Tool Security Advisory (TSA)

A Security Advisory Standard for MCP Tool Ecosystems

Version 1.0.0 (Draft)

Jason M. Lovell

Independent Researcher

jase.lovell@me.com

ORCID: 0009-0001-6300-9155

February 2026

Abstract

The Model Context Protocol (MCP) enables AI agents to invoke tools that can read files, access networks, execute processes, and integrate with external services. These tools introduce security risks that are not fully captured by traditional vulnerability formats. This paper specifies the Tool Security Advisory (TSA) standard, a strict JSON advisory format with a feed model and trust framework designed for MCP tool ecosystems. TSA adds MCP-specific primitives such as semantic drift, capability abuse, agent execution context, and Tool Bill of Materials (TBOM) content hash binding. TSA also defines deterministic canonicalization and cryptographic signing for integrity and provenance, and provides interoperability with OSV through loss-minimized mappings. A reference implementation and SDK are available for evaluation and early adoption.

Status of This Document

This document defines a community specification for MCP tool security advisories. It is independent of the MCP core protocol but designed to integrate with MCP registries and host environments. Distribution of this document is unlimited. Comments and suggestions should be directed to the working group.

Contents

1. Executive Summary

MCP tools execute inside AI agent workflows where a single vulnerability can enable prompt injection pivots, data exfiltration, and unauthorized actions at scale. Existing advisory formats (CSAF, OSV, VEX) provide structure but do not encode the MCP-specific attributes required for automated enforcement. TSA addresses these gaps with a strict JSON schema, a lightweight feed index, deterministic hashing, and signature-based trust anchors.

TSA introduces MCP-native fields that capture semantic drift, capability abuse, agent execution context, and TBOM content hash binding. These fields allow advisories to model the actual security posture of AI tools rather than only package metadata. TSA also defines enforcement actions (`BLOCK`, `WARN`, `UPDATE`, `INVESTIGATE`, `REVOKE`) so registries and hosts can apply security policy programmatically.

The standard is operationally practical: publishers can generate advisories with a CLI, consumers can enforce actions with a registry SDK, and feeds enable low-latency synchronization. Consumers can require signatures for `BLOCK` enforcement and downgrade unsigned advisories to `WARN` by policy. TSA is interoperable with OSV to integrate with existing vulnerability pipelines.

2. Introduction

2.1 Motivation

MCP tools are invoked inside AI agent workflows. A vulnerability or malicious change in a tool can result in immediate operational impact: unauthorized execution, data leakage, policy bypass, and supply-chain compromise. These tools often execute with explicit capabilities (filesystem, network, process execution) and are frequently composed in chains that amplify risk.

Traditional vulnerability formats were built for conventional software. They do not encode several MCP-specific attributes that directly affect risk and enforcement decisions:

- Tool identity beyond package name (content-hash binding)
- Semantic drift across versions
- Explicit capability abuse
- Agent execution context and prerequisites
- Enforcement semantics for registries and hosts

TSA addresses these gaps while remaining compatible with existing vulnerability infrastructure.

2.2 Scope

This document specifies:

- Security vulnerabilities in MCP tools (servers, clients, transports)
- Behavioral anomalies, including semantic drift
- Integrity concerns and supply-chain compromise
- Automated enforcement actions (`BLOCK`/`WARN`/`UPDATE`/`REVOKE`)
- Cryptographic integrity and provenance

The following are explicitly **out of scope**:

- Vulnerabilities in LLMs or AI models themselves
- General software vulnerabilities unrelated to MCP tools
- Compliance or policy attestations (use CSAF/VEX or policy-specific formats)

2.3 Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHOULD**", "**SHOULD NOT**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119 [1].

Advisory A structured document describing a security issue affecting one or more MCP tools.
Canonical Payload The advisory JSON with `signature` and `canonical_hash` fields removed, used for computing hashes and signatures.
Feed A lightweight index of advisories for efficient synchronization.
Publisher An entity that creates and signs advisories.
Consumer A registry, host, gateway, or security system that ingests and enforces advisories.
Trust Anchor A public key associated with a publisher, used to verify advisory signatures.
Semantic Drift Behavioral changes in a tool between versions that may indicate supply-chain compromise.
TBOM Tool Bill of Materials—a content hash and metadata binding a tool to its build artifacts.
Attack Context Contextual information about the agent execution environment where a vulnerability can be exploited (captured in the `attack_context` field).
Capability Abuse Misuse of legitimate tool capabilities beyond their intended purpose, often through prompt injection or agent manipulation.

2.4 Document Organization

Section 3 describes design requirements. Section 4 specifies the TSA data model. Section 5 covers MCP-specific extensions. Section 6 defines enforcement semantics. Section 7 describes the trust model. Section 8 covers the feed format. Section 9 addresses interoperability. Section 10 provides implementation guidance. Section 11 presents case studies. Section 12 offers deployment guidance. Section 13 discusses security considerations. Section 14 outlines future work. Section 15 concludes.

3. Requirements and Design Goals

TSA is designed to satisfy the following requirements:

1. **Machine-readable**: Advisory content **MUST** be structured and strict. Unknown fields **MUST** be rejected to guarantee consistent parsing.
2. **Verifiable**: Integrity **MUST** be checkable via canonicalization, hashing, and optional signatures.
3. **Interoperable**: TSA **SHOULD** integrate with OSV and remain compatible with established advisory ecosystems.
4. **MCP-native**: The standard **MUST** encode MCP-specific semantics (agent context, semantic drift, capability abuse, TBOM binding).
5. **Minimal and precise**: Required fields **SHOULD** be minimal; optional fields **MUST** be well-defined and unambiguous.

3.1 Non-Goals

The following are explicitly **not** goals of this specification:

- Replacing existing vulnerability databases or formats
- Defining a complete capability taxonomy (deferred to future work)
- Specifying policy languages or access control models
- Mandating specific cryptographic implementations beyond algorithm identifiers

4. TSA Data Model

TSA advisories are JSON documents conforming to a strict schema (JSON Schema Draft 2020-12). Unknown fields **MUST** be rejected (`additionalProperties: false`). This guarantees consistent parsing and reduces ambiguity for enforcement systems.

4.1 Required Fields

An advisory **MUST** include the following fields:

Field	Description
`tsa_version`	Specification version (e.g., “1.0.0”)
`id`	Unique identifier (format: `TSA-YYYY-NNNN`)
`published`	RFC 3339 timestamp of initial publication [2]
`modified`	RFC 3339 timestamp of last modification
`publisher`	Object containing publisher `name` and `namespace`
`title`	Human-readable summary of the advisory
`affected`	Array of affected tool entries
`actions`	Array of enforcement actions

Table 1: Required fields in a TSA advisory.

The tuple (`publisher.namespace`, `id`) forms the globally unique identifier for an advisory. Publishers **MUST** ensure that `id` values are unique within their namespace.

4.2 Optional Fields

Optional fields include:

- `description`: Extended description of the vulnerability
- `severity`: CVSS object with `score`, `vector`, and `version`
- `impact_statement`: Human-readable impact description
- `references`: Array of reference objects with `type` and `url`
- `related_vulnerabilities`: Array of related CVE/GHSA identifiers
- `workarounds`: Array of mitigation objects
- `credits`: Array of contributor objects
- `signature`: Cryptographic signature object (excluded from canonical payload)
- `withdrawn`: RFC 3339 timestamp if the advisory has been withdrawn
- `canonical_hash`: SHA-256 hash of the canonical payload (excluded from canonical payload)

4.3 Affected Tool Entries

Each entry in the `affected` array describes one tool and the range of impacted versions. An affected entry **MUST** include:

- `tool`: Object with `name`, `registry`, and optionally `purl`
- `versions`: Object with version range metadata
- `status`: One of `AFFECTED`, `NOT_AFFECTED`, `UNDER_INVESTIGATION`, or `FIXED`

The `versions` object **SHOULD** include `introduced`, `fixed`, `affected_range`, or `last_affected` as applicable. Consumers **MUST** interpret `affected_range` as a semver range expression conforming to the grammar defined in Appendix C.

4.4 Enforcement Actions

Actions encode machine-enforceable security decisions. Each action **MUST** include the base fields shown in Table 2. Additionally, certain action types require type-specific fields as specified in Table 3.

Field	Description
`type`	Action type: `BLOCK`, `WARN`, `UPDATE`, `INVESTIGATE`, or `REVOKE`
`scope`	Enforcement scope: `REGISTRY`, `HOST`, `GATEWAY`, or `ALL`
`urgency`	Priority level: `IMMEDIATE`, `HIGH`, `MEDIUM`, or `LOW`
`message`	Human-readable enforcement message

Table 2: Base fields required in all enforcement actions.

Type	Required Field	Description
`BLOCK`	`condition`	Semver range expression for matching versions
`WARN`	`condition`	Semver range expression for matching versions
`UPDATE`	`condition`	Semver range expression for matching versions
`UPDATE`	`target_version`	Recommended version to update to
`INVESTIGATE`	`condition`	Semver range expression for matching versions
`REVOKE`	`revoked_key_id`	Identifier of the compromised signing key
`REVOKE`	`replacement_key_id`	(Optional) Identifier of the replacement key

Table 3: Type-specific fields for enforcement actions.

Action types are defined as follows:

`BLOCK` Prevent installation or execution of affected versions. **MUST** include `condition`.

`WARN` Allow operation but surface a warning to users or operators. **MUST** include `condition`.

`UPDATE` Recommend updating to a specific target version. **MUST** include `condition` and `target_version`.

`INVESTIGATE` Flag for manual security review. **MUST** include `condition`.

`REVOKE` Indicate that a signing key has been compromised. **MUST** include `revoked_key_id`; **MAY** include `replacement_key_id`.

5. MCP-Specific Extensions

TSA extends traditional advisory formats with MCP-specific primitives that capture the unique risk profile of AI agent tools.

5.1 Semantic Drift

Semantic drift captures behavioral changes between tool versions. This is critical for MCP tools where behavior changes can introduce covert data exfiltration or capability expansion without any corresponding vulnerability disclosure.

An affected entry **MAY** include a `semantic_drift` object with:

- `description_changed`: Boolean indicating tool description modification
- `capabilities_changed`: Boolean indicating capability modification
- `input_schema_changed`: Boolean indicating input schema modification
- `details`: Human-readable description of the drift

TSA encodes drift explicitly, allowing consumers to flag changes even when no CVE exists. This enables proactive detection of supply-chain compromise.

5.2 Capability Abuse

TSA can enumerate `capabilities_abused` to describe the specific privileges or actions that are misused in an attack. This field is an array of capability identifiers following a namespace:action pattern (e.g., `network:oauth`, `process:exec`, `filesystem:write`).

This allows fine-grained policy responses. For example, a gateway might block tools with `process:exec` abuse while allowing tools with only `network:fetch` abuse to operate with warnings.

5.3 Attack Context

The `attack_context` field captures agent execution context—conditions under which a vulnerability can be exploited. An affected entry **MAY** include an `attack_context` object with:

- `requires_agent_execution`: Boolean indicating if agent execution is required
- `requires_user_interaction`: Boolean indicating if user interaction is required
- `requires_network_access`: Boolean indicating if network access is required
- `requires_specific_configuration`: Boolean indicating if specific configuration is required
- `prerequisites`: Array of string descriptions of other prerequisites

Consumers **SHOULD** tailor enforcement based on deployment context. For example, a host in an isolated environment might downgrade a BLOCK action if requires_network_access is true and no network access is available.

5.4 TBOM Content Hash Binding

Advisories **MAY** include tbom_binding with a content hash and signature key ID. This binds the advisory to a specific tool build rather than only a package name.

```json
{
  "tbom_binding": {
    "content_hash": "sha256:7d865e959b2466918c9863afca942d0fb89d7c9ac0c99bafc3749504ded97730",
    "signature_key_id": "mcp-security:key1",
    "tbom_version": "1.0.0"
  }
}
```

This reduces ambiguous attribution and strengthens supply-chain integrity by ensuring that an advisory applies to a specific, verifiable tool artifact.

6. Enforcement Semantics

TSA encodes enforcement semantics so automated systems can apply consistent policy. Consumers **MUST** interpret condition fields as semver ranges (see Appendix C) and **MUST** respect scope values.

6.1 Scope Interpretation

REGISTRY Action applies at package installation time. Registries **SHOULD** enforce BLOCK by preventing installation of affected versions.

HOST Action applies at runtime. Hosts **SHOULD** enforce WARN by displaying warnings before tool execution.

GATEWAY Action applies at the enterprise gateway level. Gateways **SHOULD** enforce policy based on organizational requirements.

ALL Action applies at all scopes. All consumers **SHOULD** enforce the action.

6.2 Urgency Levels

`IMMEDIATE` Critical severity requiring immediate action. Consumers **SHOULD** apply enforcement without delay.

`HIGH` High severity. Consumers **SHOULD** apply enforcement within 24 hours.

`MEDIUM` Moderate severity. Consumers **SHOULD** apply enforcement within 7 days.

`LOW` Low severity. Consumers **MAY** defer enforcement to the next maintenance window.

6.3 Example Enforcement Actions

Example `BLOCK` action:

```json
{
  "type": "BLOCK",
  "scope": "REGISTRY",
  "condition": ">=0.0.5 <0.1.16",
  "urgency": "IMMEDIATE",
  "message": "Critical RCE vulnerability. Update to 0.1.16 or later."
}
```

Example `UPDATE` action with required `target_version`:

```json
{
  "type": "UPDATE",
  "scope": "HOST",
  "condition": ">=0.0.5 <0.1.16",
  "target_version": "0.1.16",
  "urgency": "HIGH",
  "message": "Security update available. Upgrade to 0.1.16."
}
```

Example `REVOKE` action with required `revoked_key_id`:

```json
{
  "type": "REVOKE",
  "scope": "ALL",
  "revoked_key_id": "mcp-security:key1",
  "replacement_key_id": "mcp-security:key2",
  "urgency": "IMMEDIATE",
  "message": "Signing key compromised. Reject advisories signed with
     key1."
}
```

7. Trust Model and Integrity

7.1 Canonical Payload Definition

The **canonical payload** is defined as the advisory JSON with the `signature` and `canonical_hash` fields removed (if present). All hash computations and signature operations **MUST** be performed over the canonical payload, not the complete advisory.

This definition resolves the otherwise circular dependency where embedded signatures or hashes would include themselves in the computation input.

7.2 Canonicalization and Hashing

TSA uses RFC 8785 JSON Canonicalization Scheme (JCS) [7]. The canonical form of the canonical payload **MUST** be used as the input for both `canonical_hash` computation and signature generation. The `canonical_hash` field is always SHA-256; signature algorithms use their own internal hash functions as specified in RFC 7518 [5]. This separation ensures deterministic verification across implementations.

The hash computation procedure is:

1. Remove the `signature` field from the advisory (if present)
2. Remove the `canonical_hash` field from the advisory (if present)
3. Apply RFC 8785 canonicalization to the resulting JSON
4. Compute the SHA-256 hash of the canonical byte sequence

Consumers **MUST** verify the `canonical_hash` value in the feed entry after fetching an advisory. Hash mismatches **MUST** trigger rejection or quarantine of the advisory.

7.3 Signatures

Advisories **MAY** include signatures using one of the following algorithms (identifiers per JOSE [5]):

- `EdDSA` with Ed25519 (recommended for new implementations)
- `ES256` (ECDSA with P-256 and SHA-256)
- `ES384` (ECDSA with P-384 and SHA-384)
- `RS256` (RSASSA-PKCS1-v1_5 with SHA-256)

Note: Ed25519 denotes the elliptic curve and key type; `EdDSA` is the corresponding JOSE algorithm identifier per RFC 7518 [5]. CLI tooling may reference keys by curve name while the advisory format uses JOSE identifiers.

The `signature` object follows JWS conventions [4] and **MUST** include:

- `algorithm`: The signature algorithm identifier from the list above
- `key_id`: Identifier for the signing key
- `value`: Base64URL-encoded signature (RFC 4648 [3] Section 5, without padding)

The signature **MUST** be computed over the RFC 8785 canonical byte sequence of the canonical payload. The signature algorithm's internal hash function is applied to this byte sequence as specified by the algorithm (e.g., Ed25519 uses SHA-512 internally; ES256 uses SHA-256). Implementations **MUST NOT** conflate the `canonical_hash` field (always SHA-256) with the signature algorithm's internal hashing.

Consumers **SHOULD** enforce `BLOCK` actions only when the advisory is signed by a trusted key. If unsigned, the consumer **SHOULD** downgrade `BLOCK` to `WARN` by default.

7.4 Trust Anchors and Key Rotation

Consumers maintain a list of trusted publisher keys (trust anchors). Each trust anchor associates a `key_id` with a public key and publisher namespace.

Publishers **SHOULD** rotate keys periodically. Key rotation is signaled using a `REVOKE` action that includes the `revoked_key_id` field. Registries **MUST** reject or downgrade advisories signed by revoked keys.

Trust anchors **MAY** be distributed via:

- Out-of-band configuration
- Well-known endpoints per RFC 8615 [6] (e.g., `/.well-known/tsa-trust-anchors.json`)
- Feed metadata

8. Feed Model

A TSA feed is a lightweight index for efficient synchronization. Feeds enable consumers to discover new and updated advisories without polling individual advisory endpoints.

8.1 Feed Structure

A feed **MUST** include:

- `feed_version`: Feed schema version (e.g., "1.0.0")
- `generated`: RFC 3339 timestamp of feed generation
- `publisher`: Publisher object matching advisory publisher format

- `advisories`: Array of advisory index entries

Each advisory index entry **MUST** include:

- `id`: Advisory identifier
- `uri`: Relative or absolute URI to the advisory document
- `canonical_hash`: SHA-256 hash of the canonical payload (full 64 hex characters)
- `modified`: RFC 3339 timestamp of last modification

Entries **MAY** include `title`, `severity`, and `cve` for filtering without fetching full advisories.

8.2 Feed Synchronization

Consumers **MUST** verify `canonical_hash` values after fetching advisories. Consumers **SHOULD** warn on stale feeds (feeds not updated within a configurable threshold) or hash mismatches.

Feeds **MAY** include inline `advisory` payloads for single-file distribution. This simplifies deployment for small advisory sets.

9. Interoperability

9.1 OSV Mapping

TSA advisories can be converted to OSV format. MCP-specific fields are preserved under `database_specific` in the OSV output:

```
{
  "database_specific": {
    "tsa_id": "TSA-2025-0001",
    "semantic_drift": { },
    "capabilities_abused": [ ],
    "attack_context": { },
    "actions": [ ]
  }
}
```

When importing OSV to TSA, publishers **SHOULD** assign a canonical TSA ID and **SHOULD** preserve OSV identifiers under `related_vulnerabilities` or `references`.

9.2 CSAF and VEX Compatibility

While TSA does not directly map to CSAF or VEX, the semantic model is compatible. Organizations **MAY** reference TSA advisories from CSAF documents or internal policy workflows. The `references` field in TSA advisories can link to CSAF or VEX documents for organizations that maintain parallel advisory systems.

10. Reference Implementation

The TSA repository includes a reference implementation intended for evaluation and early adoption:

CLI (`tsactl`) Command-line tool for advisory operations: validate, canonicalize, hash, sign, verify, and match.
Registry SDK Python SDK for registry integration: sync feeds, verify signatures, enforce actions against tool inventories.
Feed Builder Utility for generating feed indexes from advisory directories.
OSV Converter Bidirectional converter between TSA and OSV formats.

10.1 CLI Usage Examples

```
# Validate an advisory
python3 tsactl.py validate advisory.tsa.json

# Generate signing keys
python3 tsactl.py generate-keys my-org --algorithm EdDSA

# Sign an advisory
python3 tsactl.py sign advisory.tsa.json my-org_private.pem \
    --key-id "my-org:key1"

# Verify a signed advisory
python3 tsactl.py verify advisory-signed.tsa.json my-org_public.pem

# Match against a tool inventory
python3 tsactl.py match advisory.tsa.json inventory.json
```

10.2 Registry SDK Integration

```
from tsa_registry_sdk import TSARegistry

```

```python
# Initialize with trust anchors
registry = TSARegistry(
    trust_anchors_path="trust-anchors.json",
    require_signatures=True
)

# Subscribe to and sync feeds
registry.subscribe_feed("https://example.com/tsa-feed.json")
registry.sync()

# Check a package
result = registry.check_package("mcp-remote", "0.1.14", "npm")
if result.blocked:
    print(result.message)
```

11. Case Studies

11.1 Vulnerability Advisory Flow

This case study illustrates the end-to-end flow for a critical vulnerability advisory.

1. **Discovery**: A security researcher identifies an OS command injection vulnerability in `mcp-remote` versions 0.0.5 through 0.1.15.
2. **Advisory Creation**: The publisher creates a TSA advisory with `BLOCK` action, `IMMEDIATE` urgency, and `REGISTRY` scope.
3. **Signing**: The advisory is signed with the publisher's Ed25519 key and the canonical hash is computed over the canonical payload.
4. **Feed Publication**: The advisory is added to the publisher's feed with its canonical hash.
5. **Registry Sync**: Registries sync the feed, verify the hash and signature, and enforce the `BLOCK` action for affected versions.
6. **Enforcement**: Installation attempts for `mcp-remote@0.1.14` are blocked with the advisory message.

11.2 Semantic Drift Alert

This case study illustrates detection and response to semantic drift.

1. **Detection**: Automated monitoring detects that a tool's description and declared capabilities changed significantly between versions 1.2.3 and 1.2.4.
2. **Analysis**: No CVE exists, but the changes suggest potential supply-chain compromise (expanded network capabilities, modified description).

3. **Advisory Creation**: A TSA advisory is created with `semantic_drift` populated and a `WARN` action with `HOST` scope.
4. **Distribution**: The advisory is signed and added to the feed.
5. **Enforcement**: Hosts surface warnings to security operators when the tool is invoked, prompting manual investigation.

12. Deployment Guidance

12.1 For Publishers

- Use stable TSA IDs (`TSA-YYYY-NNNN`) unique within your namespace, and include authoritative references (CVE, GHSA).
- Sign advisories and publish public keys as trust anchors.
- Maintain accurate `modified` timestamps for feed delta updates.
- Include machine-readable `condition` fields (semver ranges) for all version-scoped actions.
- For `UPDATE` actions, always include `target_version`.
- Provide clear, actionable `message` text for enforcement actions.

12.2 For Registries and Hosts

- Validate advisories against the schema in strict mode.
- Verify canonical hashes for integrity after fetching advisories.
- Require signatures for `BLOCK` enforcement; downgrade to `WARN` when missing.
- Cache advisories and track `modified` for incremental feed syncs.
- Implement fallback behavior for unreachable feeds (use cached advisories with warnings).

12.3 Policy Defaults

Recommended policy defaults for consumers:

- Default to `WARN` for unsigned advisories.
- Require signatures for `BLOCK` actions with `IMMEDIATE` or `HIGH` urgency.
- Use `scope` to apply consistent enforcement across registry, host, and gateway.
- Alert on stale feeds (not updated within 7 days).
- Quarantine advisories with hash mismatches pending investigation.

13. Security Considerations

TSA reduces risk but does not eliminate it. Implementers **MUST** consider the following failure modes:

13.1 Unsigned Advisories

Unsigned advisories **MUST** be treated as lower trust. An attacker who compromises a feed distribution channel could inject malicious advisories. Consumers **SHOULD** require signatures for high-impact actions and **SHOULD** provide clear warnings when enforcing unsigned advisories.

13.2 Key Compromise

Publisher key compromise **MUST** be mitigated with `REVOKE` actions (including the `revoked_key_id` field) and trust anchor updates. Publishers **SHOULD** maintain incident response procedures for key compromise, including:

- Issuing a `REVOKE` advisory with the `revoked_key_id` set to the compromised key
- Rotating to a new signing key and including it in `replacement_key_id`
- Notifying consumers through out-of-band channels
- Re-signing existing advisories with the new key

13.3 Hash Mismatches

Hash mismatches **MUST** trigger rejection or quarantine of advisories. A hash mismatch indicates potential tampering or transmission error. Consumers **SHOULD** log hash mismatches for security investigation.

13.4 Stale Feeds

Stale feeds **SHOULD** generate alerts for operational review. An attacker who can prevent feed updates could suppress new advisories. Consumers **SHOULD** implement freshness checks and alert operators when feeds exceed staleness thresholds.

13.5 Advisory Tampering

The combination of canonicalization, hashing, and signing provides defense-in-depth against advisory tampering. Consumers **MUST** verify all three layers when signatures are present:

1. Remove `signature` and `canonical_hash` fields to obtain the canonical payload

2. Canonicalize the payload using RFC 8785
3. Compute the SHA-256 hash of the canonical form
4. Verify the signature over the canonical form using the trusted public key

14. Future Work

The following areas are identified for future standardization:

- **Standardized Capability Taxonomy**: A registry of capability identifiers with formal definitions and hierarchical relationships.
- **Automated TBOM Generation**: Tooling and specifications for automated TBOM generation and binding during tool build processes.
- **Trust Anchor Distribution**: Standards for trust anchor discovery and distribution, including DNSSEC-based or certificate transparency-based approaches.
- **Extended Feed Validation**: Large-scale feed validation tooling and performance benchmarks.
- **Policy Language Integration**: Mappings between TSA actions and policy languages such as OPA Rego or Cedar.

15. Conclusion

TSA provides a security advisory standard that is MCP-native and operationally practical. It fills critical gaps in existing formats by encoding semantic drift, capability abuse, agent context, and enforcement semantics. With deterministic canonicalization, cryptographic signing, and a lightweight feed model, TSA enables registries and hosts to enforce security decisions automatically and reliably.

The reference implementation demonstrates feasibility today. The standard remains compatible with existing vulnerability ecosystems through OSV interoperability, allowing organizations to adopt TSA incrementally alongside their existing security infrastructure.

By providing machine-readable, verifiable, and enforceable advisories, TSA bridges the gap between AI agent tool risk and operational security controls.

References

[1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997.

[2] G. Klyne, C. Newman, "Date and Time on the Internet: Timestamps," RFC 3339, July 2002.

[3] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648, October 2006.

[4] M. Jones, J. Bradley, N. Sakimura, "JSON Web Signature (JWS)," RFC 7515, May 2015.

[5] M. Jones, "JSON Web Algorithms (JWA)," RFC 7518, May 2015.

[6] M. Nottingham, "Well-Known Uniform Resource Identifiers (URIs)," RFC 8615, May 2019.

[7] A. Rundgren, B. Jordan, S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, June 2020.

[8] JSON Schema, "JSON Schema Draft 2020-12," https://json-schema.org/specification.html.

[9] Open Source Vulnerability format, "OSV Schema," https://ossf.github.io/osv-schema/.

[10] FIRST, "Common Vulnerability Scoring System v3.1," https://www.first.org/cvss/v3.1/specification-document.

[11] FIRST, "Common Vulnerability Scoring System v4.0," https://www.first.org/cvss/v4.0/specification-document.

[12] Anthropic, "Model Context Protocol Specification," https://modelcontextprotocol.io/.

[13] OASIS, "Common Security Advisory Framework (CSAF) Version 2.0," 2022.

[14] CISA, "Vulnerability Exploitability eXchange (VEX)," 2022.

[15] T. Preston-Werner, "Semantic Versioning 2.0.0," https://semver.org/.

[16] npm, Inc., "node-semver: The semver parser for node," https://github.com/npm/node-semver.

[17] package-url, "Package URL (PURL) Specification," https://github.com/package-url/purl-spec.

Implementation Note

The JSON examples in these appendices are formatted for readability. PDF rendering may introduce line wrapping in long strings (hashes, URIs). Implementers **SHOULD** use the machine-readable schema and example files in the reference repository as the canonical source. The repository files are guaranteed to be syntactically valid JSON.

A. Appendix: Example Advisory

The following is a complete example of a TSA advisory (canonical payload shown without `signature` or `canonical_hash`):

```json
{
  "tsa_version": "1.0.0",
  "id": "TSA-2025-0001",
  "published": "2025-07-09T00:00:00Z",
  "modified": "2025-07-09T18:00:00Z",
  "publisher": {
    "name": "MCP Security Working Group",
    "namespace": "https://github.com/mcp-security"
  },
  "title": "mcp-remote OS Command Injection via OAuth Callback",
  "affected": [
    {
      "tool": {
        "name": "mcp-remote",
        "registry": "npm",
        "purl": "pkg:npm/mcp-remote"
      },
      "versions": {
        "introduced": "0.0.5",
        "fixed": "0.1.16",
        "affected_range": ">=0.0.5 <0.1.16"
      },
      "status": "AFFECTED",
      "capabilities_abused": ["network:oauth", "process:exec"],
      "attack_context": {
        "requires_agent_execution": false,
        "requires_user_interaction": true
      }
    }
  ],
  "actions": [
    {
      "type": "BLOCK",
      "scope": "REGISTRY",
      "condition": ">=0.0.5 <0.1.16",
      "urgency": "IMMEDIATE",
      "message": "Critical RCE vulnerability. Update to 0.1.16."
    },
    {
      "type": "UPDATE",
      "scope": "HOST",
      "condition": ">=0.0.5 <0.1.16",
      "target_version": "0.1.16",
      "urgency": "IMMEDIATE",
      "message": "Update to 0.1.16 to resolve CVE-2025-6514."
    }
  ]
}
```

B. Appendix: Example Feed

The following is a complete example of a TSA feed with a valid canonical hash:

```
{
  "feed_version": "1.0.0",
  "generated": "2025-07-10T12:00:00Z",
  "publisher": {
    "name": "MCP Security Working Group",
    "namespace": "https://github.com/mcp-security"
  },
  "advisories": [
    {
      "id": "TSA-2025-0001",
      "uri": "advisories/TSA-2025-0001.tsa.json",
      "canonical_hash": "sha256:338bf258a4c1b3fdc2e96c6346d19359a5c10d5850985cc04eac82c633aa3e7f",
      "title": "mcp-remote OS Command Injection",
      "modified": "2025-07-09T18:00:00Z",
      "severity": "CRITICAL",
      "cve": ["CVE-2025-6514"]
    }
  ]
}
```

C. Appendix: Semver Range Grammar

TSA adopts the node-semver range grammar [16] for version matching. This section provides a normative summary.

C.1 Primitives

A **comparator** consists of an operator and a version:

- `<1.2.3` – Less than
- `<=1.2.3` – Less than or equal
- `>1.2.3` – Greater than
- `>=1.2.3` – Greater than or equal
- `=1.2.3` or `1.2.3` – Exactly equal

C.2 Comparator Sets

A **comparator set** is one or more comparators joined by whitespace. All comparators in a set **MUST** match (logical AND).

Example: `>=1.0.0 <2.0.0` matches versions from 1.0.0 (inclusive) to 2.0.0 (exclusive).

C.3 Ranges

A **range** is one or more comparator sets joined by `||` (logical OR).

Example: `>=1.0.0 <1.5.0 || >=2.0.0` matches 1.0.0 through 1.4.x or 2.0.0 and above.

C.4 Prerelease Handling

Prerelease versions (e.g., `1.0.0-alpha`) only match comparators that explicitly include a prerelease tag on the same major.minor.patch tuple. For example, `>1.0.0-alpha` matches `1.0.0-beta` but `>1.0.0` does not match `1.0.1-alpha`.

C.5 Normative Requirement

Consumers **MUST** implement the node-semver range grammar or document deviations. Implementations **SHOULD** use established semver libraries (e.g., `semver` for Node.js, `python-semver` for Python) to ensure consistent matching behavior.

D. Appendix: JSON Schema Reference

The complete JSON Schema for TSA advisories is available in the repository at:

`schema/tsa-v1.0.0.schema.json`

The feed schema is available at:

`schema/tsa-feed-v1.0.0.schema.json`

Both schemas conform to JSON Schema Draft 2020-12 and enforce strict validation with `additionalProperties: false`.

E. Appendix: Glossary

Advisory A structured document describing a security issue affecting one or more MCP tools.
Attack Context Contextual information about the agent execution environment where a vulnerability can be exploited.
Canonical Form The deterministic byte representation of a JSON document per RFC 8785.
Canonical Payload The advisory JSON with `signature` and `canonical_hash` fields removed.
Capability Abuse Misuse of legitimate tool capabilities beyond their intended purpose.
Consumer A registry, host, gateway, or security system that ingests and enforces advisories.
Feed A lightweight index of advisories for efficient synchronization.
MCP Model Context Protocol—a protocol enabling AI agents to invoke external tools.
Publisher An entity that creates and signs advisories.
Semantic Drift Behavioral changes in a tool between versions that may indicate compromise.
TBOM Tool Bill of Materials—content hash and metadata binding a tool to build artifacts.
Trust Anchor A public key associated with a publisher, used to verify signatures.