Beyond the Mandate: A Systematic Security Analysis of the Agent Payments Protocol (AP2)

Avital Aviv*
avitalos6@gmail.com
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Parth A. Gandh*
gandhip@post.bgu.ac.il
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Ron Bitton
ron_bitton@intuit.com
Intuit
Petah Tikva, Israel

Asaf Shabtai
shabtaia@bgu.ac.il
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Abstract

The Agent Payments Protocol (AP2), introduced by Google, enables large language model (LLM)-driven shopping agents to authorize and execute payments on behalf of users. Its signed Checkout and Payment Mandates protect the integrity of transaction data after signing. Agent interactions and external inputs that shape a transaction before authorization remain outside that protection, including Agent-to-Agent Protocol (A2A) messages and Model Context Protocol (MCP) tool calls. Prior work identified replay and prompt-injection attacks in AP2 v0.1. AP2 v0.2 addresses some of these issues but adds capabilities and deployment assumptions that require renewed analysis. We present a systematic security analysis of AP2 v0.2 based on its roles, transaction lifecycle, deployment architectures, and trust boundaries. We divide the lifecycle into five phases and identify five deployment architectures. Using MAESTRO (Multi-Agent Environment, Security, Threat, Risk, Outcome), we model four threat actors, eleven attack surfaces, eighteen adversary capabilities, and six attacker goals. The resulting catalog contains 48 threats spanning five attack families. We score these threats with the Artificial Intelligence Vulnerability Scoring System (AIVSS), identifying eight that reach the High band in at least one architecture. Because no complete public AP2 deployment was available, we build a testbed spanning all five architectures and develop five proof-of-concept demonstrations covering all eight High-risk threats and their mitigations. We also develop a deployment-aware scanner that maps applicable threats to static, cross-role consistency, and adversarial checks. Our analysis shows that valid mandate signatures alone do not ensure that an agent-mediated transaction reflects the user's intent when its pre-authorization context is manipulated.

1 Introduction

Large language model (LLM)-driven agents are beginning to act on users' behalf in the financial sector to browse catalogs, assemble carts and execute payments without the need for a human to approve each step. Google published the Agent Payments Protocol (AP2) to provide a secure framework for agent-led payments [8]. AP2 v0.1, released in 2025, introduced the protocol's roles and a chain of signed mandates built on Verifiable Digital Credentials (VDCs), focusing mainly on Human-Present (HP) flows in which the user remains in session and explicitly authorizes each closed mandate through a Trusted Surface [8]. AP2 v0.2, released in April 2026, adds Human-Not-Present (HNP) flows for autonomous transactions and defenses against replay attacks [8].

Because AP2 uses LLM-driven agents in protocol roles, it faces threats that classical payment protocols never encounter, such as prompt injection, tool-result poisoning and agent manipulation threats. The specification treats all LLM-driven agentic roles as potential attackers [8]. AP2 mainly secures the signed mandates and receipts; the pre-signing context that shapes them, including catalog data, tool results and A2A messages, remains outside the signed mandates. Since AP2 also leaves deployment choices to implementers, including whether Model Context Protocol (MCP) is used and whether tool infrastructure is shared, the practical threat surface depends on the architecture. Consequently, an attacker may not need to forge a mandate or receipt to cause harm; instead, the attacker may manipulate the agent's context, tools or communications so that a validly signed mandate encodes an unsafe or unintended action. These risks connect AP2's agentic attack surface with its protocol-level security assumptions and show the need for a threat model that covers both signed artifacts and the unsigned context that produces them.

Prior research has addressed individual components of AP2 security, but none provides a systematic threat model for AP2. MCP security research has identified tool poisoning and shadowing attacks [10, 21], and Louck et al. examined the security risks in the A2A protocol [18], yet none of these studies examined whether these threats propagate into AP2 itself. Two other studies examine AP2 v0.1 directly, red-teaming prompt injection in an AP2 shopping agent [6] and characterizing replay and context-binding failures [16], while Mao et al. systematize attack vectors across seven agentic commerce protocols including AP2 [20]. None of these studies analyzed threats across AP2's phases, trust boundaries, and deployment architectures. This paper presents the first threat model that spans AP2 v0.2's phases, trust boundaries, and deployment architectures, validating the high-risk threats on a testbed and providing a security scanner for implementers.

We map AP2's protocol design to concrete security risks. We segment the AP2 transaction lifecycle into five analytical phases and identify five deployment architectures, then use MAESTRO [13] to enumerate threats across actors, attack surfaces, capabilities, and goals. We subsequently map each threat to the architectures. This yields 48 threats grouped five architectures and attack families and assessed with the AIVSS [24] risk-scoring mechanism.

Because no public AP2 deployment existed at the time of our evaluation, we built a custom testbed and developed proof-of-concept attacks against all high-risk threats. This paper presents five of these attacks in detail. A single attack scenario can chain several threats, so the five demonstrations together cover all eight high-risk threats and their mitigations.

*Both authors contributed equally to this research.

We also develop a security scanner that derives a deployment profile, executes the applicable static, cross-role, and adversarial checks, and reports detected AP2 threats. We evaluate the scanner on the same controlled testbed. Our evaluation covers the threat catalog, the AIVSS assessment, and the scanner. A structured STRIDE-GPT comparison tests the catalog's coverage against a separate threat-modeling method. An eight-rater study tests whether independent experts reproduce the derived AIVSS severity bands and finally we perform a layer-ablation study of the scanner.

In summary, this paper makes five contributions. **(1)** We segment the AP2 transaction lifecycle into five analytical phases and identify five deployment architecture classes, treating a deployment as a distinct class when it exposes at least one threat not represented by an already-defined class. **(2)** We construct the first phase- and architecture-spanning threat model for AP2 using the MAESTRO seven-layer framework, spanning four threat actors, eleven attack surfaces, fourteen access capabilities in four families, four knowledge capabilities, six attacker goals, and the five architectures. **(3)** We catalog 48 AP2 threats and organize them into five attack families based on the AP2 security object each family corrupts, and apply AIVSS separately to each applicable architecture, identifying 8 of the 48 threats as high risk in at least one architecture. **(4)** We develop PoC attacks against all high-risk threats, demonstrating that these threats are realizable along with presenting mitigations for the same. **(5)** We implement a security scanner that profiles AP2 deployments, selects applicable threat checks, and reports detected findings, and evaluate it on our controlled AP2 testbed.

2 Related Work

2.1 AP2 and Agentic Commerce Security

Prior work identifies broad risks in autonomous AI systems [23, 25]. In AP2, these risks threaten mandate integrity, user authorization, and transaction accountability [8].

Two studies analyzed AP2 v0.1. Debi et al. showed that prompt injection through malicious content in an AP2-style shopping agent can manipulate rankings and leak user data despite signed mandates [6]. At the execution layer, Lan et al. identified replay and context-binding flaws that allow mandates to be reused or rebound without detection [16]. These studies expose failures both above and below the cryptographic layer.

Other work examines AP2 within agentic commerce more broadly. Acharya investigated trustless agent payments using decentralized identifiers, verifiable credentials, and zero-knowledge proofs, concluding that protocol-level trust is possible without a central payment intermediary [1]. Mao et al. mapped recurring attack classes across seven agentic commerce protocols, including AP2 [20]. Hu and Rong found that no protocol fully ensures both authorization integrity and settlement accountability [11].
However, the newly released AP2 v0.2 remains largely unexplored. Existing work does not systematically identify its threats, affected deployment architectures, or architecture-dependent impacts.

2.2 Security of Supporting Agent Protocols

AP2 relies on A2A for inter-agent messaging and discovery and on MCP for tool invocation. Because agents consume information from both protocols during transactions, vulnerabilities in either can propagate into AP2.

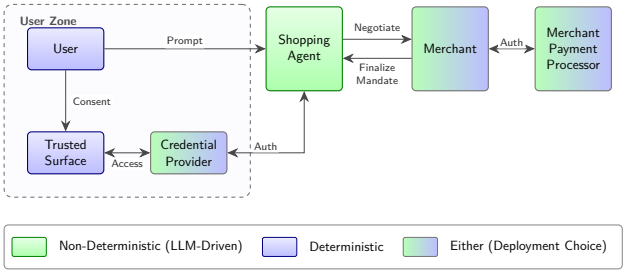

Figure 1: The five AP2 protocol roles and the flow of mandates and receipts among them.

MCP research identifies tool poisoning, tool shadowing, rug pulls, weak capability attestation, sampling risks, and client exposure to malicious tool metadata [10, 12, 14, 19, 21]. Louck et al. analyzed authentication, authorization, integrity, confidentiality, and availability in A2A, ACP, and CORAL [18]. However, no study has examined how these threats propagate into AP2 or provided a comprehensive AP2 threat model supported by proof-of-concept demonstrations and mitigations.

This paper addresses that gap and consolidates its findings into a security scanner for AP2 implementers.

3 AP2 Transaction Lifecycle & Architectures

3.1 AP2 Transaction Lifecycle

AP2 secures AI-agent payments with non-repudiable cryptographic proof of user authorization. Its Human-Present (HP) and Human-Not-Present (HNP) modes differ in when authorization occurs and who signs the mandates. This section presents the roles, flows, and five analytical lifecycle phases.

3.1.1 Roles. AP2 specifies five protocol roles [8]:

- **Shopping Agent (SA).** Performs product discovery, cart assembly, mandate construction, and payment execution on behalf of the user.
- **Merchant (M).** Provides the merchant-signed checkout (`checkout_jwt`), verifies the Checkout Mandate, and completes the checkout. When the merchant side is implemented by an LLM-driven agent, we refer to it as the Merchant Agent (MA).
- **Credential Provider (CP).** Verifies the Payment Mandate and issues a scoped payment credential.
- **Merchant Payment Processor (MPP).** Verifies that the payment credential is appropriately scoped to the checkout, processes settlement, and returns a Payment Receipt.
- **Trusted Surface (TS).** Renders mandate content to the user, obtains informed consent, and initiates mandate signing.

Figure 1 maps their interactions. A role is *agentic* when a non-deterministic LLM handles communication to or from it, making it a potential compromise point that requires tamper-evident defenses. We exclude payment-network and issuer processing, which AP2 does not define or change.

3.1.2 Mandates. Mandates are cryptographically signed, tamper-evident AP2 authorization artifacts implemented using Selective Disclosure for JSON Web Tokens (SD-JWTs). AP2 defines Checkout and Payment Mandates, each with open and closed forms [8].

The **Checkout Mandate** proves to the Merchant that the user authorized a purchase. Its open form captures the user's constraints and goals before a cart is finalized and is used in HNP mode. Its closed form authorizes a specific finalized checkout and includes a `checkout_hash` that binds it permanently to that cart.

The **Payment Mandate** proves to the CP, MPP, and any involved card networks that the user authorized payment for a checkout. Its open form records payment constraints for HNP execution; its closed form authorizes a specific amount. The Payment Mandate's `transaction_id` and the Closed Checkout Mandate's `checkout_hash` are both hashes of `checkout_jwt`. This common digest binds both mandates to the merchant-signed checkout, so applying a Payment Mandate to another checkout fails verification. In HNP flows, a mandate chain also binds each closed mandate to its parent open mandate.

3.1.3 Receipts. The Merchant returns a Checkout Receipt, and the payment-processing side returns a Payment Receipt. Each records acceptance or an error and is bound to its closed mandate. The mandates and receipts form the transaction evidence trail.

3.1.4 HP Flow. In HP mode, the user is present throughout the transaction and directly approves both closed mandates. The flow proceeds as follows [8].

(1) The user initiates a session with the SA.
(2) The SA communicates with the Merchant and assembles a cart.
(3) Upon checkout, the Merchant creates and signs `checkout_jwt`. The hash of this signed checkout is included in the closed Checkout and Payment Mandates.
(4) The SA retrieves the available payment options from the CP and selects one.
(5) The SA constructs the Checkout and Payment Mandate content and requests approval via the TS.
(6) The TS renders the content to the user and obtains consent. The resulting mandates are signed under the applicable AP2 authorization model, using either a User Credential or a key held by a trusted Agent Provider. The `checkout_hash` in the Checkout Mandate and the `transaction_id` in the Payment Mandate carry the same checkout digest.
(7) The SA passes the signed Payment Mandate to the CP, which verifies it and issues a payment token.
(8) The SA presents the Checkout Mandate and token to the Merchant, which verifies the checkout against what it originally signed and initiates payment via the MPP.
(9) On completion, a Checkout Receipt is provided to the SA, and a Payment Receipt is provided to the SA, CP, and any involved network.

3.1.5 HNP Flow. HNP mode differs from the HP flow in two ways. First, the user authorizes the SA through open mandates before leaving the session. Second, the SA signs the corresponding closed mandates autonomously with its own key rather than routing them through the TS. The flow proceeds as follows [8].

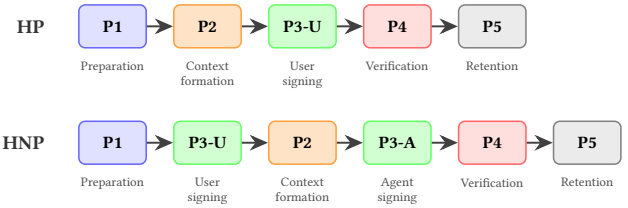

Figure 2: Phase ordering in the two AP2 flows (Table 1). Phases are analytical stages, not a fixed chronology.

(1) The SA assembles open Checkout and Payment Mandate content for the shopping session and requests approval via the TS.
(2) The TS renders the open mandate content and obtains the user's authorization. The resulting open mandates are signed under the applicable AP2 authorization model. Each open mandate includes the agent's public key (`agent_pk`) as a confirmation claim, constraining the mandate to that SA.
(3) The user leaves the session.
(4) The SA communicates with the Merchant and assembles a cart. The Merchant signs `checkout_jwt` as in HP.
(5) The SA selects the open mandates whose constraints cover the assembled checkout, constructs the closed Checkout and Payment Mandates, and signs them with `agent_sk`. The `checkout_hash` and `transaction_id` claims carry the same checkout digest, while the mandate chain binds each closed mandate to its parent open mandate.
(6) The SA presents the open and closed Payment Mandates to the CP and the corresponding Checkout Mandates to the Merchant. Each verifier validates the relevant signatures, bindings, and open mandate constraints. Credential issuance, settlement, and receipt generation then proceed as in HP.

3.1.6 Phase Segmentation. The AP2 v0.2 distinctions between HP (Direct) and HNP (Autonomous) modes and between open and closed mandates are too coarse for threat modeling [8]. We therefore divide a transaction into five phases based on how mandates are prepared, formed, authorized, consumed, and retained. P1 establishes the identities, keys, endpoints, credentials, and trust anchors needed for mandate operations. P2 assembles the cart, payment options, and merchant-signed checkout. P3 converts intent and constraints into signed mandates. P4 verifies and consumes them to authorize payment and settlement. P5 retains mandates, checkout records, and receipts for audits and disputes. These phases are analytical stages, not a chronology shared by both modes. P3-U denotes user authorization through the TS. P3-A is the HNP-only stage in which the SA derives and signs closed mandates constrained by the user's open mandates. Figure 2 shows the order in each flow. HP is sequential; in HNP, open-mandate signing (P3-U) precedes cart assembly (P2). Table 1 maps each phase to both flows.

3.1.7 Security Requirements and Baseline Assumptions. Our threat model separates three baseline assumptions (BA1–BA3), which are outside our scope, from eight security requirements (SR1–SR8),

Table 1: Five analytical phases of the AP2 transaction lifecycle.

Phase	Lifecycle function	Flow mapping
P1 Pre-Mandate Preparation	Before transaction-specific mandates exist, identities, signing keys, credentials, AgentCards (A2A discovery documents listing agent identity, endpoints, and capabilities), endpoint bindings, and trust anchors are established for mandate creation and verification.	Precondition for HP and HNP.
P2 Mandate Context Formation	The cart, payment options, and merchant-signed checkout are assembled for closed mandate formation. In HNP, prior open mandates constrain this context.	HP steps 1–4; HNP step 4.
P3 Mandate Authorization	Unsigned intent or constraints → signed mandates. P3-U creates user-signed closed mandates in HP and user-signed open mandates in HNP; P3-A derives SA-signed HNP closed mandates constrained by their parents.	P3-U: HP steps 5–6 and HNP steps 1–2. HNP step 3 transitions from P3-U to P2. P3-A: HNP step 5.
P4 Mandate Verification & Consumption	Signed, unconsumed mandates → verified signatures, mandate chain, constraints, checkout binding, expiration, and prior-consumption status, followed by credential issuance, settlement, and receipts.	HP steps 7–9; HNP step 6.
P5 Post-Transaction Retention	After execution, completed mandates, checkout records, and receipts are integrity-protected and retained as audit and dispute evidence.	Postcondition for both flows, beginning after receipt generation.

whose violations we evaluate. For the TS, BA3 assumes only that it is not LLM-driven, whereas SR1 requires faithful mandate rendering and authorization capture.

Baseline Assumptions.

BA1. Standard cryptographic primitives are sound when used: signature schemes are unforgeable and hash functions are collision resistant. Parameter adequacy is a deployment property and remains in scope under T-38.

BA2. Implementations follow all AP2-defined requirements for cryptographic constructions and message formats; attackers therefore cannot forge signatures, substitute mandates, or bypass required bindings. This assumption covers only AP2-defined properties. We evaluate implementer-controlled properties, including salt generation, key lifetime, and optional-field integrity.

BA3. The TS is not LLM-driven.

Security Requirements.

SR1. Faithful TS rendering and authorization capture. The TS faithfully renders mandate content to the user and captures the user's authorization action.

SR2. Signer attribution. Each mandate is attributable to its signing key, and the verifier establishes that the key is authorized for the relevant role. A mandate created through the TS represents user authorization only if the verifier validates the applicable User Credential or trusts the Agent Provider responsible for the TS. In HNP, a closed mandate signed with `agent_sk` represents user authorization only if it is bound to the agent key named in a valid, user-authorized open mandate and satisfies that mandate's constraints.

SR3. Independent verifier checks. Each verifier independently performs the signature, binding, and constraint checks assigned to its protocol role rather than relying solely on another party's assertion.

SR4. Checkout and mandate-chain binding. The Closed Checkout Mandate carries the digest of `checkout_jwt` in `checkout_hash`; the Closed Payment Mandate carries the same digest in `transaction_id`. Verifiers confirm that both values bind the mandates to the same merchant-signed checkout, compute every required hash over the AP2-prescribed representation, and reject mismatches. In HNP, each closed mandate also commits to its parent open mandate.

SR5. HNP constraint enforcement. In HNP, verifiers enforce each open mandate's constraints when deciding whether a closed mandate signed with `agent_sk` remains within the user-approved limits. This requirement does not assume that the encoded constraints faithfully represent the user's intent.

SR6. Key confidentiality and configuration integrity. Signing keys remain confidential to authorized holders and bound to their permitted roles. Credentials, AgentCards, registry records, endpoint configurations, and other security information established during preparation remain authentic.

SR7. Replay prevention and single-use authorization. Verifiers reject expired or replayed closed mandates, each of which may authorize execution only once. An open mandate may authorize multiple closed mandates only when its recurrence and budget constraints permit reuse and all cumulative limits remain satisfied.

SR8. Evidence retention and later verification. Signed mandates, checkout records, and receipts needed for disputes remain available, attributable, and protected against tampering.

Relationship to the Lifecycle Phases. The lifecycle maps each requirement to changes in mandate state: key and credential preparation in P1, mandate formation and authorization in P2–P3, verification and consumption in P4, and evidence retention in P5. Failures of key binding, constraint enforcement, serialization, or hash computation are attributed to the phase containing the affected mandate operation and remain in scope.

3.2 AP2 Deployment Architectures

In the AP2-based deployments studied here, A2A carries shopping and checkout messages between agents, while MCP may expose their tools and external services [8]. Agentic and deterministic roles, A2A patterns, and MCP configurations yield many architectural variants. We focus on agentic roles to characterize threats from LLM-mediated behavior. We define a distinct architecture class when a deployment introduces a new cross-role trust boundary, a new shared state domain, a new tool-execution surface, or a compromise path that reaches a different set of AP2 roles. Each such class exposes at least one threat that no already-defined class

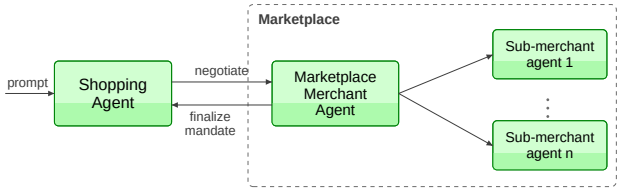

Figure 3: A4: Marketplace architecture.

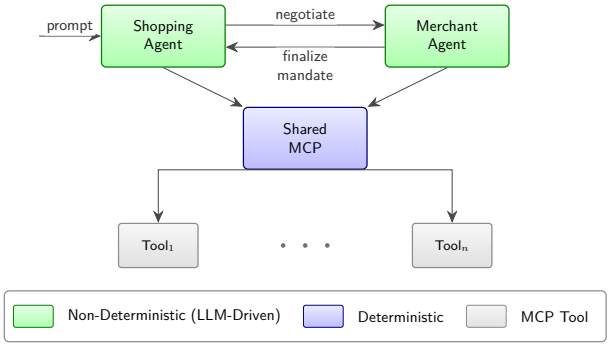

Figure 4: A5: Shared MCP architecture.

exposes. A configuration that only combines the surfaces of existing classes, and therefore exposes no threat beyond their union, is treated as a composition rather than a new class. Applying this rule yields five architecture classes.

A1: Single-Agent SA and Single-Agent MA. A1 is our baseline and matches Google's official HP samples [8]. One LLM-driven SA communicates with one LLM-driven MA over A2A; the CP and MPP are independent peers reachable through A2A or direct API calls. A1 has no MCP server, internal sub-agent delegation, or marketplace layer. Adversarial input reaches the SA through merchant-side A2A, user input, CP/MPP responses, and external discovery content. Merchant-side A2A is the primary transaction-formation channel because the SA uses it to assemble the cart and negotiate the checkout used in mandate construction. The other channels expand the attack surface without adding a trust boundary.

A2: Single-Agent SA and Single-Agent MA with Isolated MCP. A2 retains A1's single-agent structure but adds an isolated MCP server to each side. The servers share no state, cache, storage, credentials, or authorization context across the SA/MA boundary. The added tool-execution surface exposes MCP threats absent from A1 even though it adds no SA/MA boundary.

A3: Multi-Agent SA and Multi-Agent MA. A3 implements the SA and MA as specialized sub-agents. An SA may have monitoring, purchase, and consent sub-agents under an orchestrator, while an MA may have catalog, pricing, and checkout sub-agents. The external AP2 interface remains unchanged, with the SA communicating with the MA over A2A. A3 introduces vulnerabilities from malicious sub-agent propagation and orchestration failures. Adding isolated MCP yields A3×A2, a composition rather than a separate class because it exposes no threat beyond the union of A3 and A2.

A4: Marketplace. In A4, shown in Figure 3, the SA interacts with one marketplace identity that routes transactions to sub-merchants and mediates the AP2 lifecycle. During discovery and cart assembly, the SA uses one marketplace-facing agent instead of negotiating with each merchant as an independent AP2 counterparty. Behind that identity, the marketplace resolves the user's request internally by selecting one or more sub-merchants, catalog entries, fulfillment options, and fee rules.

Unlike A1–A3, A4 can separate the visible AP2 merchant identity from the sub-merchant supplying the item, price, fulfillment state, or dispute facts. A user may approve a marketplace-scoped mandate while hidden actors handle execution and evidence, creating an opaque merchant-side trust boundary. Using one or many internal agents changes compromise scale but not the class. Shared MCP produces an A4×A5 composition because it combines the marketplace and shared-MCP boundaries without exposing threats beyond their union.

A5: Shared MCP. A5, shown in Figure 4, retains the SA–MA A2A channel but gives both roles the same MCP server during discovery, cart assembly, and checkout preparation. Unlike architectures with no tools or isolated tool use, this server is an operational surface and influence point for both roles. The SA may search products, compare offers, or construct a candidate cart, while the MA may resolve inventory, pricing, merchant policies, cart state, or checkout records. AP2 messages still use A2A, but some message state and resulting artifacts derive from a shared tool layer that may produce product data, price quotes, cart identifiers, and tool results. A compromised or malicious server can therefore influence both transaction sides through common MCP attacks. Internal agent variations change the compromise footprint but not the class.

4 Threat Model

We characterize each threat as a tuple of four elements: the actor initiating it, the surface on which they act, the capability they wield, and their goal. Phase applicability is recorded per threat in Figure 6 rather than in the tuple, since several threats span phases (e.g., P2–P4).

Threats are identified using MAESTRO [13], whose seven layers cover foundation models (L1), data operations (L2), agent frameworks (L3), deployment infrastructure (L4), evaluation and observability (L5), security and compliance (L6), and the agent ecosystem (L7). The MAESTRO layers are tagged per surface in Section 4.3 and per threat in Figure 6. We assign L1 only when a threat depends on foundation-model behavior. We classify a threat that changes MCP or tool metadata as L7, L3, or L2, according to the affected object. Therefore, we assign T-2 to L7 because it changes MCP tool metadata. We assign T-1 to L1 because it exploits the model's response to poisoned context.

Notation Access capabilities use the prefix AC followed by a family letter. We use K for key access, M for mandate and token access, C for communication-channel access and R for runtime access. Knowledge capabilities use the prefix AK. Surfaces are labeled S1–S11, threat actors are labeled TA1–TA4, and attacker goals are labeled AG1–AG6. We refer to the five deployment architectures described in Section 3.2 as A1-A5 and to the five protocol phases described

in 3.1.6 as P1–P5. Security requirements are labeled SR1–SR8 and baseline assumptions BA1–BA3; both are defined in Section 3.1.7 and cited by number throughout.

4.1 Attacker Model

An attacker is characterized by their access capabilities, which describe what they can read, write, invoke or control, and knowledge capabilities, which describe what they know about a deployment. Access capabilities are required to realize a threat. Knowledge capabilities do not realize threats by themselves, but they reduce attack complexity or increase the probability of success. The capability codes used in the threat model are summarized in Table 2.

A compromised component gives the attacker the same permissions and access that component has. If several components are compromised, their permissions combine. In Architecture A5, both the SA and MA call the same MCP server, so compromising that server may let an attacker interfere with both agents' tool interactions in a single transaction. Therefore, threats must be analyzed by system architecture, not just by agent role.

4.2 Threat Actors

A threat actor may be an individual, a group, or an organization. Each actor is assigned a security posture: honest, honest-but-curious, or malicious for TA1–TA3, and malicious by assumption for TA4. Honest postures are included so the actor set also identifies principals appearing as victims, counterparties, or compromised surfaces in a given threat. We derive four threat actors by grouping the AP2 roles defined in Section 3.1.1, along with the User principal, according to their adversarial objectives.

[TA1] User. The customer who initiates the transaction and authorizes consent at the TS. An honest-but-curious User probes to infer verifier policy or merchant-side state that was not intentionally exposed. A malicious User on the other hand repudiates consent, or tampers the credential store, or submits transaction inputs they later dispute. The User has AK4 for their own mandates and inherently holds ACK1, along with ACR2–ACR3 on a user-controlled SA host, but does not have build-pipeline access (ACR4).

[TA2] Merchant. The legal entity occupying the Merchant role, Merchant Agent, Merchant BE, or merchant MCP server. A malicious Merchant may overcharge, substitute items, manipulate catalog content, or return tool output that steers mandate construction away from the user's intent. The Merchant inherently holds ACK3, ACR2–ACR4 over its own infrastructure and release builds. In A4, the Merchant role may comprise multiple merchant-side principals, including a marketplace operator and one or more sub-merchants. We model these as instances of the same actor class, TA2, rather than as separate actor labels. When the distinction matters, we identify the responsible principal from the affected S4 backend: a marketplace-front backend points to the marketplace operator, a sub-merchant backend to a sub-merchant, and a shared backend to shared merchant infrastructure.

[TA3] Developer / Operator. Parties that implement, deploy, or operate AP2 surfaces, including SA providers, TS vendors, CP and MPP operators, and registry operators. A malicious operator may ship a backdoored release, weaken verifier logic, exfiltrate keys, or route traffic through an unintended dependency. Per operated surface, this actor holds the relevant key, runtime, release and configuration capabilities.

[TA4] External Attacker. Any principal outside the named transaction parties and operators, including a network attacker, supply-chain attacker, on-host malware operator, or third-party MCP provider outside the deployment trust boundary. Its capabilities come from reachable or compromised surfaces, not from a default AP2 role.

Modeling choices. The model assigns intent to threat actors, not to surfaces. We therefore model the TS, SA, MA, CP, MPP, and AgentCard registry as surfaces (S1–S7), not as threat actors. We attribute malicious behavior to the actor that operates or compromises the surface (TA1–TA4, as applicable). When actors collude, we combine their capabilities and assign their shared goal. We do not define a separate actor class for collusion.

4.3 Attack Surfaces

An attack surface is a point at which a threat actor's capabilities can affect AP2 security. We identify 11 AP2 attack surfaces and organize them into three categories: role and runtime surfaces (S1–S7), communication-channel surfaces (S8–S9), and data and artifact surfaces (S10–S11). These surfaces are described in Table 3.

One surface, the S9 MCP channel, varies across architectures: it is absent in A1, pure A3, and pure A4; isolated per side in A2 and A3×A2; and shared in A5 and A4×A5. We nonetheless keep it as a single surface, so the deployment architecture determines whether an S9 threat applies and whether its scope is confined to a single side or spans both. The tuple's architecture element records the applicable case.

4.4 Attacker Goals

We summarize the attacker objectives as six goals. Each goal is realized by one or more threats. Targets denotes the security properties whose violation can realize the goal; SR3 (independent verification) is a latent target for any goal requiring a malformed or out-of-scope artifact to pass verification; we list it explicitly only where it is the primary failure.

[AG1] Unauthorized Payment. Cause settlement with no valid corresponding authorization (forged, absent, or invalid), such as a payment the user never approved or approved for a different amount, merchant, or instrument. Targets SR2, SR4, and SR6.

[AG2] Mandate-Content Manipulation. Obtain a valid signature over content whose effective semantics differ from the user's intent, rendered view, or pre-authorized constraints, such as a hidden charge, substituted item, or relaxed allowed_merchants list; the manipulation occurs before or during authorization, including mandate construction, rendering, constraint encoding, or signing mediation. Targets SR1, SR4, and SR5.

[AG3] Authorization-Scope Inflation. Extend a legitimately constructed authorization beyond its scope after signing (replayed, reused, or rebound). Targets SR3, SR4, SR5 and SR7.

[AG4] Confidentiality Breach. Recover selectively disclosed claims, observe payment metadata, or link transactions across the dispute-evidence retention window. Targets the confidentiality of S10 and communication-channel metadata on S8/S9.

Table 2: Summary of the capability codes used in the AP2 threat model. `I` means inherent to the actor in normal operation, `C` means obtainable through compromise or reachability, and – means not typically held.

Code	Meaning	TA1	TA2	TA3	TA4
Keys					
ACK1	Control of `user_sk`; can produce user-signed mandates.	I	-	C	C
ACK2	Control of `agent_sk`; can produce HNP closed mandates.	C	-	C	C
ACK3	Control of `merchant_sk`; can sign checkout artifacts and receipts.	-	I	C	C
ACK4	Control of CP/MPP verifier keys; can forge payment tokens or receipts.	-	-	I	C
Mandates & Tokens					
ACM1	Access to public mandate, token, or receipt claims.	I	I	I	C
ACM2	Access to selectively disclosed mandate or receipt claims.	I	I	I	C
ACM3	Access to a scoped payment token in flight.	-	C	C	C
Channels					
ACC1	Passive observation of A2A or MCP traffic.	-	C	C	I[b]
ACC2	Active modification of A2A or MCP messages in transit.	-	C	C	C
Runtime					
ACR1	RPC-level access to an exposed AP2 role or tool interface.	I	I	I	C
ACR2	Process-owner access to memory, local state, or verifier logic.	I[a]	I	I	C
ACR3	Host-level control of the target role or infrastructure.	I[a]	I	I	C
ACR4	Build-pipeline or release influence on AP2 software.	-	I	I	C
ACR5	MCP-server-operator access; control over tool descriptions or tool results.[c]	-	C	I	C
Knowledge					
AK1	Knowledge of deployed architecture, agentic roles, and shared infrastructure.	C	C	I	C
AK2	Knowledge of exposed tools, capabilities, and MCP resources.	-	I	I	C
AK3	Knowledge of the LLM model, version, or agent framework.	-	I	I	C
AK4	Knowledge of mandate contents, constraints, or disclosed claims.	I	I	I	C

[a] TA1 inherency for ACR2/ACR3 assumes a user-controlled SA host; under hosted-agent deployments these become C and the corresponding runtime capability shifts to TA3. [b] TA4's inherent ACC1 reflects an assumed on-path network vantage; absent that vantage it is C. [c] ACR5's blast radius is architecture-conditional: in A2 and A3×A2 it influences one role's tool calls; in A5 and A4×A5 (shared server) it influences both.

Table 3: Systemic AP2 attack surfaces (S_1–S_{11}).

Surface	Core role	Risk / failure mode	MAESTRO
Role and Runtime Surfaces			
S1: TS	UI surface trusted to render mandate content and obtain user consent before signing.	Mandate mismatch, consent bypass, or local key exposure (`user_sk`, ACR2–4).	L4, L6
S2: SA	SA for cart assembly, mandate construction, and payment execution.	Prompt injection, jailbreaks, or tool poisoning altering mandate generation fields.	L1–L4, L6, L7
S3: MA	LLM-driven MA interacting with the SA over A2A.	Catalog/quote manipulation or tool tampering; signing compromise maps to ACK3.	L1–L4, L6, L7
S4: Merchant BE	Deterministic merchant backend system.	`merchant_sk` exposure, checkout state alteration, or marketplace isolation failure (A4: marketplace-front vs. sub-merchant backend instance).	L4, L6
S5: CP	Scoped credential release authority.	Mandate check bypass, token leakage, or agentic prompt injection.	L4, L6 (Agentic: L1–L3, L7)
S6: MPP	Settlement verification and receipt signer.	Settlement/token corruption, faulty receipt generation, or audit tampering.	L4, L6 (Agentic: L1–L3, L7)
S7: Registry	Identity and discovery infrastructure (AgentCard).	Impostor routing, extension suppression, or discovery provenance manipulation.	L4, L7
Communication-Channel Surfaces			
S8: A2A Channel	Inter-agent messaging substrate.	Protocol Downgrade, endpoint hijacking, or conversational state replays.	L4, L6, L7
S9: MCP Channel	Tool execution interface.	Tool poisoning, result tampering, or cross-role state bleeding (shared server A5).	L2–L4, L6, L7
Data and Artifact Surfaces			
S10: Artifacts	Signed credentials and evidence (SD-JWTs, tokens, receipts, audits).	Information leakage, disclosure confusion, stale replay, or token capture.	L2, L5, L6
S11: Knowledge	External data, catalog caches, and retrieval databases.	Ingestion/retrieval poisoning, cross-tenant bleeding, or corpus modification.	L1, L2, L6

[AG5] Repudiation. Deny a legitimately authorized transaction through weaknesses in evidence retention or mandate-chain auditability. Targets SR2 and evidence retention in P5.

[AG6] Denial of Authorized Payment. Prevent an authorized transaction from being completed, for example, by misrouting discovery, or stalling MCP tool calls. Targets availability of the authorized transaction across P2–P4 (SR3/SR4 when caused by verifier or binding failure).

Table 4: Mapping from AP2 attacker goals to CVSS impact dimensions.

Goal	CVSS impact interpretation
AG1 Unauthorized payment	Integrity impact on the AP2 transaction and subsequent payment system.
AG2 Mandate-content manipulation	Integrity impact on mandate semantics and signed user intent.
AG3 Authorization-scope inflation	Integrity impact on the scope or reuse of authorization.
AG4 Confidentiality breach	Confidentiality impact on disclosed claims, payment metadata, or linkage data.
AG5 Repudiation	Integrity impact on evidence, receipts, and dispute artifacts.
AG6 Denial of authorized payment	Availability impact on completion of an authorized transaction.

4.5 Risk Assessment

Risk is assessed separately for each deployment architecture. Each threat uses the tuple ⟨actor, surface, capability, architecture, goal⟩ defined above and applies only when the architecture exposes the required surface and its preconditions hold. We score inherent risk using AIVSS v0.5 [24], which combines a CVSS v4.0 base score [7] with Agentic AI Risk Score (AARS) amplification factors. We assume PoC threat maturity and apply no mitigation discount, ranking inherent risk before deployment-level controls. A threat is high if its AIVSS score reaches the high-severity band in at least one architecture. The complete per-threat breakdown for all 48 threats along with their full AIVSS score table and the full threat matrix is provided in the companion repository.[1]

4.5.1 Scoring Method. We score each applicable threat in two layers. First, we assign a CVSS v4.0 base score [7] from the protocol-level attacker path. Exploitability is derived from the minimum required access capability, the number of prerequisite compromises, and whether the attack requires a timing window, privileged role, or ordinary AP2 transaction flow. Impact is derived from the AP2 attacker goal mapped to CVSS dimensions as shown in Table 4.

Second, we assign the ten AARS amplification factors on a three-level scale. A value of 0 means the factor is not applicable to the scored threat path, 0.5 means it is partially present or deployment-dependent, and 1 means it is fully present. The factors are autonomy of action, tool use, memory use, dynamic identity, multi-agent interaction, non-determinism, self-modification, goal-driven planning, contextual awareness, and opacity/reflexivity, as defined by AIVSS [24]. AARS is used only as the agentic amplification component inside AIVSS; it is not treated as a separate severity score.

We apply two guardrails when assigning these factors. We reserve non-determinism for attacks in which an LLM decision point contributes to exploit success. We reserve self-modification for cases where the agent can alter its own code, prompt, tool configuration, model configuration, or policy at runtime. Deterministic protocol downgrades, signing-key compromise, registry errors, and historical evidence failures therefore do not receive these factors by default.

The AARS amplification sum is reported on a 0–10 scale:

$$\text{AARS} = \sum_{i=1}^{10} f_i, \qquad f_i \in \{0, 0.5, 1\}.$$

The AIVSS score combines the CVSS v4.0 base score with the AARS amplification sum, scaled by a threat-maturity multiplier ThM and a mitigation factor MF:

$$\text{AIVSS} = \text{round}\left(\frac{\text{CVSS}_{base} + \text{AARS}}{2} \times \text{ThM} \times \text{MF},\ 1\right).$$

Here $\text{CVSS}_{base} \in [0, 10]$ is the protocol-level CVSS v4.0 base score, AARS is the factor sum above, $\text{ThM} \in (0, 1]$ captures observed threat maturity, and $\text{MF} \in (0, 1]$ captures residual exposure after deployed mitigations. The arithmetic mean is used because both CVSS and AARS are on a 0–10 scale; averaging keeps AIVSS bounded within the same range while weighting protocol-level severity and agentic amplification equally. The round$(\cdot, 1)$ operator denotes round-half-up to one decimal place.

Following the AIVSS documentation [24], we use ThM = 0.97 because this assessment evaluates PoCs rather than exploits observed in production deployments. To rank inherent risk before deployment-level controls are applied, we assume the lack of mitigations and set MF = 1.0. Substituting these values gives the formula used for every scored row:

$$\text{AIVSS} = \text{round}\left(\frac{\text{CVSS}_{base} + \text{AARS}}{2} \times 0.97 \times 1.0,\ 1\right).$$

For example, T-1 (pre-signing context poisoning) has the CVSS vector `CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:H/VA:N/SC:N/SI:H/SA:N`, which gives $\text{CVSS}_{base} = 8.3$. Its AARS factor tuple is $(0.5, 0.5, 0.5, 0.5, 0.5, 1, 0, 1, 1, 1)$, so AARS = 6.5. Therefore,

$$\text{AIVSS} = \text{round}\left(\frac{8.3 + 6.5}{2} \times 0.97 \times 1.0,\ 1\right) = 7.2,$$

placing the threat in the High band.

Rows are labeled using the standard severity bands: None for 0.0, Low for 0.1–3.9, Medium for 4.0–6.9, High for 7.0–8.9, and Critical for 9.0–10.0. Figure 5 summarizes the 8 threats that reach the High band, showing each threat's CVSS base score, AARS factor profile, and final AIVSS score.

4.5.2 Architecture-specific amplification. Consider T-31 (mandate replay), which is applicable in both the Single-Agent architecture (A1) and the Multi-Agent architecture (A3). In A1, one agent can sequentially reuse a valid mandate. With the factor order used above, this path has the AARS tuple $(0.5, 0.5, 1, 0, 0, 0, 0, 0.5, 0.5, 1)$, so AARS = 4.0. Holding its $\text{CVSS}_{base} = 8.9$ constant gives AIVSS = 6.3. In A3, the same authorization can be fanned out concurrently across several agents. This activates the multi-agent-interaction factor while the other factors remain unchanged, producing the tuple $(0.5, 0.5, 1, 0, 1, 0, 0, 0.5, 0.5, 1)$, AARS = 5.0, and AIVSS = 6.7.

[1] https://anonymous.4open.science/r/AP2_Beyond_the_Mandate

ID	Threat	CVSS v4.0 vector	Aut	Tool	Mem	ID	Multi	ND	Self	Plan	Ctx	Opaque	CVSS	AARS	AIVSS
F1: Semantic Manipulation															
T-1	Pre-signing Context Poisoning	AV:N AC:L AT:P PR:N UI:P VC:L VI:H VA:N SC:L SI:H SA:N	1	1	.5	0	1	1	0	1	1	1	7.0	7.5	**7.0**
T-4	Tool Argument Manipulation	AV:N AC:L AT:N PR:N UI:N VC:N VI:H VA:N SC:N SI:H SA:N	1	1	0	.5	1	.5	0	.5	1	.5	9.2	6.0	**7.4**
F2: Authority Spoofing															
T-15	Caller Role Confusion	AV:N AC:L AT:N PR:L UI:N VC:H VI:H VA:L SC:L SI:H SA:N	.5	1	1	1	1	.5	0	.5	1	.5	8.6	7.0	**7.6**
F3: Supply Chain and Trust Root Subversion															
T-23	Sub-Agent Prompt Drift	AV:N AC:L AT:P PR:H UI:N VC:H VI:H VA:N SC:H SI:H SA:N	1	0	.5	.5	1	1	0	1	1	1	8.7	7.0	**7.6**
T-27	A2A Extension Downgrade	AV:N AC:H AT:P PR:N UI:N VC:H VI:H VA:N SC:N SI:H SA:N	1	1	0	0	1	0	0	.5	.5	1	9.4	5.0	**7.0**
T-29	Cross-Server Tool Confusion	AV:N AC:L AT:P PR:N UI:N VC:H VI:H VA:N SC:N SI:H SA:N	1	1	0	0	1	.5	0	0	1	1	9.4	5.5	**7.2**
F4: State-Binding Failures															
T-34	risk_data Poisoning	AV:N AC:L AT:P PR:L UI:N VC:L VI:H VA:L SC:L SI:H SA:L	1	1	.5	.5	1	.5	0	.5	1	1	8.3	7.0	**7.4**
F5: Accountability Failures															
T-42	Multi-Tenant Isolation Failure	AV:N AC:L AT:N PR:L UI:N VC:H VI:H VA:L SC:H SI:H SA:N	.5	.5	1	1	1	.5	0	.5	.5	.5	8.7	6.0	**7.1**

AARS factor abbreviations: Aut=Autonomy, Tool=Tool use, Mem=Memory, ID=Dynamic identity, Multi=Multi-agent interactions, ND=Non-determinism, Self=Self-modification, Plan=Goal-driven planning, Ctx=Contextual awareness, Opaque=Opacity/reflexivity. 0 absent .5 partial 1 strong

Figure 5: AIVSS risk scores for the 8 high-risk AP2 threats.

4.5.3 Threat Matrix reading protocol. The matrix in Figure 6 can be read in three ways. Reading within an attack-family band shows the common actor, surface, and capability profile for that family. Reading down an architecture column shows which threats become High under that deployment, where the Single-Agent baseline (A1) is dominated by F3 trust-root and F4 state-binding threats, while the Shared-MCP architecture (A5) amplifies the largest number of F1 threats. Reading across a single row reconstructs the threat tuple used to connect the threat model, the risk assessment, and the PoC descriptions.

5 Taxonomy of AP2 Attack Families

Using the threat tuple from Section 4.5, we devided the 48 MAESTRO threats into five attack families, each defined by the AP2 *security object* it corrupts. A security object represents the core logical and cryptographic mandates, such as signing authority and transaction state, vital to protocol security. Within each family we further divide the threats into sub-families, where a sub-family shares a common attacker goal and a common fix. Section 6 then focuses on the 8 threats that reach the High band by AIVSS.

5.1 Classification Methodology

We categorize the threats using a two-stage bottom-up methodology. First, we cluster the discovered threats bottom-up based on the specific AP2 security object they corrupt. These objects represent the core logical and cryptographic mandates, such as signing authority and transaction state, vital to protocol security. Second, we verify these clusters by ensuring that each resulting family shares a distinct AP2 failure mode, overlapping attacker goals, and a common fix. We then refine each family one level down into sub-families, naming each sub-family by its *mechanism* of failure rather than by its symptom. We rejected alternative taxonomy models because they organize threats by surface-level attributes, such as when the attack occurs, what the attacker ultimately wants, or which capability it uses, rather than by the protocol function that fails:

Phases merely track an operational timeline rather than exploit mechanics.

Goals are too broad, grouping unrelated vulnerabilities under a single objective.

Capabilities over-fragments the catalog, since identical protocol failures can be reached through entirely different vectors.

These dimensions remain essential for threat attribution and scoring (Sections 3.1 and 4); we reject them only as the basis for the family taxonomy, which is organized by corrupted security object.

This bottom-up process yields five distinct families, each mapped to core security objects: *Semantic Manipulation* (corrupting conversation context), *Authority Spoofing* (corrupting signing authority), *Supply-Chain & Trust-Root Subversion* (corrupting trust roots), *State-Binding Failures* (corrupting transaction state), and *Accountability Failures* (corrupting accountability records). If a threat corrupts more than one security object, we assign it to the object whose corruption constitutes the initial AP2-specific failure, the point at which the protocol's guarantees first break and from which the remaining corruptions follow. For instance, a poisoned tool result under shared MCP (T-1) simultaneously injects adversarial content into the receiving agent's context and exfiltrates transaction state from the originating one. We classify it under Semantic Manipulation rather than State-Binding or Accountability, because the contextual corruption is what first diverts mandate construction from user intent, whereas the state leakage is a downstream confidentiality effect. Figure 7 summarizes the resulting taxonomy of families and sub-families.

			Actor				Surface											Capability														Know.				Goal						Arch.				
ID Attack Technique	Layer	Phase	TA1 User	TA2 Merch	TA3 Dev	TA4 Ext	S1 TS	S2 SA	S3 MA	S4 MBE	S5 CP	S6 MPP	S7 Registry	S8 A2A	S9 MCP	S10 Artifacts	S11 Knowledge	ACK1	ACK2	ACK3	ACK4	ACM1	ACM2	ACM3	ACC1	ACC2	ACR1	ACR2	ACR3	ACR4	ACR5	AK1	AK2	AK3	AK4	AG1	AG2	AG3	AG4	AG5	AG6	A1 Single agent	A2 Single agent MCP	A3 Multi agent	A4 Marketplace	A5 SharedMCP
F1: Semantic Manipulation																																														
T-1 Pre-signing Context Poisoning	L1	P2–P3	-	•	-	•	-	•	•	-	-	-	-	-	•	-	•	-	-	-	-	-	-	-	-	•	-	-	-	-	•	-	-	•	•	-	•	-	-	-	-	•	•	•	•	+
T-4 Tool Argument Manipulation	L3	P3–P4	-	-	-	•	-	-	-	-	•	•	-	-	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	•	-	•	•	-	-	•	-	-	-	-	-	•	-	-	+
F2: Authority Spoofing																																														
T-15 Caller Role Confusion	L3	P4	-	-	-	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	•	-	•	-	-	-	-	-	-	-	-	-	•
F3: Supply-Chain & Trust-Root Subversion																																														
T-23 Sub-Agent Prompt Drift	L1	P1	-	-	•	-	-	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	•	-	-	-	•	-	-	•	-	-	-	-	-	-	•	-	-
T-27 A2A Extension Downgrade	L7	P2	-	•	-	•	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-	•	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	•	•	•	•	•
T-29 Cross-Server Tool Confusion	L3	P2–P4	-	-	-	•	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	•	-	•	-	-	•	-	-	-	-	-	-	-	-	-	•
F4: State-Binding Failures																																														
T-34 risk_data Poisoning	L6	P4	•	•	-	•	-	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-	•	-	-	-	-	-	-	-	•	-	-	-	-	-	•	•	•	•	•
F5: Accountability Failures																																														
T-42 Multi-Tenant Isolation Failure	L6	P4	-	-	-	•	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	-	•	-	-	-	-	•	-	-	-	-	-	-	-	-	-	-	•	-	-	•	•	•	•	•

• applies + applies and is architecture-amplified - not applicable

Figure 6: High risk AP2 threat matrix.

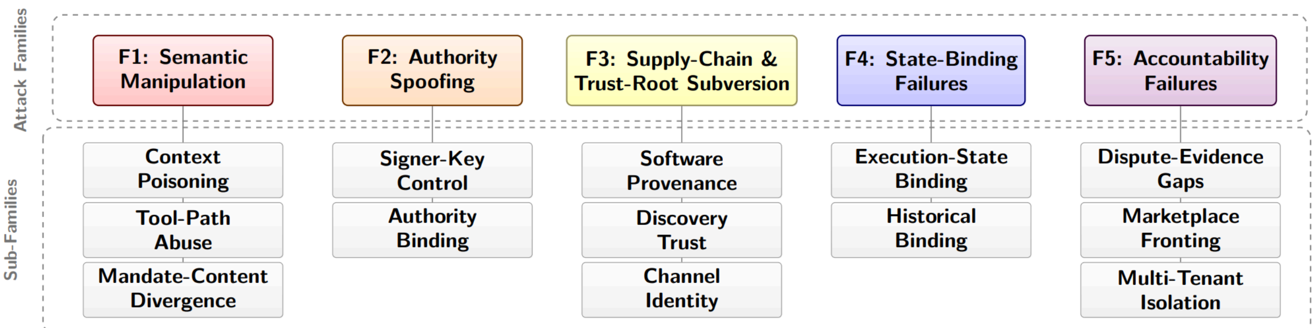

Figure 7: Taxonomy of AP2 attack families and sub-families.

5.2 F1: Semantic Manipulation

Semantic Manipulation attacks cause AP2 to accept, construct, authorize, or execute mandates whose effective semantics differ from what the user intended to perform. This family contains 10 of the 48 threats (T-1–T-9, T-48). Their primary goal is AG2, with AG1 and AG4 as common consequences.

Context Poisoning. This sub-family targets the agent's working context, the accumulated inputs from which a mandate is formed. By seeding that context with adversarial content before construction, an attacker leads the SA or MA to encode its own intent as though it were the user's. What defines the sub-family is that this single failure mode is reachable through several distinct entry points. The first is pre-signing context poisoning, in which adversarial text arrives via catalog content, an A2A handshake or reply, a RAG corpus, or a cross-zone tool-result fan-out; under shared MCP, that same fan-out both injects content into the receiving zone and leaks state back from the originating one (T-1). The second is MCP metadata poisoning, in which malicious tool descriptions persist and are interpreted with high instruction weight (T-2). The third involves no boundary-crossing injection at all, instead weakening the system prompt through long-context attention dilution or slow multi-turn drift (T-3). The fourth drops injection entirely for persuasion: guardrail-passing language that embeds a malicious intent in the agent's long-running memory or convinces a peer agent to lower its defenses (T-48). Although the vectors may differ, the resulting failure is the same i.e., the generated mandate aligns with the attacker's context, bypassing the user's intent.

Tool-Path Abuse. The attacker does not need to poison the entire conversational history; instead, they can redirect framework's control flow simply by tampering with tool configurations or schemas. This includes tool argument manipulation, including verifier-side CP-MCP tools that can bias credential release (T-4); tool calls fired

before consent or in adversarial order (T-5); and cross-agent semantic collusion, where sub-agent chains compose locally valid actions into a mandate that exceeds user intent (T-9). These attacks exploit the architectural reality that LLM agents treat tool metadata and structural outputs as operational instructions, altering framework outputs or verifier-facing actions to bypass user intent.

Mandate-Content Divergence. In this sub-family, the mis-match appears inside the cryptographic mandate itself or between the mandate and its user-facing representation, and it can arise at three points in the authorization pipeline. During construction, the SA fills in missing constraints on its own instead of asking the user, producing an over-broad mandate (T-6). At the rendering layer, the user-facing summary diverges from the structured fields that are actually signed (T-7). And inside the SD-JWT disclosure flow, one value is disclosed to the TS while a different signed value is later consumed by the verifier (T-8). Although signature verification succeeds in each case, the protocol still fails, because AP2 binds a mandate the user never authorized.

5.3 F2: Authority Spoofing

Authority Spoofing tricks AP2 verifiers into attributing a mandate, checkout_jwt or receipt to the wrong signing authority. A legitimate principal appears responsible, although an unauthorized party controlled the signing path or the upstream decision. This family contains 9 of the 48 consolidated threats (T-10–T-17, T-45) and primarily realizes AG1, with specific instances targeting AG2, AG4, and AG5.

Signer-Key Control. The attacker obtains control of a long-lived AP2 signing key, or causes that key to be used outside its legitimate authorized scope. This includes signing-key exfiltration of `user_sk`, `agent_sk`, or `merchant_sk` through host or memory compromise (T-10); weak or unsealed key generation, such as low-entropy keygen or an in-memory window before HSM seal (T-11); and MCP credential exposure, where committed OAuth secrets or API keys let an attacker impersonate the operator's MCP client (T-17). In all cases a valid signature proves only that the relevant key was used; it does not establish that the principal it identifies actually approved the action.

Authority Binding. The attacker exploits an attribution or attestation gap that is missing, too coarse, or checked at the wrong role, caller, or boundary. This includes unanchored agent-key provenance (T-12), unscoped key identifiers that let one role's key verify as another (T-13), sub-agent attestation gaps (T-14), caller-role confusion at shared components (T-15), verifier decisions that leave no proof of checking (T-16), and TS consent-surface hijacking where the user approves a decoy while the real mandate signs underneath (T-45). This threat is close to rendered-vs-signed divergence (T-7) but sits in a different family: in T-7 the TS legitimately belongs to the user and the fault is that the signed fields diverge from the rendered summary, so the corruption is of the mandate *content* (Semantic Manipulation); in T-45 the consent surface itself is hijacked, so the user's approval action is captured for a mandate they never saw, and the corruption is of the *authority* the signature is attributed to. In all cases, AP2 accepts an action as authoritative without being able to prove that the right authority actually produced or approved it.

5.4 F3: Supply-Chain & Trust-Root Subversion

These attacks force AP2 to trust or run illegitimate dependencies before the mandate checks can protect the transaction. This family contains 13 of the 48 threats (T-18–T-30). They primarily target AG1 and AG2, with AG4 and AG6 as further consequences.

Software Provenance. This sub-family covers cases where an AP2 role runs, loads, or trusts code, models, prompts or tool infrastructure that the operator did not intend to deploy. It includes AP2 build or CI/CD compromise (T-18), non-attestable builds that make swapped artifacts hard to detect (T-19), runtime compromise of an agentic role (T-20), malicious MCP server images (T-21), unverified LLM model substitution (T-22), and sub-agent prompt drift across releases (T-23). The common failure is provenance: AP2 relies on software or model state whose origin and integrity are not reliably bound to the transaction.

Discovery Trust. The SA establishes trust with an illegitimate peer or registry during discovery, before any AP2 artifacts are generated. This includes AgentCard integrity failures on both the publish and consume side, since AgentCards are unsigned by spec and can be tampered at origin, in transit, or in cache (T-24) ; discovery transport MITM or redirection via a rogue root CA, host network misconfiguration or DNS/well-known hijack (T-25); a malicious registry or registry caching that subverts the trust root itself (T-26); and an A2A AP2-extension downgrade where a look-alike or fail-open extension mismatch silently drops AP2 enforcement (T-27).

Channel Identity. Trust is broken at the channel or tool-identity layer mid-session, after discovery. This includes an MCP server identity that is not session-bound, so a mid-session redirect to a malicious MCP breaks no AP2 invariant (T-28); cross-server tool or resource confusion through typosquatted tool names or overlapping resource handles that route data to the wrong server (T-29); and missing message-level role authentication, where post-TLS AP2 channels carry no app-layer role signature, leaving handshakes, status replies, and side-channel fields role-spoofable (T-30).

5.5 F4: State-Binding Failures

State-binding failures occur when AP2 fails to maintain a secure and consistent transaction state across its operational phases. Their goals include AG3 as primary and AG1, AG6, AG4 and sometimes AG5, depending on whether the attacker reuses a valid state, creates an inconsistent state or destabilizes historical verification. This family contains 9 of the 48 threats (T-31–T-38, T-46).

Execution-State Binding. These attacks break the relationship between the conditions the user authorized and the state actually consumed during execution. This includes replaying a closed mandate or fanning out one open mandate across multiple executions (T-31); mutating cart or constraint state after user review but before signing (T-32); shared-MCP races or ordering attacks that leave peers with conflicting transaction state (T-33); shared-MCP cache bleed across peers without concurrency (T-46); and unsigned `risk_data` fields that bias CP/MPP step-up or fraud decisions (T-34).

Historical Binding. These attacks break cryptographic re-verification over extended periods. This includes linkability from weak SD-JWT salts (T-35); loss of historical verifiability when key rotation or

missing trust snapshots prevent reconstruction of the signing-time trust state (T-36); stale mandate reuse, where AP2 does not tie a receipt to the validity window of the mandate that authorized it, so a stale receipt can settle a lapsed authorization (T-37); and long-term cryptographic aging, where retained ECDSA-P256 mandates may outlive their security time frame (T-38). The common failure is that AP2 cannot reliably bind a past transaction to the trust and cryptographic state that existed when it occurred.

5.6 F5: Accountability Failures

Accountability failures occur when AP2 validates a proxy identity or keeps an incomplete audit trail, allowing the actual entity responsible for execution, fees, or liability to remain hidden. Their primary goal is AG5 (Repudiation); depending on the hidden principal's action, they also realize AG4 and AG1. This family contains 7 of the 48 threats (T-39–T-44, T-47).

Dispute-Evidence Gaps. These attacks exploit structural gaps in what AP2 can prove after the fact. This includes missing evidence that AP2 cannot prove ever existed (T-39); absent per-role, per-decision or consent-ceremony audit trails (T-40); missing behavioral telemetry that lets compromise resemble normal traffic (T-41); deletion requirements that destroy dispute evidence (T-44); and MCP elicitation prompts that mimic AP2 consent without producing a TS audit trail (T-47). The dispute resolver is left with a partial evidence chain and cannot identify who saw, approved, changed, or retained the relevant state.

Marketplace Fronting. While the marketplace identity is visible to AP2, the sub-merchant that actually shaped the transaction remains hidden. A single marketplace `merchant_identity` can hide the sub-merchant or intermediary responsible for transaction terms, fees, fulfillment, dispute facts, or post-signing behavior (T-43). The signed cryptographic chain points exclusively to the front identity, even though the accountable principal sits behind it.

Multi-Tenant Isolation. Here a CP or MPP serves many tenants from one platform instance but fails to isolate their state. Session context, cached decisions, or scoped tokens issued for one tenant bleed into another, either exposing that tenant's payment data or biasing an authorization or step-up decision made on its behalf. Because every request is signed and processed under the shared platform's identity, the AP2 mandate chain attributes the action only to that platform service and cannot name the tenant whose state actually shaped the outcome (T-42). The failure is architectural rather than cryptographic: AP2 binds signatures to platform identities, not to the tenants, sub-merchants, and evidence chains that actually shape and account for the transaction

6 Attack Demonstrations and Mitigations

This section presents five attack demonstrations and mitigations (AM1–AM5) that together cover all eight high risk threats identified in Figure 6. The first two are chains, in which each link creates a precondition the next needs or disables a defense that would otherwise stop it, so the attack fails without any one link; the remaining three each realize a single high risk threat in isolation. Each demonstration describes the attack scenario, attacker requirements, access and knowledge capabilities, deployment architecture, the threats realized, attack families and attacker goals, and the mitigations.

AM1. Authorization Beyond Stated Intent (T-23 → T-1 → T-4)

A malicious merchant wants the SA to obtain broader payment authority than the user intended. The user asks the SA to buy one camera for no more than $50. The deployment uses HNP mode and a multi-agent SA whose consent sub-agent compares proposed open mandates with the user's request. Its original rule requires escalation whenever a proposed amount exceeds the user's limit or a requested restriction is absent. A prompt-only release changes that rule so that the sub-agent accepts deviations when the surrounding transaction context appears to justify them (P1, T-23). The merchant may exploit drift already present in the deployment.

The merchant then supplies catalog terms claiming that a $50 purchase may require up to $80 of authorization headroom and that the payee will be resolved at checkout (P2 and P3, T-1). Its MCP quote tool returns an amount range from $50 to $80 but no payee field. The SA maps that result into a `payment.amount_range` constraint with `min=5000`, `max=8000`, and `currency=USD`. Because the tool supplies no payee, the mapper emits no `payment.allowed_payees` constraint (P3, T-4). The drifted consent sub-agent treats the merchant-controlled terms and quote as sufficient justification for both deviations and does not escalate.

The TS presents the open mandates for approval. The review identifies the task as buying one camera expected to cost $50 and presents the $80 ceiling and unrestricted payee status as checkout headroom. Relying on the consent workflow's lack of a warning and the merchant's explanation, the user authorizes the open mandates. This scenario does not require the TS to hide signed fields. If the TS displayed only the $50 purchase while signing the broader constraints, the deployment would also realize rendered-versus-signed divergence (T-7).

The merchant later creates a merchant-signed checkout for the same camera at $80, and the SA signs the corresponding closed Checkout and Payment Mandates with its bound agent key.

Requirements.	HNP mode and a prompt-defined consent decision.
Access.	ACR4 on S2; TA2 catalog-authoring influence on S11; ACR5 on S9.
Knowledge.	AK1, AK2, AK3, AK4.
Architecture.	A3×A2 in HNP mode; amplified under A5.
Threats.	T-23, T-1, T-4.
Family / goals.	F3/T-23; F1/T-1, T-4. AG2; AG1 if the $80 checkout settles.

Mitigation. For T-23, sub-agent prompts, thresholds, and tool grants become versioned release artifacts, with a provenance record for every change. The SA rejects a prompt identity that is not tied to an approved release, and a deterministic versioned policy outside the model enforces the escalation rule. For T-1, catalog terms and tool results retain their origin and remain transaction data rather than authorization. Merchant content

may report a price or fee, but it cannot justify increasing a ceiling or removing a restriction. For T-4, the deployment pins the tool schema and passes results through a typed mapper. Before signing, the TS checks and explicitly renders the exact value of `payment.amount_range.max`, rather than a generic amount range, and states when `payment.allowed_payees` is absent. It rejects a maximum above the user's limit or an absent payee restriction that the user did not explicitly authorize, and signs only the bytes it rendered.

AM2: Role Confusion at a Shared Tool Layer (T-29 → T-15)

Attack Scenario. A camera store and its largest competitor sell through the same payments platform, where the shopping agents, merchant agents, and the Credentials Provider all reach their tools through one shared MCP layer (a generalized A5 instance, Section 3.2). The competitor runs one of the tool servers on that shared layer, ordinary for a merchant peer under TA2, and uses it to make the CP act on its behalf. The opening is that AP2 authenticates artifacts but never checks who is behind a tool call: which server answered it, and which role made it.

It starts with a name collision. The competitor registers `checkout.quote_v2` under a name matching the canonical `checkout.quote`, and because the shopping agent resolves tool names against every connected server with nothing tying a name to the one server allowed to answer it, some quote requests reach the competitor's server. It returns correct quotes, so nothing looks wrong, but every request reveals the fields the shopping agent attached: the live transaction and cart identifiers, the amounts, and the CP handle in use (T-29).

That context enables the second step. Holding a real transaction identifier, the competitor calls the CP's credential tools and binds them to it. The CP authorizes on the transport channel the call arrives on, since AP2 carries role identity in signed artifacts and not in tool calls, and a merchant peer's channel looks legitimate; unable to tell the caller is reaching for CP-reserved operations, it answers (T-15). Neither step suffices alone: the collision only leaks context and confers no authority, while a role-confused call with no live transaction identifier binds to nothing that matters. Together they let a merchant peer drive CP-reserved operations, and because every call is signed under the platform's identity, the audit trail names only the platform, never the server that answered or the role that called.

Requirements.	Tool names must resolve across servers without a binding to an authoritative server identity, and the shared component must authorize by transport rather than by AP2 role.
Access.	ACR5 on S9 to operate the shadowing tool server, and ACR1 on the shared S9.
Knowledge.	AK1, AK2.
Architecture.	Generalized A5: one MCP tool layer shared across the SA, MA, and CP.
Threats.	T-29, T-15.
Family / goals.	F3/T-29; F2/T-15. AG1.

Mitigation. Both steps close by binding an identity that AP2 currently leaves to the channel: the server that answers, and the role that calls. For T-29, the SA pins each tool to the one server allowed to answer it and rejects name collisions across servers. The binding is fixed at session start and cannot be re-resolved mid-session, so a look-alike tool never receives the call. For T-15, every MCP call carries an application-layer signature over caller identity, AP2 role, and transaction identifier, and the CP authorizes on that signature rather than on the transport channel. CP-reserved tools are further gated by per-caller capability tokens issued for the transaction, so a legitimate peer channel alone never confers a role.

AM3: Cross-Tenant Credential Theft (T-42)

Attack Scenario. A payments platform runs one CP instance for all the merchant programs it hosts, and a small merchant on that platform is the attacker. It has a legitimate storefront, a legitimate AP2 role, and, like every tenant, a legitimate channel to the CP. Its goal is a rival store's customer payment credentials, the tokens the CP mints when it approves a checkout.

The weakness is in how the CP remembers approvals. It issues each payment credential as a scoped token and files it in one shared store keyed by the token and the transaction reference, with no tenant in the key. The attacker learns this indexing is flat: nothing in a lookup says which merchant a reference belongs to. So during its own ordinary checkouts it harvests transaction references, then replays them to the CP's credential interface under its own valid channel. The CP resolves each reference against the shared store, finds the matching approval, and returns the token, without ever asking whether the caller is the merchant that reference was created for. One of those tokens belongs to the rival's customer, and the CP hands it over. Because the call is signed under the platform's own identity and the store carries no tenant, the audit trail records only that the platform issued a credential, never that a competitor pulled a rival customer's token from shared state.

Requirements.	The CP must serve multiple tenants without per-tenant cryptographic scoping of session, cache, and token state.
Access.	ACR1 on S5 at the verifier interface. Isolation being absent, the standalone path assumed ACR2 is not required.
Knowledge.	AK1.
Architecture.	A4: a multi-tenant CP verifier.
Threats.	T-42.
Family / goals.	F5. AG4 for the cross-tenant disclosure; AG3 where the bled decision inflates the released scope.

Mitigation. Every entry in the token and decision store carries the tenant it belongs to, and the CP admits a lookup only when the caller's authenticated tenant matches that entry, so a token or approval filed for one merchant is never returned to another. The token itself is derived under a per-tenant key, and the responsible tenant, not only the platform, is named in the audit trail.

AM4: AP2 Extension URI Downgrade (T-27)

Attack Scenario. A shopper's SA begins a purchase, and an attacker positioned on the discovery path answers first. At P2, before any AP2 artifact exists to be signed, the attacker returns an unverified A2A AgentCard advertising a Merchant endpoint

whose AP2 extension URI differs from the canonical one by a single character: https://registry.ap2-protoco1.org/v2/secure against https://registry.ap2-protocol.org/v2/secure, the digit "1" standing in for the letter "l". The SA matches this URI loosely rather than exact-matching it against a pinned canonical value, so the look-alike passes and the SA binds the attacker's endpoint to the session. The trap is that this endpoint speaks only AP2 v0.1: the transaction silently downgrades and loses the context-binding and replay defenses v0.2 introduced [16] (T-27). The whole substitution happens before artifact validation begins, so every later signature check still passes over a session that is already replayable and unbound, and neither the shopper nor the merchant sees a downgrade occur.

Requirements. The SA must match the extension URI loosely rather than exact-match it against a pinned value, and accept the downgraded discovery result before AP2 artifact validation begins.
Access. ACC2 or ACR1 on S8.
Knowledge. AK1.
Architecture. A1 and above; the attack depends on no MCP, multi-agent, or marketplace surface.
Threats. T-27.
Family / goals. F3. AG1, AG2.
Mitigation. During P2, the SA exact-matches the extension URI against a pinned canonical value or a signed AgentCard rather than matching loosely. A URI that does not match exactly makes the SA fail closed: the legacy fallback is refused and the transaction dropped rather than silently downgraded.

AM5: Parameter Poisoning in `risk_data` (T-34)

Attack Scenario. A fraudster holds a stolen card number and wants to spend it without triggering the step-up challenge that would ask for something only the real cardholder has. Running its own SA, it drives the transaction itself, and the SA controls the one input the verifiers trust for that decision: the `risk_data` field carried with the Payment Mandate. In the AP2 reference implementation [8], the SA populates `risk_data` as a mutable field forwarded with the mandate, and no verifier checks its origin or integrity, so the fraudster simply writes the values it wants: a low risk score and a step-up-completed claim that never happened. The CP reads the supplied score and releases the payment credential as low risk; the MPP reads the step-up-completed flag and skips the OTP or 3DS2 challenge, the exact control meant to stop a stolen-card charge. Where the deployment forwards the same `risk_data` to a payment-network risk engine, the planted low-risk signal further lowers the chance the charge is ever flagged as fraud.

Requirements. The deployment must accept SA-authored or A2A-carried `risk_data` as verifier input.
Access. ACR2/ACR3 on the user's own S2, with write influence over `risk_data` carried with S10.
Knowledge. AK1, AK4.
Architecture. A1 and above.
Threats. T-34.
Family / goals. F4. AG1; AG3 where the suppressed step-up widens the effective authorization.
Mitigation. During P3, `risk_data` is accepted only inside the signed Payment Mandate or as a signed attestation the mandate references, conforming to a registered schema and issued by an authorized source such as the TS. The SA may not attach it as a mutable field. At P4, the CP and MPP reject any transaction whose risk payload is unsigned, off-schema, or unattributable to an authorized source.

7 The AP2 Security Scanner Tool

We built an AP2 security scanner to evaluate implementations against the threat catalog using static source analysis and targeted runtime checks. Given an implementation's source code, the scanner reports evidence-backed findings mapped to concrete attack paths across its roles, tools, verifier logic, and protocol surfaces. The scanner is available in the companion repository.[2] The scanner has four stages. First, it profiles the deployment's architecture, active AP2 roles, agentic roles, and exposed surfaces. Second, it filters the threat taxonomy matrix to threats whose architecture and surface requirements match that profile. Third, it runs threat-specific checks tied to each retained threat's surface and failure mode. For example, it checks whether MCP tools enforce per-request authorization through role-bound capability tokens rather than shared caller state, and whether AP2 control fields come only from typed policy or quote objects rather than open schema boundaries. Some failures require cross-role analysis. The Cross-Role Differential Specification Evaluator (CDSE) compares the constraint schemas, mandate fields, and accepted tool schemas that the SA, MA, CP, and MPP use for the same transaction. It flags cases in which one role signs, verifies, or executes a semantic object that differs from the object used by another role. This detects Semantic Manipulation failures in which a mandate is cryptographically valid but the transaction details differ across roles or diverge from the user's intent. Implementations that run in a test harness can also use an optional adversarial mode. The scanner starts a temporary test instance, reuses the discovered role wiring and tool definitions, selects a user-defined top-K set of threats with testable runtime paths, and runs fixed, reproducible probes against those paths. It records a finding only when a probe meets its threat-specific success condition. Finally, the report records each finding's evidence, affected role, surface, phase, threat ID, attack family, and required mitigation. This lets implementers identify exposed threats and determine whether the corresponding mitigations are enforced in code or at runtime.

8 Evaluation

We evaluate the threat catalog, the AIVSS risk assessment, and the scanner's layered design through three research questions.

RQ1. Catalog comparison. To what extent does a structured STRIDE-GPT analysis elicit the threats in the MAESTRO-derived catalog?

RQ2. Risk-score reproducibility. To what extent do independent experts reproduce the AIVSS ratings when applying the CVSS and AARS rubric defined in Section 4.5.1?

[2] https://anonymous.4open.science/r/AP2_Beyond_the_Mandate

RQ3. Scanner-layer contribution. Which threats does each scanner layer detect that the other layers miss?

8.1 Threat Catalog Comparison (RQ1)

Method. We used STRIDE-GPT [2] as an independent threat-modeling baseline. STRIDE [26] examines components and data flows, whereas MAESTRO examines agents and their interactions. This comparison tests which catalog threats a structured STRIDE analysis elicits and whether it finds any threat absent from our catalog. We considered three agent-specific alternatives. ATFAA provides an agent-focused threat taxonomy but no automated comparison procedure [22]. STRIDE-AI and ASTRIDE were relevant, but we found no public artifacts that allowed us to reproduce their analyses [4, 5]. We excluded the official MAESTRO Analyzer because it uses the framework from which we built our catalog and would not provide an independent baseline [3].

We used STRIDE-GPT v0.18.0 and ran it once for each deployment architecture from A1 through A5. Both configured generation roles used `gpt-5.3`. Each input described the same protocol roles, HP/HNP cryptographic artifacts, security properties, and AP2 lifecycle. Only the architecture block changed. The inputs excluded MAESTRO terminology, catalog identifiers and titles, target mechanisms, examples, expected counts, and prior mappings. The catalog and STRIDE-GPT runs used the same lifecycle and security-property definitions, which may increase measured agreement. One run per architecture also does not measure run-to-run variation.

An LLM judge compared every catalog threat with all STRIDE-GPT rows. Each row represented a generated threat scenario. The judge processed one catalog item per call and used `gpt-5.5` at temperature 0 with a fixed seed, so the generator did not grade its own output. For each item, the judge proposed a verdict, supporting-row identifiers, and a rationale. The mapping was many-to-many. Several rows could jointly support one catalog item, and one row could support several items. Full required compatible rows to cover the affected component or artifact, action or failure, security effect, and relevant architecture conditions. Partial covered the core failure and security effect but missed or generalized at least one AP2-specific detail without contradicting the item. None meant that no row, alone or in combination, could derive the threat. We manually checked every proposed Full and Partial mapping against the evidence and rules. We did not include None labels in that review. We also checked every STRIDE-GPT row for a catalog mapping. An unmapped row would be a candidate addition.

Two third-party annotators independently assessed every None item. Each had five years of cybersecurity experience and had developed multi-agent applications. They received the threat catalog, the AP2 documentation supplied to the judge, and an XLSX form requesting a label, confidence rating, and notes. *Distinct* meant that the item was a valid AP2 threat and did not duplicate a positively matched item. *Overlap* meant that a positively matched item already covered substantially the same threat. *Unsupported* meant that the AP2 system model did not support the threat.

Results The five STRIDE-GPT runs produced 124 rows, with 24 each for A1, A2, A3, and A5 and 28 for A4. The judge labeled 32 of the 48 catalog items Full or Partial and the remaining 16 None. We agreed with 31 of the 32 judgments and changed one from Full to Partial. Full-or-partial coverage was 32/48 = 66.7%. All 124 STRIDE-GPT rows mapped to at least one catalog item, so none qualified as a candidate addition.

Table 5: Catalog coverage from five frozen structured STRIDE-GPT runs.

Family	Full	Partial	Not elicited	Total
F1 Semantic	4	2	4	10
F2 Authority	5	3	1	9
F3 Supply-chain/trust-root	5	4	4	13
F4 State binding	3	3	3	9
F5 Accountability	2	1	4	7
Total	**19**	**13**	**16**	**48**

For the 16 None items, Annotator 1 assigned 15 *Distinct* labels and one *Overlap* label. Annotator 2 assigned all 16 items *Distinct*. They agreed on 15 items, giving raw agreement of 15/16 = 93.75%. They resolved the remaining disagreement as *Distinct*. The final labels were 16 *Distinct*, zero *Overlap*, and zero *Unsupported*.

8.2 Risk Assessment Reproducibility (RQ2)

Method. One rater assigned the AIVSS scores reported in Section 4.5.1. Eight independent participants from academia and industry then applied the same rubric. We tested whether the bands derived from their ratings reproduced the original severity bands. Our primary hypothesis was that the lower bound of the 95% confidence interval for Gwet's AC2 would be at least 0.61. For each threat, raters completed the 11 CVSS v4.0 base metrics [7] and ten AARS factors [24]. They did not calculate an AIVSS score or select a band. We reconstructed each score and band using the procedure in Section 4.5.1. Disagreement therefore reflects metric and factor assignments rather than arithmetic or band selection.

Each rater first completed two calibration examples, which we excluded from the analysis. The instrument then presented 25 threats. It included five threats per family, spanned the represented bands, and included threats near the High threshold. For each threat, the questionnaire fixed the architecture with the highest band. Five threats (`T-1`, `T-15`, `T-21`, `T-34`, `T-42`) formed a common set rated by all participants. We split the remaining 20 into two non-overlapping batches of ten, each covering all five families. Every participant rated the five common threats and one batch. Four participants rated Batch A and four rated Batch B. Each group included two participants from academia and two from industry. They worked independently, and we pseudonymized their responses.

We measured agreement with quadratically weighted Gwet's AC2 [9]. This coefficient gives partial credit to adjacent-band disagreements and is less sensitive than κ to skewed frequencies. We cross-checked it with quadratically weighted Krippendorff's α [15] and exact pairwise agreement. The AC2 interval uses a leave-one-threat-out jackknife over all 25 threats. For α, we resample the 25 threats with replacement and report the middle 95% of the resulting estimates. The 0.61 hypothesis threshold is an operational reference drawn from the Landis and Koch interpretation of κ [17]. It is not a validated cutoff for quadratically weighted AC2, so we interpret AC2 together with the two complementary agreement measures.

Results Table 6 reports agreement on the derived severity bands.

Table 6: Inter-rater agreement on AIVSS severity bands, reported as estimates with 95% confidence intervals.

Statistic	Scope	Est. [95% CI]	Interpretation
Gwet's AC2 (quad.)	all 25	**0.984** [0.963, 1.000]	LB exceeds 0.61 benchmark
Krippendorff's α (quad.)	all 25	0.802 [0.520, 0.978]	High estimate; CI crosses 0.61
Exact pairwise agreement	all 25	0.923 [0.846, 0.980]	High observed agreement; no chance correction

Quadratic weighting gives substantial credit to adjacent-band disagreements. With that qualification, the AC2 lower bound exceeds the 0.61 reference point. Exact pairwise agreement and Krippendorff's α provide complementary evidence. Participants also reproduced the single-rater reference band in 111 of 120 ratings (92.5%).

8.3 Scanner-Layer Ablation (RQ3)

Method. The scanner in Section 7 has three layers. The deterministic *Passive* layer checks source code and configuration. The LLM-driven *CDSE* layer compares specifications across AP2 roles. The LLM-driven *Adversarial* layer runs threat-specific probes against a test instance. We used an ablation study to measure each layer's contribution.

No public AP2 deployment was available, so we used our AP2 testbed. Section 9.2 discusses this limitation. Recall is the fraction of applicable threats detected for each architecture. The evaluation includes five threats for A2, five for A3, two for A4, and six for A5. A threat counts as detected when any enabled layer reports it. Each finding records the layer that produced it.

We evaluated seven configurations. **Full** enables all three layers. Three leave-one-out configurations disable one layer (**No-Passive**, **No-CDSE**, and **No-Adv**), and three isolated configurations enable one (**Only-Passive**, **Only-CDSE**, and **Only-Adversarial**). We ran every configuration five times per architecture. Every run that invoked CDSE or Adversarial used OpenAI `gpt-5.5`. We provide the scanner prompts in the GitHub repository.

The Adversarial layer tests only threats with an executable Bounded Adversarial Sandbox strategy. Each strategy defines an attack procedure and success condition for a shadow agent configured with the target's system prompt and tool schemas. Eligible threats must also have an architecture-specific AIVSS score of at least 4.0. If more than eight qualify, the scanner tests the eight highest-ranked threats. None of the evaluated architectures exceeded this limit. We classify a detection as *unique* when one isolated layer reports the threat in all five runs and the other two report it in none.

Results. The Full configuration attained the highest mean recall on every architecture, although other configurations tied it on A4 and A5 (Table 7).

The isolated configurations showed which detections each layer added. Passive uniquely detected `T-4`, `T-8`, and `T-24`. CDSE uniquely detected `T-14`, `T-1` on A4, and `T-29`. Adversarial uniquely detected `T-5`. Each layer detected at least one seeded threat that both other layers missed, so no layer subsumed the others. All three layers missed `T-2` on A2, `T-1` and `T-9` on A3, and `T-3` on A4. Table 8 gives the full per-threat attribution.

Table 7: Recall of the seven scanner configurations, reported as mean ± population standard deviation across five runs.

Arch	Full	No-Pass.	No-CDSE	No-Adv	Only-Pass.	Only-CDSE	Only-Adv
A2	.80	.36±.08	.80	.60	.60	.00	.36±.08
A3	.60	.28±.10	.40	.60	.40	.20	.08±.10
A4	.50	.50	.00	.50	.00	.50	.00
A5	1.00	.93±.08	.83	1.00	.83	.87±.16	.17

Table 8: Per-threat detection by the isolated scanner layers. • denotes detection in all five runs. A fraction gives the number of detections across five runs. •* denotes detection in all five runs by one layer and none by the other two.

Threat	Passive	CDSE	Adversarial
A2, Single-Agent with Isolated MCP			
T-2	0/5	0/5	0/5
T-4	•*	0/5	0/5
T-8	•*	0/5	0/5
T-34	•	0/5	4/5
T-5	0/5	0/5	•*
A3, Multi-Agent			
T-1	0/5	0/5	0/5
T-9	0/5	0/5	0/5
T-24	•*	0/5	0/5
T-27	•	0/5	2/5
T-14	0/5	•*	0/5
A4, Marketplace			
T-3	0/5	0/5	0/5
T-1	0/5	•*	0/5
A5, Shared MCP			
T-1	•	•	0/5
T-15	•	•	0/5
T-31	•	•	0/5
T-33	•	3/5	•
T-46	•	3/5	0/5
T-29	0/5	•*	0/5

9 Discussion and Future Work

9.1 Discussion

The threat modeling, attack taxonomy, and PoC validation yield three observations relevant to AP2 deployments.

9.1.1 Agentic authorization must bind operational context, not only signed intent. Across our PoCs, AP2 artifacts remain cryptographically valid even when a transaction no longer reflects the user's intent. Signatures cover the mandate but may omit the operational context from which the agent constructs it, including discovery choices, intermediate cart state, tool outputs, and verifier-facing disclosures. When an LLM-driven role selects this context, the pre-signing path becomes an attack surface.

Several PoCs succeed without forging signatures or breaking cryptographic checks. They steer the agent before it produces a protected AP2 artifact, so mandate-chain verification may not reveal the failure. The same limitation affects Tamarin or ProVerif analyses

when their models cover signed artifacts but omit the agentic execution path. Agentic authorization protocols should bind relevant pre-signature context to issued artifacts or reject execution paths that do not produce such bindings.

9.1.2 AP2 security depends on its protocols and deployment architecture. AP2 defines the authorization layer but leaves A2A transport and MCP tool-access architecture to implementers. AP2 consumes data from these protocols before committing it to a signed mandate, and many high-severity threats in our catalog originate in those exchanges. We therefore examined how A2A and MCP vulnerabilities affect mandate semantics and verifier decisions. This dependency is asymmetric: AP2 mitigations can validate the final mandate but cannot retroactively secure the flawed protocol exchange that produced it. Several mitigations must therefore be implemented as structural requirements in the supporting protocols.

The same asymmetry applies to deployment architecture. AP2 does not prescribe how A2A endpoints, MCP servers, agents, and merchant components are deployed. These choices determine the capabilities an attacker gains from a compromise and the trust boundaries between roles. As a result, a threat may be absent in one architecture, present in another, and amplified in a third. Deployments using the same protocols can therefore face different threats.

9.1.3 Different threat classes require different validation techniques. The scanner maps each threat class to a suitable validation strategy. Reasoning-layer vulnerabilities require adversarial testing, whereas supply-chain, cryptographic, and data-retention flaws are better suited to static analysis and configuration review. Because no single method covers the full attack surface, agentic-protocol security tools must combine detection methods matched to specific threat classes.

9.2 Limitations

9.2.1 No complete public AP2 deployment. At the time of evaluation, no complete public production or prototype AP2 deployment was available; Google's public AP2 implementation covers only part of the protocol [8]. We therefore built the AP2 testbed used for our PoCs. This limits our ability to claim that the same failures will appear unchanged in future third-party deployments (T-34). However, the testbed allowed us to evaluate AP2 before large-scale adoption, while design and implementation guidance can still influence how the protocol is deployed. The testbed, threat catalog, and scanner can help implementers identify trust boundaries, attack surfaces, and failure modes before production release. Our PoCs show that the attacks can affect authorization within AP2, including attacks that originate in A2A or MCP interactions. They do not measure whether or how the resulting AP2-layer failures affect card networks, issuers, acquirers, or other payment-rail systems.

9.3 Future Work

Once public AP2 deployments or reference implementations become available, future work should run the scanner against them to audit the threats identified here. Such evaluations would provide a more realistic basis for estimating false-positive and false-negative rates and help turn the scanner from a research prototype into a production-ready tool.

10 Conclusion

AP2 introduces a cryptographic mandate chain that makes user intent verifiable, binds checkout and payment mandates, and yields artifacts verifiable after a transaction. In this work, we analyze how unsecured catalog data, tool results, A2A messages or some other context information can manipulate agents before signing. A manipulated pre-signing path can yield a valid mandate chain authorizing a transaction the user did not intend. We therefore analyzed AP2 end to end as an agentic payment system. We decomposed AP2 v0.2 into five lifecycle phases and five deployment architectures; built a MAESTRO-based threat model covering four threat actors, eleven attack surfaces, and six attacker goals; and grouped 48 threats into five attack families. AIVSS rated 8 threats high in at least one architecture. We also built a testbed for the PoC demonstrations and developed a scanner to operationalize the security analysis. As AP2 moves from specification to production, the threat catalog and scanner can help implementers identify architecture-specific threats and enforce the required mitigations.

References

[1] V. Acharya. 2025. Secure Autonomous Agent Payments: Verifying Authenticity and Intent in a Trustless Environment. https://arxiv.org/abs/2511.15712. arXiv:2511.15712

[2] Matthew Adams. 2024. STRIDE-GPT: An AI-powered threat modeling tool based on the STRIDE methodology. https://github.com/mrwadams/stride-gpt. Accessed: 2026-07-08.

[3] Cloud Security Alliance. 2024. GitHub - CloudSecurityAlliance/MAESTRO — github.com. https://github.com/{C}loud{S}ecurity{A}lliance/{M}{A}{E}{S}{T}{R}{O}. [Accessed 22-08-2026].

[4] Eranga Bandara, Amin Hass, Ross Gore, Sachin Shetty, Ravi Mukkamala, Safdar H. Bouk, Xueping Liang, Ng Wee Keong, Kasun De Zoysa, Aruna Withanage, and Nilaan Loganathan. 2025. ASTRIDE: A Security Threat Modeling Platform for Agentic-AI Applications. arXiv:2512.04785 [cs.AI] https://arxiv.org/abs/2512.04785

[5] Tsafac Nkombong Regine Cyrille and Franziska Schwarz. 2026. STRIDE-AI: A Threat Modeling Framework for Generative AI Security Assessment. arXiv:2605.17163 [cs.CR] https://arxiv.org/abs/2605.17163

[6] T. Debi, W. Zhu, and P. S. Gupta. 2026. Whispers of Wealth: Red-Teaming Google's Agent Payments Protocol via Prompt Injection. https://arxiv.org/abs/2601.22569. arXiv:2601.22569

[7] Forum of Incident Response and Security Teams. 2023. Common Vulnerability Scoring System version 4.0: Specification Document. Accessed: 2026-05-25. https://www.first.org/cvss/v4.0/specification-document

[8] Google Agentic Commerce. 2026. Agent Payments Protocol (AP2). https://github.com/google-agentic-commerce/AP2. Release v0.2.0, Apr. 28, 2026; accessed 2026-05-25.

[9] Kilem L. Gwet. 2008. Computing Inter-Rater Reliability and Its Variance in the Presence of High Agreement. *Brit. J. Math. Statist. Psych.* 61, 1 (2008), 29–48.

[10] X. Hou, Y. Zhao, S. Wang, and H. Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. https://arxiv.org/abs/2503.23278. arXiv:2503.23278

[11] B. A. Hu and H. Rong. 2025. Inter-Agent Trust Models: A Comparative Study of Brief, Claim, Proof, Stake, Reputation and Constraint in Agentic Web Protocol Design – A2A, AP2, ERC-8004, and Beyond. https://arxiv.org/abs/2511.03434. arXiv:2511.03434

[12] C. Huang, X. Huang, N. P. Tran, and A. M. Fard. 2026. Model Context Protocol Threat Modeling and Analyzing Vulnerabilities to Prompt Injection with Tool Poisoning. https://arxiv.org/abs/2603.22489. arXiv:2603.22489

[13] K. Huang. 2025. Agentic AI Threat Modeling Framework: MAESTRO. Cloud Security Alliance blog. Accessed: 2026-05-25. https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro

[14] S. Jamshidi, A. M. Dakhel, K. W. Nafi, and F. Khomh. 2025. Semantic Attacks on Tool-Augmented LLMs: Securing the Model Context Protocol against Descriptor-Level Manipulation. https://arxiv.org/abs/2512.06556. arXiv:2512.06556

[15] Klaus Krippendorff. 2004. *Content Analysis: An Introduction to Its Methodology* (2nd ed.). Sage.

[16] Q. Lan, A. Kaul, S. Jones, and S. Westrum. 2026. Zero-Trust Runtime Verification for Agentic Payment Protocols: Mitigating Replay and Context-Binding Failures in AP2. https://arxiv.org/abs/2602.06345. arXiv:2602.06345

[17] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174.

[18] Y. Louck, A. Stulman, and A. Dvir. 2025. Security Analysis of Agentic AI Communication Protocols: A Comparative Evaluation. https://arxiv.org/abs/2511.03841. arXiv:2511.03841

[19] N. Maloyan and D. Namiot. 2026. Breaking the Protocol: Security Analysis of the Model Context Protocol Specification and Prompt Injection Vulnerabilities in Tool-Integrated LLM Agents. https://arxiv.org/abs/2601.17549. arXiv:2601.17549

[20] Q. Mao, J. Wang, Y. Liu, L. Zhu, C. Ma, and J. Yan. 2026. SoK: Security of Autonomous LLM Agents in Agentic Commerce. https://arxiv.org/abs/2604.15367. arXiv:2604.15367

[21] V. S. Narajala and I. Habler. 2025. Enterprise-Grade Security for the Model Context Protocol (MCP): Frameworks and Mitigation Strategies. https://arxiv.org/abs/2504.08623. arXiv:2504.08623

[22] Vineeth Sai Narajala and Om Narayan. 2025. Securing Agentic AI: A Comprehensive Threat Model and Mitigation Framework for Generative AI Agents. arXiv:2504.19956 [cs.CR] https://arxiv.org/abs/2504.19956

[23] OWASP Agentic Security Initiative. 2025. Agentic AI – Threats and Mitigations. Accessed: 2026-05-25. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/

[24] OWASP Agentic Security Initiative. 2026. AI Vulnerability Scoring System (AIVSS). Accessed: 2026-05-25. https://aivss.owasp.org/

[25] OWASP GenAI Security Project. 2025. OWASP Top 10 for LLM Applications 2025. Accessed: 2026-05-25. https://genai.owasp.org/llm-top-10/

[26] Adam Shostack. 2014. *Threat Modeling: Designing for Security.* Wiley.