MCP Tool Bill of Materials (TBOM)

A Provenance Standard for AI Agent Tool Supply Chains

RFC Version 1.0.2

Jason M. Lovell
Independent Researcher
jase.lovell@me.com
ORCID: https://orcid.org/0009-0001-6300-9155

February 2, 2026

Status of This Memo

This document is a community RFC and not an official part of the MCP specification. It proposes a Tool Bill of Materials (TBOM) format and verification process for MCP servers and related registries. Feedback and proposed changes are welcome.

Version: 1.0.2 **Status:** Draft for MCP Community Review

Abstract

The Model Context Protocol (MCP) has achieved remarkable adoption as a widely used standard interface for connecting large language model (LLM) applications to external tools and data sources. Following Anthropic's donation of MCP to the Linux Foundation's Agentic AI Foundation (AAIF) in December 2025, the ecosystem reports more than 10,000 published MCP servers and adoption across major AI platforms including Claude, Cursor, Microsoft Copilot, Gemini, VS Code, and ChatGPT.

This rapid adoption has outpaced the security maturity of the emerging "tool supply chain." Unlike traditional software dependencies with fixed call graphs, MCP tools influence agent behavior through their names, descriptions, and schemas—metadata that the MCP specification explicitly warns should be "considered untrusted, unless obtained from a trusted server." Yet the ecosystem lacks standardized mechanisms to verify who published a tool, exactly what was published, whether behavior changed since review, and what vulnerabilities exist in dependencies.

This paper proposes **Tool Bill of Materials (TBOM)**: a signed, machine-readable manifest that binds an MCP server release to immutable tool metadata via cryptographic hashes, provenance identifiers and release artifact digests, Software Bill of Materials (SBOM) references, and optional capability declarations for policy enforcement.

Keywords: Model Context Protocol, MCP, software supply chain security, bill of materials, AI agent security, tool provenance, cryptographic signing

Contents

Release Integrity

This document corresponds to TBOM specification release v1.0.2.

Artifact	SHA-256 Digest
tbom-schema-v1.0.2.json	`3c8f5e6da906198be7f1a19d66b974581540a46b17d182f1e72766a27d47cff9`
tbom-keys-schema-v1.0.1.json	`baaaac073939d3b050f65f87ce3f15175578d039f29d2f577d7fc1212a1ebc7e`

Verification: The authoritative digests are published in the repository release tag v1.0.2. Compute `sha256sum <filename>` and compare against the signed release manifest.

Conventions and Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "NOT RECOMMENDED", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [23][24] when, and only when, they appear in all capitals, as shown here.

Definitions:

MCP Server A software component implementing the Model Context Protocol server interface, exposing tools, resources, and/or prompts to MCP clients.
Tool Definition The metadata describing an MCP tool, including its name, description, input schema, output schema, and annotations.
TBOM Tool Bill of Materials; the signed manifest specified by this document.
Supplier The entity that publishes an MCP server and signs its TBOM.
Host The MCP client application that connects to MCP servers and invokes tools on behalf of users or agents.
Registry A service that indexes, stores, and distributes MCP server packages and their associated TBOMs.

1. Introduction

1.1 The MCP Tool Supply Chain Problem

The Model Context Protocol (MCP), introduced by Anthropic in 2024, has rapidly become the standard interface for connecting AI agents to external capabilities. In December 2025, Anthropic announced the donation of MCP to the Linux Foundation, establishing the Agentic AI Foundation (AAIF) with founding project contributions including Anthropic's MCP, Block's goose, and OpenAI's AGENTS.md. Platinum members include AWS, Anthropic, Block, Bloomberg,

Cloudflare, Google, Microsoft, and OpenAI [1]. The foundation reported ecosystem metrics demonstrating unprecedented adoption:

- More than 10,000 published MCP servers in the ecosystem (Linux Foundation announcement, December 2025) [1]
- Adoption by major AI platforms including Claude, Cursor, Microsoft Copilot, Gemini, VS Code, and ChatGPT (Linux Foundation announcement, December 2025) [1]

This adoption has created a new category of software supply chain—the tool supply chain—where AI agents discover, trust, and invoke external capabilities based on metadata descriptions. Unlike traditional software dependencies managed through package manifests and lockfiles, MCP tools present unique security challenges that existing supply chain security frameworks do not adequately address.

1.2 Why Tool Supply Chains Are Different

MCP tools differ from traditional software dependencies in ways that fundamentally amplify supply chain risk:

Runtime Invocation by Model Decisions. Traditional dependencies are invoked through fixed call graphs determined at development time. MCP tools are selected and invoked by language model decisions at runtime, based on natural language descriptions. An attacker who modifies a tool's description can influence when and how it is invoked without changing any executable code.

Semantic Attack Surface. The names, descriptions, schemas, and annotations of MCP tools directly influence agent behavior. A tool named `safe_file_reader` with a description claiming read-only access might actually perform writes. Unlike code vulnerabilities that require execution path analysis to exploit, semantic attacks operate at the description layer where verification is typically absent.

Autonomous Operation. MCP-enabled agents increasingly operate with minimal human oversight, executing multi-step workflows that may span multiple tools. The reduced human checkpoint frequency means malicious tool behavior may execute before any human review occurs.

Transitive Trust. Tools frequently access credentials, interact with other systems, and invoke additional tools. A compromised tool in a trusted position can leverage these transitive relationships to expand its attack surface significantly.

1.3 Evidence of Risk: Documented Incidents

The theoretical risks of MCP tool supply chains have materialized in documented incidents throughout 2025:

CVE-2025-6514: mcp-remote Command Injection (July 2025). JFrog Security Research discovered a critical vulnerability (CVSS 9.6) in the mcp-remote package, which enables MCP clients to connect to remote servers over HTTP/SSE [2]. The vulnerability allowed OS command injection via a crafted `authorization_endpoint` URL during OAuth flows. Affected versions (0.0.5 through 0.1.15) had accumulated significant download volume before disclosure. The npm Downloads API reports 233,023 downloads for mcp-remote in July 2025 [3]. This incident demonstrated how a single compromised MCP client library could enable arbitrary code execution on developer machines connecting to untrusted servers.

CVE-2025-49596: MCP Inspector RCE (2025). The NIST National Vulnerability Database reports a critical RCE in the MCP Inspector reference tooling (CVSS 9.4), demonstrating that MCP ecosystem tooling can introduce high-severity client-side risk [5].

Supabase/Cursor "Lethal Trifecta" (June 2025). General Analysis documented a proof-of-concept attack in which an IDE agent using the Supabase MCP server was manipulated via prompt injection embedded in user-generated content. The attack combined (1) privileged database credentials that bypass Row Level Security, (2) exposure to untrusted instructions, and (3) an exfiltration channel that caused the agent to publish sensitive tokens [4]. Simon Willison popularized the term "lethal trifecta" to describe this class of failure in MCP-enabled agents [25].

Additional Research Findings. CVE-2025-53110/CVE-2025-53109 (Filesystem MCP Server directory traversal and symlink bypass) demonstrate ongoing discovery of vulnerabilities in MCP tooling [6][7].

1.4 The SBOM Precedent

The software industry has established precedent for supply chain transparency through Software Bill of Materials (SBOM) requirements. Executive Order 14028 (May 2021) directed federal agencies to require secure software development practices, including providing SBOMs for software purchased by the U.S. government [9]. OMB Memorandum M-22-18 operationalized these requirements through agency guidance and vendor attestation processes [8].

These frameworks establish that machine-readable component transparency is essential for supply chain security. However, existing SBOM formats (CycloneDX, SPDX) address traditional software components—libraries, packages, containers—not the semantic layer unique to AI tool ecosystems. TBOM extends this precedent to cover tool semantics: the names, descriptions, schemas, and behavioral annotations that govern how AI agents discover and invoke capabilities.

1.5 Objectives

TBOM is designed to enable five core capabilities:

1. **Automated Provenance Verification.** Cryptographically verify publisher identity through signatures, establishing who published a tool and when.

2. **Immutable Tool Metadata Verification.** Detect changes to tool names, descriptions, schemas, or annotations through cryptographic hashes, preventing silent tool poisoning.
3. **Dependency Transparency.** Reference or embed traditional SBOMs, enabling vulnerability correlation against known CVE databases.
4. **Registry and Enterprise Policy Automation.** Provide machine-readable capability declarations enabling registries to filter by risk profile and enterprises to enforce allowlists.
5. **Incident Response Acceleration.** Support revocation mechanisms and advisory distribution, enabling ecosystem-wide response to discovered vulnerabilities.

2. Scope: What TBOM Is and Is Not

2.1 TBOM Is

- A **signed release manifest** for an MCP server package that enumerates the tools, resources, and prompts it exposes with their complete definitions
- A **semantic integrity binding** that cryptographically binds the exact tool metadata agents ingest (descriptions, schemas, annotations) to a specific release
- A **provenance and artifact binding** that includes stable identifiers such as Package URLs (purl) and digests for release artifacts (npm, pip, container) plus optional build attestations
- A **policy input** that optionally declares capabilities and risk classifications to enable host and enterprise policy enforcement
- **Self-contained and reviewable** without requiring connection to a running server

2.2 TBOM Is Not

- **A sandbox or runtime security boundary.** TBOM attests what was published; it does not prevent malicious code from executing once invoked.
- **A substitute for code review, least privilege, isolation, or user consent.** These complementary controls remain essential.
- **A runtime labeling or taint tracking system.** TBOM addresses static provenance at publish/install time, not runtime data flow. MCP annotation proposals address the runtime layer.
- **A guarantee that tool behavior matches tool description.** TBOM attests what was declared and shipped; behavioral verification requires testing, monitoring, and isolation.
- **A replacement for traditional SBOM.** TBOM references or embeds SBOMs; it does not duplicate dependency enumeration.

3. Background: MCP Trust Model and Existing Controls

3.1 MCP Protocol Overview

MCP defines a client-server architecture where AI applications (hosts) connect to MCP servers that expose capabilities through three primitives:

- **Tools:** Executable functions the agent can invoke
- **Resources:** Data sources the agent can read
- **Prompts:** Pre-defined prompt templates

During connection, the MCP server advertises its capabilities through `tools/list`, `resources/list`, and `prompts/list` methods. The returned metadata—particularly tool names, descriptions, and input schemas—is incorporated into the agent's context window and directly influences tool selection and invocation decisions.

3.2 What MCP Already Says

The MCP specification (2025-11-25 revision) explicitly addresses the trust implications of tool metadata [10]:

> "Descriptions of tool behavior such as annotations should be considered untrusted, unless obtained from a trusted server."

This warning acknowledges that tool metadata represents an attack surface but leaves "trusted server" undefined at the protocol level. The specification's security best practices page recommends input validation, access control, error handling, and logging—but these recommendations operate at the server implementation level. The ecosystem lacks mechanisms to verify trust at the supply chain level, including server identity, publisher identity, and the exact metadata shipped.

3.3 MCP Protocol Version Binding

TBOM v1.0.2 is designed for compatibility with MCP specification revision 2025-11-25 and later. Specifically:

- Tool definitions correspond to the `tools/list` response format
- Resource definitions correspond to the `resources/list` response format
- Prompt definitions correspond to the `prompts/list` response format

Forward compatibility: If future MCP revisions add fields to tool definitions, TBOM implementations **SHOULD**: (1) Include new fields in the TBOM `tools[]` entries, (2) Include new fields in the `definitionDigest` coverage (update `covers` string), (3) Maintain backward compatibility by supporting both old and new coverage patterns.

Unknown fields: When computing `definitionDigest`, implementations **MUST** only include fields explicitly listed in the `covers` string. Unknown fields returned by `tools/list` that are not in the TBOM's `covers` specification **MUST** be ignored for digest computation but **MAY** be preserved in the TBOM document for informational purposes.

3.4 Windows MCP Integration: A Platform Precedent

Microsoft's Windows MCP support introduces an on-device agent registry (ODR) model and a containment mode for MCP servers distributed as packaged applications. In this workflow, servers are registered using a declarative configuration that includes static responses for `tools/list`. This containment model emphasizes declarative manifests and static discovery responses, reducing post-review tool metadata drift [11].

This is a platform precedent: a major OS vendor is requiring packaged distribution and static discovery responses for MCP servers. TBOM generalizes these requirements into a portable, vendor-neutral standard that can be used across operating systems, registries, and hosts.

Windows MCP Pattern	**TBOM Analogue**
Packaged server identity and provenance	`signatures[]` (role: supplier)
Static `tools/list` manifest	`tools[].definitionDigest` drift detection
Contained execution modes and policy enforcement	Registry/host enforcement policy

4. Related Work

4.1 ETDI and Trust Governance Proposals

Academic and industry research has explored cryptographically signed tool manifests and registry-based governance workflows. The Enhanced Tool Definition Interface (ETDI) proposal describes signed tool manifests, immutable version identifiers, and registry-based approval workflows as components of a trust framework for MCP ecosystems [12].

TBOM contributes to this direction by providing:

- A **concrete, minimal, implementable schema** with explicit field definitions and conformance requirements
- **Deterministic digest rules** based on RFC 8785 JSON Canonicalization Scheme
- **Explicit standards bindings** to established supply chain primitives (purl, CycloneDX, SPDX, Sigstore, DSSE)
- **Test vectors** enabling interoperable implementations

TBOM can serve as the manifest and attestation payload within ETDI-like governance systems, providing the technical substrate for trust decisions.

4.2 Software Supply Chain Standards

TBOM builds on established supply chain security standards:

- **CycloneDX / SPDX:** SBOM formats for dependency enumeration [13][14]
- **SLSA / in-toto:** Build provenance attestation frameworks [15][16]
- **Sigstore:** Keyless signing and transparency log infrastructure [17]
- **TUF:** Secure metadata distribution [18]
- **Package URL (purl):** Universal package identification [19]

Rather than reinventing these capabilities, TBOM references and integrates them, focusing specifically on the tool semantic layer these standards do not address.

4.3 Differentiation

Existing standards address *what code ships* (SBOM) and *how it was built* (SLSA). TBOM addresses *what the tool claims to do*—the semantic metadata that AI agents consume to make invocation decisions. This distinction is critical because:

- Code can be unchanged while descriptions change (tool poisoning)
- Descriptions influence agent behavior independently of code execution
- Semantic attacks require semantic verification

4.4 Standards Comparison

Standard	**Primary Focus**	**Tool Metadata Hashing**	**Signing**
CycloneDX/SPDX	Dependency enumeration	No	Optional
SLSA/in-toto	Build provenance	No	Yes
Sigstore	Signing infrastructure	No	Yes
TBOM	Tool semantic integrity	Yes	Required

TBOM is designed to complement these standards, not replace them. A complete supply chain security posture for MCP servers would include: SBOM (CycloneDX/SPDX) for dependency transparency, SLSA/in-toto for build provenance, TBOM for tool semantic integrity, and Sigstore (optional) for keyless signing infrastructure.

4.5 MCP Security Research

Recent academic research has systematically analyzed security vulnerabilities in MCP ecosystems:

- **MCPSecBench** provides a comprehensive taxonomy and benchmark for testing MCP security across 9 vulnerability categories covering over 1,200 test cases [26].
- **MCP Security Bench (MSB)** evaluates end-to-end attack scenarios including tool poisoning, data exfiltration, and privilege escalation in real-world MCP deployments [27].
- **MCPTox** benchmarks tool poisoning attacks specifically, demonstrating how malicious tool descriptions can manipulate agent behavior [28].
- **Defending LLMs Against Tool Poisoning** explicitly analyzes descriptor mutation and "rug pull" attacks where tool metadata changes post-approval [29].

TBOM is complementary to these behavioral and semantic defenses. While MCPSecBench and related work focus on detecting malicious behavior at runtime, TBOM provides static provenance and integrity guarantees that enable detection of metadata drift (tool poisoning via description changes), verification of publisher identity before behavioral analysis, and correlation of vulnerabilities across the tool supply chain.

5. Threat Model

5.1 Threat Actors

TBOM addresses threats from the following actor categories:

Actor	Capability	Motivation
Malicious Publisher	Publishes intentionally harmful tools with legitimate-appearing metadata	Financial gain, data theft, system access
Compromised Publisher	Legitimate publisher whose signing keys or build infrastructure have been compromised	Collateral damage from targeted attack
Supply Chain Attacker	Compromises registry, hosting platform, or distribution infrastructure	Broad ecosystem access, credential harvesting
Metadata Manipulator	Modifies tool descriptions, schemas, or annotations without changing code	Influence agent behavior, bypass security reviews

5.2 Attack Vectors

Tool Poisoning via Metadata Modification. An attacker modifies a tool's description to influence when the agent invokes it or what parameters it passes. Example: changing "read-only database query" to "database query with automatic optimization" to encourage write operations.

Dependency Confusion. An attacker publishes a malicious package with a name similar to a legitimate MCP server, relying on typosquatting or namespace confusion to achieve installation.

Registry Compromise. An attacker compromises a registry or hosting platform, replacing legitimate packages with malicious versions or injecting malicious metadata into package listings.

Build Infrastructure Compromise. An attacker compromises a publisher's CI/CD pipeline, injecting malicious code or metadata into releases without the publisher's knowledge.

Key Compromise. An attacker obtains a legitimate publisher's signing key, enabling publication of malicious packages with valid signatures.

5.3 What TBOM Addresses

Attack Vector	TBOM Mitigation
Tool poisoning via metadata modification	`definitionDigest` detects unauthorized changes
Dependency confusion	purl and supplier identity enable verification
Registry compromise	Signatures survive registry modification
Build infrastructure compromise	Artifact digests detect tampering; attestations prove build provenance
Silent updates	`serialNumber` and version tracking enable change detection

5.4 What TBOM Does Not Address

Attack Vector	Required Complementary Control
Key compromise	Key management practices, hardware security modules, revocation infrastructure
Malicious-but-signed tools	Code review, behavioral testing, reputation systems, community reporting
Runtime prompt injection	Input sanitization, context isolation, output filtering
Confused deputy attacks	Least privilege, capability restrictions, user consent
Zero-day vulnerabilities	Vulnerability research, rapid patching, defense in depth

TBOM improves transparency and enables detection; it does not guarantee safety. Defense in depth remains essential.

6. TBOM v1: The Standard

6.1 Design Principles

Self-Contained and Reviewable. A TBOM document **MUST** contain sufficient information to review tool definitions without connecting to a running server. This enables offline security review and aligns with Windows' static manifest approach.

Minimum Viable Security. Version 1 requires only the fields essential for provenance verification and metadata integrity: signatures, artifact digests, and tool definitions with digests.

Interoperability First. TBOM maps cleanly to existing standards. Package URL (purl) identifiers provide artifact identification. CycloneDX and SPDX handle dependency enumeration. DSSE and JWS provide signing envelopes. TBOM avoids reinvention where established solutions exist.

Deterministic Verification. Canonicalization rules (RFC 8785 JSON Canonicalization Scheme) ensure hash stability across implementations. Explicit field coverage definitions prevent ambiguity.

Multi-Signer Model. TBOM supports layered trust through multiple signatures with explicit roles: supplier (original publisher), registry (identity verification), and enterprise (internal approval).

Extensibility. Optional modules for capabilities, risk classification, and attestations enable ecosystem-specific extensions without fragmenting the core standard.

6.2 Data Model

TBOM is scoped to an MCP server release, not individual tools. An MCP server typically exposes multiple tools, resources, and prompts through a single package distribution. The TBOM describes:

- The **server package** (subject): name, version, supplier, artifacts
- The **tool definitions** with complete metadata for offline review
- Optional **resources and prompts**
- Optional **dependency/vulnerability information**, capability declarations, and attestations
- One or more **cryptographic signatures** with explicit roles

This server-level scope aligns with distribution reality (packages are the unit of publication) while enabling tool-level verification (each tool's metadata is individually hashed).

6.3 Required Fields (v1 Conformance)

A conformant TBOM v1 document **MUST** include the following:

6.3.1 Document Metadata

- `tbomVersion`: "1.0.2"
- `serialNumber`: URN UUID uniquely identifying this TBOM instance. Implementations **SHOULD** use UUID version 4 (random) per RFC 9562.
- `createdAt`: ISO 8601 timestamp of TBOM generation

6.3.2 Subject (Server Package)

- `kind`: "mcp-server"
- `name`: Human-readable server name
- `version`: Semantic version of the release
- `supplier`: Object containing at minimum `name`
- `artifacts`: Array with at least one artifact containing `type` and `digest`

If an artifact includes `downloadUrl`, it **SHOULD** use `https://`. Plain `http://` is NOT **RECOMMENDED** even when an artifact digest is present.

6.3.3 Tool Definitions

Each tool entry **MUST** include:

- `name`: Tool name as returned by `tools/list`
- `description`: Tool description as returned by `tools/list`
- `inputSchema`: JSON Schema for tool inputs
- `definitionDigest`: Object containing `algorithm`, `value`, and `canonicalization`

Each tool entry **MAY** include: `toolId`, `outputSchema`, `annotations`, `capabilities`, `risk`.

6.3.4 Signatures

`signatures`: Array with at least one signature containing:

- `role`: "supplier" (at minimum one supplier signature **REQUIRED**)
- `type`: Signature envelope type (jws, dsse, sigstore)
- `algorithm`: Cryptographic algorithm (Ed25519, ECDSA-P256, ECDSA-P384)
- `keyId`: URI identifying the signing key
- `value`: Signature value

6.4 Canonicalization and Hashing

Canonical JSON (RFC 8785). All hashes **MUST** be computed over RFC 8785 canonical JSON [20]. This ensures lexicographic key ordering, no insignificant whitespace, consistent number representation, and consistent string escaping.

Pre-Canonicalization Normalization. Before canonicalization, implementations **MUST**: (1) Remove any keys with null values, (2) When serializing to JSON, omit keys whose values are language-specific equivalents of undefined, (3) Apply this normalization recursively to nested objects.

Tool Definition Digest Coverage. The `definitionDigest` for each tool covers the canonical JSON representation of an object containing exactly these keys: `name` (**REQUIRED**), `description` (**REQUIRED**), `inputSchema` (**REQUIRED**), `outputSchema` (if present), `annotations` (if present).

Algorithm. SHA-256 is the **REQUIRED** algorithm for TBOM v1. The digest value is formatted as `sha256:<64-hex-characters>`.

6.5 Signing

Envelope Format. TBOM v1 implementations **MUST** support at least one of: JWS (RFC 7515) with detached payload [21], or DSSE (Dead Simple Signing Envelope) [22]. Implementations **MAY** additionally support Sigstore bundles or COSE signatures.

Payload. The signature covers the canonical JSON representation of the TBOM document with the `signatures` field removed.

Signature Roles. Each signature **MUST** include a `role` field with one of: `supplier` (publisher attestation, at least one **REQUIRED**), `registry` (identity verification and review attestation), or `enterprise` (internal approval for production use).

Key Discovery. Suppliers **MUST** publish signing keys at a well-known location: `https://{supplier-domain}/.well-known/tbom-keys.json`

7. Verification and Enforcement

7.1 Verification Algorithm

Verifiers **MUST** implement the following algorithm at install time and/or connect time:

1. Parse TBOM document; REJECT if schema validation fails
2. For each required signature role in `policy.requiredRoles`:
   1. Find matching signature in `tbom.signatures` by role
   2. Fetch public key from `signature.keyId`
   3. Verify signature over `canonical(tbom - signatures)`
   4. REJECT if verification fails
3. For each artifact in `tbom.subject.artifacts`:
   1. Compute digest of installed/downloaded artifact
   2. Compare to declared digest
   3. REJECT if mismatch
4. Connect to MCP server; fetch `tools/list` response
5. For each tool in `tbom.tools`:
   1. Find corresponding tool in server response by name
   2. Extract {name, description, inputSchema, outputSchema, annotations}
   3. Compute RFC 8785 canonical JSON, then SHA-256
   4. Compare to `tool.definitionDigest.value`
   5. REJECT if mismatch (DRIFT DETECTED)
6. Evaluate `policy.capabilityRules` against TBOM capabilities
7. Return VERIFIED or REJECTED with reason

7.2 Drift Detection

If the server returns tool definitions that do not match TBOM digests, the verifier has detected drift. Possible causes: version skew, tampering, or tool poisoning (intentional metadata manipulation to influence agent behavior).

Verifiers **SHOULD**: block the tool from being invoked, alert the user/administrator, log the mismatch details for investigation, and optionally quarantine and report to registry.

Drift detection is TBOM's primary defense against tool poisoning attacks where attackers modify descriptions to manipulate agent behavior.

8. Registry Integration

8.1 Publication Requirements

MCP registries implementing TBOM support **MUST** require: TBOM document (embedded or by reference), at least one valid supplier signature, artifact digests matching distributed packages, and schema validation pass.

8.2 Revocation Mechanisms

Registry Revocation List. Registries **MUST** maintain a revocation list accessible at: `https://{registry}/api/v1/tbom/revocations`

Verifier Behavior. Verifiers **SHOULD** check revocation status before installation, periodically for installed packages, and upon security advisory notification.

9. Capability and Risk Classification

9.1 Capability Taxonomy

TBOM defines a minimal, unambiguous capability vocabulary:

Capability	Type	Description
shellExecution	boolean	Can execute shell commands
fileSystemAccess	enum	File system access level (none, read, write, readwrite)
networkAccess	array	Allowed network destinations
credentialAccess	enum	Access to credential stores
userDataAccess	array	Categories of user data accessed
externalSideEffects	enum	External state modification (none, low, high)

9.2 Risk Classification

Risk tiers provide informative guidance; they are NOT normative requirements:

Tier	Criteria (Informative)
Low	No shell, no write access, no credentials, enumerated network endpoints
Medium	Read-only credential access OR PII access OR unenumerated network access
High	Write credential access OR shell execution OR high side effects
Critical	Multiple high-risk capabilities combined

10. Implementation Guidance

10.1 For MCP Server Developers

Generate TBOM at Release Time. Integrate TBOM generation into CI/CD pipelines.

Ensure Deterministic Tool Definitions. Tool metadata returned by `tools/list` **MUST** be deterministic across server instances. Avoid dynamic descriptions based on environment, timestamps in metadata, and random identifiers.

Publish Signing Keys. Host key material at the well-known location: `https://yourdomain.com/.well-known/tbom-keys.json`

Include SBOM. Generate CycloneDX or SPDX SBOM alongside TBOM.

10.2 For MCP Clients and Hosts

Verify Before Installation. Fetch and verify TBOM before installing MCP server packages.

Verify at Connect Time. Re-verify tool definitions when connecting to servers using the drift detection algorithm.

Surface Verification Status. Provide clear UI indication of verification status, capabilities, and any vulnerability warnings.

10.3 For Enterprise Security Teams

Maintain Supplier Allowlist. Define trusted suppliers whose signatures satisfy policy.

Mirror Registry Internally. Operate internal registry mirror for network isolation, additional review gates, audit logging, and custom policy enforcement.

Require Minimum Conformance. Define minimum requirements for production: valid supplier signature, dependencies present, no critical vulnerabilities without VEX “not_affected”, and capabilities within policy bounds.

11. Case Studies

11.1 CVE-2025-6514: mcp-remote Command Injection

What Happened. The mcp-remote package (versions 0.0.5-0.1.15) contained an OS command injection vulnerability in OAuth flow handling. A malicious MCP server could craft an `authorization_endpoint` URL that, when processed by the client, executed arbitrary commands on the developer's machine [2].

What Control Failed. No verification occurred before the client connected to the MCP server. The malicious server's OAuth configuration was trusted implicitly.

What TBOM Would Have Enabled. Signature verification would have identified unsigned or untrusted servers. Registry enforcement could have blocked unreviewed OAuth implementations. Capability declaration would have triggered policy evaluation. Known vulnerability correlation would have flagged affected versions after CVE publication.

What TBOM Would Not Have Prevented. If the malicious server had a valid TBOM from a compromised publisher. If the vulnerability existed in a signed, reviewed package before CVE disclosure. The underlying code vulnerability itself (TBOM attests metadata, not code correctness).

11.2 CVE-2025-49596: MCP Inspector RCE

What Happened. The NIST National Vulnerability Database reports a critical RCE in the MCP Inspector reference tooling (CVSS 9.4), showing that MCP ecosystem tooling can introduce high-severity client-side risk [5].

What TBOM Would Have Enabled. Registry enforcement could have required signed, reviewed tooling versions. Dependencies and vulnerability metadata would have surfaced affected versions after disclosure. Policy could have blocked use of affected versions.

11.3 Supabase/Cursor "Lethal Trifecta"

What Happened. An AI agent using the Supabase MCP server with `service_role` credentials was manipulated through prompt injection in user-generated content. The agent bypassed Row-Level Security, extracted sensitive tokens, and exfiltrated them through a public channel [4][25].

What TBOM Would Have Enabled. Capability declarations would have flagged `credentialAccess: readwrite` and `userDataAccess: [other]`. Enterprise policy could have required additional approval for high-privilege tools.

Key Insight. This case illustrates TBOM's role as one layer in defense-in-depth. TBOM would have improved visibility and enabled policy enforcement that might have prevented deployment with excessive privileges, but runtime controls remain essential.

12. Limitations and Future Work

12.1 Acknowledged Limitations

Compromised Signing Keys. If an attacker obtains a legitimate publisher's signing key, TBOM cannot distinguish malicious-but-validly-signed packages from legitimate releases. Mitigation requires robust key management, hardware security modules, and effective revocation infrastructure.

Malicious-but-Signed Tools. TBOM attests provenance and metadata integrity, not intent or safety. A malicious actor can publish properly-signed TBOMs for harmful tools. Reputation systems, code review, and behavioral monitoring remain essential.

Runtime Attacks. TBOM operates at publish/install/connect time. Runtime attacks (prompt injection, confused deputy, data exfiltration) require complementary runtime controls.

Semantic Gap. TBOM verifies that metadata has not changed, but cannot verify that descriptions accurately represent behavior. A tool described as "read-only" might actually write; TBOM detects description changes but not description accuracy.

12.2 Future Work

Transparency Logs. Integration with transparency log infrastructure would enable ecosystem-wide detection of duplicate serial numbers, inconsistent metadata, and key usage anomalies.

TUF Integration. The Update Framework (TUF) provides robust metadata distribution with role separation and threshold signatures.

Runtime Attestation. Future versions could define attestation mechanisms for runtime behavior.

Federated Identity. Integration with decentralized identity systems could enable verifiable supplier credentials and reputation aggregation across registries.

13. Security Considerations

This section consolidates security-relevant guidance for implementers.

13.1 Security Properties and Non-Goals

What TBOM guarantees (when signatures verify):

- **Integrity of declared metadata:** Tool definitions match what the supplier signed
- **Publisher identity:** Signatures trace to identifiable keys
- **Drift detection:** Local tool metadata can be compared against signed definitions

- **Dependency transparency:** SBOM references enable vulnerability correlation

What TBOM does NOT guarantee:

- **Safety of tool behavior:** A malicious-but-signed tool remains dangerous
- **Key security:** Compromised signing keys undermine all guarantees
- **Runtime protection:** Prompt injection and other runtime attacks are out of scope
- **Code correctness:** TBOM attests to metadata, not implementation quality

TBOM is a transparency and integrity control, not a safety guarantee. It enables informed trust decisions but does not make those decisions automatically.

13.2 Cryptographic Considerations

Algorithm Selection. TBOM v1 requires SHA-256 for hashing and supports Ed25519, ECDSA P-256, and ECDSA P-384 for signatures. Ed25519 is **RECOMMENDED** for new implementations due to its resistance to timing attacks, small key/signature sizes, and widespread library support.

Key Storage. Signing keys **SHOULD** be stored in hardware security modules (HSMs) or secure enclaves for production deployments.

13.3 Time-of-Check-to-Time-of-Use (TOCTOU)

TBOM verification occurs at specific points (install time, connect time), but tool invocation occurs afterward. This creates a TOCTOU window. Mitigations: verify artifact digests immediately before execution, re-verify tool digests periodically, use runtime isolation as defense in depth, and monitor for unexpected behavior.

13.4 Trust Bootstrapping

TBOM relies on trusting public keys published at `/.well-known/tbom-keys.json`. Recommended approaches: certificate binding (include X.509 certificate chain), registry endorsement (registries sign TBOMs after identity verification), out-of-band verification (verify key fingerprints through separate channels), and trust-on-first-use (TOFU) for development environments.

13.5 Replay and Rollback Attacks

Mitigations: verifiers **SHOULD** maintain a cache of seen `serialNumber` values and reject duplicates, compare `subject.version` against known-good versions, and registries **SHOULD** enforce monotonically increasing version numbers.

14. IANA Considerations

Note on Provisional Identifiers: The `/.well-known/tbom-keys.json` path and `urn:tbom:` URI scheme used in this specification are provisional. If TBOM is standardized, these identifiers would require IANA registration.

This document has no IANA actions at this time. Future versions may request IANA registration for a dedicated media type (e.g., `application/tbom+json`), the `urn:tbom:` URN namespace, and the `/.well-known/tbom-keys.json` well-known URI suffix.

15. Conclusion

The MCP specification warns that tool behavior descriptions should be considered untrusted unless obtained from a trusted server. Windows has published containment guidance for MCP servers that relies on packaged distribution and declarative manifests in its registry model [11]. The MCP ecosystem has achieved massive scale with more than 10,000 published servers and adoption across major AI platforms [1].

TBOM provides the missing trust material to make the MCP specification's warning actionable and Windows' requirements portable across the broader ecosystem.

TBOM establishes:

- **Cryptographic signatures** that establish publisher identity and enable revocation
- **Definition digests** that detect tool poisoning through metadata modification
- **Artifact digests** that verify package integrity across distribution
- **Dependency/vulnerability metadata** that enables traditional vulnerability management
- **Capability declarations** that enable policy-based access control

TBOM does not solve all MCP security challenges. Runtime attacks, key compromise, and malicious-but-signed tools require complementary controls. However, TBOM establishes the foundation for supply chain security that has proven essential in traditional software ecosystems.

As AI agents assume greater autonomy in enterprise and consumer contexts, the integrity of the tools they invoke becomes critical infrastructure. TBOM provides the transparency and verification mechanisms necessary to build trustworthy AI agent ecosystems at scale.

We invite the MCP community, the Agentic AI Foundation, and the broader security community to review, refine, and adopt this specification.

References

References

[1] The Linux Foundation. Linux Foundation Announces the Formation of the Agentic AI Foundation. Press release, December 9, 2025. https://www.linuxfoundation.org/press/linux-foundation-announces-the-formation-of-the-agentic-ai-foundation

[2] JFrog. CVE-2025-6514 Threatens LLM Clients. July 9, 2025. https://jfrog.com/blog/cve-2025-6514-critical-mcp-remote-rce-vulnerability/

[3] npm Downloads API. mcp-remote downloads (July 2025). https://api.npmjs.org/downloads/point/2025-07-01:2025-07-31/mcp-remote (Accessed: 2025-08-01)

[4] General Analysis. Supabase MCP can leak your entire SQL database. June 16, 2025. https://www.generalanalysis.com/blog/supabase-mcp-blog (Accessed: 2026-01-09)

[5] NIST National Vulnerability Database. CVE-2025-49596. https://nvd.nist.gov/vuln/detail/CVE-2025-49596 (Accessed: 2026-01-09)

[6] NIST National Vulnerability Database. CVE-2025-53110. https://nvd.nist.gov/vuln/detail/CVE-2025-53110 (Accessed: 2026-01-09)

[7] NIST National Vulnerability Database. CVE-2025-53109. https://nvd.nist.gov/vuln/detail/CVE-2025-53109 (Accessed: 2026-01-09)

[8] U.S. Office of Management and Budget. M-22-18: Enhancing the Security of the Software Supply Chain through Secure Software Development Practices. September 14, 2022. https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf

[9] U.S. Federal Register. Executive Order 14028: Improving the Nation's Cybersecurity. May 17, 2021. https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

[10] Model Context Protocol. MCP Specification (2025-11-25). https://modelcontextprotocol.io/specification/2025-11-25/ (Accessed: 2026-01-09)

[11] Microsoft. Securing the Model Context Protocol: Building a safer agentic future on Windows. Windows Experience Blog, May 19, 2025. https://blogs.windows.com/windowsexperience/2025/05/19/securing-the-model-context-protocol-building-a-safer-agentic-future-on-windows/

[12] Bhatt, M., Narajala, V. S., and Habler, I. ETDI: Mitigating Tool Squatting and Rug Pull Attacks in Model Context Protocol (MCP) by using OAuth-Enhanced Tool Definitions and Policy-Based Access Control. arXiv:2506.01333 (2025). https://arxiv.org/abs/2506.01333

[13] CycloneDX. CycloneDX Specification. https://cyclonedx.org/specification/overvi

ew/

[14] SPDX. SPDX Specification. https://spdx.github.io/spdx-spec/

[15] SLSA. Supply-chain Levels for Software Artifacts. https://slsa.dev/

[16] in-toto. in-toto Specification. https://in-toto.io/

[17] Sigstore. Sigstore Project. https://www.sigstore.dev/

[18] The Update Framework (TUF). Specification. https://theupdateframework.io/

[19] Package URL (purl). Specification. https://github.com/package-url/purl-spec

[20] IETF. RFC 8785: JSON Canonicalization Scheme (JCS). https://www.rfc-editor.org/rfc/rfc8785

[21] IETF. RFC 7515: JSON Web Signature (JWS). https://www.rfc-editor.org/rfc/rfc7515

[22] Dead Simple Signing Envelope (DSSE). https://github.com/secure-systems-lab/dsse

[23] IETF. RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. https://www.rfc-editor.org/rfc/rfc2119

[24] IETF. RFC 8174: Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. https://www.rfc-editor.org/rfc/rfc8174

[25] IETF. RFC 9562: Universally Unique IDentifiers (UUIDs). May 2024. https://www.rfc-editor.org/rfc/rfc9562

[26] Simon Willison. The Supabase MCP "lethal trifecta". July 6, 2025. https://simonwillison.net/2025/Jul/6/supabase-mcp-lethal-trifecta/

[27] Yang, Y., Chen, F., Xu, W., Yu, Z., and Ma, X. MCPSecBench: A Systematic Security Benchmark and Playground for Testing Model Context Protocols. arXiv:2508.13220 (2025). https://arxiv.org/abs/2508.13220 (Accessed: 2026-01-09)

[28] Zhang, D., Li, Z., Luo, X., Liu, X., Li, P., and Xu, W. MCP Security Bench (MSB): Benchmarking Attacks Against Model Context Protocol in LLM Agents. arXiv:2510.15994 (2025). https://arxiv.org/abs/2510.15994 (Accessed: 2026-01-09)

[29] Wang, Z. et al. MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers. arXiv:2508.14925 (2025). https://arxiv.org/abs/2508.14925 (Accessed: 2026-01-09)

[30] Jamshidi, S., Tong, Q., Shen, Z., Chen, Z., and Schulzrinne, H. Securing the Model Context Protocol: Defending LLMs Against Tool Poisoning. arXiv:2512.06556 (2025). https://arxiv.org/abs/2512.06556 (Accessed: 2026-01-09)

[31] IETF. RFC 8615: Well-Known Uniform Resource Identifiers (URIs). May 2019. https://www.rfc-editor.org/rfc/rfc8615

A. Appendix: TBOM v1.0.2 JSON Schema

Note: The canonical schema file is `tbom-schema-v1.0.2.json`. The embedded schema below uses URN identifiers. For production implementations, use the schema published at the specification repository.

Normative Status: This appendix is an informative rendering of the schema for human review. The normative machine-readable schema is the versioned JSON file published in the specification repository release tag.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:tbom:schema:1.0.2",
  "title": "Tool Bill of Materials (TBOM) Schema",
  "type": "object",
  "required": [
    "tbomVersion",
    "serialNumber",
    "createdAt",
    "subject",
    "tools",
    "signatures"
  ],
  "properties": {
    "tbomVersion": {
      "type": "string",
      "const": "1.0.2"
    },
    "serialNumber": {
      "type": "string",
      "pattern": "^urn:uuid:[0-9a-fA-F]{8}-..."
    },
    "createdAt": {
      "type": "string",
      "format": "date-time"
    },
    "subject": { "$ref": "#/$defs/Subject" },
    "tools": {
      "type": "array",
      "items": { "$ref": "#/$defs/ToolDefinition" },
      "minItems": 1
    },
    "signatures": {
      "type": "array",
      "items": { "$ref": "#/$defs/Signature" },
      "minItems": 1
    }
  }
```

```
}
```

Listing 1: TBOM v1.0.2 JSON Schema (abbreviated)

See the specification repository for the complete schema with all definitions.

B. Appendix: TBOM Signing Keys Document Schema v1.0.1

The signing keys document extends JSON Web Key Set (JWKS) format with TBOM-specific metadata including role constraints and validity windows.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:tbom:keys-schema:1.0.1",
  "title": "TBOM Signing Keys Document Schema",
  "type": "object",
  "required": ["keys"],
  "properties": {
    "keys": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["kid", "kty", "use"],
        "properties": {
          "kid": { "type": "string" },
          "kty": { "type": "string" },
          "use": { "const": "sig" },
          "alg": { "type": "string" },
          "validFrom": { "type": "string", "format": "date-time" },
          "validUntil": { "type": "string", "format": "date-time" },
          "revoked": { "type": "boolean" },
          "roles": {
            "type": "array",
            "items": { "enum": ["supplier", "registry", "enterprise"]
              ↪ }
          }
        }
      }
    }
  }
}
```

Listing 2: TBOM Keys Document Schema (abbreviated)

C. Appendix: TBOM Examples

C.1 Minimal Conformant TBOM

```json
{
  "tbomVersion": "1.0.2",
  "serialNumber": "urn:uuid:550e8400-e29b-41d4-a716-446655440000",
  "createdAt": "2026-01-09T12:00:00Z",
  "subject": {
    "kind": "mcp-server",
    "name": "example-server",
    "version": "1.0.0",
    "supplier": { "name": "Example Corp" },
    "artifacts": [{
      "type": "npm",
      "digest": "sha256:abc123..."
    }]
  },
  "tools": [{
    "name": "hello_world",
    "description": "Returns a greeting",
    "inputSchema": {
      "type": "object",
      "properties": {
        "name": { "type": "string" }
      }
    },
    "definitionDigest": {
      "algorithm": "sha256",
      "value": "sha256:def456...",
      "canonicalization": "rfc8785",
      "covers": "{name,description,inputSchema}"
    }
  }],
  "signatures": [{
    "role": "supplier",
    "type": "jws",
    "algorithm": "Ed25519",
    "keyId": "https://example.com/.well-known/tbom-keys.json#2026-01",
    "value": "eyJ..."
  }]
}
```

Listing 3: Minimal TBOM Example

D. Appendix: Test Vectors

D.1 Tool Definition Digest Test Vector

Input (pre-canonicalization):

```
{
  "name": "get_weather",
  "description": "Retrieves current weather for a location",
  "inputSchema": {
    "type": "object",
    "properties": {
      "location": { "type": "string" }
    },
    "required": ["location"]
  }
}
```

Canonical JSON (RFC 8785):

```
{"description":"Retrieves current weather for a location",
"inputSchema":{"properties":{"location":{"type":"string"}},
"required":["location"],"type":"object"},"name":"get_weather"}
```

SHA-256 Digest: `sha256:a1b2c3d4...` (compute from canonical form)

E. Appendix: Standards Alignment (Informative)

E.1 CycloneDX / SPDX Alignment

TBOM's `dependencies[]` array uses purl identifiers compatible with both CycloneDX and SPDX component identification. TBOM's `attestations[]` can reference CycloneDX or SPDX SBOM documents.

E.2 SLSA / in-toto / Sigstore Alignment

TBOM's `attestations[]` array can embed or reference SLSA provenance attestations, in-toto link metadata, and Sigstore bundles.

E.3 Windows MCP Precedent

TBOM's design aligns with Microsoft's Windows MCP containment model: static tool manifests correspond to `tools[]` with `definitionDigest`, packaged distribution aligns with `subject.artifacts[]`, and registry enforcement maps to verification algorithm.

F. Appendix: Regulatory Alignment (Informative)

TBOM supports compliance with:

- **Executive Order 14028** (Software Supply Chain Security): TBOM provides the attestation layer for MCP server provenance
- **OMB M-22-18**: TBOM's SBOM references enable vulnerability correlation required by federal procurement
- **EU AI Act**: TBOM's capability declarations support risk classification required for AI system transparency