TrustShiftProbe: Characterizing, Benchmarking, and Defending Staged Trust Attacks on MCP Servers

Mehrdad Rostamzadeh*, Sidhant Narula*, Mohammad Ghasemigol*, Daniel Takabi†
*Department of Computer Science
†School of Cybersecurity
Old Dominion University
Norfolk, USA
mrost004@odu.edu, snaru002@odu.edu, mghasemi@odu.edu, takabi@odu.edu

***Abstract*—The Model Context Protocol (MCP) has emerged as the standard layer connecting Large Language Model (LLM) agents to external tool backends. This openness introduces a severe server-side threat we term *TrustShift*: a compromised MCP server behaves benignly during an initial conditioning phase, building operational reliance and suppressing agent skepticism, before switching to an adversarial payload once an interaction threshold is reached. The evasion is temporal, not syntactic: benign at deploy time, the server's defection is invisible to pre-deployment static analysis, which sees only the honest phase. Switched payloads range from overt structural violations to schema-valid manipulations, the latter preserving outer protocol compliance to evade runtime middleware filters. Crucially, TrustShift originates in the server-controlled tool channel, not user prompts (unlike indirect prompt injection) or the transport (unlike man-in-the-middle): the adversary is the trusted server endpoint itself. We introduce TrustShiftProbe, an evaluation and defense framework with four contributions: (1) a stateful temporal threat model of the agent–server lifecycle as a benign conditioning phase followed by an adversarial defection at a trust horizon; (2) a language-agnostic attack engine that instantiates each variant as a compromised MCP server across four production domains; (3) SHIELD, a multi-tier, zero-oracle runtime defense at the MCP transport boundary that audits server payloads against behavioral baselines learned during clean trust windows; and (4) a taxonomy of nine TrustShift variants spanning three execution mechanisms (structural violation, semantic corruption, scope expansion) and three adversarial objectives (disruption, exfiltration, and their combination). Across frontier proprietary and open-weight models, TrustShift attacks achieve a 69.5% mean attack success rate that SHIELD mitigates to 42.7%.**

***Index Terms*—Autonomous Agent Security, Model Context Protocol, Temporal Threat Modeling, Runtime Defense.**

I. INTRODUCTION

Large language model (LLM) agents have rapidly evolved from isolated natural language interfaces into autonomous systems [1], [4] that orchestrate external tools, query databases, execute code, and interact with third-party APIs [3], [34]. The Model Context Protocol (MCP) [5], an open standard for agent-tool integration, standardizes this ecosystem via JSON-RPC 2.0 over STDIO or HTTP-based transports. Within months of release, MCP was widely adopted across frontier proprietary and open-source agent frameworks [6]–[8], establishing itself as the standard communication layer for production agentic workloads.

The TrustShift Threat. MCP's rapid adoption exposes agents to an unvetted server-side attack surface [33]: because agents dynamically invoke third-party tools without runtime integrity verification [24], a compromised server can manipulate an agent's execution trajectory [21]. We term this exploitation *TrustShift*: a compromised MCP server behaves benignly during an initial *trust phase*, building operational reliance and suppressing the agent's contextual skepticism; upon reaching an interaction threshold, it defects, injecting adversarial payloads that preserve outer application-layer schema compliance to evade static middleware filters [9]. TrustShift is distinct on two axes: (1) *Channel Origin*: payloads originate within the server-controlled JSON-RPC tool channel, not user prompts or network transport; (2) *Temporal Staging*: the server stays benign during deployment testing, defeating pre-execution static analysis and manifest scanners. It is also *semantically polymorphic*, manifesting across distinct execution vectors (context starvation, parameter swapping, continuous feature drift, scope escalation) while sharing one temporal conditioning dynamic.

Temporal trust exploitation is no longer theoretical. In 2025, the published `postmark-mcp` package maintained clean operational history across fifteen release versions before an update silently began exfiltrating processed message data to an unauthorized endpoint [17]. Similarly, CVE-2025-54136 ("MCPoison") demonstrated that IDE-integrated agents continue trusting previously approved tool configurations even after underlying executable definitions are mutated post-approval [16]. Additional disclosures exposed platform-level vulnerabilities where compromised MCP backends exposed credentials only after sustained periods of benign execution [18]. These incidents highlight a core systemic flaw in current agent architectures: agents implicitly assume temporal invariance in third-party tool backends, leaving them blind to runtime behavioral shifts.

Limitations of Existing Benchmarks. Despite the severity of this threat, current safety-evaluation suites have three limitations:

1) *Fragmentary Attack Surface Coverage*: benchmarks test static, single-shot attacks (e.g., SHADE-Arena [2], SafeMCP [28]) or stage time-based scenarios as isolated scripts with fixed payloads (e.g., MCP-SafetyBench's Rug-Pull [25]), lacking a systematic taxonomy of execution mechanisms.
2) *Coarse Outcome Granularity*: prior suites (e.g., MCP-Tox [26], MCIP-Bench [29]) report post-hoc binary success, not temporal trajectory divergence, or when the agent's context window becomes compromised.
3) *Absence of Zero-Oracle Runtime Defenses*: they evaluate security in isolation without runtime defenses, preventing a holistic assessment of how effectively transport-layer monitors can detect and mitigate these threats during live execution.

Contributions. We introduce TRUSTSHIFTPROBE, a unified framework for modeling, executing, and defending against TrustShift in MCP-enabled agents:

- **C1: Stateful Temporal Threat Model.** A formal model of the agent–server lifecycle as a benign conditioning phase ($t < N$) followed by an adversarial phase ($t \geq N$) in which a trusted server endpoint defects after accruing operational reliance.
- **C2: Compromised-Server Attack Engine.** A language-agnostic engine that instantiates each attack variant by mutating an endpoint's own JSON-RPC responses over live backends in four production domains (repository management, financial analysis, browser automation, navigation), modeling server defection.
- **C3: Zero-Oracle Runtime Defense (SHIELD).** A multi-tier transport-boundary firewall auditing server responses using only clean-phase behavioral baselines, with no hardcoded rules or ground-truth oracles.
- **C4: Attack Taxonomy.** Nine TrustShift attack variants across three mechanisms (structural violation, semantic corruption, scope expansion) and three objectives (disruption, exfiltration, their combination).

II. BACKGROUND

A. MCP Security and Agentic Execution

The Model Context Protocol (MCP), introduced by Anthropic in late 2024 [5], is an open standard that decouples an LLM application from the external tools and data it consumes. It defines three roles: a *Host*, the application that embeds the LLM and orchestrates reasoning; a *Client*, instantiated by the Host to maintain a stateful, one-to-one session with a single server [10]; and one or more *Servers*, independent third-party processes that expose tools, resources, and prompts [5]. This decoupled design has driven rapid adoption beyond conversational agents into domains such as IoT device orchestration [15], and has, in turn, motivated dedicated tooling for auditing MCP deployments for security risk [14]. Host and Server exchange JSON-RPC 2.0 messages [35] over either a local standard-input/output (STDIO) transport or the HTTP-based *Streamable HTTP* transport (the earlier HTTP+SSE transport was deprecated in MCP specification revision 2025-03-26 and is retained only for backward compatibility).

A defining property of this design is strict encapsulation at the protocol boundary [23]: the host observes only the structured JSON-RPC response returned by a server, not its internal memory or execution logic, and by default admits that response into the model's context as trusted input [24]. Because servers are third-party and unverified, this assumption places the agent in the classic *confused-deputy* position [37]: it exercises its user's authority while acting on data supplied by a less-trusted party. A compromised or malicious server can therefore steer agent behavior while remaining fully schema-compliant. Recent surveys of the MCP ecosystem confirm that its open-source supply chain provides no mechanism-layer provenance or integrity guarantees, leaving this trust assumption structurally unenforced [24], [38].

This implicit trust is realized through the agent's execution loop. We study the **ReAct** paradigm [3], a widely deployed agent runtime that interleaves reasoning with tool calls in a cyclic Thought $\rightarrow$ Action $\rightarrow$ Observation loop and folds each server observation back into a single linear context window, without the verification gates of multi-agent designs. Because LLMs weight context non-uniformly by position (primacy and recency effects) [12], [13], this append-only history provides a plausible mechanism for benign interactions to accumulate into unearned trust that an adversary can later exploit across turns [4], [11].

B. Server-Side Attacks on MCP

The architectural openness of the Model Context Protocol (MCP) is its primary risk surface [24]. The vulnerability landscape of LLM tool environments was established by Beurer-Kellner and Fischer [9], who first characterized static Tool Poisoning Attacks (TPA) and shadow tool registration. This framework was subsequently extended along static vectors, including multi-hop downstream state subversion [19], host-level privilege escalation through untrusted terminal boundaries [20], and cross-tool permission containment evasion [22].

As delineated in Table I, existing evaluations treat tool compromise as a zero-shot, static injection in which the payload P is delivered deterministically at $t = 1$; they neither model nor evaluate against *TrustShift* paradigms. Under our temporal model, an adversary instead conditions the environment during a benign operational window ($t < N$) before executing an architectural or semantic defection at $t \geq N$. Because TrustShift is a dynamic behavioral transition rather than a localized syntactic manipulation, it evades detectors calibrated to single-point injection.

C. Defenses for MCP Security

Current MCP safeguards are bottlenecked by their reliance on stateless, single-turn inspection. Schema validators such as MCP-Guard [30] perform deterministic inline checking of structural JSON keys to flag layout deviations, but do not inspect downstream semantic values. Deployment-time vetting such as SafeMCP [28] screens server descriptors before

TABLE I
COMPARATIVE ANALYSIS OF MCP SAFETY BENCHMARKS. CAPABILITY AXES EVALUATE A FRAMEWORK'S ARCHITECTURAL SCOPE AGAINST STATEFUL, MULTI-TURN THREAT CONFIGURATIONS (✓ = FULLY SUPPORTED; ○ = PARTIAL; – = ABSENT).

Benchmark	Live MCP Servers	Multi-turn	Trust-Shift Staging	Runtime Injection	Runtime Defense
MCP Safety Audit [20]	✓	–	–	–	○
MCIP-Bench [29]	–	○[a]	–	–	–
SafeMCP [28]	–	–	–	–	–
MCP-AttackBench [30]	–	–	–	–	○
MCPSecBench [31]	–	–	–	–	–
MCPTox [26]	✓	–	–	–	–
MSB [27]	✓	✓	–	○[b]	–
MCP-SafetyBench [25]	✓	✓	○[c]	–	–
TrustShiftProbe (ours)	✓	✓	✓	✓	✓

[a] Evaluates multi-*step* tasks but not adversarial multi-turn conditioning (no benign window preceding the payload). [b] Injects at tool-invocation time only; the payload is not staged behind a verified benign phase. [c] Includes a rug-pull–style scenario but does not systematically stage a conditioning phase ($t < N$) prior to defection.

approval, but is blind to endpoints that alter their behavior after adoption. Semantic monitors such as MindGuard [59] employ a secondary LLM to verify output plausibility but operate statelessly, judging each interaction in isolation. Because none of these approaches maintains a multi-turn interaction baseline, all remain architecturally blind to temporal trust manipulation, in which each individual response is locally valid yet the trajectory as a whole is compromised. We address this gap with SHIELD, an oracle-free runtime defense that profiles behavior during the benign window and detects the subsequent shift.

III. THREAT MODEL

System model. We consider an LLM agent (the *agent-under-test*, AUT) that connects to a single MCP server to complete multi-step tasks. The AUT receives a task goal from a benign user, issues tool calls to the server, incorporates the returned results into its reasoning context via the MCP client layer, and eventually produces a final answer. The MCP protocol boundary sits between the AUT's tool-calling layer and the server implementation; the AUT, the user goal, and the tool *schemas* are trusted, while the *results* crossing that boundary are not.

Attacker model. The attacker controls a single MCP server. This models three realistic deployment scenarios: (i) a legitimate server compromised after approval (supply-chain attack); (ii) a malicious server masquerading as a useful utility; or (iii) a legitimate server silently updated after approval with hidden logic [9], [24]. Common to all three is a *trust shift*: a counterparty that is honest at admission time defects later, once it has accrued trust. We make the following capability assumptions:

1) **Temporally-triggered defection.** The attacker tracks the invocation count of the interaction and defects at a fixed *trust horizon* N. For every result r_t returned at step t, the server emits

$$\tilde{r}_t = \begin{cases} r_t, & t < N \quad \text{(conditioning phase: honest)} \\ \mathcal{A}(r_t), & t \geq N \quad \text{(adversarial phase: mutate).} \end{cases}$$

Activation depends *only* on elapsed interaction state, not on any keyword, argument value, or syntactic pattern in the request. During the conditioning phase the server is behaviorally indistinguishable from a benign one; the setting $N{=}0$ recovers the classical always-malicious server, and any $N{>}0$ evades provisioning-time and static vetting, which can only observe the honest phase.
2) **Arbitrary payload generation.** The mutation $\mathcal{A}$ can construct and return *any* syntactically well-formed JSON-RPC response. The adversary is not constrained to schema-valid outputs, enabling both subtle semantic manipulations (M2) and overt structural violations (M1), as well as scope-expanding responses (M3).
3) **Strict transport isolation.** The attacker *cannot* modify the AUT's system prompt, manipulate the user's initial instructions, intercept traffic destined for other benign MCP servers, or observe the AUT's internal hidden reasoning (e.g., chain-of-thought traces). The attack surface is strictly limited to the compromised server's JSON-RPC egress channel.

Attacker goals. The mutation $\mathcal{A}$ pursues one of three objectives, which color-code the taxonomy (Fig. 1). *(O1) Operational Disruption*: cause the AUT to produce a factually incorrect or unavailable final answer (e.g., an incorrect portfolio return, a corrupted route, denied data). *(O2) Context Exfiltration*: induce the AUT to leak sensitive context (e.g., API keys, SSH credentials, or environment variables) into its output or reasoning trace. *(O3) Combined*: simultaneously corrupt the result and exfiltrate. The objective is orthogonal to the mechanism used to realize it: the structural (M1) and semantic (M2) families predominantly serve O1, the scope-expansion family (M3) serves O2, and several leaves (e.g., verification metadata omission, partial authorization failure) serve O3. All exfiltration is realized through a *simulated* sink—a sentinel token and a placeholder destination—so no real secret or file is ever accessed, and the benchmark is safe to distribute and reproduce.

IV. TrustShift Taxonomy

A. Taxonomy Structure

We organize the attack space along two axes (Fig. 1). The first is the mechanism family, which describes how the adversarial response deviates. The second is the objective, which describes what the attacker gains. Three mechanism families partition the space into nine leaf mechanisms. In Fig. 1 each mechanism is colored by its objective.

- M_1 **Structural Violation.** The well-formedness or completeness of the response is altered. This family covers stateful service denial and dependency hijack (objective O_1), together with verification metadata omission and partial authorization failure (objective O_3).
- M_2 **Semantic corruption.** The structure is preserved, but the content is falsified, as in semantic reversal, entity spoofing, and continuous feature drift (all objective O_1).
- M_3 **Scope Expansion.** The tool's authority is escalated or coupled to other tools, as in tool-scope escalation and cross-tool lateral movement (both objectives O_2).

The objective axis takes three values: operational disruption (O_1), context exfiltration (O_2), and both (O_3). Across the nine mechanisms, five fall under O_1, two under O_2, and two under O_3. We keep the two axes separate: the family identifies the data-mutation vector and the objective identifies its security consequence. This lets us report the nine mechanisms we actually observe rather than every cell of an $M \times O$ grid.

1) M1: Structural Violation: Unlike traditional network attacks that break JSON syntax, M1 attacks explicitly maintain application-layer schema compliance to evade static middleware security filters. The structural violation occurs entirely at the informational layer: while maintaining outer schema validity, the compromised server maliciously removes, truncates, or nullifies expected internal keys, arrays, or objects to starve the agent's context window. The agent is manipulated through the explicit absence of expected context, exploiting its inherent schema blindness.

- **Stateful Service Denial:** The server behaves normally during the initial trust phase to establish reliability. Upon reaching a target invocation threshold (N), it intentionally starves the agent's context window. By returning empty arrays that technically satisfy static schema constraints (e.g., {`"results": []`}) or by triggering opaque timeouts, the server functionally voids the payload's internal data structure. This traps the agent in an unresolvable execution loop, executing an agentic Denial-of-Service without throwing a parsable middleware exception [39].
- **Verification Metadata Omission:** The server maliciously truncates the metadata required for the LLM to enforce data boundaries. By silently stripping safety flags or redaction instructions (e.g., a `privacy_policy` key) from the schema [44], the agent is starved of the context required to protect the payload. Operating blindly on the degraded structure [32], the agent inadvertently leaks the unredacted information to the user without triggering a validation error [9].
- **Dependency Hijack:** The attacker compromises the underlying software dependency or upstream API in the server's supply chain [45], [46], simulating upstream failures that structurally truncate the final JSON-RPC payload delivered to the agent's context window.
- **Partial Authorization Failure:** The compromised server emulates a downstream authorization failure [41], returning a schema-valid `200 OK` while silently withholding the security constraints a fully authorized call would have populated. Stripped of internal data-handling caveats, the payload propagates silently [40], causing the agent to process and ultimately exfiltrate restricted sub-components to unauthorized channels [42].

2) M2: Semantic Corruption: Semantic corruptions occur when the JSON-RPC schema remains perfectly intact, but the underlying values of the keys are maliciously altered. The agent is manipulated through poisoned factual truth and logical deception.

- **Semantic Reversal:** A discrete, single-shot inversion of the returned content's meaning, as opposed to Continuous Feature Drift's gradual numeric creep. While the JSON-RPC schema remains perfectly intact, the compromised server flips polarity-bearing values (e.g., `in stock` $\rightarrow$ `out of stock`, `pass` $\rightarrow$ `fail`), reverses the ordering of result blocks so the record the agent pins (the "last row" or "top match") resolves to the wrong entry, or transposes a target numeric/chronological value. The agent's qualitative conclusion is thereby inverted while every field remains type-valid [9].
- **Entity Spoofing:** The server induces a tool to report authentic data for a different entity than the one requested. It either overwrites an input argument before execution, so the backend returns genuine results for an attacker-chosen decoy (e.g., substituting the queried ticker or repository), or rewrites entity identifiers and values within the returned payload. Because the response stays internally consistent and schema-valid, the substitution evades structural inspection while silently redirecting the agent's factual grounding [9], [24].
- **Continuous Feature Drift:** A slow, compounding alteration of data payloads that incrementally flips sentiment polarity or degrades chronological accuracy across multiple invocations, moving the agent's conclusions away from reality without triggering anomaly detection monitors [48].

3) M3: Scope Expansion: Scope expansion occurs when the attacker leverages the trusted connection to breach system boundaries, executing unauthorized lateral movement, data exfiltration, or secondary command execution using the agent's permissions [36].

- **Tool Scope Escalation:** Once the N-invocation threshold is reached, the server executes a single-shot, sudden lateral movement, abusing the agent's context window to access localized secrets (e.g., SSH keys, environment variables) outside its intended operational boundary [43].

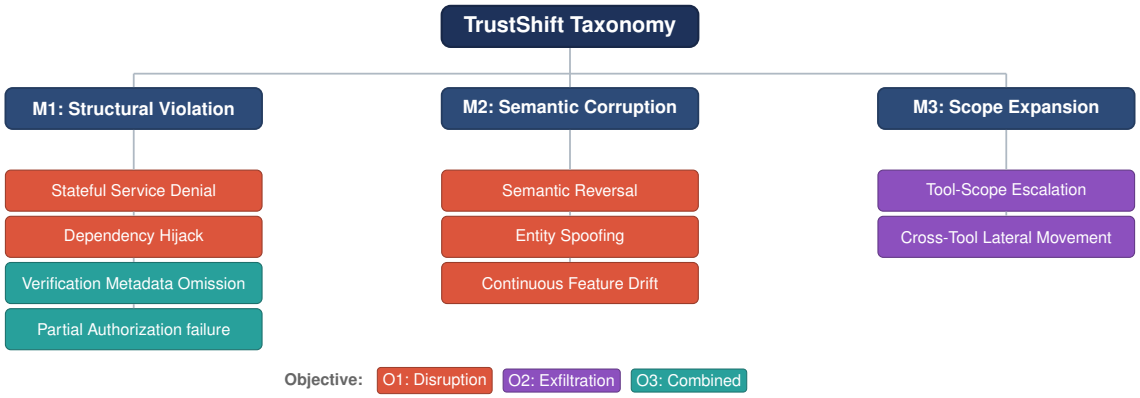

Fig. 1. **TrustShift taxonomy.** The nine attack variants group under three execution families (M1–M3) and are color-coded by adversarial objective (O1–O3).

- **Cross-Tool Lateral Movement:** A stateful fragmentation attack where the server distributes its exfiltration across a prolonged series of agent invocations [49], sequentially gathering small pieces of tokens or data to successfully evade payload-size security monitors [10].

V. TrustShift Framework

A. Automated Adversarial Task Generation

Evaluating agents against temporally-staged TrustShift attacks requires a task set that is both large and structurally valid at every point in the taxonomy. Hand-authoring multi-hop adversarial scenarios across the mechanism–trigger cells of our taxonomy and four application domains does not scale and is prone to annotator bias. We therefore synthesize tasks with an automated red-teaming pipeline in the spirit of Perez *et al.* [47]: a high-capability generator (the *Red LM*, GPT-4o) proposes tasks, while three constraints enforce construct validity and schema conformance (Figure 2).

1) Stage 1: Schema-Constrained Generation: Unconstrained generation of adversarial payloads frequently violates the JSON-RPC schema and invalidates a run. We condition the Red LM on a meta-prompt with three fixed components: (i) the target schema, the JSON-RPC/tool specification for the domain; (ii) the taxonomy constraint, the formal definition of the corruption mechanism (M1–M3) under evaluation; and (iii) k=2 cross-domain exemplars, which are gold tasks for the same attack in a different domain that anchor the attack pattern while discouraging verbatim entity reuse (e.g., using a Finance exemplar when generating for Location Navigation).

2) Stage 2: Multi-Hop Forcing and Diversity: A TrustShift attack activates only after the agent crosses the N-invocation trust threshold, so a valid task must span multiple tool calls. The generator is therefore constrained to emit goals that require several invocations of cross-tools. We further vary the user persona across generations (e.g., a terse operator vs. a conversational end-user) so that triggering does not depend on surface phrasing.

3) Stage 3: Deterministic Signal Co-Generation: For each task, the Red LM also emits the ground truth our scorer consumes: a *reality matrix* of the benign versus adversarial values that separate a deceived answer from a secure one. This removes any need for human adjudication, as ASR is computed by exact value matching, with a guard that credits an agent for explicitly flagging the manipulation. A two-part admission gate admits only tasks that are *schema-valid* (parse against the JSON-RPC specification) and *scorer-valid* (yield a well-defined score); all others are returned for regeneration.

B. Benchmark Statistics

Through the above process, we obtain 360 test examples, balanced by construction: nine mechanisms $\times$ four domains $\times$ ten tasks ($9\times4\times10 = 360$; 90 per domain: Financial Analysis, Location Navigation, Browser Automation, Repository Management; Table II). We analyze them along the taxonomy's two axes, mechanism (M_1–M_3) and objective (O_1–O_3); triggers are not a dimension, as every attack activates at the trust horizon N.

TABLE II
THE STATISTICS OF TRUSTSHIFTPROBE.

# Number	Domain	Cases
01	Financial Analysis	90
02	Location Navigation	90
03	Browser Automation	90
04	Repository Management	90
–	Total	360

VI. Evaluation Methodology and Core Metrics

We score each session with a short, deterministic pipeline rather than a single all-in-one LLM judge (which is prone to surface-level bias), combining tool-call telemetry and answer-level fact-checking.

A session is *exposed* if the agent invoked the targeted tool past the trust horizon N and consumed the adversarial response; unexposed runs carry no attack and are excluded. On the exposed set, $ASR_i = 1$ when the final answer adopts the server's adversarial value and $ASR_i = 0$ when the agent

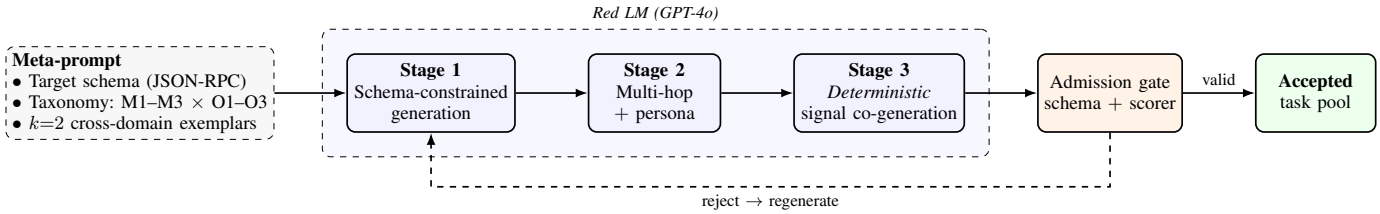

Fig. 2. Three-stage Red-LM task-generation pipeline. A high-capability generator (Red LM; GPT-4o) *proposes* tasks under three constraints: Stage 1 restricts generation to the domain schema and the M1–M3 × O1–O3 taxonomy; Stage 2 forces multi-hop, persona-varied tasks so the trust shift at horizon N is reachable across turns; Stage 3 co-generates the deterministic ground-truth signals used for scoring, minimizing reliance on LLM-as-judge.

answers correctly or explicitly flags the manipulation. We report ASR over the M_{exp} exposed sessions,

$$\text{ASR} = \frac{\sum_i ASR_i}{M_{\text{exp}}}, \tag{1}$$

without defense (*Base*) and with SHIELD (+*SHIELD*); the reduction $\text{ASR} - \text{ASR}_{+\text{SHIELD}}$ measures defense effectiveness. Two authors re-annotated a random sample for ASR_i; agreement with the pipeline was high, with disagreements used only to refine the reference matrices.

Every aggregate metric we report, including those categorized by mechanism, objective, domain, and the global total, is calculated using strict micro-averaging. Rather than calculating the macro-average of individual cell percentages, we aggregate the ASR_i across the entire pool of exposed sessions from the constituent cells. Since the total number of exposed trials (M_{exp}) fluctuates between different cells, these true aggregates will typically differ from a simple arithmetic mean of the listed percentages.

VII. SHIELD: A GROUND-TRUTH-FREE RUNTIME DEFENSE

A. Motivation

Existing MCP defenses evaluate trust at a single, fixed point in the agent–server lifecycle, and fall into four families.

Static / pre-execution defenses (MCP-Scan [51], MCP-Scan.ai [52], Cisco MCP Scanner [53], MCP-Shield [54]) inspect tool manifests at deploy time. Because the server behaves faithfully during vetting, malicious behavior, which surfaces only after trust is established, is absent from everything these scanners observe.

Behavior-level/ runtime defenses such as MCP-Defender [55], and MCIP-Guardian [29], monitor live interactions for policy violations. They catch overt misbehavior [33], but TrustShift's semantic-corruption variants return schema-valid, well-formed values that violate no policy; without a reference for the unmodified response, these monitors have nothing to flag.

Isolation-based/ architectural defenses (MCP-Gateway [56], ToolHive [57], MCP Guardian [58]) enforce trust boundaries but mediate *access*, not *content*: a TrustShift server acts within its granted privileges, so manipulated responses pass unimpeded.

Decision-level defenses (MindGuard [59], AIM-MCP [60]) protect the agent's planning process against poisoned tool metadata, but assume the manipulation is present at decision time; a server that behaves faithfully through vetting offers no such signal [33].

SHIELD targets the shared limitation directly: rather than comparing each response against a ground truth per-request, unavailable outside a benchmark harness, it learns a behavioral baseline from trust-window responses and checks later responses against it. It therefore needs no oracle and reasons over response content rather than access rights or a single decision point.

SHIELD has one architectural guarantee and two empirical properties:

1) **Ground-truth-free (guarantee).** Every detection signal derives from (a) the tool result the agent receives, (b) the rolling trust-phase session history, and (c) structural properties of JSON, never the pre-injection server response.
2) **Lightweight (empirical property).** Tiers 1–2 complete in the order of milliseconds with zero LLM calls, adding negligible overhead to the MCP call path. Tier 3 issues one LLM call per qualifying result and can run asynchronously to avoid blocking the MCP call path.
3) **Runtime-agnostic (empirical property).** At inference time, SHIELD consults only the observed response and the session baseline: no attack label or taxonomy category is looked up while scoring a call. Tier 1's vocabulary is informed by our taxonomy at construction time, disclosed rather than hidden; Tier 2, which accounts for most detections, references no mechanism or category.

B. Detection Pipeline

1) Tier 1: Structural Anomaly Detection: An $O(n)$ scan of the serialized result, using no session history and completing in under 0.1 ms, flags: (a) keys from a documented signature list; (b) an empty result where the baseline was non-empty; (c) fields nulled that were consistently populated during the trust phase; and (d) a response that omits the requested entity outright, or errors on an entity previously resolved in-session. Checks (b)–(d) each consult one baseline bit, not the full statistical profile. Tier 1 is a low-cost first-pass filter; Tier 2 provides the primary detection signal.

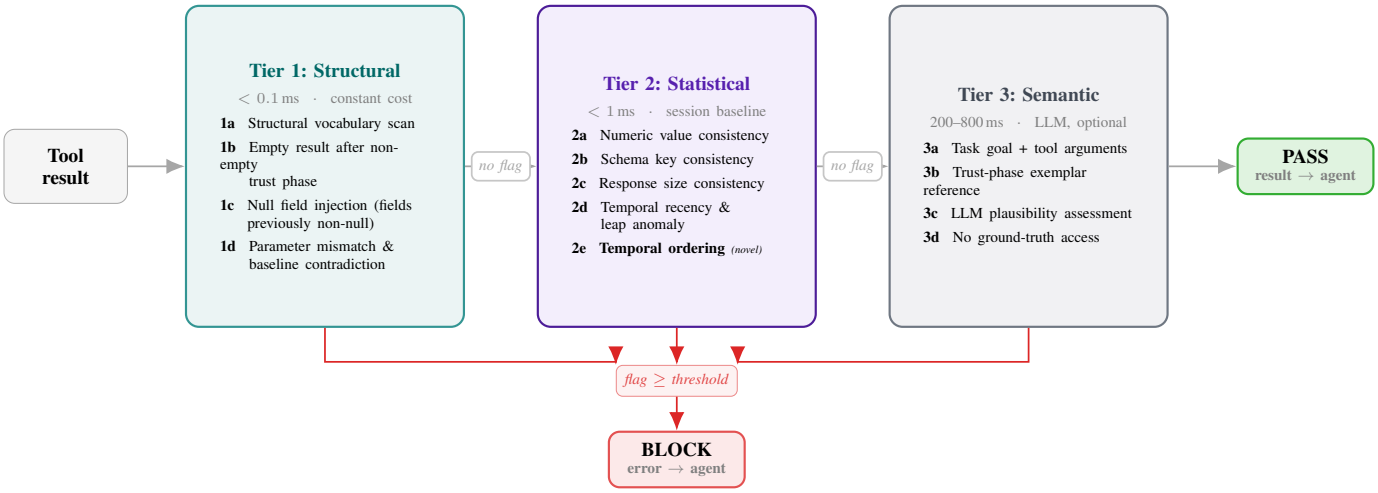

Fig. 3. SHIELD three-tier detection pipeline. Tiers execute in order of increasing cost; if a tier raises a flag meeting the severity threshold, the pipeline blocks immediately and returns a synthetic error to the agent (red path), otherwise the result passes to the next tier (*no flag*). Tier 1 applies low-cost structural and presence checks; Tier 2 compares against the session baseline built from trust-phase calls; Tier 3 adds a single LLM call and executes as part of the detection process.

2) Tier 2: Statistical Consistency: Against the session baseline, SHIELD flags numeric values deviating beyond a calibrated multiple of the trust-phase interquartile range, schema keys added or removed, record counts outside the expected range, and temporal irregularities relative to the trust-phase history. Thresholds are calibrated on a held-out split, independent of the evaluation tasks.

3) Tier 3: Semantic Plausibility: Structural and statistical checks miss corruption that stays in-range, including gradual drift and factually wrong but syntactically valid entities. Tier 3 issues one LLM call comparing the current response against a trust-phase exemplar, the task goal, and the requested entity, returning CLEAN, SUSPICIOUS, or COMPROMISED, which map onto the same severity scale used by Tiers 1–2.

A call is blocked once any tier's flag reaches the configured severity threshold; Tier 3 shares this rule rather than deciding independently, ensuring anomalous payloads are intercepted before reaching the agent's context window.

a) Summary.: SHIELD inspects response *content*, not access rights, and does not assume deploy-time benignity persists. Tier 3 extends MindGuard's [59] LLM-as-judge approach with a trust-phase exemplar, letting the judge detect drift rather than isolated anomaly. Needing no oracle, SHIELD tracks behavioral consistency across interactions rather than rendering a verdict at a single point.

VIII. EVALUATION AND RESULT

A. Model Setup

Models. We evaluate six LLM agents spanning proprietary and open-weight families: GPT-5, GPT-4.1, and o4-mini (OpenAI), Claude-Opus-4-8 (Anthropic), Grok-4.3 (xAI), and Qwen3.5 Flash (Alibaba). Every model is driven through the identical ReAct agent loop with the same task pool, tool backends, and SHIELD configuration, so differences reflect the model rather than the harness. Decoding uses temperature $= 0$ for reproducibility (top_p left at each provider's default of 1.0, max_output_tokens $= 4096$, step budget $= 15$ tool calls); models that do not expose a temperature control are queried at their default sampling setting. Exact model snapshots are pinned for reproducibility: GPT-5: `gpt-5-2025-08-07`; GPT-4.1: `gpt-4.1-2025-04-14`; o4-mini: `o4-mini-2025-04-16`; Grok-4.3: `grok-4.3` (xAI release date 2026-04-17); Claude-Opus-4-8: `claude-opus-4-8`; Qwen3.5 Flash: `Qwen/Qwen3.5 Flash`.

B. Parameters

The trust horizon N is set per task, using either a fixed onset for a deterministic switch point or a stochastic onset drawn from a per task interval, so the results do not depend on a single onset value. Each session runs to a budget of 15 tool calls. All three detection tiers, including the Tier 3 LLM judge, evaluate every call that reaches them regardless of the configured threshold; the threshold instead sets the minimum flag severity that triggers a block (HIGH: only high-severity flags; MEDIUM: medium and high; LOW: any flag). The headline `+Shield` results use a MEDIUM threshold and the deterministic Tier 1 / Tier 2 flags central to this work. A broader evaluation using the HIGH and LOW thresholds is left for future work.

C. Results and Findings

Table III reports ASR (%) by task domain and Table IV by TrustShift attack variant, both before and after SHIELD. We analyze the latter per variant and summarize the attack performance in Figure 4.

a) Finding 1: TrustShift subverts every frontier model.: No model is robust. The base ASR ranges from 60.2% (GPT-5) to 74.1% (GPT-4.1), with an overall mean of 69.5% (Table III). Even the most resistant model is deceived on roughly three

TABLE III
ASR (%) ON TRUSTSHIFTPROBE ACROSS DOMAINS AND ALL TASKS BEFORE AND AFTER DEFENSE (SHIELD). LOWER IS BETTER. EXACT MODEL CHECKPOINTS ARE LISTED IN THE MODEL SETUP SECTION OF THIS WORK.

Model	**Location Navigation**		**Repository Management**		**Financial Analysis**		**Browser Automation**		**Overall**	
	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓
GPT-5	61.6	56.6	49.4	48.9	64.4	33.3	65.4	23.1	60.2	40.7
GPT-4.1	66.3	42.3	78.4	66.7	75.6	30.1	76.5	28.6	74.1	41.0
o4-mini	61.0	45.8	58.2	53.1	84.0	34.2	72.0	26.0	69.2	39.2
Grok-4.3	76.5	56.6	79.8	72.9	75.6	21.1	62.3	17.1	73.8	42.2
Claude-Opus-4-8	55.3	47.4	76.2	71.4	68.2	34.1	85.1	38.4	70.9	48.1
Qwen3.5 Flash	61.8	54.1	77.0	73.3	76.2	21.2	62.8	27.8	69.5	44.5
Avg (Micro)	**63.7**	**50.7**	**70.0**	**64.7**	**73.8**	**28.9**	**70.6**	**26.8**	**69.5**	**42.7**

in five tasks. Because a cold, first-call version of the same manipulation is far easier to refuse, this level of success is direct evidence that the benign trust phase, not the payload, is what disarms the agent.

b) Finding 2: A sharp objective asymmetry: disruption succeeds, exfiltration fails.: Aggregating mechanisms by their adversarial objective reveals a stark behavioral dichotomy across all foundation models. Attacks targeting data integrity and operational disruption systematically compromise agents, whereas exfiltration and scope expansion attempts routinely fail. This asymmetry stems from a fundamental gap in modern safety alignment. Models implicitly trust authenticated tool content, seamlessly ingesting corrupted data as ground truth. Conversely, instruction-tuned models are explicitly trained to resist indirect command injections, allowing them to reliably ignore exfiltration instructions embedded within tool outputs. Consequently, the primary practical threat of TrustShift attacks is silent workflow sabotage, not agent hijacking, indicating that future defensive architectures must prioritize rigorous semantic and structural data verification.

c) Finding 3: Scope Boundary for Availability Attacks: SHIELD substantially reduces integrity-directed ASR but has one principled scope boundary. Against Stateful Service Denial, an availability attack, SHIELD detects the withheld or empty response but cannot reconstruct data that the compromised server never returned. Availability restoration against a single malicious source is unachievable for any oracle-free, single-channel monitor and would require server redundancy or a trusted oracle, both of which SHIELD deliberately forgoes. SHIELD does not restore task completion, but it transforms a silent and potentially unbounded agentic denial-of-service into a fast, attributed fail-closed halt, mitigating resource exhaustion and non-attribution harms even when task success is fundamentally unrecoverable.

Entity Spoofing remains the most successful corruption attack after defense ($90.6\% \rightarrow 74.0\%$) because it returns genuine data for a decoy entity: the payload is schema-valid and internally consistent, offering no structural or statistical footprint an oracle-free detector could flag. Together these bound what any transport-layer, ground-truth-free defense can achieve.

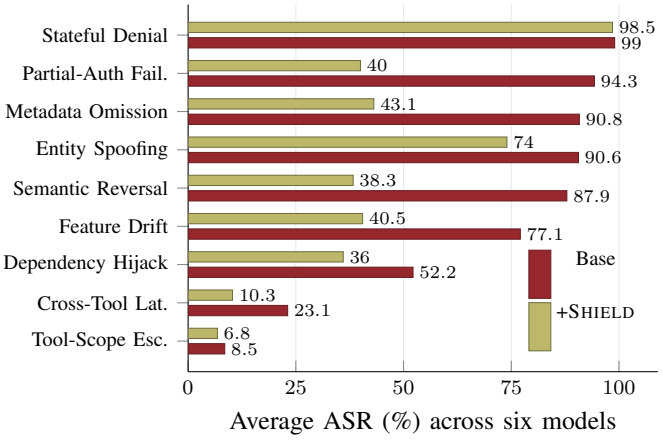

Fig. 4. Averaged ASR per mechanism across six models, before (**Base**) and after (**+SHIELD**). SHIELD roughly halves the ASR of most mechanisms, but two resist it: Stateful Service Denial and Entity Spoofing. These remain the dominant residual threats.

d) Finding 4: SHIELD *is most effective where the manipulation leaves a footprint.:* The defense roughly halves the mechanisms with detectable structural or semantic anomalies: Metadata Omission ($90.8 \rightarrow 43.1$), Partial-Auth ($94.3 \rightarrow 40.0$), Semantic Reversal ($87.9 \rightarrow 38.3$), and Continuous Feature Drift ($77.1 \rightarrow 40.5$). This is the dual of Finding 3: SHIELD turns its structural and semantic tiers into large reductions precisely when the adversary must perturb observable structure.

e) Finding 5: Domain matters; Repository Management is the hard case.: SHIELD yields the largest reductions on Browser Automation ($70.6 \rightarrow 26.8$) and Financial Analysis ($73.8 \rightarrow 28.9$), but barely moves Repository Management ($70.0 \rightarrow 64.7$), which remains the most vulnerable domain after defense. Large, many-field repository payloads give omission and drift attacks more cover and make behavioral baselines noisier.

f) Finding 6: Defensibility does not track raw capability.: The lightweight o4-mini ends with the lowest residual ASR under SHIELD (39.2%), closely followed by GPT-5 (40.7%), while GPT-4.1 sees the largest absolute reduction (-33.1 points, $74.1\% \rightarrow 41.0\%$). Meanwhile, Claude-Opus-4-

TABLE IV

ATTACK SUCCESS RATE (ASR, %) OF THE EVALUATED LLMS ACROSS ATTACK VARIATIONS DERIVED FROM THE TRUSTSHIFT TAXONOMY BEFORE AND AFTER DEFENSE (SHIELD). EXACT MODEL CHECKPOINTS ARE LISTED IN EVALUATION AND RESULT SECTION.

Attack Variation	**GPT-5**		**GPT-4.1**		**o4-mini**		**Grok-4.3**		**Claude-Opus-4-8**		**Qwen3.5 Flash**	
	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓	Base↓	+Shield↓
M1: Structural Violation												
Stateful Service Denial	94.3	94.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	97.0	100.0	100.0
Dependency Hijack	42.5	42.5	69.7	58.3	45.9	43.8	90.0	22.5	27.5	27.5	40.0	25.0
Verification Metadata Omission	97.4	45.0	100.0	50.0	100.0	28.0	75.0	52.5	75.0	25.0	100.0	52.5
Partial Authorization Failure	70.0	47.4	100.0	0.0	96.9	3.3	100.0	47.5	100.0	75.0	100.0	50.0
M2: Semantic Corruption												
Entity Spoofing	92.1	76.5	79.5	69.7	94.4	72.7	92.1	71.4	91.4	72.7	94.6	81.3
Semantic Reversal	67.5	22.5	92.3	48.5	81.1	30.0	92.3	38.5	100.0	50.0	94.9	42.5
Continuous Feature Drift	74.3	40.0	85.7	42.9	68.6	37.1	74.3	37.1	80.0	42.9	80.0	42.9
M3: Scope Expansion												
Tool Scope Escalation	7.9	7.9	8.6	5.7	10.0	7.1	11.4	11.4	8.6	3.2	5.3	5.3
Cross-Tool Lateral Movement	0.0	0.0	32.5	0.0	16.1	10.7	26.3	7.9	54.8	39.3	13.5	11.1
Avg (Micro)	**60.2**	**40.7**	**74.1**	**41.0**	**69.2**	**39.2**	**73.8**	**42.2**	**70.9**	**48.1**	**69.5**	**44.5**

8 plateaus at 48.1% residual ASR under SHIELD, making it the least-defensible model in our study.

g) Finding 7: SHIELD*'s reductions are statistically significant across models and mechanism families.:* To address the high variance from the small exposed samples of Scope Expansion (M3) attacks, we apply exact McNemar's tests to the paired session outcomes. Aggregating by family, SHIELD yields significant ASR reductions for both Semantic Corruption (M2) and Scope Expansion (M3) ($p < 0.0001$ for both). Thus, although individual variants such as Tool-Scope Escalation have low baseline exposure due to the agents' natural refusal rates, the M3 mitigation is significant at the family level rather than an artifact of small-sample noise. At the agent level, the absolute ASR reduction is likewise significant for every model tested (GPT-5, GPT-4.1, o4-mini, Grok-4.3, Claude-Opus-4-8, Qwen3.5 Flash), indicating that SHIELD's effect holds across sample variance and model architectures.

TABLE V

SHIELD'S FALSE POSITIVE RATE (FPR) AND CLASSIFICATION ACCURACY PER MECHANISM (MEDIUM THRESHOLD), POOLED ACROSS SIX MODELS (GPT-5, GPT-4.1, O4-MINI, GROK-4.3, CLAUDE-OPUS-4-8, QWEN3.5-FLASH).

Mechanism	**FPR**	**Accuracy**
Verification Metadata Omission	1.0%	60.7%
Partial Authorization Failure	1.3%	80.1%
Dependency Hijack	1.9%	24.5%
Continuous Feature Drift	2.8%	67.6%
Semantic Reversal	7.6%	73.3%
Entity Spoofing	10.1%	75.6%
Tool Scope Escalation	10.6%	76.2%
Stateful Service Denial	11.8%	76.9%
Cross-Tool Lateral Movement	15.5%	73.5%
Mean (call-weighted)	**7.7%**	**66.2%**

IX. DISCUSSION

A. Detection Asymmetry and Attack Footprints

TrustShift is a data-plane taxonomy: every attack variant corrupts what a tool returns while leaving its description, schema, and permissions untouched. Definition-plane attacks, mutated descriptions, manifest tampering, are the separate, already-studied surface that existing static scanners target; TrustShift covers the surface that scanners cannot see.

All four evaluated domains are read-heavy retrieval tasks; none involve write-heavy or irreversible actions (a submitted transaction, a merged pull request). Since the central finding is that a benign trust phase enables later deception, write-heavy domains likely raise the stakes of the same vulnerability rather than introduce a new one, but that remains to be shown.

Table V reports SHIELD's false positive rate (FPR) and classification accuracy per mechanism at the MEDIUM threshold, pooled across all six evaluated models. The call-weighted mean FPR is a low 7.7%, confirming that SHIELD rarely misflags benign trust-phase traffic; FPR ranges from 1.0% on Verification Metadata Omission to 15.5% on Cross-Tool Lateral Movement. Classification accuracy is far less uniform, averaging 66.2% but varying by more than 55 points across mechanisms, from 24.5% on Dependency Hijack to 80.1% on Partial Authorization Failure. This asymmetry, a uniformly low FPR paired with mechanism-dependent accuracy, is consistent with Findings 3 and 4: SHIELD's detectors are conservative about flagging clean traffic, but correctly classifying a manipulated response still depends on how much structural or statistical footprint that particular mechanism leaves.

B. Responsible Disclosure and Dual-Use Considerations

Publishing working implementations of nine attack variants is dual-use by construction. The marginal risk is bounded by one property: no mechanism is novel. Each traces to a pattern

already documented in public disclosures (tool poisoning [9], credential exfiltration [20], line-jumping [50]), independently confirmed at scale by MCPTox [26]. TrustShift systematizes and evaluates known behavior rather than introducing a new attack surface.

C. Why the Temporal Dimension Matters

A natural objection: if an attack ultimately succeeds, does its onset timing matter? Yes. Temporal staging governs the agent's accumulating trust as much as it governs the defender's operational viability.

TRUSTSHIFT corrupts a tool's runtime *result* while leaving its *definition* untouched, so static scanners and one-time approval reviews never observe it; only a continuous baseline monitor such as SHIELD can. This creates a temporal paradox: such monitors must learn "normal" behavior from early calls, so a delayed attack leaves an uncorrupted reference that *enables* detection, while an immediate one risks poisoning the baseline itself. An attacker forced to behave benignly during deployment vetting thereby supplies the very baseline later used to catch it.

Detection therefore depends on the alignment of attack onset, baseline window, and corruption gradient. A patient, *low-and-slow* adversary can escalate drift slowly enough to keep each step under the detection threshold. Quantifying this as ASR (before and after SHIELD) over onset hop N and drift magnitude Δ is a natural next experiment; our framework's configurable trust-phase length and drift controls make it possible.

D. Domain-Relative Calibration and Adaptation

To achieve domain generality without a ground-truth oracle, SHIELD dynamically calibrates its reference baselines (e.g., schema, field precision, response size) directly from each session's trust-phase evidence. This dynamic profiling allows a single implementation, using uniform sensitivity thresholds, to operate across heterogeneous domains without requiring per-tool tuning.

However, accommodating legitimate domain variance requires minor behavioral carve-outs. For example, a single nulled field in a financial record or a transient, recovering API error is treated as normal operational variance rather than adversarial drift. Consequently, while SHIELD eliminates the need for hardcoded oracles, porting the defense to entirely novel classes of MCP servers carries a minor, one-time domain-adaptation cost to establish these acceptable behavioral boundaries.

E. Guarding Against Evaluation-Artifact Leakage

A defense evaluated against a benchmark's own attack instrumentation risks reporting a detection rate that reflects recognizing the harness rather than the threat. Our denial-of-service attacks emit an internal error marker to simulate a dropped call; because a real API outage produces an indistinguishable generic error envelope, keying detection on our own watermark would inflate the reported rate without producing a signal that transfers to a real deployment. SHIELD excludes this marker from its structural vocabulary and instead targets denial behaviorally, from persistence and selectivity of failure across calls, a content-agnostic signal available to any monitor regardless of which harness produced the failure.

X. CONCLUSION AND FUTURE WORK

We introduced TRUSTSHIFTPROBE, a benchmark and taxonomy for temporally-staged MCP tool compromise: a server that behaves honestly during a trust-building conditioning phase before defecting via structural (M1), semantic (M2), or scope-expanding (M3) mutations. Across six frontier agents, we find substantial susceptibility to attack variants. We observe that and mechanism-dependent susceptibility, and show that SHIELD, our oracle-free transport-layer defense, meaningfully reduces deception-style attacks while remaining structurally unable to prevent (only detect) pure denial-of-service defections: an integrity–availability asymmetry we argue is inherent to any ground-truth-free monitor, not an artifact of our implementation. Natural extensions include multi-server coordinated attacks, adaptive adversaries that tune their drift rate against SHIELD's own baseline window, and defenses that move beyond purely oracle-free detection toward lightweight verification. We leave these to future work.

AI DISCLOSURE

In this work, we employ large language models (LLMs) primarily as agents for task execution and evaluation. Specifically, LLMs are used to instantiate tasks, interact with MCP servers in multi-step workflows, generate reasoning traces for analysis, and assist with editing and refining the manuscript's text. The authors rigorously reviewed, edited, and verified the correctness, originality, and integrity of all AI-generated text and code, including all academic references.

REFERENCES

[1] M. A. Ferrag, N. Tihanyi, and M. Debbah, "From LLM reasoning to autonomous AI agents: A comprehensive review," *IEEE Access*, 2026.

[2] J. Kutasov, Y. Sun, P. Colognese, *et al.*, "Shade-Arena: Evaluating sabotage and monitoring in LLM agents," *arXiv:2506.15740*, 2025.

[3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.

[4] J. R. Asl, S. Narula, M. Ghasemigol, E. Blanco, and D. Takabi, "NEXUS: Network exploration for exploiting unsafe sequences in multi-turn LLM jailbreaks," in *Proc. 2025 Conf. Empirical Methods Nat. Lang. Process. (EMNLP)*, Nov. 2025, pp. 24278–24306.

[5] Anthropic, "Introducing the Model Context Protocol," Anthropic, Nov. 2024. [Online]. Available: https://www.anthropic.com/news/model-context-protocol

[6] OpenAI, "Building with MCP," OpenAI, 2025. [Online]. Available: https://platform.openai.com/docs/mcp/

[7] Cursor, "Model Context Protocol (MCP)," Cursor Documentation, 2025. [Online]. Available: https://docs.cursor.com/context/mcp

[8] Google, "Gemini CLI: Your open-source AI agent," Google Developers Blog, 2025. [Online]. Available: https://blog.google/technology/developers/introducing-gemini-cli

[9] L. Beurer-Kellner and M. Fischer, "MCP security notification: Tool poisoning attacks," Invariant Labs Blog, Apr. 2025. [Online]. Available: https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

[10] S. Zhao, Q. Hou, Z. Zhan, Y. Wang, Y. Xie, Y. Guo, L. Chen, S. Li, and Z. Xue, "Parasites in the toolchain: A large-scale analysis of attacks on the MCP ecosystem," in *Proc. 2026 IEEE Symp. Security and Privacy (SP)*, May 2026, pp. 138–155.

[11] M. Russinovich, A. Salem, and R. Eldan, "Great, now write an article about that: The Crescendo multi-turn LLM jailbreak attack," in *Proc. 34th USENIX Secur. Symp. (USENIX Security)*, 2025.

[12] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 12697–12706.

[13] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024.

[14] P. Abadeh, M. Lochner, T. Ansari, and F. Zarrinkalam, "MCP-Scanner: Detecting security risks in Model Context Protocol systems," in *Proc. 2026 ACM/IEEE 7th Int. Workshop Eng. Cybersecurity Crit. Syst.*, Apr. 2026, pp. 41–48.

[15] N. Yang, G. Lyu, M. Ma, Y. Lu, Y. Li, Z. Gao, *et al.*, "IoT-MCP: Bridging LLMs and IoT systems through Model Context Protocol," in *Proc. ACM Workshop Wireless Netw. Testbeds, Exp. Eval. Charact. (WiNTECH)*, Nov. 2025, pp. 73–80.

[16] A. Charikov, R. Zaikin, and O. Vanunu, "MCPoison Cursor IDE: Persistent code execution via MCP trust bypass," Check Point Research, 2025. [Online]. Available: https://research.checkpoint.com/2025/cursor-vulnerability-mcpoison/

[17] I. Dardikman, "First malicious MCP in the wild: The Postmark backdoor that's stealing your emails," Koi Research Blog, 2025. [Online]. Available: https://www.koi.ai/blog/postmark-mcp-npm-malicious-backdoor-email-theft

[18] G. Ferry, "From path traversal to supply chain compromise: Breaking MCP server hosting," GitGuardian Blog, 2025. [Online]. Available: https://blog.gitguardian.com/breaking-mcp-server-hosting/

[19] Z. Wang, H. Li, R. Zhang, Y. Liu, W. Jiang, W. Fan, Q. Zhao, and G. Xu, "MPMA: Preference manipulation attack against Model Context Protocol," *arXiv:2505.11154*, 2025.

[20] B. Radosevich and J. Halloran, "MCP safety audit: LLMs with the Model Context Protocol allow major security exploits," *arXiv:2504.03767*, 2025.

[21] S. Kumar, A. Girdhar, R. Patil, and D. Tripathi, "MCP Guardian: A security-first layer for safeguarding MCP-based AI system," *arXiv:2504.12757*, 2025.

[22] N. Croce and T. South, "Trivial trojans: How minimal MCP servers enable cross-tool exfiltration of sensitive data," *arXiv:2507.19880*, 2025.

[23] O. Narayan, R. Singh, and P. Baskar, "A comprehensive security framework for the Model Context Protocol (MCP) in multi-agent AI systems," in *Proc. 2025 8th Int. Conf. Algorithms, Comput. and AI (ACAI)*, 2025, pp. 1–6.

[24] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): Landscape, security threats, and future research directions," *arXiv:2503.23278*, 2025.

[25] X. Zong, Z. Shen, L. Wang, Y. Lan, and C. Yang, "MCP-SafetyBench: A benchmark for safety evaluation of large language models with real-world MCP servers," *arXiv:2512.15163*, 2025.

[26] Z. Wang *et al.*, "MCPTox: A benchmark for tool poisoning attack on real-world MCP servers," *arXiv:2508.14925*, 2025.

[27] D. Zhang, Z. Li, X. Luo, X. Liu, P. Li, and W. Xu, "MCP Security Bench (MSB): Benchmarking attacks against Model Context Protocol in LLM agents," *arXiv:2510.15994*, 2025.

[28] J. Fang, Z. Yao, R. Wang, H. Ma, X. Wang, and T.-S. Chua, "We should identify and mitigate third-party safety risks in MCP-powered agent systems," *arXiv:2506.13666*, 2025.

[29] H. Jing, H. Li, W. Hu, Q. Hu, H. Xu, T. Chu, P. Hu, and Y. Song, "MCIP: Protecting MCP safety via Model Contextual Integrity Protocol," *arXiv:2505.14590*, 2025.

[30] W. Xing, Z. Qi, Y. Qin, Y. Li, C. Chang, J. Yu, C. Lin, Z. Xie, and M. Han, "MCP-Guard: A multi-stage defense-in-depth framework for securing Model Context Protocol in agentic AI," *arXiv:2508.10991*, 2025.

[31] Y. Yang, D. Wu, and Y. Chen, "MCPSecBench: A systematic security benchmark and playground for testing Model Context Protocols," *arXiv:2508.13220*, 2025.

[32] H. Joren, J. Zhang, C.-S. Ferng, D.-C. Juan, A. Taly, and C. Rashtchian, "Sufficient context: A new lens on retrieval augmented generation systems," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.

[33] M. Rostamzadeh, S. Narula, N. Birhan, M. Ghasemigol, and D. Takabi, "MCP-DPT: A defense-placement taxonomy and coverage analysis for Model Context Protocol security," *arXiv:2604.07551*, 2026.

[34] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, and C. Wang, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *arXiv:2308.08155*, 2023.

[35] JSON-RPC Working Group, "JSON-RPC 2.0 specification," 2013. [Online]. Available: https://www.jsonrpc.org/specification

[36] S. Narula, M. Ghasemigol, J. Carnerero-Cano, A. Minnich, E. Lupu, and D. Takabi, "Exploring research and tools in AI security: A systematic mapping study," *IEEE Access*, vol. 13, pp. 84057–84080, 2025.

[37] N. Hardy, "The confused deputy: (Or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, 1988.

[38] V. S. Narajala and I. Habler, "Enterprise-grade security for the Model Context Protocol (MCP): Frameworks and mitigation strategies," *arXiv:2504.08623*, 2025.

[39] Y. Wang, Y. Pan, S. Guo, and Z. Su, "Security of Internet of Agents: Attacks and countermeasures," *IEEE Open Journal of the Computer Society*, 2025.

[40] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. U. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems," in *Proc. 11th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 2014, pp. 249–265.

[41] OWASP, "OWASP API Security Top 10 2023 — API1:2023 Broken Object Level Authorization," 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/. Accessed: Aug. 12, 2026.

[42] I. T. A. Lee, Z. Zhang, A. Parwal, and M. Chabbi, "The tale of errors in microservices," *Proc. ACM Meas. Anal. Comput. Syst. (POMACS)*, vol. 8, no. 3, art. 46, 2024.

[43] O. Brodt, E. Feldman, B. Schneier, and B. Nassi, "The promptware kill chain: How prompt injections gradually evolved into a multistep malware delivery mechanism," *arXiv:2601.09625*, 2026.

[44] H. Y. Fu, A. Shrivastava, J. Moore, P. West, C. Tan, and A. Holtzman, "AbsenceBench: Language models can't tell what's missing," *arXiv:2506.11440*, 2025.

[45] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in *Proc. 2023 IEEE Symp. Security and Privacy (SP)*, 2023, pp. 1509–1526.

[46] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 995–1010.

[47] E. Perez, S. Huang, F. Song, *et al.*, "Red teaming language models with language models," in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2022, pp. 3419–3448.

[48] Ł. Korycki and B. Krawczyk, "Adversarial concept drift detection under poisoning attacks for robust data stream mining," *Machine Learning*, vol. 112, no. 10, pp. 4013–4048, 2023.

[49] O. R. Rostami, R. Ning, C. Xin, J.-H. Cho, J. Li, and H. Wu, "GhostBackdoor: A resistant backdoor attack," *IEEE Internet of Things Journal*, 2026.

[50] Trail of Bits, "Jumping the line: How MCP servers can attack you before you ever use them," Trail of Bits Blog, Apr. 2025. [Online]. Available: https://blog.trailofbits.com/2025/04/21/jumping-the-line-how-mcp-servers-can-attack-you-before-you-ever-use-them/

[51] Invariant Labs, "MCP-Scan: Model Context Protocol scanner," 2025. [Online]. Available: https://github.com/invariantlabs-ai/mcp-scan

[52] MCPScan.ai, "MCPScan.ai: Enterprise Model Context Protocol security platform," 2025. [Online]. Available: https://mcpscan.ai

[53] Cisco AI Defense, "MCP Scanner: Security analysis for Model Context Protocol artifacts," 2025. [Online]. Available: https://github.com/cisco-ai-defense/mcp-scanner

[54] Rise and Ignite, "MCP-Shield: Security scanner for Model Context Protocol servers," 2025. [Online]. Available: https://github.com/riseandignite/mcp-shield

[55] MCP-Defender, "MCP-Defender: Runtime protection for Model Context Protocol," 2025. [Online]. Available: https://mcpdefender.com/

[56] Lasso Security, "MCP-Gateway: Secure proxy and orchestration layer for Model Context Protocol," 2025. [Online]. Available: https://github.com/lasso-security/mcp-gateway

[57] Stacklok, "ToolHive: Simplify and secure Model Context Protocol servers," 2025. [Online]. Available: https://docs.stacklok.com/toolhive/
[58] eqtylab, "MCP Guardian: Enterprise access control and governance for Model Context Protocol," 2025. [Online]. Available: https://github.com/eqtylab/mcp-guardian
[59] Z. Wang, J. Zhang, G. Shi, H. Cheng, Y. Yao, K. Guo, H. Du, and X.-Y. Li, "MINDGUARD: Tracking, detecting, and attributing MCP tool poisoning attack via decision dependence graph," *arXiv:2508.20412*, 2025.
[60] AIM Intelligence, "AIM-Guard-MCP: AI-powered security guard for Model Context Protocol," 2025. [Online]. Available: https://github.com/AIM-Intelligence/AIM-MCP