

A2ABreak: Systematic Security Analysis of the A2A Protocol

Alireza Lotfi*, Mirza Masfiquir Rahman*, Imtiaz Karim†, Elisa Bertino*

*Purdue University, †The University of Texas at Dallas

*{lotfia, rahman75, bertino}@purdue.edu, †imtiaz.karim@utdallas.edu

Abstract—The Agent2Agent (A2A) protocol, now governed by the Linux Foundation, is an open standard that enables autonomous AI agents to discover, authenticate with, and delegate tasks to one another across organizational boundaries. Designed to complement the Model Context Protocol (MCP) for tool integration, A2A is rapidly emerging as the horizontal communication layer of the multi-agent ecosystem. Yet the protocol’s security has received no systematic analysis. This paper presents A2ABREAK, the first rigorous systematic security analysis of the A2A protocol. We introduce a novel framework that utilizes an LLM-assisted extraction of a verified finite-state machine directly from the natural-language specification, producing a unified model of 37 states and 76 transitions from 929 formalized statements, and then systematically reasons over this model to discover protocol-level vulnerabilities through adversarial verification, under a full-compliance assumption. Our analysis uncovers 11 new vulnerabilities, each exploitable by a specification-compliant adversary without requiring any implementation flaw. Among the findings are cross-client context injection through unprotected context identifiers, credential harvesting via multi-hop identity loss in delegation chains, and data exfiltration through rogue agents advertising unattested capability claims. A2ABREAK achieves 73.3% precision and 84.6% F1 against independent expert review, while a zero-shot LLM baseline operating over the same specification produces zero confirmed findings, demonstrating that explicit formal grounding is essential for sound protocol security analysis.

1. Introduction

The emergence of autonomous AI agents, systems capable of planning, reasoning, and executing multi-step tasks with minimal human oversight, has created a pressing need for standardized inter-agent communication. While individual agents have grown increasingly capable through frameworks such as LangChain [1], AutoGen [2], and CrewAI [3], their ability to collaborate across organizational and vendor boundaries has remained fundamentally ad hoc: each integration requires bespoke connectors, proprietary message formats, and manually negotiated trust assumptions. The Agent2Agent (A2A) protocol, introduced by Google in April 2025, directly targets this gap by defining an open, HTTP-based standard for agent discovery, authentication, task delegation, and

result exchange [4]. Within months of its release, A2A attracted endorsement from over 50 technology partners, including Salesforce, SAP, ServiceNow, MongoDB, and LangChain, and in June 2025, governance was transferred to the Linux Foundation as an open-source project [5], [6], [7].

A2A is explicitly designed to complement, rather than replace, the Model Context Protocol (MCP), which Anthropic released in November 2024 to standardize the interface between an agent and its local tools [8]. Where MCP governs the *vertical* relationship between an agent and its data sources, APIs, and function calls, A2A governs the *horizontal* relationship between autonomous peers. In a typical production deployment, an orchestrator agent uses MCP to query internal databases and invoke local tools, then uses A2A to delegate specialized subtasks, billing, compliance review, and document generation to remote agents operated by entirely separate organizations. Together, the two protocols form the foundational communication stack for the emerging multi-agent ecosystem, and their joint adoption trajectory suggests that A2A-mediated interactions will soon be the basis of critical enterprise workflows in finance, healthcare, supply-chain management, and government operations.

This trajectory, however, demands scrutiny. A2A introduces a fundamentally different and substantially broader attack surface than prior agent integration paradigms. The protocol’s central design principle is *opaque execution*: a client agent delegates a task to a remote agent without any visibility into the remote agent’s internal reasoning, tool invocations, or intermediate state [4]. This opacity enables cross-vendor collaboration and protects intellectual property, but it also means that a client has no mechanism to verify what a remote agent actually does with a delegated task, the credentials it accumulates, or the artifacts it returns. The security model compounds this exposure by relying predominantly on non-normative guidance for protective measures: the specification expresses critical security controls at the *SHOULD* or *MAY* level rather than as mandatory requirements, and leaves key mechanisms—including the sole primitive for verifying agent identity at discovery time—entirely optional. The combination of opaque execution with this permissive security model creates a potentially large attack surface, whose boundaries have not been formally characterized.

Despite the protocol’s rapid adoption, the security properties of A2A have received virtually no systematic anal-

ysis in the research literature. Existing work has either surveyed the protocol at a descriptive level alongside other agent interoperability standards [9], [10] or proposed high-level security architectures for A2A-based systems without grounding them in a formal analysis of the protocol’s state machine [11]. Louck et al. [12] addressed token-lifecycle privacy gaps without examining broader structural vulnerabilities. More broadly, recent work has identified multi-agent interaction security as a fundamental open challenge, taxonomizing threats that emerge from inter-agent communication such as cascading prompt injection, credential harvesting, and coordinated manipulation [13], but no study has performed specification-level analysis of any individual protocol. No prior work has systematically modeled the A2A interaction lifecycle, enumerated the protocol’s attack surface from its specification, or constructed a formal model suitable for structured vulnerability discovery. Because no open-source reference implementation of A2A exists, implementation-level techniques such as static analysis, fuzzing, or symbolic execution cannot be applied. Specification-only analysis using large language models is also insufficient on its own: in our experiments, a zero-shot baseline with Claude OPUS 4.6 and extended thinking produced nine candidate attacks across two independent runs, yet every candidate was rejected upon adversarial verification. Each attack trace violated normative requirements that the model failed to account for without an explicitly constructed formal model as a grounding mechanism.

Challenges. Performing a rigorous, specification-driven security analysis of A2A requires overcoming three technical challenges. First, natural-language protocol specifications do not define behavioral rules in a logically precise form; constraints, triggers, guard conditions, and state transitions are expressed in prose that lacks formal semantics, and the same sentence may simultaneously describe a precondition, a normative requirement, and a data-type constraint without syntactic distinction, making it infeasible to extract a faithful formal model through direct interpretation (C1). Second, LLM-based generation in protocol analysis is inherently open-ended: outputs have no ground-truth reference model to validate against, and there is no mechanism to detect when a generated artifact lacks a normative basis or introduces inconsistencies. Manual construction avoids this but is prohibitively expensive; prior work on cellular protocol specifications required approximately 2,800 person-hours of expert annotation [14] and cannot track rapid specification updates (C2). Third, vulnerability analysis must be *sound*: every reported finding must be grounded in genuine normative gaps, extracted rules must correspond to actual normative statements, formal representations must preserve their semantics, and the reported missing primitive must be genuinely absent from the specification rather than defined in a different section (C3).

Our Approach. We present A2ABREAK, a framework that combines LLM-assisted formal modeling with adversarial verification to systematically discover protocol-level vulnerabilities from natural-language specifications. A2ABREAK

is organized into three stages. Stage A formalizes the natural-language specification into a verified corpus of 929 structured statements through dual-pass semantic separation: an independent structural extraction pass and a behavioral extraction pass, reconciled and verified against the specification before downstream use. Stage B constructs a unified finite-state machine (FSM) from this corpus, building six per-stage FSMs independently, resolving inter-stage transitions, and applying deterministic and semantic deduplication to produce a final model of 37 states and 76 transitions, a 92% end-to-end reduction from the 929 extracted statements. Stage C reasons over the unified FSM to discover vulnerability candidates and subjects each to adversarial verification that grounds every attack trace in valid FSM states. It further confirms executability under full protocol compliance and verifies that the identified missing primitive is genuinely absent from the specification. All LLM-assisted steps operate under a domain-specific constraint language that forces machine-verifiable formal representations and prevents free-text hallucination, while human checkpoints at stage boundaries ensure that errors do not propagate silently into downstream analysis.

Findings. A2ABREAK identifies eleven protocol-level vulnerabilities spanning discovery, initiation, task execution, and interruption, each exploitable under full specification compliance without requiring any implementation flaw or misconfiguration. The analysis reveals that the specification’s non-normative security posture leaves concrete, exploitable gaps across the protocol lifecycle. Conversation contexts carry no ownership binding or access control (spec sections §3.4.1, §3.4.3), enabling an authenticated adversary to inject tasks into another client’s context, access the victim’s accumulated state, and poison subsequent interactions. Identity is established exclusively at the transport layer and is not propagated across delegation hops (spec sections §7.2, §7.6.2), allowing a malicious intermediary to silently harvest forwarded credentials because downstream agents and credential providers have no means to verify the original principal. The *AgentSkill* data model consists entirely of self-asserted fields with no verification or attestation mechanism (spec sections §4.4.5, §8.4), enabling a fully compliant rogue agent to advertise false capabilities, receive delegated tasks containing confidential data, and return fabricated artifacts without triggering any protocol-level error. The analysis achieves a precision of 73.3% and an F1 score of 84.6% against independent expert manual review, substantially outperforming a zero-shot baseline that produces zero confirmed findings.

Contributions. This paper makes the following contributions:

- **LLM-assisted protocol analysis framework.** We design and implement A2ABREAK, a three-stage framework that transforms natural-language protocol specifications into verified finite-state machines and systematically discovers protocol-level vulnerabilities through constrained LLM reasoning and adversarial verification. We validate the FSM construction stage against the TCP ground-

truth benchmark from PSMBench [15], recovering all 11 protocol states and 19 of 20 transitions (precision 0.76, recall 0.95, F1 0.84), and demonstrate that the framework substantially outperforms zero-shot LLM analysis (Section 4).

- **Formal lifecycle model.** We construct the first finite-state machine model of the complete A2A protocol interaction lifecycle, decomposing it into six stage-specific FSMs that are merged into a unified model of 37 states and 76 transitions, capturing all states, transitions, and failure modes defined by the specification (Section 4).
- **Systematic vulnerability analysis.** We identify and validate eleven protocol-level vulnerabilities in A2A, mapped to specific FSM stages, specification sections, and STRIDE categories, providing the first structured threat model for the protocol. The analysis achieves a precision of 73.3% and an F1 score of 84.6% against expert manual review (Section 6).
- **Open-source framework and artifacts.** We release A2ABREAK’s full source code, all LLM prompts, stage-specific FSM models, and experimental artifacts to support reproducibility. <https://github.com/arlotfi79/A2ABreak>

Responsible disclosure. We have reported all findings to the A2A project maintainers under the Linux Foundation and are waiting for their response. We are committed to working with the A2A maintainers to improve standards through our findings and interactions. Upon publication, we will open-source all artifacts of A2ABREAK, including the extracted statement corpus, the unified FSM, and the full vulnerability analysis traces.

Roadmap. Section 2 provides background on MCP, A2A, and their complementary roles. Section 3 defines the scope, threat model, and problem statement, and discusses the key technical challenges addressed by A2ABREAK. Section 4 details the three-stage framework: specification formalization, FSM construction, and security analysis. Section 5 evaluates the framework’s correctness, refinement contribution, and cost. Section 6 presents the eleven validated vulnerabilities with representative attack scenarios. Section 7 surveys related work, Section 8 concludes the paper and discusses future directions.

2. Background

In this section, we discuss the relevant details of A2A and MCP protocol and further highlight their distinction.

2.1. Agent2Agent Protocol (A2A)

The Agent2Agent (A2A) protocol is an open standard introduced by Google in April 2025 to enable interoperability between autonomous AI agents across frameworks, vendors, and organizational boundaries [16], [17]. A2A defines a client-server model in which a *Client Agent* discovers, authenticates with, and delegates tasks to a *Remote Agent* over HTTPS. A central design principle is *opaque*

execution: agents collaborate based on declared capabilities and exchanged data, without exposing internal reasoning, memory, or tool implementations to one another [16]. The v1.0 specification defines the data model normatively in Protocol Buffers and provides three protocol bindings: JSON-RPC 2.0, gRPC, and HTTP+JSON/REST. The specification uses RFC 2119 keywords [18] to express its requirements at varying normative strengths. A2A launched with support from over 50 technology partners; governance was transferred to the Linux Foundation in June 2025 [17], [19].

A complete A2A interaction proceeds through six stages that structure our FSM model (Section 4) and vulnerability analysis (Section 6). In *Stage 1 (Discovery)*, the client fetches the *AgentCard* a self-asserted JSON document that declares the agent’s identity, skills, endpoint, and authentication requirements which serve as the sole trust anchor for all subsequent interaction; signing is optional [9], [16]. In *Stage 2 (Authentication)*, the client authenticates using the scheme declared in the AgentCard (OAuth 2.0, API keys, mTLS); all credentials are carried at the transport layer, and authentication is strictly hop-by-hop [16], [20]. In *Stage 3 (Initiation)*, the client sends an initial message and the agent responds with either a stateless reply or a stateful *Task* that enters the lifecycle. In *Stage 4 (Task Execution)*, the task is acknowledged and actively processed, with updates delivered via polling, streaming, or push notifications. In *Stage 5 (Interruption)*, the agent may pause execution to request additional input or re-authentication; the client responds, and the task returns to Stage 4. This optional loop may repeat unboundedly. In *Stage 6 (Termination)*, the task reaches a terminal state: completed, failed, canceled, or rejected [16].

2.2. Model Context Protocol (MCP)

Introduced by Anthropic in November 2024 [21], the Model Context Protocol (MCP) standardizes the interface between an LLM application and its local tools and data sources via a client-server architecture over JSON-RPC 2.0 [22], [23], [24]. The protocol addresses the so-called $N \times M$ *integration problem*: without a shared standard, each combination of AI model and external tool requires a bespoke connector, causing integration complexity that grows quadratically with the number of models and services [21]. MCP reduces this to an $N+M$ architecture by defining a universal capability surface through three protocol primitives, namely, *tools* (executable functions an agent can invoke), *resources* (read-only data sources that supply context), and *prompts* (reusable templates that structure model-server interaction); all discoverable at runtime via a machine-readable handshake [23]. MCP was adopted by OpenAI and Google DeepMind in early 2025 [25], [26], and the November 2025 specification revision added support for asynchronous operations, stateless transports, and server identity, broadening its applicability to cloud-hosted and enterprise deployments. In December 2025, Anthropic donated the protocol to the Linux Foundation [27], and by mid-2026, a release candidate for a further major revision,

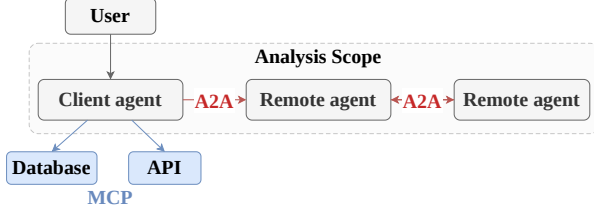


Figure 1: MCP vs. A2A protocol scope

introducing a fully stateless protocol core and an extensions framework, was under community review.

2.3. MCP vs. A2A

MCP and A2A are complementary [17] (Figure 1): MCP governs the *vertical* relationship between an agent and its tools, while A2A governs the *horizontal* relationship between autonomous peers. The critical distinction lies in their trust models. MCP servers are subordinate tool providers under full host control; A2A remote agents are opaque, autonomous peers whose internal state is hidden by design [16]. This opacity prevents clients from verifying what a remote agent does with delegated data, accumulated credentials, or returned artifacts, introducing the broader attack surface analyzed in this paper.

3. Overview

In this section, we define the scope and threat model of our analysis, formally state the problem, and discuss the critical challenges tackled by A2ABREAK.

3.1. Scope

This work targets protocol-level security analysis of the A2A specification [4]. The analysis treats the protocol’s normative rules as the sole evidence base and determines whether they are sufficient to guarantee key security properties, or whether gaps permit attacks even under full compliance. We do not enumerate all possible vulnerabilities, verify specific agent implementations or address transport-layer security concerns.

3.2. Threat Model

We consider an adversary who is computationally bounded and cannot break standard cryptographic primitives but exploits gaps in the A2A protocol specification by sending protocol-conformant messages. The adversary has complete knowledge of the specification and operates exclusively through A2A-defined operations. The adversary may act as a client initiating requests to a legitimate agent, as a server receiving tasks from a legitimate client, or as an intermediary occupying both roles within a multi-agent delegation chain. In all cases, the adversary is fully authenticated and compliant; their advantage derives entirely from what the protocol fails to define, not from any deviation from the specification.

3.3. Problem Statement

Given a protocol specification in natural language, A2ABREAK aims to generate a formal model and systematically discover protocol-level vulnerabilities.

Specification corpus. Let $\mathcal{D} = \{d_1, d_2, \dots, d_M\}$ denote the set of the M sections of the A2A specification. Each section d_k is parsed into a sequence of statements $C_k = \langle c_1^{(k)}, c_2^{(k)}, \dots \rangle$, where each statement is either a structural definition or a normative rule expressed using RFC 2119 keywords. The full corpus is $\mathcal{C} = \bigcup_k C_k$ with $|\mathcal{C}| = N$ after deduplication.

Protocol FSM. From $\mathcal{C}$, we derive a finite-state machine $\mathcal{F} = (S, s_0, T, \Sigma, S_f)$, where S is the set of protocol states, $s_0 \in S$ the initial state, $S_f \subseteq S$ the terminal states, Σ the event alphabet, and $T \subseteq S \times \Sigma \times \mathcal{G} \times S$ the transition relation with guard conditions $\mathcal{G}$. Each transition $\tau = (s, \sigma, g, s') \in T$ represents a state change from s to s' triggered by event σ under guard g .

Protocol Vulnerability. A protocol-level vulnerability is a tuple $\mathcal{V} = (\pi, \rho, \phi)$: a feasible attack trace $\pi = \langle \tau_1, \dots, \tau_k \rangle$ through $\mathcal{F}$, a violated security property ρ , and a missing primitive ϕ whose absence permits the violation. A trace is *feasible* if every transition exists in T and every guard can be satisfied by a fully compliant party.

3.4. Challenges

C1: Infeasibility of semantic interpretation of natural-language specifications. Natural-language protocol specifications do not define behavioral rules in a logically or mathematically precise form. Constraints, triggers, guard conditions, and state transitions are expressed in prose that lacks formal semantics, i.e., the same sentence may simultaneously describe a precondition, a normative requirement, and a data type constraint without syntactic distinction. This ambiguity makes it infeasible to extract a faithful formal model through direct interpretation.

C2: Ensuring correctness of open-ended model generation. LLM-based generation in protocol analysis is inherently open-ended: outputs have no formal reference model to validate against, and there is no mechanism to detect when a generated artifact lacks a normative basis or introduces inconsistencies. Standard models lack the sustained reasoning capacity to make interdependent decisions reliably over large inputs, and errors propagate invisibly into downstream analysis. Manual construction avoids this, but is prohibitively expensive. Prior work on cellular protocols required approximately 2,800 man-hours of expert annotation [14] and cannot track rapid specification updates. Directly prompting an LLM to identify vulnerabilities without a formal model produces plausible but incorrect findings. From our experiments, we see zero-shot baselines generate nine candidates across two runs, all rejected upon manual review.

C3: Ensuring soundness of automatic vulnerability analysis. For vulnerability analysis, soundness requires that every output of the framework is faithful to the specification—

extracted rules must correspond to actual normative statements, formal representations must preserve their semantics, and reported vulnerabilities must represent genuine design flaws. Without explicit constraints on what the model can produce at each stage, errors accumulate silently. For instance, an extraction stage may invent a guard condition that does not appear in the specification, an FSM construction stage may connect states that the protocol never links, and a vulnerability analysis stage may report a primitive as absent when it is defined in a different section. Each error is individually plausible, and downstream stages have no mechanism to detect that their inputs are wrong.

3.5. Our Solution Approaches

S1: Dual-pass semantic separation. A2ABREAK addresses the semantic interpretation challenge by separating specification interpretation into two independent passes. The key insight is that structural definitions and behavioral rules, though interleaved in prose, impose fundamentally different extraction constraints; a single pass conflates them, producing artifacts where type definitions are misread as transitions and vice versa. The first pass targets structural content: data types, field definitions, object hierarchies, and enumerations. The second targets behavioral content: state transitions, guard conditions, normative requirements, and protocol operations. The two passes are executed independently, and their outputs are reconciled before further processing, preventing cross-contamination between definitional and behavioral semantics and allowing each pass to apply an extraction schema optimized for its specific content category.

S2: Input reduction and extended thinking. A2ABREAK addresses this challenge by decomposing the analysis into a sequence of focused steps, each operating over a narrow, task-specific input slice rather than the full specification. The insight is that open-ended generation over large inputs is where LLM reasoning is most error-prone; by restricting each step’s scope, we convert an unconstrained synthesis problem into a series of bounded decisions that are individually verifiable. As a concrete example, our framework begins with an extraction stage that parses the specification into individual formal statements—929 statement in total—but only 283 (30%) of these carry transition structure and enter FSM construction; the remainder are filtered out before downstream steps. Within each step, all generation is delegated to a model with extended thinking capabilities, providing the sustained reasoning depth needed to make interdependent decisions reliably over the reduced inputs.

S3: Domain-specific constraint language for analysis soundness. A2ABREAK embeds formally specified constraints across all framework prompts that collectively function as a domain-specific language governing both extraction and analysis. In the extraction stages, the constraints define a formal transition schema ($s_1 \xrightarrow{c/a} s_2$) with typed fields for guards, triggers, and post-conditions expressed in a logical notation ($\&$, $|$, $!$, dot-qualified attributes), forcing the model to produce machine-verifiable formal representations rather than free-text summaries. In the analysis stages, the con-

straints define what counts as a valid finding: the flaw must exist in what the protocol does not say; vague delegation is not intentional exclusion; and an outcome mandate without a mechanism is a missing primitive. The verification stage enforces these through four adversarial steps: ground the trace in FSM states, confirm executability under full compliance, verify no normative rule at any modality prevents the trace, and confirm the primitive is genuinely absent. Candidates that fail any step are discarded with specific falsifying evidence.

4. Design of A2ABREAK

A2ABreak is organized into three stages (shown in Figure 2): Stage A formalizes the natural-language specification into a verified corpus of structured statements; Stage B constructs a unified finite-state machine from that corpus, and Stage C reasons over the FSM to discover and validate protocol-level security vulnerabilities. Human checkpoints at the stage boundaries ensure that errors do not propagate silently into the security analysis.

4.1. Stage A: Specification Formalization

Stage A transforms the A2A protocol specification from natural language into a machine-processable corpus of formal statements. Because the specification mixes definitional structure with RFC 2119 behavioral rules, extraction is split into two independent passes targeting these orthogonal concerns separately. Every extracted statement is encoded as a component of the FSM tuple $\mathcal{F} = (S, s_0, T, \Sigma, S_f)$: states populate S , transitions populate T , and trigger strings contribute to the event alphabet Σ . Each transition is represented as a tuple $\langle s_1, \text{pre_cond}, \text{trigger}, \text{post_cond}, s_2 \rangle$ instantiating the formal notation $s_1 \xrightarrow{c/a} s_2$, where c is the guard condition and a is the resulting effect.

A1: Structural Extraction. Step A1 extracts the implicit formal structure that all subsequent behavioral analysis depends on. It produces three categories of statement: FSM_STATE entries name every distinct lifecycle condition a protocol entity can occupy and classify each as initial, intermediate, or terminal; CONSTRAINT entries capture global data invariants that hold across all states; and IMPLICIT_TRANSITION entries encode state changes implied by definition tables and protocol flow diagrams, even when no RFC 2119 keyword accompanies them. Each statement receives a stable identifier and is annotated with the protocol stage it belongs to.

A2: Behavioral Extraction. Step A2 performs an independent pass over the same source, targeting exclusively sentences that contain an RFC 2119 keyword. Each such sentence is encoded as a BEHAVIORAL transition statement using the identical schema used by Step A1, preserving the normative obligation strength (must, shall, should, may) for downstream analysis. Step A2 deliberately does not re-extract definition tables or enumeration values, keeping the two corpora semantically disjoint.

A3: Reconciliation. The outputs of Steps A1 and A2 are merged into a single unified statement list. Exact and near-duplicate statements arising from the same normative rule

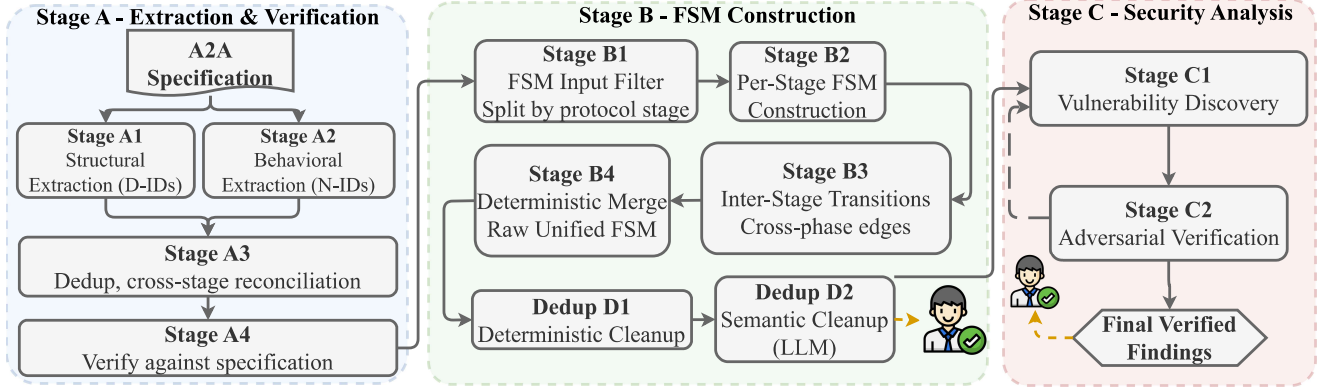


Figure 2: Overview of the A2ABreak.

being captured by both passes are detected and collapsed into a canonical form, preserving source identifiers from both origins for traceability. Ambiguous cases are resolved by an LLM pass and flagged for mandatory human review before A2ABREAK advances.

A4: Specification Verification. Every statement in the reconciled corpus is independently verified against the live specification source text before any FSM construction begins. Each statement’s extracted fields are confirmed against the originating specification section, with automated checks detecting hallucinated states, misattributed normative keywords, and guard conditions absent from the source. Statements that fail verification are corrected or removed, and the outcome is recorded in a verification report that forms a mandatory human checkpoint before Stage B.

4.2. Stage B: FSM Construction

Stage B converts the verified statement corpus into a single unified FSM. To manage the complexity of a six-stage protocol, per-stage FSMs are constructed independently and then connected by resolving cross-stage transitions before a final deterministic merge.

B1: FSM Input Filtering. Step B1 is a deterministic, LLM-free filter that partitions the verified corpus by protocol stage and retains only statements that carry sufficient transition structure to contribute a state or edge to the graph. Its output is six per-stage input bundles, one for each protocol stage.

B2: Per-Stage FSM Synthesis. Step B2 constructs a complete FSM for each of the six protocol stages from its B1 input bundle. For each stage, the synthesizer deduplicates states, discards entries that are not protocol session states, and infers implicit states referenced in transitions but absent from explicit definitions. Every state is assigned an actor field, `client`, `server`, `shared`, or `unknown`, derived by reasoning about which principal controls entry into and exit from that state. Shared-actor states mark trust boundaries and are primary targets for the security analysis in Stage C. Where the input statements leave gaps, the synthesizer queries the live specification before recording unresolvable cases in diagnostic lists that form a mandatory human checkpoint.

B3: Inter-Stage Transition Resolution. Once all six per-stage FSMs exist, Step B3 identifies the cross-stage edges that connect them. It examines exit points in each stage of the FSM and determines which entry points of the downstream stage they connect to, along with the trigger event and guard condition governing each handoff. The seven-stage boundaries resolved include the bidirectional cycle between `task_execution` and `interruption`. Step B3 produces only inter-stage transitions and does not modify the intra-stage graphs from B2.

B4: Unified FSM Assembly. Step B4 deterministically assembles the unified FSM by combining all six per-stage FSMs from Step B2 and the inter-stage edges from Step B3 into a single connected graph. The resulting unified FSM is then refined through two deduplication sub-passes. Sub-pass D1 is a deterministic workflow that normalizes naming inconsistencies, merges states duplicated across stages, removes identity inter-stage transitions that are artifacts of stage-scoped modeling, and merges semantically equivalent duplicates transitions. It concludes with structural validation and uses a heuristic detectors to flag candidate issues, meta-states, absence states, and unreachable non-protocol states for the subsequent pass. Sub-pass D2 is a targeted LLM pass that triages these candidates: removing abstract meta-states and redistributing their behavior, collapsing trivial pass-through states, discarding deployment-lifecycle states that no protocol operation can reach, and adding missing incoming transitions for disconnected sub-lifecycles by consulting the specification provenance attached to each state. The output is a fully connected FSM with zero orphaned states and full referential integrity. Figure 9 (available in Appendix A) illustrates the derived unified FSM for the A2A protocol, comprising 37 states and 76 transitions. For better resolution, we refer to the artifact repository.

4.3. Stage C: Security Analysis

Stage C reasons over the unified FSM to identify and validate protocol-level security vulnerabilities. The analysis operates under a full compliance assumption: every `MUST`, `SHOULD`, and `MAY` is satisfied by all parties, implementations are correct, and transport security is properly estab-

lished. Under this assumption, any surviving finding is a flaw in the protocol design itself. The two-step structure separates generative discovery from adversarial falsification to prevent confirmation bias.

C1: Vulnerability Discovery. Step C1 examines the unified FSM for structural properties associated with known security vulnerabilities: missing authorization guards on sensitive transitions, absent checks at stage boundaries, states reachable without traversing required predecessors, and event-alphabet gaps that permit unanticipated message sequences. Before reporting any candidate, two filters are applied. First, the attack trace must be executable entirely within the A2A protocol as defined, with no out-of-band mechanisms. Second, a web search of the original specification must confirm that the identified gap is a genuine missing primitive rather than an intentional design choice, vague delegation to implementations does not qualify as intentional exclusion. C1 is executed multiple times as independent discovery passes; candidates from all runs are merged before Step C2, improving recall across the candidate set.

C2: Adversarial and Manual Verification. Step C2 subjects every C1 candidate to a combined automated and human verification process. Each candidate is first grounded in the unified FSM, discarding any attack path that cannot be reconstructed from valid states and transitions. The remaining candidates are executed step by step under full protocol compliance to confirm exploitability and to ensure that no existing normative rule prevents the attack. The specification is then searched comprehensively to verify that the identified primitive is truly absent rather than defined elsewhere.

Confirmed candidates subsequently undergo an independent manual review against the specification and unified FSM. A domain expert validates the FSM trace, and confirms that the cited specification evidence supports the claimed absence of the missing primitive. Candidates that fail either automated or manual scrutiny are downgraded to inconclusive or discarded with recorded justification. The final output partitions the candidate set into confirmed findings, discarded candidates, and inconclusive cases, providing a complete audit trail from discovery through validation.

5. Evaluation

We evaluate the effectiveness of A2ABreak through the following research questions:

- **RQ1.** How does A2ABreak compare against a zero-shot analysis of the protocol?
- **RQ2.** How is the correctness of A2ABreak evaluated?
- **RQ3.** How does each pipeline stage contribute to FSM refinement?
- **RQ4.** Where does specification complexity concentrate across A2A protocol stages?
- **RQ5.** What is the execution cost of A2ABreak?

RQ1: Zero-Shot Comparison. To compare A2ABREAK against a zero-shot baseline, we prompted Claude Opus 4.6 with high reasoning enabled using only the A2A specification and instructed it to identify protocol-level attack scenarios under the same constraints enforced by our pipeline. Two

TABLE 1: TCP FSM validation against PSMBench.

	Ground Truth	Extracted	Matched	Missed
States	11	11	11	0
Transitions	20	25	19	1

independent zero-shot runs produced nine candidate attacks. Applying the same Stage C2 adversarial verification, all nine candidates were rejected, primarily because their attack traces were already prohibited by normative requirements that the model failed to account for. These results suggest that, without an explicitly constructed FSM as a grounding mechanism, even advanced reasoning models lack a precise representation of protocol behavior, limiting their ability to distinguish genuine protocol design gaps from behaviors already constrained by the specification.

RQ2: Correctness Validation. We evaluate correctness at two levels: the accuracy of the FSM construction pipeline (Stages A–B) and the precision of the vulnerability analysis pipeline (Stages C1–C2). For FSM construction, we benchmark Stages A and B against the TCP ground-truth Protocol State Machine (PSM) from PSMBench [15], which pairs cleaned RFC specifications with manually validated states and transitions across widely deployed protocols. Because no comparable benchmark exists for A2A, TCP serves as a representative proxy for assessing whether the pipeline can accurately derive FSMs from natural-language specifications. Our framework recovers all 11 protocol states and 19 of 20 ground-truth transitions, missing only a single `LISTEN` $\rightarrow$ `SYN_SENT` edge that was extracted with a variant event label, achieving a precision of 0.760, recall of 0.950, and an F1-score of 0.844. Table 1 summarizes the results.

For vulnerability analysis, across five analysis runs Stage C1 produced 17 unique candidate vulnerabilities. Stage C2 accepted 16 and correctly rejected 1. Manual expert review, applying the same compliance assumptions and evaluation criteria used throughout the pipeline, validated 11 of the 16 as true positives, identified 1 duplicate, and overturned 4 as false positives (preventable by existing normative rules under full compliance). Overall, A2ABREAK achieved a precision of 73.3% (11 TP out of 15 non-duplicate candidates), and an F1 score of 84.6%.

RQ3: Pipeline Refinement. Figure 4 shows the 929 verified statements produced by Stage A, broken down by category and protocol stage. The two extraction passes yield five statement categories: FSM States name discrete life-cycle conditions an entity can occupy (e.g., `submitted`, `working`); Implicit Transitions capture state changes implied by prose without RFC 2119 keywords; Behavioral statements are explicit normative rules containing RFC 2119 keywords (`MUST`, `SHOULD`, `MAY`); Field Presence constraints record required or optional fields on protocol objects; and OneOf Constraints capture mutually exclusive field groups where exactly one must be present (e.g., a `Part` must contain one of `TextPart`, `FilePart`, or `DataPart`). Of these, only the first three carry transition

structure and enter FSM construction: FSM States populate the state set $\mathcal{S}$, while Implicit Transitions and Behavioral statements populate the transition relation $\mathcal{T}$. Field Presence and OneOf Constraints, which dominate Discovery (73 of 165) and Task Execution (235 of 406), encode structural invariants but contribute no states or edges, and are filtered out before FSM synthesis.

Figure 3 traces the subsequent FSM refinement. Of the 929 verified statements, only 283 (30%) carry transition structure and enter FSM construction, producing a raw graph of 38 states and 245 transitions. Each subsequent stage targets a distinct class of redundancy: per-stage synthesis (B2) eliminates 168 intra-stage duplicate transitions, deterministic cleanup (D1) collapses 24 mechanically duplicate states across stage boundaries, and semantic deduplication (D2) resolves 4 naming-level equivalences detectable only by an LLM pass while recovering 8 previously obscured transitions. The final unified FSM of 37 states and 76 transitions represents a 69% transition reduction from raw synthesis and a 92% end-to-end reduction from the 929 extracted statements—confirming that each pipeline stage addresses an orthogonal source of noise whose removal is necessary for a tractable and faithful protocol model.

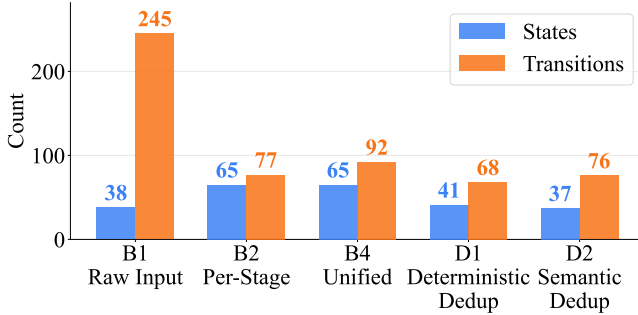


Figure 3: FSM state and transition counts across A2ABREAK stages.

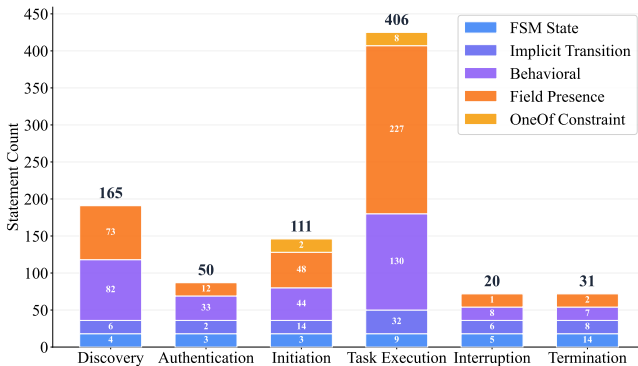


Figure 4: Distribution of 929 verified statements by category and protocol stage after Stage A extraction.

RQ4: Specification Complexity Concentration. Figure 5 shows the distribution of extracted statements, FSM states, and FSM transitions across the six protocol stages after full

deduplication. The pipeline reveals that textual volume is a misleading proxy for behavioral complexity: Authentication contains only 21 statements but produces 16 transitions (0.76 per statement), three times the ratio of Discovery (0.24). Interruption is the smallest stage by every metric yet encodes the re-entry loop central to credential accumulation in multi-hop chains. These mismatches are invisible from a linear reading of the specification and unrecoverable by monolithic FSM construction; they emerge precisely because the pipeline decomposes extraction by protocol stage, enabling quantitative cross-stage comparison that directs security analysis toward stages where behavioral density, not page count, indicates attack surface concentration.

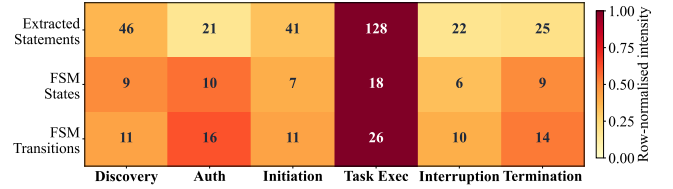


Figure 5: Specification complexity of A2A protocol stages.

RQ5: Execution Cost. Table 2 reports the API cost of a single end-to-end run of A2ABREAK. Stages A1–A4 use Claude Sonnet 4.6 with high reasoning, while the more structurally complex Stages B2, B3, D2, C1, and C2 use Claude Opus 4.6 with high reasoning. The total one-pass cost is \$40.97, of which \$34.67 is consumed by Stages A and B, with Stage B2 being the single most expensive step at \$11.64, followed by Stage A1 at \$9.04 and Stage A4 at \$5.42. The FSM deduplication pass (D2) adds \$1.08, and a single Stage C discovery and verification pass adds \$6.30. Mechanical stages involving no LLM calls (B1, B4, D1) incur zero cost.

TABLE 2: Per-stage API cost for one A2ABREAK run

Stage	Model	Reasoning	Cost (USD)
A1	Sonnet 4.6	High	\$9.04
A2	Sonnet 4.6	High	\$5.46
A3	Sonnet 4.6	High	\$0.38
A4	Sonnet 4.6	High	\$5.42
B2	Opus 4.6	High	\$11.64
B3	Opus 4.6	High	\$1.65
D2	Opus 4.6	High	\$1.08
C1	Opus 4.6	High	\$3.20
C2	Opus 4.6	High	\$3.10
Total			\$40.97

6. Security Analysis

To demonstrate that the vulnerabilities identified by A2ABREAK carry concrete security consequences, we present the full set of eleven validated findings in Table 3, each exploitable under full specification compliance without requiring any implementation flaw or misconfiguration. In this section, we discuss three representative findings in detail spanning discovery, initiation, and interruption to illustrate

the attack scenarios, message flows, and impact of the missing protocol primitives.

(1) Cross-Client Context Injection. The A2A protocol allows agents to maintain persistent conversation state across tasks via a shared `contextId`, accumulating sensitive information such as user preferences, prior approvals, and business data. However, the protocol defines no ownership model for contexts: unlike tasks, which are bound to an authenticated principal at creation, a `contextId` carries no creator field, no access token, and no authorization requirement. Any authenticated client that knows a valid `contextId` can inject a new task into that context and receive a response informed by another client’s conversational state.

Root cause. The root cause is the specification permitting clients to supply a `contextId` without ownership verification. Specification Section §3.4.1 allows servers to accept client-provided `contextId` values, and §3.4.3 defines cross-client context entry as a normative pattern: clients MAY supply a `contextId` without a `taskId` to start a new task within an existing context. The §13.1 authorization MUSTs are scoped exclusively to task operations, with no corresponding requirement at the context layer.

Attack. The adversary (Client B) first authenticates as a legitimate client. As shown in Figure 6, Client B then issues `SendMessage(taskId=∅, contextId=X)`, where `contextId=X` belongs to an ongoing session established by the victim (Client A). The server validates only `contextId-taskId` consistency, a check that passes and creates a new task in context X with no ownership verification. The server then processes Client B’s task against Client A’s accumulated conversational history and returns a response informed by that state to Client B.

Impact. The attack produces two consequences. First, the adversary receives responses informed by the victim’s prior state, leaking sensitive context, including approvals, and business data. Second, the victim’s subsequent tasks are processed against a history poisoned by the adversary’s injected messages, corrupting future results without bypassing any task-level control. The adversary can repeatedly inject into the same context, deepening the poisoning effect over time.

(2) Multi-Hop Identity Loss. The A2A protocol enables multi-agent execution by delegating tasks across a chain of agents. Identity, however, is established exclusively at the transport layer and is not propagated across hops: each agent knows only its immediate caller’s identity. While sufficient for point-to-point interactions, this design is architecturally incompatible with multi-hop delegation chains, where downstream agents and credential providers must verify the original principal’s identity and delegation provenance.

Root cause. The specification states that payloads do not carry user or client identity directly and that identity is established at the transport layer (spec section §7.2). No delegation context token, principal identity field, or chain-of-custody primitive exists to bridge this hop-by-hop model with the multi-hop delegation chains the protocol explicitly enables via `TASK_STATE_AUTH_REQUIRED` (spec sections §7.5, §7.6).

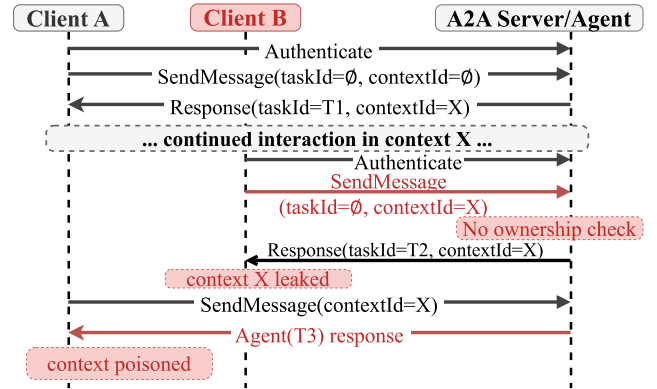


Figure 6: Cross-Client Context Injection

Attack. As shown in Figure 7, the User authenticates with Agent A and delegates a task. Agent A forwards the task to Agent B, controlled by the adversary, using its own credentials; the User’s identity is lost at this hop boundary. Agent B delegates the task onward to Agent C, which requires credentials for a third-party resource and returns `TASK_STATE_AUTH_REQUIRED`. This signal propagates back through the chain carrying no delegation context: neither a principal identity field, nor a chain identifier, nor any indication of which downstream agent originated the request. Agent A receives the authorization request with no information about who triggered it or on whose behalf, and forwards it to the User. The User is presented with an authorization request that appears to originate from Agent A, with no protocol-supplied means to verify the actual requester, the delegation depth, or whether the request is legitimate. The User obtains the required credentials out-of-band (spec section §7.6), but the credential provider likewise receives no protocol-level context about the originating principal or the delegation chain, and therefore cannot make an informed authorization decision.

Impact. The attack produces two consequences. First, credential providers cannot make informed authorization decisions, as the original principal’s identity and the delegation chain are invisible to them the out-of-band credential acquisition occurs without any protocol-supplied context about who initiated the chain or which downstream agent requires authorization. Second, intermediary agents receive forwarded credentials with no protocol-level restriction on misuse, a compromised Agent B can retain and replay credentials beyond the intended delegation scope. An adversary controlling any intermediate agent can silently harvest credentials across repeated delegation chains.

(3) Unattested Skill Claims. The A2A protocol’s `AgentSkill` data model (spec section §4.4.5) consists entirely of self-asserted string fields: `id`, `name`, `description`, `tags` with no protocol-level mechanism for verification, attestation, or challenge-response validation.

TABLE 3: Validated A2A Protocol-Level Vulnerabilities

Stage	#	Vulnerability	STRIDE Category	Impact (CIA)	Spec Reference
<i>Discovery</i>	1	Unattested Skill Claims	Spoofing	Confidentiality, Integrity	§4.4.5, §8.4
	2	JWS Key Trust Model Gap	Spoofing	Confidentiality	§8.4
<i>Initiation</i>	3	Cross-Client Context Injection	Info. Disclosure, Tampering	Confidentiality, Integrity	§3.4.1, §3.4.3
<i>Task Execution</i>	4	SSE Post-Revocation Leakage	Info. Disclosure	Confidentiality	§7.4, §3.2.3
	5	Artifact Chunk Integrity Gap	Tampering	Integrity	§4.2.2, §3.1.6
	6	Concurrent Access TOCTOU	Tampering, Elevation of Privilege	Integrity, Confidentiality	§3.1.1, §4.1
	7	Unverified Webhook URL	DoS, Info. Disclosure	Availability, Confidentiality	§13.2
<i>Interruption</i>	8	Auth Scope Amplification	Elevation of Privilege	Confidentiality, Integrity	§7.6.1, §7.6.2
	9	Multi-Hop Identity Loss	Spoofing, Info. Disclosure	Confidentiality	§7.2, §7.6.2
	10	Circular Delegation Deadlock	DoS	Availability	§7.6.2, §4.1.3
	11	No Timeout from Interrupted States	DoS	Availability	§4.1.3

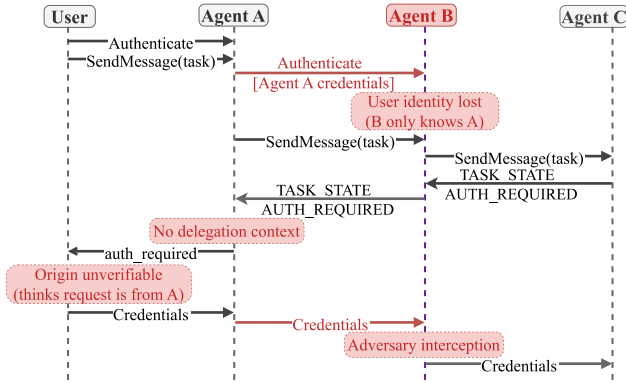


Figure 7: Identity Loss in Multi-Hop Delegation Chains

Client agents use these unverified claims to select downstream agents for task delegation. Agent Card signing (JWS, spec section §8.4) authenticates identity but does not attest capability: it confirms *who* published the card, not *whether* the advertised skills are truthful.

Root cause. The specification provides no skill verification primitive. The `AgentSkill` object is a self-declared advertisement with no integrity binding to actual capability. The specification introduces no planned mechanism to close this gap, confirming that skill claims remain entirely trust-based in the current protocol design.

Attack. As shown in Figure 8, a fully compliant malicious agent publishes an Agent Card advertising false skill claims for a sensitive domain. A client agent discovers the card, selects the malicious agent based on its unverified skill advertisement, authenticates, and delegates a task containing sensitive domain-specific data. The malicious agent accepts the task, satisfying every normative requirement, exfiltrates the data, and returns fabricated artifacts. In multi-agent chains, fabricated artifacts propagate downstream undetected.

Impact. The attack produces two consequences. First, sensitive task data is exposed to an agent that has no actual capability to process it, enabling direct exfiltration. Second, fabricated artifacts injected into multi-agent workflows corrupt

downstream task results without triggering any protocol-level error, as the specification defines no mechanism to distinguish genuine skill execution from fabrication.

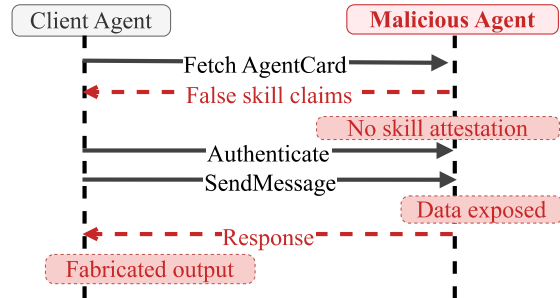


Figure 8: Unattested Skill Claims

7. Related Work

Our work intersects two research areas: the emerging literature on agentic AI security, and the established tradition of finite state machine-based security analysis of communication protocols. We survey each in turn and position our contributions relative to the state of the art.

7.1. Security of Agentic Systems

Security analyses of agent communication protocols have emerged alongside the rapid deployment of LLM-powered agents [10], [28], [29], [30], [31], [32]. For MCP, Radosevich and Halloran [28] demonstrated tool poisoning and cross-server exfiltration exploits; Maloyan and Namiot [29] identified three architectural vulnerabilities and proposed ATTESTMCP; and Hou et al. [30] constructed a threat taxonomy across the full MCP server lifecycle. However, A2A security has received limited attention. Habler et al. [11] applied the MAESTRO framework to A2A deployments but provided no formal protocol model. Louck et al. [12] addressed token-lifecycle privacy gaps without examining broader structural vulnerabilities. Anbiaee et al. [33] performed a comparative threat analysis across MCP, A2A, Agora, and ANP based on literature review rather than direct specification analysis. More broadly, Ferrag et al. [31] and Deng et al. [32] surveyed agentic AI threat landscapes,

cataloging attack surfaces from prompt injection to cross-agent manipulation, but neither performed specification-level analysis of any individual protocol.

7.2. Protocol Analysis and Security Verification

Protocol specifications have long been a de facto source of security analysis [14], [34], [35], [36], [37]. Finite state machines have received widespread attention [14], [15], [34], [35], [37], [38]. De Ruiter and Poll [39] pioneered protocol state fuzzing, using learned state machines to discover flaws in TLS implementations; Fiterau-Brostean et al. [40] extended this to DTLS, uncovering a full authentication bypass; and Shi et al. [41] extracted protocol format specifications as state machines from implementation code to enhance downstream fuzzers. Moving from inferred to formally specified state machines, Fett et al. [38] conducted the first comprehensive formal analysis of OAuth 2.0, proving security properties while discovering two new attacks, and tools such as ProVerif [42], Scyther [43], LTEInspector [35], and 5GReasoner [34] enable automatic verification of security protocols from their specifications. Shen et al. [15] introduced PSMBench, a benchmark pairing cleaned RFC text with manually validated protocol state machines across 14 protocols, establishing a ground truth for evaluating automated FSM extraction from natural-language specifications. Beyond analysis, FSMs have also been deployed as runtime enforcement mechanisms: Che et al. [44] modeled BLE attack patterns as malicious transition paths and used lightweight eBPF-based monitors to mitigate session-based attacks across IoT devices. These works collectively demonstrate the effectiveness of FSM-based approaches across the protocol security lifecycle, from specification analysis to runtime enforcement.

To our knowledge, no prior work has constructed a formal state machine model of the A2A protocol, or performed a specification-driven vulnerability analysis using such a model. This paper addresses all these gaps.

8. Conclusion and Future Work

We presented A2ABREAK, the first systematic security analysis of the Agent2Agent protocol. Through dual-pass specification formalization, LLM-assisted finite-state machine construction, and adversarial verification under a full-compliance assumption, A2ABREAK uncovered eleven protocol-level vulnerabilities spanning discovery, initiation, task execution, and interruption, each exploitable by a specification-compliant adversary without requiring any implementation flaw. Our findings establish that the A2A specification’s treatment of critical security properties, context ownership, delegation provenance, capability attestation, credential scoping, as implementation concerns rather than protocol-level invariants produces design gaps that propagate directly as exploitable vulnerabilities in any compliant deployment.

Future Work. This paper establishes a threat model and demonstrates concrete vulnerabilities in the A2A protocol. Several directions remain open for future investigation.

A2A Simulation Framework. To our knowledge, no dedicated framework exists for simulating multi-agent A2A deployments at scale. A systematic simulation framework would enable reproducible security testing across the full protocol lifecycle, support automated fuzzing of A2A message flows, and allow researchers to evaluate mitigations under controlled conditions. Building such a framework is a natural and necessary next step for the community.

Security Gap Analysis between A2A and MCP. While this paper treats A2A and MCP as complementary protocols operating at distinct layers, their combination in production deployments creates a composite attack surface that neither specification addresses in isolation. Future work should systematically characterize the security properties of A2A–MCP integration points: specifically, how trust established at the A2A layer propagates into MCP tool invocations, whether prompt injection at the MCP layer can influence A2A task delegation decisions, and what authorization invariants must hold across the boundary between the two protocols to prevent cross-layer privilege escalation.

Protocol-Level Mitigations. The vulnerabilities identified by A2ABreak trace to specific missing primitives in the A2A specification. Future work should design and formally evaluate concrete protocol extensions that mitigate these vulnerabilities while preserving backward compatibility.

Acknowledgments

The work reported in this paper has been supported by NSF under grants 2229876, National Artificial Intelligence Research Resource Pilot award 250336, the University of Texas System Rising STARs Award (No. 40071109), and the startup funding from the University of Texas at Dallas.

References

- [1] “Langchain: Powering the agent development lifecycle,” 2026. [Online]. Available: <https://www.langchain.com/>
- [2] “Autogen: A framework for building ai agents and applications,” 2026. [Online]. Available: <https://microsoft.github.io/autogen/stable/index.html>
- [3] “Crewai for enterprise agents,” 2026. [Online]. Available: <https://crewai.com/>
- [4] “Google agent2agent protocol specification.” [Online]. Available: <https://a2a-protocol.org/latest/specification/>
- [5] “Ibm think: What is a2a protocol.” [Online]. Available: <https://www.ibm.com/think/topics/agent2agent-protocol>
- [6] “Linux foundation launches the agent2agent protocol.” [Online]. Available: <https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents>
- [7] “A2a protocol lands in major cloud platforms, and sees enterprise production use.” [Online]. Available: <https://www.linuxfoundation.org/press/a2a-protocol-surpasses-150-organizations-lands-in-major-cloud-platforms-and-sees-enterprise-production-use-in-first-year>
- [8] “Anthropic introduces the model context protocol.” [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>

- [9] A. Ehtesham, A. Singh, G. K. Gupta, and S. Kumar, "A survey of agent interoperability protocols: Model context protocol (mcp), agent communication protocol (acp), agent-to-agent protocol (a2a), and agent network protocol (anp)," 2025. [Online]. Available: <https://arxiv.org/abs/2505.02279>
- [10] Z. Anbiaee, M. Rabbani, M. Mirani, G. Piya, I. V. Opushnyev, A. A. Ghorbani, and S. Dadkhah, "Security threat modeling for emerging AI-agent protocols: A comparative analysis of MCP, A2A, Agora, and ANP," *arXiv preprint arXiv:2602.11327*, 2026. [Online]. Available: <https://arxiv.org/abs/2602.11327>
- [11] I. Habler, K. Huang, V. S. Narajala, and P. Kulkarni, "Building a secure agentic ai application leveraging a2a protocol," *arXiv preprint arXiv:2504.16902*, 2025.
- [12] Y. Louck, A. Stulman, and A. Dvir, "Improving Google A2A protocol: Protecting sensitive data and mitigating unintended harms in multi-agent systems," *arXiv preprint arXiv:2505.12490*, 2025. [Online]. Available: <https://arxiv.org/abs/2505.12490>
- [13] C. S. de Witt, K. Krawiecka, I. Krawczuk, B. Hagag, W. L. Anderson, P. Belcak, B. Bucknall, X. Cai, A. Chopra, D. Cohen, R. F. D. Rosario, A. Draguns, A. Gray, K. Katz, V. Mavroudis, J. Mink, S. R. Motwani, J. Petit, L.-S. Rembeck, C. Smith, J. Sotiropoulos, S. Young, S. Scheffler, and M. Llewellyn, "Open challenges in multi-agent security: Towards secure systems of interacting AI agents," *arXiv preprint arXiv:2505.02077*, 2025, revised April 2026. [Online]. Available: <https://arxiv.org/abs/2505.02077>
- [14] A. A. Ishtiaq, S. S. S. Das, S. M. M. Rashid, A. Ranjbar, K. Tu, T. Wu, Z. Song, W. Wang, M. Akon, R. Zhang, and S. R. Hussain, "Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications," 2023. [Online]. Available: <https://arxiv.org/abs/2310.04381>
- [15] Z. Shen, X. Luo, I. Karim, and E. Bertino, "Psmbench: A benchmark and dataset for evaluating llms extraction of protocol state machines from rfc specifications," in *Advances in Neural Information Processing Systems*, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, Eds., vol. 38. Curran Associates, Inc., 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/file/521bd9583b3a39e92f662ee57e81e5ce-Paper-Datasets_and_Benchmarks_Track.pdf
- [16] A2A Project, "Agent2Agent (A2A) protocol specification," Linux Foundation, Tech. Rep., 2025, <https://a2a-protocol.org/latest/specification/>.
- [17] IBM, "What is Agent2Agent (A2A) protocol?" <https://www.ibm.com/think/topics/agent2agent-protocol>, 2025, accessed: 2026-04-28.
- [18] S. O. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Mar. 1997. [Online]. Available: <https://www.rfc-editor.org/info/rfc2119>
- [19] Platform Engineering, "Google Cloud unveils Agent2Agent protocol: A new standard for AI agent interoperability," <https://platformengineering.com/editorial-calendar/best-of-2025/google-cloud-unveils-agent2agent-protocol>, Apr. 2025, accessed: 2026-04-28.
- [20] Galileo, "Google's Agent2Agent (A2A) protocol explained," <https://galileo.ai/blog/google-agent2agent-a2a-protocol-guide>, 2025, accessed: 2026-04-28.
- [21] Anthropic, "Introducing the Model Context Protocol," <https://www.anthropic.com/news/model-context-protocol>, Nov. 2024, accessed: 2026-04-28.
- [22] JSON-RPC Working Group, "JSON-RPC 2.0 specification," <https://www.jsonrpc.org/specification>, 2010, accessed: 2026-05-01.
- [23] Anthropic, "Model context protocol specification," <https://modelcontextprotocol.io/specification/2025-11-25>, 2025, revision 2025-11-25. Accessed: 2026-05-01.
- [24] InfoQ, "Anthropic publishes model context protocol specification for LLM app integration," <https://www.infoq.com/news/2024/12/anthropic-model-context-protocol/>, Dec. 2024, accessed: 2026-04-28.
- [25] Techcrunch, "Openai adopts anthropic's standard for connecting ai models to data," <https://techcrunch.com/2025/03/26/openai-adopts-rival-anthropics-standard-for-connecting-ai-models-to-data/>, Nov. 2025, accessed: 2026-04-28.
- [26] —, "Google to embrace anthropic's standard for connecting ai models to data," <https://techcrunch.com/2025/04/09/google-says-it-l-embrace-anthropics-standard-for-connecting-ai-models-to-data/>, Apr. 2025, accessed: 2026-04-28.
- [27] Wikipedia contributors, "Model context protocol — Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Model_Context_Protocol, 2026, accessed: 2026-04-28.
- [28] B. Radosevich and J. Halloran, "Mcp safety audit: LLMs with the model context protocol allow major security exploits," *ArXiv*, vol. abs/2504.03767, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:277621603>
- [29] N. Maloyan and D. Namiot, "Breaking the protocol: Security analysis of the model context protocol specification and prompt injection vulnerabilities in tool-integrated llm agents," *ArXiv*, vol. abs/2601.17549, 2026. [Online]. Available: <https://api.semanticscholar.org/CorpusID:285051495>
- [30] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (mcp): Landscape, security threats, and future research directions," *ACM Trans. Softw. Eng. Methodol.*, Feb. 2026, just Accepted. [Online]. Available: <https://doi.org/10.1145/3796519>
- [31] M. A. Ferrag, N. Tihanyi, D. Hamouda, L. A. Maglaras, and M. Debbah, "From prompt injections to protocol exploits: Threats in llm-powered ai agents workflows," *ArXiv*, vol. abs/2506.23260, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:280011490>
- [32] Z.-H. Deng, J. Gui, and W. Zhang, "From secure agentic ai to secure agentic web: Challenges, threats, and future directions," *ArXiv*, vol. abs/2603.01564, 2026. [Online]. Available: <https://api.semanticscholar.org/CorpusID:286223842>
- [33] Z. Anbiaee, M. Rabbani, M. Mirani, G. Piya, I. V. Opushnyev, A. A. Ghorbani, and S. Dadkhah, "Security threat modeling for emerging ai-agent protocols: A comparative analysis of mcp, a2a, agora, and anp," *ArXiv*, vol. abs/2602.11327, 2026. [Online]. Available: <https://api.semanticscholar.org/CorpusID:285540552>
- [34] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 669–684. [Online]. Available: <https://doi.org/10.1145/3319535.3354263>
- [35] S. R. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "Lteinspector: A systematic approach for adversarial testing of 4g LTE," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02A-3_Hussain_paper.pdf
- [36] M. M. Rahman, I. Karim, and E. Bertino, "CellularLint: A systematic approach to identify inconsistent behavior in cellular network specifications," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5215–5232. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/rahman>
- [37] M. L. Pacheco, M. v. Hippel, B. Weintraub, D. Goldwasser, and C. Nita-Rotaru, "Automated attack synthesis by extracting finite state machines from protocol specification documents," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 51–68.
- [38] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of oauth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1204–1215. [Online]. Available: <https://doi.org/10.1145/2976749.2978385>

- [39] J. de Ruiter and E. Poll, "Protocol state fuzzing of TLS implementations," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 193–206. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter>
- [40] P. Fiterau-Brosteau, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS implementations using protocol state fuzzing," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2523–2540. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brosteau>
- [41] Q. Shi, X. Xu, and X. Zhang, "Extracting protocol format as state machine via controlled static loop analysis," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 7019–7036. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity23/presentation/shi-qingkai>
- [42] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Found. Trends Priv. Secur.*, vol. 1, no. 1–2, p. 1–135, Oct. 2016. [Online]. Available: <https://doi.org/10.1561/33000000004>
- [43] C. J. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Proceedings of the 20th International Conference on Computer Aided Verification*, ser. CAV '08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 414–418. [Online]. Available: https://doi.org/10.1007/978-3-540-70545-1_38
- [44] X. Che, Y. He, X. Feng, K. Sun, K. Xu, and Q. Li, "Blueswat: A lightweight state-aware security framework for bluetooth low energy," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2087–2101. [Online]. Available: <https://doi.org/10.1145/3658644.3670397>

Appendix A.

A2A FSM

Figure 9 represents the extracted FSM.

