11036v1

[14] Strata-Sword: A Comprehensive Framework for Evaluating the Security of Large Language Models. https://arxiv.org/abs/2509.01444

[15] StegoRedTeam https://github.com/lhppppp/StegoRedTeam