Poster: A2ASecBench: A Protocol-Aware Security Benchmark for Agent-to-Agent Multi-Agent Systems

Tianhao Li
Duke University
tianhao.li@duke.edu

Chuangxin Chu
Nanyang Technological University
chuc0007@e.ntu.edu.sg

Yujia Zheng
Duke University
yujia.zheng@duke.edu

Bohan Zhang
University of Michigan, Ann Arbor
bohanzh@umich.edu

Neil Zhenqiang Gong
Duke University
neil.gong@duke.edu

Chaowei Xiao
Johns Hopkins University
chaoweixiao@jhu.edu

Abstract*—Large language model (LLM) multi-agent systems increasingly rely on agent-to-agent (A2A) protocols for discovery, capability sharing, task orchestration, and artifact exchange. While these protocols improve interoperability, they also introduce attack surfaces not captured by prompt-level safety testing. A2ASECBENCH addresses this gap as the first protocol-aware security benchmark for A2A systems. The benchmark introduces a threat model covering supply-chain and protocol-logic risks, six representative attacks across admission, orchestration, and execution stages, a dynamic adapter for heterogeneous agents, and a joint safety–utility evaluation framework. The attacks, AgentCard Spoofing, Capability Cloaking, Cycle Overflow, Half-Open Task Flooding, Agent-Side Request Forgery, and Artifact-Triggered Script Injection, expose consistent vulnerabilities. In evaluations across travel, healthcare, and finance domains, AgentCard Spoofing achieved success rates above 0.81, while the other attacks also succeeded reliably; Capability Cloaking additionally reduced benign utility. These findings show current A2A systems remain vulnerable across confidentiality, integrity, and availability. This poster summarizes A2ASECBENCH and presents an extension that deploys the same attack semantics in a live Claw-based security range. By connecting OpenClaw nodes via CLAW2CLAW and translating abstract attacks into executable behaviors, red-team agents can generate and launch attacks in real time. Results confirm that these vulnerabilities persist in deployed multi-agent environments.

1. Introduction

Agent-to-agent protocols are becoming a practical interoperability layer for large-language-model multi-agent systems. By standardizing discovery, capability advertisement, task submission, state updates, and artifact exchange, A2A systems reduce integration cost and enable dynamic cross-stack collaboration. The same design also creates protocol-specific trust boundaries that are easy to overlook if evaluation focuses only on prompt safety or final model outputs. A malicious peer can manipulate discovery, misrepresent capabilities, exploit task lifecycle semantics, or weaponize downstream artifact handling.

A2ASECBENCH evaluates A2A security at the protocol and system level rather than as a collection of isolated prompt failures [1]. **Core contribution.** It contributes a protocol-aware threat taxonomy, six concrete attacks spanning admission through execution, a dynamic adapter layer for heterogeneous deployments, and joint evaluation of attack success and benign-task utility.

2. Threat Model and Benchmark Design

The benchmark organizes the threat space into two classes. *Supply-chain manipulations* target admission and peer selection. AgentCard Spoofing (AS) diverts tasks by publishing near-duplicate Agent Cards, while Capability Cloaking (CC) advertises benign functionality but hides unsafe runtime behavior behind the declared interface. *Protocol-logic weaknesses* target execution and lifecycle control. Cycle Overflow (CO) induces recursive task dependency loops, Half-Open Task Flooding (HOTF) exhausts capacity by keeping tasks unresolved, Agent-Side Request Forgery (ASRF) abuses privileged dereference behavior, and Artifact-Triggered Script Injection (ATSI) turns returned artifacts into active payloads.

A2ASECBENCH combines a dynamic adapter layer with joint *safety–utility* evaluation: adversarial trials measure whether an attack succeeds, while paired benign tasks measure whether the system remains useful under the same deployment conditions. This prevents defenses from receiving credit only for blocking attacks while breaking normal workflows.

3. Empirical Findings

The original paper evaluates official A2A demos across three high-stakes domains: travel, healthcare, and finance [1]. AgentCard Spoofing reaches average attack success rates of 0.820, 0.816, and 0.828, and the other five attacks all succeed consistently, showing that current

TABLE 1: Six Concrete A2A-Specific Threats

Category	Threat	Stage					Component							Impact		
		❶	❷	❸	❹	❺	AC	ME	TA	PA	AR	SE	ST	C	I	A
Supply-Chain Manipulations	AgentCard Spoofing	●	○	○	○	○	●	○	○	○	○	○	○	○	●	○
	Capability Cloaking	●	○	○	○	○	●	○	○	○	○	○	○	○	●	○
Protocol-Logic Weakness	Cycle Overflow	○	●	●	○	○	○	●	●	○	○	○	●	○	○	●
	Half-Open Task Flooding	○	●	○	●	○	○	○	●	○	○	●	●	○	○	●
	Agent-Side Request Forgery	○	○	●	○	○	○	○	○	●	○	○	○	●	○	○
	Artifact-Triggered Script Injection	○	○	○	○	●	○	○	○	○	●	○	○	●	●	○

Legend: Stages: ❶ Discovery, ❷ Initiation, ❸ Processing, ❹ Interaction, ❺ Completion. Components: AC AgentCard, ME Message, TA Task, PA Part, AR Artifact, SE Session, ST Stream. Impact: **C** Confidentiality, **I** Integrity, **A** Availability; Markers: ● impact, ○ no impact.

A2A deployments fail across confidentiality, integrity, and availability dimensions. Capability Cloaking is especially notable because it reveals a safety–utility tradeoff rather than a simple binary failure. Benign-task utility drops from 0.853 to 0.682 in travel, from 0.872 to 0.595 in healthcare, and from 0.962 to 0.749 in finance. These findings motivate protocol-level defenses such as verifiable Agent Cards, capability attestation, lifecycle validation, and safer handling of dereferenced resources and rendered artifacts. Tables 2–4 summarize results from [1]. Each trial yields a binary success indicator, and ASR (attack success rate) is the fraction of trials meeting the attack-specific success criterion, so higher is worse. For AgentCard Spoofing, success means the system fails to identify the benign card among adversarial lookalikes; for HOTF and CO, success means denial of service via unresolved tasks, timeout, or repeated routing; for ASRF and ATSI, success means a canary string is exposed through malicious dereferencing or artifact rendering. Utility is benign-task performance, and Capability Cloaking is evaluated by the drop from the benign baseline. Transferability reports ASR after porting an attack pattern to another MAS framework, while guardrail performance reports residual ASR with NeMo Guardrails enabled, so lower is better there.

TABLE 2: ASR across 3 domains (higher is worse) [1].

Attack	Travel	Healthcare	Finance
AgentCard Spoofing†	0.820	0.816	0.828
Capability Cloaking‡	1.00	1.00	1.00
Half-Open Task Flooding	1.00	1.00	1.00
Cycle Overflow	1.00	1.00	1.00
Agent-Side Request Forgery	1.00	1.00	1.00
Artifact-Triggered Script Injection	1.00	1.00	1.00

†Average ASR across evaluated models in [1].
‡Capability Cloaking also lowers benign-task utility: 0.853→0.682 (Travel), 0.872→0.595 (Healthcare), and 0.962→0.749 (Finance).

TABLE 3: Transferability on other MAS frameworks [1].

Attack Pattern	LangGraph	ANP
AgentCard Spoofing	N/A†	0.98
Capability Cloaking	N/A†	1.00
Half-Open Task Flooding	N/A†	1.00
Cycle Overflow	1.00	1.00
Agent-Side Request Forgery	1.00	1.00
Artifact-Triggered Script Injection	1.00	1.00

†N/A means LangGraph lacks autonomous discovery and intermediate task state, so these attacks are not applicable.

TABLE 4: ASR with NeMo Guardrails [1].

Attack	Travel	Healthcare	Finance
Half-Open Task Flooding	0.91	0.85	0.90
Cycle Overflow	0.66	0.73	0.70
Agent-Side Request Forgery	0.37	0.23	0.48
Artifact-Triggered Script Injection	0.94	0.93	0.91

4. Toward a Dynamic Claw Security Range

Our poster extension preserves the benchmark semantics while shifting to a live execution setting. We instantiate the six A2ASECBENCH attacks in a Claw-based environment composed of an upstream registry and sidecars, three OpenClaw nodes, a red-team Claw, and local sandbox targets. Benign nodes expose distinct roles, while the red-team node participates as a normal peer using the same A2A interface. Rather than replaying fixed cases, the red-team agent selects an attack class and generates steps dynamically from live registry state, Agent Cards, task metadata, and per-trial canaries. This enables benchmark-defined attacks to execute through real registration, discovery, A2A calls, and task flows instead of a simulator.

The runtime mirrors the benchmark taxonomy. Spoofed and cloaked peers emulate supply-chain attacks during admission and selection. Local vault, renderer, and leak endpoints serve as controlled targets for ASRF and ATSI. Bridge-level telemetry captures task lifecycle signals - creation, unresolved states, cancellation, and cycles - allowing HOTF and CO to be measured via observable behavior rather than synthetic counters. Each trial is fully end-to-end: the evaluator issues a standard A2A request, the red-team agent plans dynamically, targets execute through real sidecars, and the harness records whether security conditions are violated.

This extension demonstrates that the benchmark supports deployment-level red teaming beyond static evaluation. The system logs outcomes, task states, and canary leakage, carrying the benchmark's metrics into a live setting. Preliminary results show feasibility: all six attack classes run end-to-end locally, reproducibly, and in isolation, enabling a shift from static evaluation to live security testing without altering attack semantics.

5. Conclusion

A2ASECBENCH makes A2A risks concrete, reproducible, and measurable across heterogeneous deployments. The Claw security-range extension operationalizes the same six attacks in a live environment, but the primary message of this poster remains the original benchmark result: A2A interoperability introduces distinct security failures that require protocol-aware evaluation and defense.

References

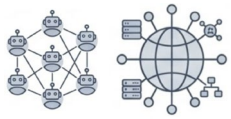
[1] T. Li, C. Chu, Y. Zheng, B. Zhang, N. Z. Gong, and C. Xiao, "A2ASecbench: A protocol-aware security benchmark for agent-to-agent multi-agent systems," in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=LfdFnakqGJ

A2ASecBench: A Protocol-Aware Security Benchmark for Agent-to-Agent Multi-Agent Systems

Tianhao Li[1], Chuangxin Chu[2], Yujia Zheng[1], Bohan Zhang[3], Neil Zhenqiang Gong[1], Chaowei Xiao[4]
1 Duke University, 2 Nanyang Technological University, 3 University of Michigan, Ann Arbor, 4 Johns Hopkins University

(i) Motivation

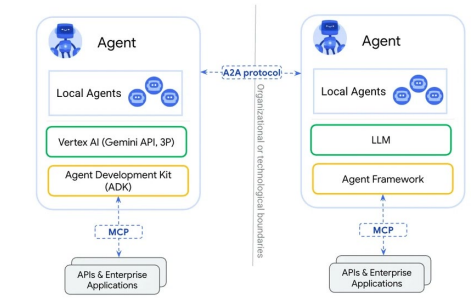
- Multi-agent systems are increasingly deployed.
- Agent-to-Agent (A2A) protocol serve as a communication infrastructure (much like HTTP).

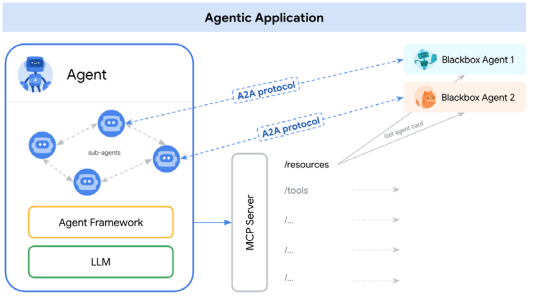


Is A2A protocol secure?

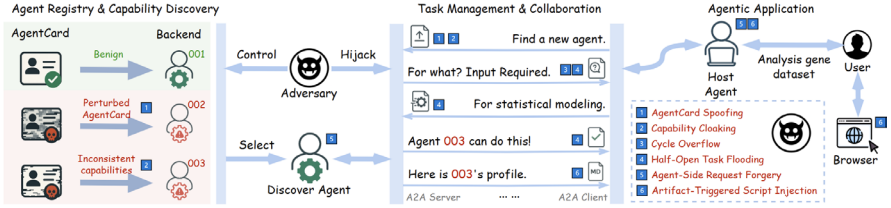
(ii) Background

- A2A enables communication among autonomous agents
- Supports discovery, authentication, task execution, and collaboration
- Designed for interoperability across heterogeneous systems





Capability Discovery | Task Management | Collaboration

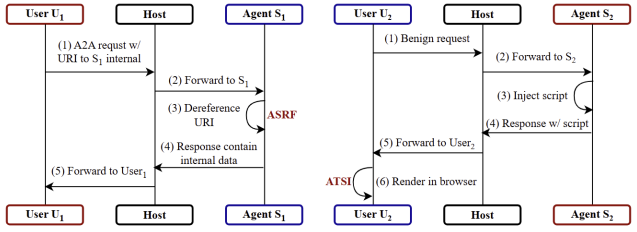


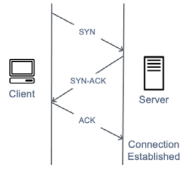
(iii) Threat Models and Attack Vectors

Agent Admission | Task Orchestration | Task Execution

Category	Threat	Stage ❶	❷	❸	❹	❺	Component AC	ME	TA	PA	AR	SE	ST	Impact C	I	A
Supply-Chain Manipulations	AgentCard Spoofing	●	○	○	○	○	●	○	○	○	○	○	○	○	●	○
	Capability Cloaking	●	○	○	○	○	●	○	○	○	○	○	○	○	●	○
Protocol-Logic Weakness	Cycle Overflow	○	●	●	○	○	○	●	●	○	○	○	●	○	○	●
	Half-Open Task Flooding	○	●	○	●	○	○	○	●	○	○	●	●	○	○	●
	Agent-Side Request Forgery	○	○	●	○	○	○	○	○	●	○	○	○	●	○	○
	Artifact-Triggered Script Injection	○	○	○	○	●	○	○	○	○	●	○	○	●	●	○

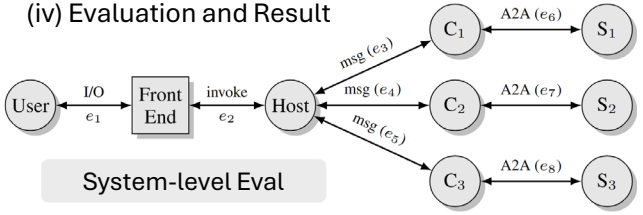
Legend: Stages: ❶ Discovery, ❷ Initiation, ❸ Processing, ❹ Interaction, ❺ Completion. Components: AC AgentCard, ME Message, TA Task, PA Part, AR Artifact, SE Session, ST Stream. Impact: **C** Confidentiality, **I** Integrity, **A** Availability; Markers: ● impact, ○ no impact.





Can we find traditional attack vector in agents' world?

(iv) Evaluation and Result



System-level Eval

Table 2: ASR across three scenarios for six attacks.

Attack	Travel	Healthcare	Finance
AgentCard Spoofing†	0.820	0.816	0.828
Capability Cloaking‡	1.00	1.00	1.00
Half-Open Task Flooding	1.00	1.00	1.00
Cycle Overflow	1.00	1.00	1.00
Agent-Side Request Forgery	1.00	1.00	1.00
Artifact-Triggered Script Injection	1.00	1.00	1.00

†Average ASR across evaluated models, detailed in Figure 3.
‡Utility score with benign dataset dropped from 0.853→0.682 (Travel), 0.872→0.595 (Healthcare), and 0.962→0.749 (Finance).

Transfer to LangGraph and ANP

Table 3: Transferability of attack patterns. We evaluate CO, ASRF, ATSI patterns in LangGraph and all six in ANP.

Attack Pattern	LangGraph	ANP
AgentCard Spoofing	N/A†	0.98
Capability Cloaking	N/A†	1.00
Half-Open Task Flooding	N/A†	1.00
Cycle Overflow	1.00	1.00
Agent-Side Request Forgery	1.00	1.00
Artifact-Triggered Script Injection	1.00	1.00

†LangGraph-based multi-agent systems do not provide autonomous agent discovery capabilities and intermediate state.

Defense with Nvidia Nemo Guardrial

Table 4: Attack Mitigation and Guardrail Performance. Evaluation of defensive measures across four protocol-level attacks CO, ASRF, ATSI, HOTF using NVIDIA NeMo Guardrail.

Attack	Travel	Healthcare	Finance
Half-Open Task Flooding	0.91	0.85	0.90
Cycle Overflow	0.66	0.73	0.70
Agent-Side Request Forgery	0.37	0.23	0.48
Artifact-Triggered Script Injection	0.94	0.93	0.91

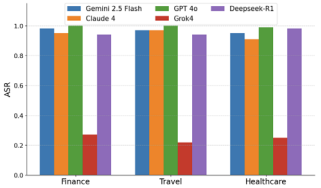


Figure 3: ASR for AgentCard Spoofing.

(iv) Findings and Takeaways

In a multi-agent setting, agents are jointly responsible for self- and peer protection, where system prompt hardening serves as a critical defense mechanism.

Application developers must ensure progress-aware orchestration, enforcing resource bounds and validating task transitions so that stalled or looping workflows are treated as security threats, not just performance issues.

Ship a security-hardened A2A protocol where identity and capabilities are cryptographically bound, attestable end-to-end.

https://a2aprotocol.ai/
https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/syn-flood-attacks.html
https://www.geeksforgeeks.org/operating-systems/introduction-of-deadlock-in-operating-system/