OWASP AIBOM
AI BILL OF MATERIALS

OWASP AIBOM Foundations Guide

SEPTEMBER 2026

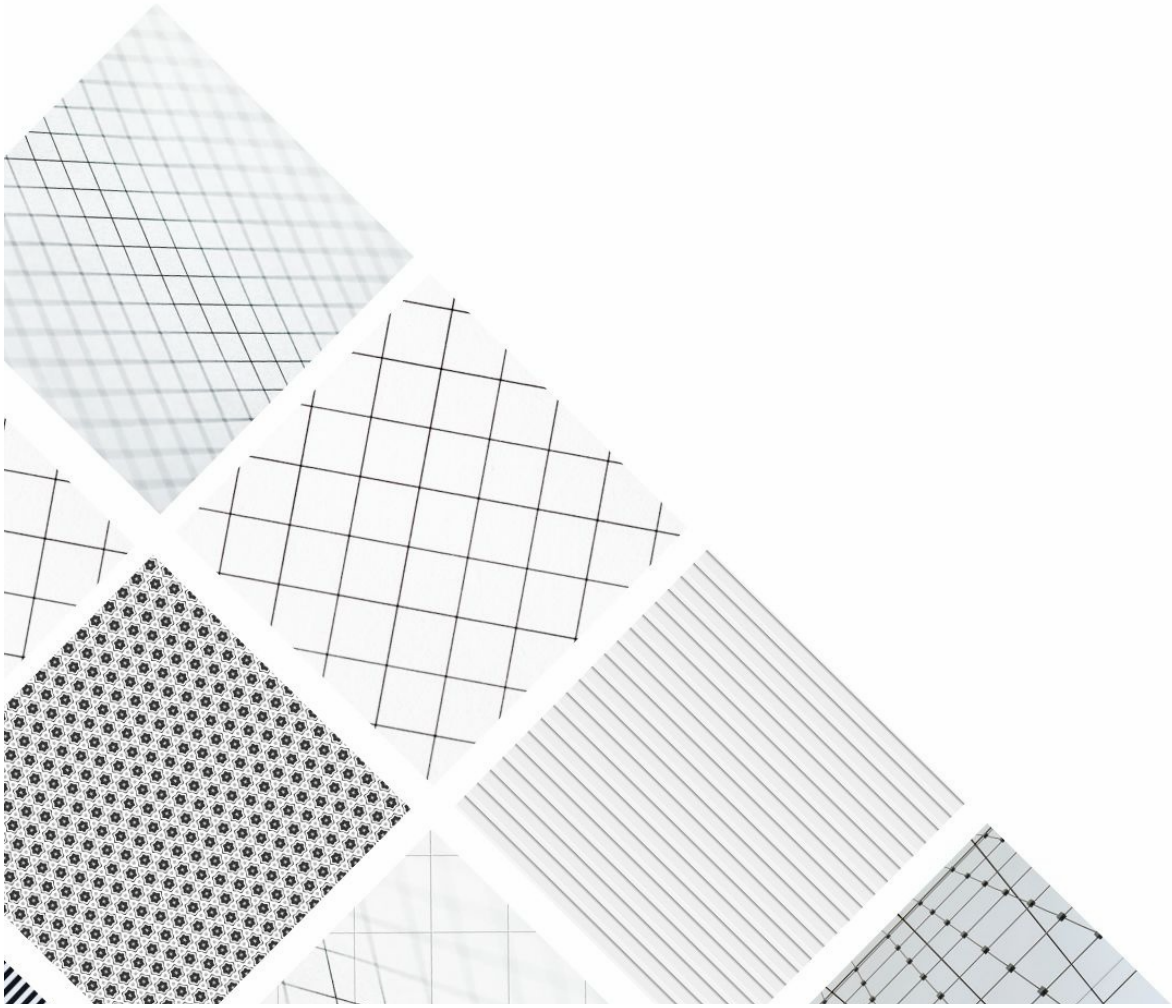


Table of Contents

From the AIBOM Project Lead

I spent time co-chairing the SBOM initiative under CISA at DHS, and the lesson that stayed with me longest was not about file formats. It was about what happens after the format ships. We got the software industry to generate SBOMs at scale, and then watched adoption stall right at that point, because generation was never the hard part. The hard part was building the operational layer around it: who consumes the artifact, what decision it feeds, what happens when it goes stale, and who is accountable when it does not exist at all. AIBOM 101 is written with that lesson in mind. It is not a schema reference. It is the operational layer for AI systems that we never fully built for software.

An AI Bill of Materials only earns its place if it changes what a security team, an auditor, or a procurement reviewer can actually do. That is why this document spends as much time on scope, completeness claims, and evidence as it does on components and data flows. A record nobody can trust and nobody is required to consume is an engineering byproduct wearing a governance label, not infrastructure. Our aim with this 101 is to hand enterprise security leaders something they can put to work immediately: a way to ask what AI they have, what it depends on, and whether anyone can vouch for those dependencies, using language that holds up in a boardroom as well as a build pipeline.

We wrote this to be read, not just referenced. The worked examples inside, a hosted inference system, a fine-tuning pipeline, walk through the reasoning a producer actually has to do: where the system boundary sits, what counts as evidence, what a completeness claim is allowed to imply. The enterprise adoption roadmap does the same for the organizational side, moving from a named inventory of AI systems through a pilot, into governance integration, and finally into continuous operation where AIBOMs regenerate automatically on defined triggers. Each phase has an exit criterion you can check, on purpose, because a roadmap without a way to know you have cleared a phase is just an aspiration.

We also built this to move. Agentic AI is not a future edge case; it is already reshaping what "dependency" means, since a system's tool calls, sub-agent delegations, and MCP connections can now be decided at runtime rather than fixed at design time. Rather than treat that as an exception to bolt on later, we designed AIBOM as a graph from the outset, one that extends naturally into what we call AgBOM: agent identities, tool permissions, and multi-agent interaction patterns captured with the same rigor as models and datasets. The architecture is meant to flex as the field does, without forcing a rewrite every time a new pattern emerges.

None of this happens in isolation. This document is the product of the OWASP AIBOM project's workstreams, Requirements, Best Practices, Integrity and Quality, Threat Intel, Policy, Content, Alliances, Engagement, Sponsorship, and AgBOM, run as one interconnected system rather than separate efforts. Each workstream has pushed us to keep the guidance honest about what an AIBOM can and cannot claim. We have tried to be direct about that boundary throughout: an AIBOM records what a system contains, not whether it is safe or compliant. That judgment stays with the reader, using their own risk framework, and it will stay that way as this project matures. The project, its schema work, and ongoing releases live at owaspaibom.org, and we welcome contributors from any of the workstreams above.

If SBOM taught the software industry anything, it is that a good format without an operating discipline around it is a shelf artifact. Our goal for AIBOM is that it never becomes one. I hope this document gives you the starting point to make that true inside your own organization.

Aruneesh Salhotra
Project Lead, OWASP AIBOM
www.owaspaibom.org

Scope and Audience

This document is an introduction to the AI Bill of Materials (AIBOM): what it is, what it must contain, how it is produced and validated, and how an organization adopts it. The intended audience is primarily the people who produce and consume AIBOMs — AI and ML engineers, MLOps and platform teams, security architects, and product security and supply-chain risk practitioners. It is written to be read end to end by someone with no or little prior AIBOM experience; familiarity with software SBOM practice is helpful but not assumed.

We also aim to inform CISOs, AI governance leads, and senior leaders accountable for AI supply-chain risk, who will benefit from the document's framing of why AIBOM matters, the security, regulatory, and governance drivers behind it, and the phased adoption roadmap and maturity indicators. In addition, this document touches on the broader regulatory and governance context around AI systems, and may be useful to compliance, legal, and procurement or third-party risk teams that set AIBOM disclosure requirements for suppliers.

This document describes AIBOM structure conceptually and shows how that structure maps onto an established, machine-readable BOM format. It is not a field-level schema reference; the format specification remains authoritative for encoding detail, and readers should consult its current release rather than treating the examples here as a schema. It does not provide a threat taxonomy for AI systems, an AI risk management methodology, or vendor- and tool-specific implementation instructions — those are maintained in companion OWASP resources and in the referenced standards, and are cited throughout.

Copyright and License

EXECUTIVE SUMMARY

Most organizations cannot reliably say which model is running in production, trace the data and upstream suppliers behind it, or show whether a newly disclosed vulnerability reaches a system they depend on. Modern AI systems are assembled from third-party models, licensed and internal data, open-source frameworks, retrieval pipelines, agents, tools, and provider-managed services - many of which change without any local release. Traditional software inventory was not built to capture these non-code artifacts and external services, yet that is where an AI system's behaviour, and its risk, is largely determined.

An AI Bill of Materials (AIBOM) closes this gap: a structured, machine-readable account of what an AI system actually contains and how information flows through it. By design it records fact rather than issuing a score or a compliance verdict, which is what lets it serve as the common, verifiable foundation for security, governance, procurement, and audit decisions alike.

For security leaders in particular, this is leverage over decisions you already own. An AIBOM turns AI systems from opaque dependencies into accountable ones - shortening vulnerability response from weeks to hours, giving procurement and third-party risk a concrete disclosure standard to hold suppliers to, and providing a single, defensible evidence base for governance reviews and board reporting. The most effective adoption is not a new parallel programme but making the AIBOM a required input to the controls you already run: release gates, vendor assessments, model risk reviews, and incident response - so one record serves security, compliance, and audit without being prepared three times.

This document is a practical entry point to producing one. It explains what an AIBOM is and why it is becoming an operational requirement rather than an optional practice, shows how an AIBOM is structured and produced through worked examples, extends the model to the added demands of agentic AI systems, and sets out a phased roadmap for adopting AIBOMs across an enterprise.

1. Understanding AI Bill of Materials

AI systems depend on more than code. A production system draws on model weights, training and fine-tuning datasets, adapters and checkpoints, prompts and templates, retrieval corpora, guardrail and moderation services, orchestration frameworks, and provider-managed APIs. Together these form the AI supply chain, and it differs from a software supply chain in one decisive respect: its most consequential dependencies are frequently not software packages at all. They are non-code artifacts and externally operated services whose behavior can change without any local release.

These dependencies do not concentrate at a single stage. They accumulate across the lifecycle. During design, teams choose model providers and data sources. During development, they introduce fine-tuning data, prompts, and evaluation assets. During deployment, they add infrastructure, safety controls, and integrations. In production, systems depend on retrieval corpora that change continuously, moderation services that materially shape outputs, and provider-side updates that arrive on someone else's schedule. A critical dependency can therefore enter at any stage, and much of it sits outside the deploying organization's direct visibility.

Consider an enterprise assistant built on a hosted foundation model API. From the application team's perspective nothing has changed: the prompts, the retrieval configuration, and the infrastructure are all as they were. Yet providers publish API changelogs and maintain lifecycle policies that distinguish active, deprecated, and retired models, and a versioned alias can be repointed at a newer snapshot. The model that was evaluated and approved may no longer be the model shaping production behavior. When output quality shifts or an incident occurs, the team cannot answer the first three questions asked of it - what changed, when, and whether the trigger was upstream - because the information needed to answer them was never recorded anywhere.

The same gap appears without a hosted provider in the picture. Fine-tuning pipelines that lack lineage tracking cannot demonstrate which base model and which data produced a deployed artifact. Retrieval sources change without versioning. Safety and moderation layers are treated as implementation detail despite their effect on outputs. The pattern is consistent: organizations inherit capability without inheriting understanding. They know which application they deployed; they cannot show which upstream model, dataset, or control layer actually shaped its behavior.

This is where conventional software inventory reaches its limit. An SBOM records software components and their declared versions, which remains necessary and does not become sufficient. Determining what an AI system will do requires knowing the pretrained models, the data they were trained and evaluated on, the evaluation results that justified deployment, and the provider-managed services in the request path - none of which are software components in the SBOM sense. Software inventory tells you what code is present. AI supply chain visibility must also tell you which non-code

artifacts and upstream services materially shaped behavior, and with enough provenance to reconstruct the system as it stood at a given point in time.

Agentic systems intensify this rather than changing it. When an agent selects tools, calls external services, or delegates to other agents at runtime, the dependency set is no longer fixed at design time, and three views of it diverge: what configuration declares, what permissions make reachable, and what execution traces show was actually used.

Understanding AI Bill of Materials as Governance Artifacts

An AIBOM is not primarily an engineering convenience. It is a governance artifact: a record authored by an accountable party, in a form another party can read later and act on. That distinction matters because the reader is usually not the author, and usually arrives under pressure - a security engineer scoping an incident at 2 a.m., a procurement reviewer assessing a vendor's disclosure, an auditor asking which datasets trained a model that made a contested decision eighteen months ago. Four properties make a record usable in those moments.

- **Accountable authorship:** Every claim in an AIBOM is asserted by someone. Document metadata records who authored, reviewed, and approved it, when, and by what method - and that metadata is what lets a reader distinguish a record generated from build telemetry from one hand-drafted during procurement review. Two documents can contain identical claims and warrant very different confidence.

- **Declared scope and an explicit completeness claim:** An AIBOM states what it covers and how confident its producer is that it covers it. This is what makes silence interpretable. Without a completeness claim, an unlisted dataset proves nothing; with one, its absence becomes a producer assertion that can be relied on and, if wrong, challenged. Chapter 4 develops these self-descriptions in detail.

- **Claims tied to evidence:** Hashes, signatures, model cards, dataset records, and evaluation results attach to individual claims, which converts assertion into something a consumer can independently check. An AIBOM need not evidence every claim; where evidence is absent, that absence is itself information the reader uses to decide how much weight the claim can bear.

- **A controlled lifecycle:** A governance artifact is versioned, reviewed, approved, updated on material change, retained, and preserved when the system it describes is retired. An AIBOM produced once and never refreshed describes a system that no longer exists, and reading it is worse than having nothing, because it invites false confidence. Chapter 7 sets out the ownership and update triggers this implies.

These properties are what allow an AIBOM to be consumed as evidence in decisions that already exist: supplier assessment and contractual disclosure requirements, release gating, vulnerability impact analysis, incident scoping, model risk review, and regulatory or audit response.

Understanding What an AIBOM is not: It is not a score, a rating, or a maturity grade; it records what a system contains, not how good or how safe that system is. It does not assert compliance, and it does not substitute for evaluation, threat modeling, or risk assessment - it is an input those activities consume, and its value depends on their being performed. Nor is an AIBOM a control in itself: it blocks nothing until an organization builds a gate, a review, or a query that reads it. An AIBOM that no one consumes is an engineering byproduct wearing a governance label.

2. What is an AIBOM?

An AI Bill of Materials, SBOM for AI, or AIBOM is a structured, machine-readable description of an AI system in four parts: a header that declares what the document covers and how complete it claims to be, an inventory of the system's components and services such as models, datasets, agents and tools, guardrails, and runtime elements, the directed data flows between them, and the evidence of origin, rights, integrity, and evaluation attached to those claims.

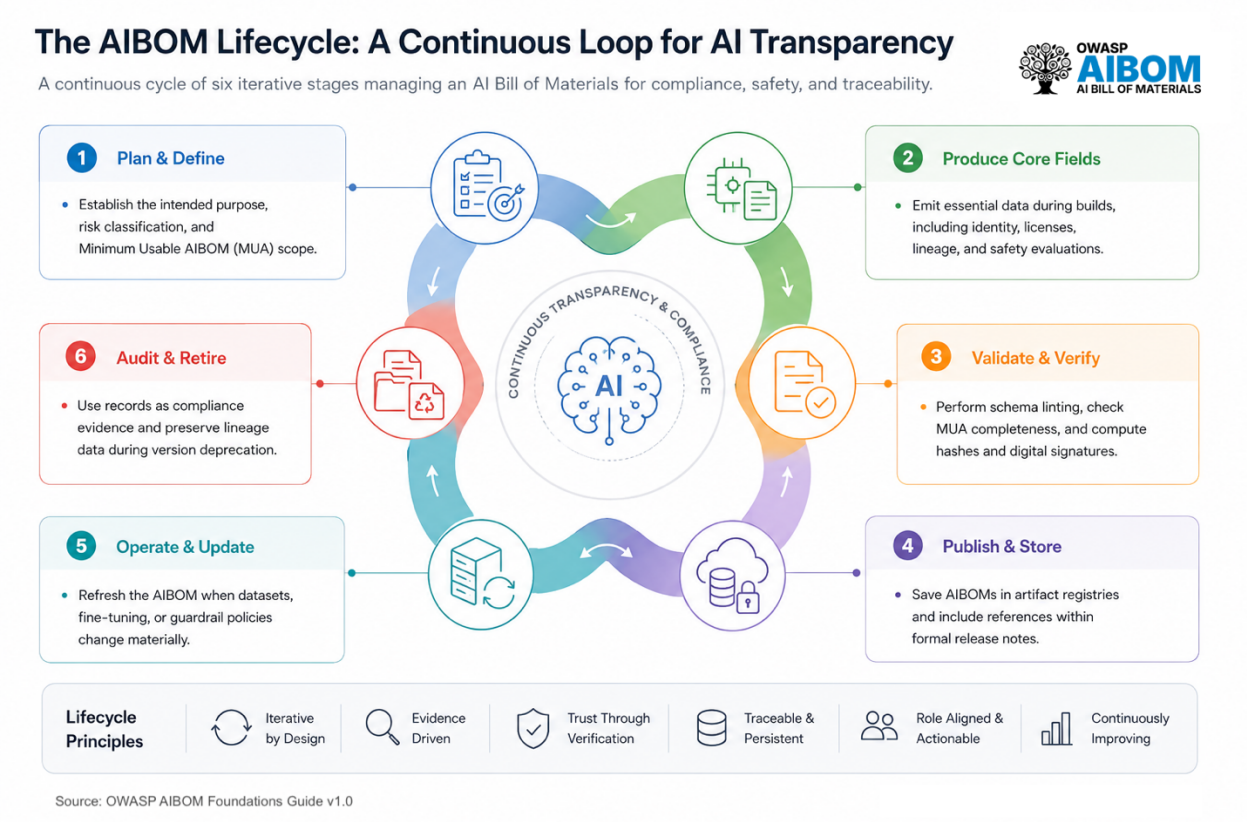
The goal of an AIBOM is to improve transparency and trust across the AI supply chain. An AIBOM provides visibility, which enables organizations to inspect and analyze every component of their AI ecosystem. AIBOM upholds developing and existing risk management practices and processes. It helps identify vulnerabilities, assess AI-related security risks, and track model provenance from development through deployment. An AIBOM supports organizational efforts to meet applicable regulatory and risk management requirements.

Core objectives
An AIBOM provides a comprehensive record of AI artifacts and lifecycle components of an AI system throughout its design, development, and deployment lifecycle, which can serve as a primary evidence used for AI audits.

AIBOM Artifacts refer to AI system components and assets which include licenses and usage obligations for models and datasets, verifiable provenance, and evaluation evidence. Examples of artifacts include:

- Models: base model digests, declared and concluded licenses, distribution channels, signatures, fine-tuned, LoRA/PEFT adapters, checkpoints
- Datasets: Dataset ID/Version, hash, licenses, source URLs, snapshots, collection methods, sensitive/anonymization
- Agents & Tools: Orchestrators, APIs, Extensions, MCP servers
- Prompts & Guardrails: Templates, Policy packs, security notes
- Runtime: Containers, dependencies, timestamps
- Build Provenance: digests of the built images, the input source locations, the build arguments, and the build duration

A core objective of an AIBOM is to systematize an iterative process of documenting artifacts of AI system components throughout the design, development, and deployment lifecycle.



Phases of AIBOM development:

1. Plan: Define intended purpose, scope, risk class and Minimum Usable AIBOM (MUA).
2. Produce: Emit core fields during training/builds including but not limited to identity, licenses, provenance, lineage, eval, safety
3. Validate: Lint the files (schema), confirm MUA completeness, compute hashes, signatures and check licenses.
4. Publish: Store with the artifact (registry/artifact store), keep a reference in release notes.
5. Operate: Update on material changes like datasets, fine-tuning. Licenses, new/updated guardrail policies.
6. Audit/Retire: Use AIBOM as primary evidence, preserve lineage when deprecating versions.

3. Why do AIBOMs Matter?

Modern AI systems are rarely built end to end by a single team. A typical production stack pulls a base model from one provider, fine-tunes it on a mixture of internal and licensed data, integrates open-source components, and connects to live data through retrieval pipelines. Each of these layers carries its own risks, licenses, and provenance questions.

Without a formal inventory, organizations frequently cannot answer fundamental questions: Which model version is deployed in production? What data was it trained on? Are any of its components affected by a newly disclosed vulnerability? Is the license compatible with commercial use? An AIBOM addresses these questions by acting as the canonical, queryable record of what an AI system actually contains.

The remainder of this document examines the five primary drivers pushing AIBOM from a useful idea into an operational requirement.

Security Drivers

AI systems introduce new attack surfaces that traditional application security tooling was not designed to cover. An AIBOM is the foundation of any credible AI security program because it tells defenders what they are protecting.

- **Vulnerability response.** When a vulnerability is disclosed in a popular model, framework, or serialization format (such as the recurring pickle exploits affecting publicly hosted model repositories), an AIBOM lets security teams identify exposure within hours instead of weeks.
- **Model and data poisoning.** Malicious actors can compromise upstream models or training datasets. An AIBOM captures provenance, hashes, and signing information so tampering can be detected before deployment.
- **Backdoor and trojan detection.** Knowing the lineage of every model artifact, including who trained it and on what data, is essential for evaluating the risk of hidden behaviors triggered by specific inputs.
- **Secrets and prompt leakage.** Inventorying prompts, system instructions, and embedded credentials helps prevent accidental exposure when models are shared or fine-tuned downstream.
- **Incident response.** When an AI system produces harmful or anomalous output, an AIBOM lets responders reconstruct exactly what was running, on what data, with what configuration at the time of the incident.

Regulatory Drivers

The regulatory landscape for AI has shifted from voluntary principles to enforceable obligations in a remarkably short time. AIBOMs are emerging as the practical mechanism for demonstrating compliance across multiple frameworks at once.

- **AI-specific regulation.** Emerging AI regulations[1] increasingly require high-risk AI systems to maintain technical documentation covering training data, model architecture, performance metrics, and known limitations. An AIBOM operationalizes these requirements in a standard format.
- **AI risk-management frameworks[2].** Their core functions explicitly call for documented inventories of AI components, data, and dependencies as a precondition for risk assessment.
- **Government procurement and agency guidance[3].** Public-sector bodies increasingly require AI system documentation when procuring or deploying AI, mirroring the SBOM mandates introduced for traditional software.
- **Sector-specific rules.** Financial services regulators, healthcare authorities, and data protection regulators are converging on similar disclosure expectations for AI-driven decisions.
- **Cross-border transfers.** Where data residency or sovereignty rules apply, an AIBOM provides the audit trail needed to prove that training data and model components meet jurisdictional requirements.

Governance Drivers

Internal governance has historically lagged behind AI adoption. Many organizations discover, often during an audit or incident, that no one can definitively say which models are in production, who owns them, or how they were validated. AIBOMs give governance functions a concrete artifact to anchor policies, reviews, and approvals.

- **Model lifecycle management.** An AIBOM tracks every version, fine-tune, and deployment of a model, replacing tribal knowledge with auditable records.
- **Bias, fairness, and ethical review.** Reviewers cannot assess whether a model is appropriate for hiring, lending, or healthcare without knowing what data trained it. The AIBOM is the input to those evaluations.
- **License and intellectual property compliance.** Models and datasets carry diverse licenses, some permissive, some restrictive, and some with unresolved

[1] EU AI Act, Regulation (EU) 2024/1689 (see Art. 11 & Annex IV): eur-lex.europa.eu/eli/reg/2024/1689/oj/eng
[2] NIST AI RMF 1.0 (NIST AI 100-1): nist.gov/publications/artificial-intelligence-risk-management-framework-ai-rmf-10
[3] NTIA *Minimum Elements for a Software Bill of Materials* (2021): ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom; and the 2025 CISA update: cisa.gov/…/2025_CISA_SBOM_Minimum_Elements.pdf

questions about training data copyright. Surfacing these obligations early prevents costly remediation later.

- **Third-party and vendor risk.** When AI capabilities are sourced from vendors, an AIBOM is increasingly part of due diligence, alongside SOC 2 reports and security questionnaires.
- **Board-level visibility.** Aggregated AIBOM data lets executives and boards see the organization's AI footprint in a single view rather than as scattered project artifacts.

Business Continuity

AI systems often become quietly critical. A recommendation engine, fraud model, or document processor can move from pilot to load-bearing infrastructure in months. When something breaks, the cost of not having an AIBOM is measured in downtime, revenue loss, and reputational damage.

- **Reproducibility.** If a production model is lost, corrupted, or needs to be retrained, an AIBOM captures the inputs, parameters, and dependencies needed to rebuild it predictably.
- **Vendor failure or deprecation.** When a model provider shuts down a service, changes pricing, or deprecates a model, an AIBOM tells you which systems are affected and what alternatives are compatible.
- **Drift and degradation.** Tracking model and data versions over time makes it possible to correlate quality regressions with specific changes rather than guessing.
- **Regulatory holds and audits.** When regulators or courts require evidence of what a system was doing on a specific date, an AIBOM provides the historical record.
- **Disaster recovery planning.** Recovery time and recovery point objectives for AI systems are only meaningful if the components needed to restore them are inventoried and accessible.

Supply Chain Threats

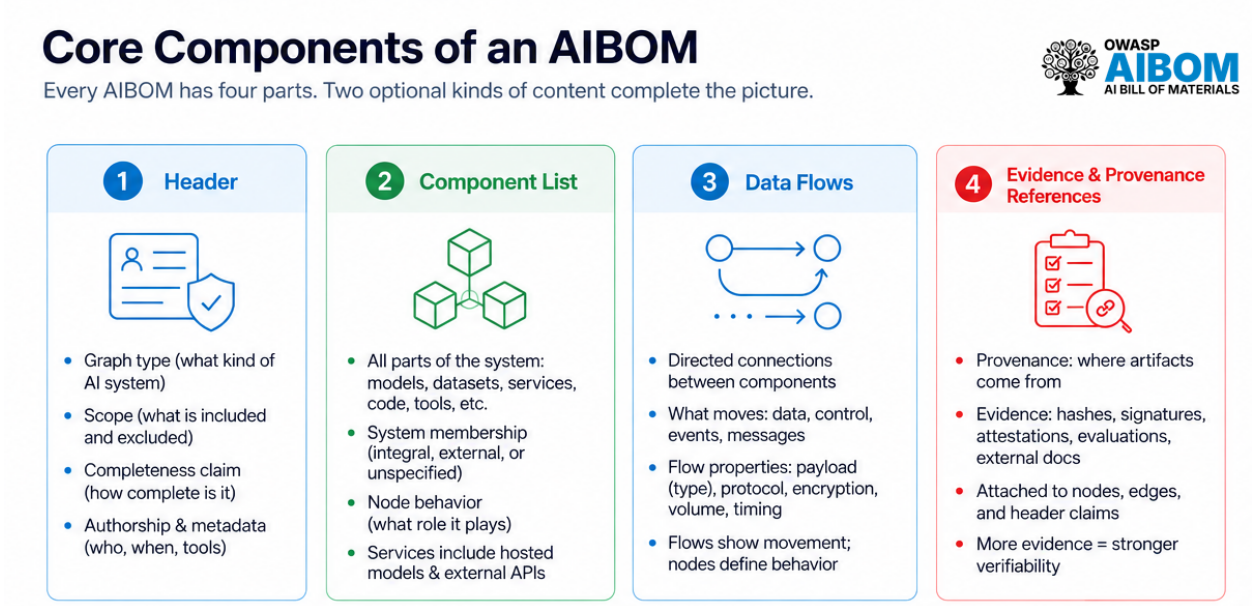
The AI supply chain is broader, deeper, and less mature than the traditional software supply chain. Components are pulled from public model hubs, dataset repositories, and inference APIs that operate with varying levels of security hygiene. AIBOMs make the supply chain visible enough to defend.

- **Compromised public model repositories.** Public hubs have repeatedly hosted models containing malicious code in their serialization formats. An AIBOM with verified hashes makes this class of attack detectable.
- **Typosquatting and impersonation.** Attackers publish look-alike models or packages designed to be pulled accidentally. Provenance tracking in an AIBOM is the primary defense.
- **Dataset contamination.** Training or fine-tuning data sourced from the public web can be deliberately seeded with poisoned content. Cataloging dataset sources and processing steps is the starting point for detection.
- **Transitive dependencies.** Every model carries upstream dependencies, on tokenizers, base weights, and supporting libraries. AIBOMs make these transitive relationships explicit.
- **Insider and developer risk.** Internal fine-tuning and data labeling pipelines can introduce risk through unvetted contributions. AIBOM-driven controls treat internal artifacts with the same rigor as external ones.

4. Core Components of an AIBOM

Earlier chapters explained what an AIBOM is and why it matters. This chapter introduces the parts of an AIBOM and what each part is for, so a reader knows what belongs in one and why.

Structurally, an AIBOM has four parts: **a header, a component list, directed data flows, and evidence and provenance references**. Every AIBOM has all four. Two optional kinds of content complete the picture where a system calls for them: trust context, which groups components into zones and marks the boundaries between them, and behaviors, which describe conditional action for systems whose data movement depends on runtime decisions. This chapter walks through each in turn, connects each to the purpose it serves, and closes with what a finished AIBOM enables.



The Header

The AIBOM header is used to describe what kind of AI system is being modeled and what claims can be made with regard to scope and completeness. This information provides context for reading the rest of the document. One producer's AIBOM might be a complete picture of a single deployed system, generated automatically from build telemetry. Another might be a partial sketch of a future system, drafted by hand during procurement review. Without a way to tell these apart, a reader cannot draw the right inferences from either one.

Every AIBOM therefore opens with **three self-descriptions**: (1) The graph type says what kind of system the AIBOM is depicting, such as a single model, a runtime system, a system extension like an MCP server, or a training pipeline. (2) The scope says what the AIBOM is trying to cover: which system, which version, which deployment, which interfaces. (3) The completeness claim says how confident the producer is that the AIBOM covers what its scope says it covers, ranging from "complete for the declared scope" through partial coverage to "we don't know what else is in here." Alongside these self-descriptions, standard document metadata records who authored, reviewed, and approved the AIBOM, when, and with what tools. That metadata is how a reader distinguishes an automatically generated record from a hand-drafted sketch.

The completeness claim guides what a reader may meaningfully conclude from the absence of information. If an AIBOM does not list a particular dataset and does not claim to be complete, the absence proves nothing. The dataset might be in use and simply unlisted. If the AIBOM does claim to be complete for the declared scope, however, the absence becomes meaningful: it is now a producer assertion that no such dataset participates within scope.

This asymmetry between positive and negative inference is why graph self-description must come before graph content. Without scope and completeness, two readers can look at the same AIBOM and reach opposite conclusions about what it says. The remaining sections of this chapter assume each AIBOM carries all three self-descriptions, even where the prose does not repeat them.

The Component List

An AI system is made up of a number of interrelated components. These are the "materials" in the AI Bill of Materials. In an AIBOM, everything in the system is a component, including services: a discrete code module, a microservice, a dataset, and a provider-hosted API endpoint are all components, distinguished by their component type rather than by separate lists. The component list answers the first question every reviewer brings to a system: what am I depending on?

Two things describe a node beyond its identity: system membership and node behavior. System membership asks whether a node is considered an integral part of the described AI system, external to it, or left unspecified. This is a scope-relative statement, and it is recorded structurally: the AIBOM's declared scope lists which components are included in, and which are excluded from, the described system. It is not the same thing as ownership or control. A third-party hosted model can be integral to the described AI system, and an internally operated database can still be external to the AI system boundary if it sits outside the declared scope. A component included in the declared scope is a constituent of the described system.

In contrast, node behavior asks what role the node plays in the graph. Some nodes perform work. A prompt processor transforms text. A model generates output. An agent decides what to call next. A guardrail filters content. A retrieval service ranks candidates. Other nodes primarily store, supply, or receive information, such as

datasets, indexes, logs, and data stores. A node that does not directly process the information may be informatively annotated as a “passive” component, or that status may be inferred from the pattern of data flows connecting to it.

Hosted models are among the most important service-type components for many AIBOMs, but many other kinds exist. A service component may allow a model or AI application to get access to data, tools, or systems through a network API. The AIBOM records these components so that a reader can see which parts of the system exchange data and where system boundaries are crossed.

The Data Flows

The data flows in an AIBOM connect components together. If the components are the "parts list," the data flows are the "construction diagram" or "wiring diagram" that shows how they fit together. Data flows are always directional: they show information moving from one component to another. When information moves in both directions between two components, such as a request and its response, each material movement is its own flow.

If the component list answers what am I depending on, the data flows answer the question that makes AI systems different: what can reach what? The same model has a completely different risk profile depending on whether customer data can reach it unfiltered and where its output goes. A parts list cannot express that difference. For AI systems, the connections are where the risk lives.

Each flow can carry information about what moves along it. A flow declares its kind — a data flow, a control flow, an event, a message — and can describe the payload it carries, either by referencing data objects described elsewhere in the AIBOM or with an inline descriptor when the payload on that specific flow differs from anything described elsewhere. A flow can also record how the movement happens: whether it is encrypted, over what protocol, at what volume, and with what timing.

What a flow does not carry is a description of what the receiving component does with the data. A model generating output, a retrieval service ranking candidates, a guardrail filtering content, or a database persisting what it receives is node behavior: it is described on the node, through its component type and metadata, not as an annotation on the flow. A flow records that data moved from one place to another; the nodes explain what happens on each end.

Evidence and Provenance References

The header, the components and services, and the data flow edges say what the system contains and how information moves through it. Evidence and provenance references say why a reader should believe those claims and where they can be checked. They are not a separate graph; they are attachments to specific nodes, edges, and self-descriptions.

Two kinds of attachment matter most. Provenance records how an artifact came to exist: which base model a fine-tuned model derives from, which training run produced it, which dataset snapshot fed a retrieval index. Evidence records the support for a claim: a cryptographic hash that anchors a component's integrity, a signature or attestation that ties an artifact to the build that produced it, an evaluation result, or an external document such as a model card or license file.

Because these references are scoped to individual claims, they interact directly with the completeness claim in the header. A component asserted without any evidence reference is a bare assertion; the same component carrying a hash, a signature, and a link to its model card is a claim a consumer can independently verify. An AIBOM need not attach evidence to every claim, but the presence or absence of evidence is itself information a reader uses to decide how much weight a given node or edge can bear.

The elements a producer records against these four parts — model metadata, dataset provenance, training environment, dependencies and libraries, APIs and toolchains, runtime configuration, and agent and sub-agent descriptors for agentic systems — are not a separate list to maintain. Each is either a node, a property of a node, an edge, or an evidence reference attached to one of them.

Trust Zones and Boundaries

Components do not all live in the same place or under the same control. An AIBOM can group components into zones of differing trust: the team's own infrastructure, the wider enterprise network, a provider's cloud, the public internet. Where a flow crosses from one zone into another is a boundary, and boundaries are where review attention concentrates. A crossing is where data changes hands, where controls such as authentication and encryption apply, and where the producer's visibility ends.

Zones exist to make a difference explicit that a bare graph hides. A flow between two internal components and a flow that carries the same payload out to a third-party API look like similar edges, but they are very different facts. Without declared zones, telling them apart depends on a reader's background knowledge of which names are internal. With them, the AIBOM itself says which movements stay home and which leave — and what each departure is required to pass through on the way out.

Behaviors

Flows say what can reach what. Behaviors say when, and under what conditions. For a linear pipeline, the flows are usually enough: the same movements happen for every request. Agentic systems are different. The interesting fact about an agent is often that it decides at runtime whether to act at all — whether to call a tool, query a database, or hand off to another agent. A flow alone either overstates what happens (the tool is invoked every time) or understates it (the connection goes unrecorded). A behavior description captures the conditional part: what triggers an action, what the component may do in response, and what follows from it. Most AIBOMs will not need behavior

descriptions; systems whose risk depends on runtime decisions will. The chapter on agentic AI systems returns to this.

What an AIBOM Enables

Everything this chapter has described is structured fact, and every piece of it — each component, flow, zone, and boundary — carries a stable identifier. That is what separates an AIBOM from an architecture diagram: other artifacts can point at exactly one thing in it.

That referenceability is what the rest of the governance stack builds on. Threat models, risk registers, control mappings, and vulnerability statements do not re-describe the system; they reference the AIBOM's graph. A threat can say that an attack path crosses a specific boundary. A control can say which flow it protects. A vulnerability analysis can say that a weakness is unreachable because a zone's controls block the path to it. Build the graph once, and the analyses attach to it — and stay attached, and stay checkable, as the system evolves.

The AIBOM itself stays on its side of that line. It records what the system is, not whether it is acceptable. Compliance, risk acceptability, and control sufficiency are judgments a reader brings, using their own policy framework, on top of the facts the AIBOM provides.

5. AIBOM Structure

The chapter "Core Components of an AIBOM" set out the four parts of an AIBOM — header, components and services, data flow edges, and evidence and provenance references — as an abstract model. This chapter makes that model concrete. It works through examples of a hosted inference system, a fine-tuning pipeline, and an end-to-end run that generates an AIBOM for a published model; states plainly what an AIBOM does and does not assert; and introduces the encoding tiers and layers a producer needs when sitting down to actually emit one.

The three self-descriptions in the header — graph type, generation method, scope, and completeness — are assumed throughout and are not re-derived here; see "The Header" in "Core Components of an AIBOM." Nothing else in the AIBOM is interpretable without them, so each example below is written as if its header were already in place, even where the prose does not repeat it.

What every AIBOM declares

Before anyone can read an AIBOM, they need to know what kind of AIBOM it is.

The same words can describe very different artifacts. One producer's AIBOM might be a complete picture of a single deployed system, generated automatically from build telemetry. Another's might be a partial sketch of a future system, drafted by hand during procurement review. Without a way to tell these apart, a reader cannot draw the right inferences from either one.

Every AIBOM therefore opens with three self-descriptions:

The **graph type** says what the AIBOM is depicting: a single model, a single dataset, a runtime system, a training pipeline, or another well-defined artifact. It tells the reader what kind of nodes and flows to expect.

The **scope** says what the AIBOM is trying to cover: which system, which version, which deployment, which interfaces. Scope matters because every other claim in the AIBOM is relative to it.

The **completeness** claim says how confident the producer is that the AIBOM covers what its scope says it covers. Completeness ranges from "complete for the declared scope" through partial coverage to "we don't know what else is in here."

Alongside these self-descriptions, the AIBOM's document metadata records who authored, reviewed, and approved it, when, with what tools, and over what validity period. That metadata is how a reader distinguishes an AIBOM hand-authored during procurement review from one generated from build configuration or captured from runtime telemetry — and it is where a producer's claims acquire an accountable author.

The completeness claim does more work than it looks like. It governs what a reader is allowed to conclude from the absence of information. If an AIBOM does not list a particular dataset, and the AIBOM does not claim to be complete, the absence proves nothing: the dataset might be in use and simply unlisted. If the AIBOM does claim to be complete for the declared scope, the absence becomes meaningful: it is now a producer's assertion that no such dataset participates within scope.

This asymmetry between positive and negative inference is why graph self-description must come before graph content. Without scope and completeness, two readers can look at the same AIBOM and reach opposite conclusions about what it says.

The remaining sections of this chapter assume each AIBOM carries these three self-descriptions, even where the prose does not repeat them.

Worked example: a hosted inference system

Consider an enterprise customer support assistant. A customer types a question into a web interface. The application assembles a prompt, retrieves relevant internal documentation, checks the input for unsafe content, sends the prompt to a hosted large language model, post-processes the response, and returns it to the customer.

The AIBOM for this system has the following nodes:

A **user-facing service** receives the customer query. It is a service-type component operated by the assistant team.

A **prompt processor** assembles the final prompt. It applies templates, redacts personal information, manages session context, and decides what to retrieve. The team owns it as a component.

A **retrieval service** holds embeddings of internal support documentation and returns relevant passages. The team operates it on internal infrastructure. The underlying corpus, a snapshot of the internal support knowledge base from a specific date, is referenced as a dataset component.

A **moderation service** checks customer input for content the assistant should refuse to engage with. In this example, the team subscribes to a third-party moderation API.

A **hosted language model** generates the assistant's reply. The team accesses it through the provider's API. This is the most interesting node in the graph, and the prose returns to it below.

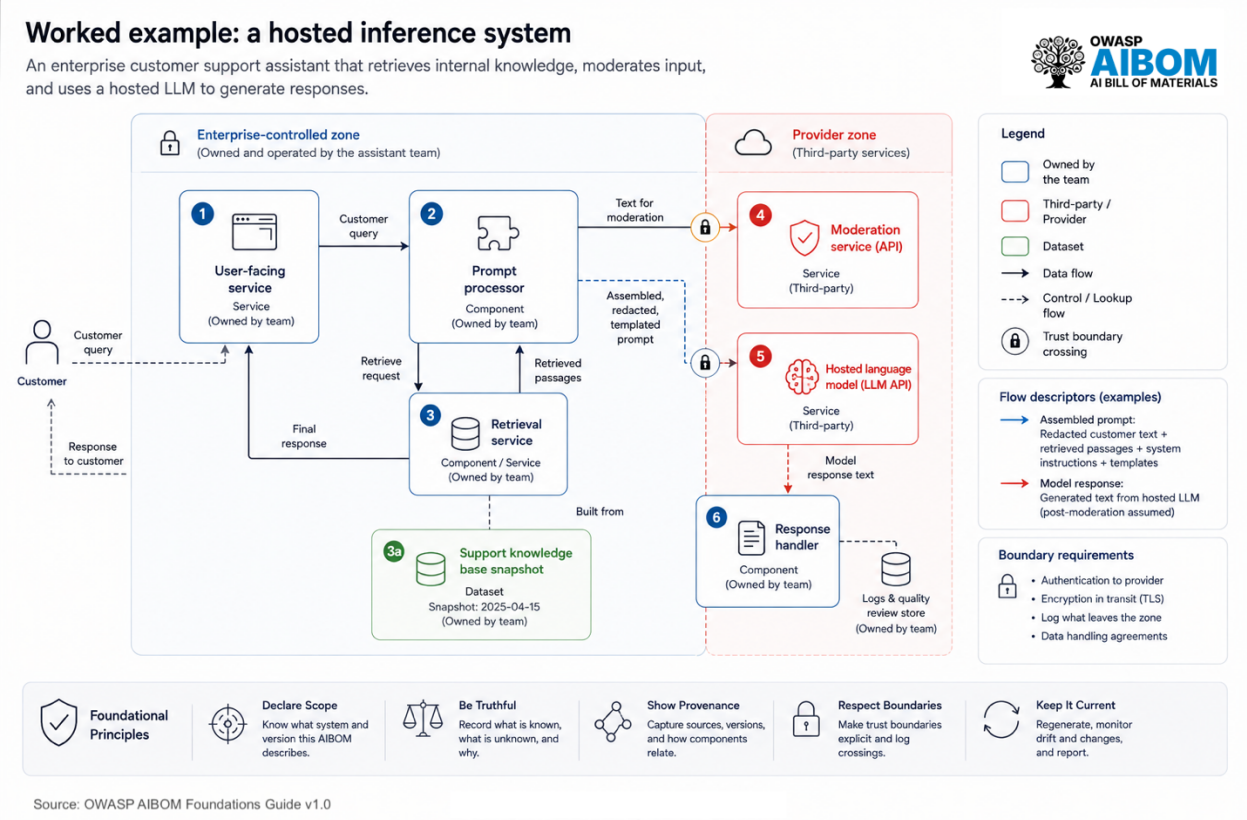
A **response handler** post-processes the model output: it formats the response, logs the exchange for quality review, and returns the final text to the user-facing service.

The AIBOM connects these nodes with directed data flows. The customer query moves from the user-facing service to the prompt processor. The processor sends text to the

moderation service, retrieves passages from the retrieval service, and sends the assembled prompt to the hosted language model. The model returns text to the response handler. The response handler returns text to the user-facing service, which returns it to the customer.

Each flow carries information about what moved along it. The prompt sent to the hosted model is not the raw customer query; it has been retrieved against, redacted, and templated. The flow that carries it can carry an inline description of the payload that traveled on that specific flow, rather than relying on a generic description of what the prompt processor outputs. Per-flow descriptors matter when the payload at one stage in the system differs materially from the payload at another stage.

The example also carries trust context. The user-facing service, prompt processor, retrieval service, and response handler sit in a zone the team controls. The moderation API and the hosted language model sit in provider zones. The two flows that leave the team's zone — the moderation check and the assembled prompt — cross declared boundaries, and each boundary records what its crossings require: authentication toward the provider, encryption in transit, logging of what left. The prompt flow is the one that deserves the most scrutiny, and the graph itself says why: it carries redacted customer text and retrieved internal documentation across a boundary into infrastructure the team cannot see.



Two structural choices in this example deserve discussion.

The hosted language model is represented as a service-type component whose boundary is the provider's endpoint. The reason is that data movement terminates at the provider's endpoint, not inside the provider's infrastructure. The team has no visibility into the provider's serving topology, and an AIBOM that pretended otherwise would assert facts the team cannot demonstrate. The endpoint component is what the team actually sees: an endpoint with an address, an authentication method, and a contract.

The model itself is not absent from the AIBOM, though. The endpoint component carries a reference to a machine learning model component, and that component carries the model's identity, version, license, intended use, training characteristics, and other model card metadata. A reader can enumerate which model the endpoint is serving in one hop. If the provider later changes the alias and points it at a newer model snapshot, the endpoint component does not move. The model component reference changes. The next revision of the AIBOM records the version delta.

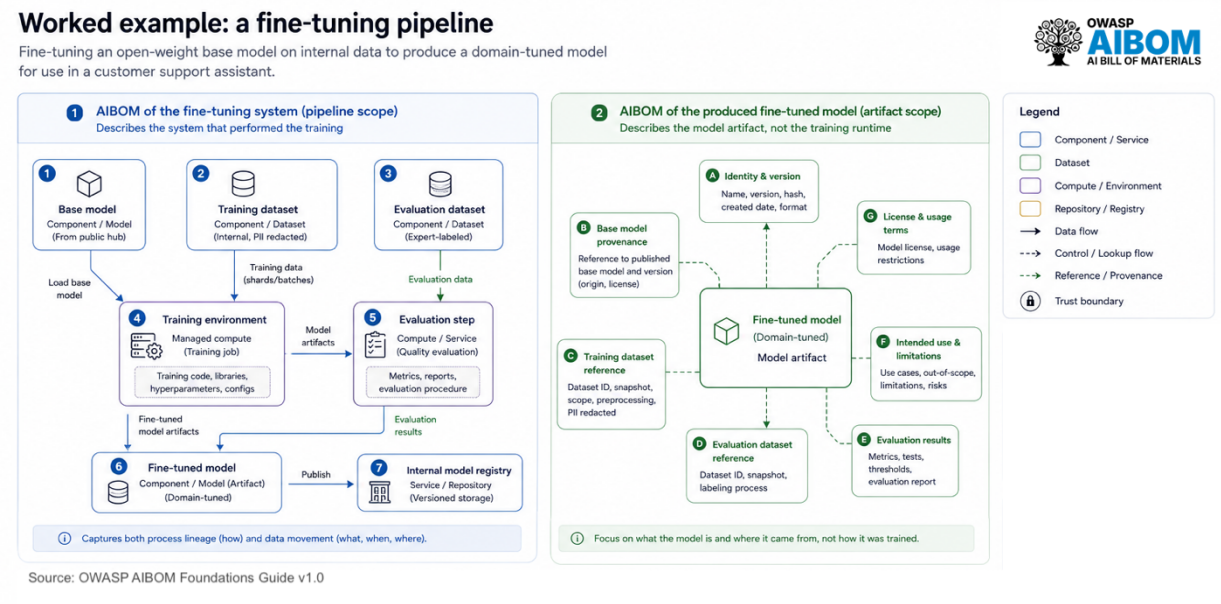
The second choice is how to mark each node as part of the system or outside it. The user-facing service, prompt processor, retrieval service, and response handler are part of the system the AIBOM describes: the team operates them, and the declared scope includes them. The third-party moderation API is more debatable. If the team treats the moderation capability as part of what makes the assistant safe to operate, the moderation service is constituent. If the team treats it as an external dependency the assistant happens to call, it is external. Both choices can be correct; the AIBOM makes the choice explicit so a reviewer knows which model the producer has in mind.

The hosted language model is also a producer choice. An AIBOM whose scope is "the assistant as a deployed capability" will treat the model endpoint as constituent: the model is part of the assistant. An AIBOM whose scope is "the integration we built around an external model" will treat it as external. The AIBOM formalizes this distinction through the declared scope itself, which lists the components included in and excluded from the described system; an introductory reader needs only the underlying idea, which is that node membership is a scope-relative producer choice and not a property of the node itself.

Worked example: a fine-tuning pipeline

Now consider a different system: a team fine-tuning an open-weight base model on internal data to produce a domain-tuned model for use in the assistant above.

The team starts with a published base model retrieved from a public model hub. The team has a training dataset assembled from internal customer support transcripts, with personal information redacted. The team has an evaluation dataset, hand-labeled by domain experts, used to measure model quality before the fine-tuned model is released downstream. The team runs the training job on a managed compute environment, produces a fine-tuned model, evaluates it, and publishes it to an internal model registry.



Source: OWASP AIBOM Foundations Guide v1.0

This system produces two distinguishable AIBOMs, and the distinction matters.

The first is an AIBOM of the fine-tuning system itself: the pipeline that did the training. Its graph includes the base model component, the training dataset component, the evaluation dataset component, the training environment, the evaluation step, and the produced fine-tuned model. The data flows describe how training data reached the training step, how the base model was loaded, how evaluation data reached the evaluation step, and how the produced model was published to the registry.

The second is an AIBOM of the produced fine-tuned model itself, treated as an artifact. Its scope is the model, not the pipeline. It carries the model's identity, its provenance back to the base model, references to the training and evaluation datasets, evaluation results, intended use, and license. It does not describe the fine-tuning runtime: by the time a downstream system loads this model, the fine-tuning runtime is no longer present.

Both AIBOMs reference the same datasets and the same base model. They are not duplicates; they describe different scopes. A consumer doing model risk review reads the artifact AIBOM. A producer auditing the training environment reads the system AIBOM. An organization that wants to demonstrate end-to-end provenance maintains both and links them.

The fine-tuning example also surfaces the distinction between process lineage and data movement. Process lineage records how an artifact came to exist: which base model was used, which training code was run, which hyperparameters governed the job, which evaluation procedure produced the reported metrics. Data movement records which

data went where, between which nodes, when. Training pipelines need both. The fine-tuned model's provenance back to the base model is process lineage. The runtime movement of training shards from a data store into the training compute is data movement. The AIBOM carries both, in their respective places. For an introductory reader, the point is that an AIBOM does not have to choose between the two views, and a training system AIBOM that uses only one will under-describe what happened.

What the AIBOM does not say

A complete AIBOM is structured fact and provenance. It is not a verdict.

It is not a statement that a system is compliant with any particular regulation. It is not a statement that the system's risk is acceptable. It is not a statement that the controls present in the system are sufficient for the risk. It is not a statement about who is legally responsible for any aspect of the system. It is not a trust score, an assurance level, or a quality rating.

This is a deliberate choice in the AIBOM design, not an oversight. A regulator, auditor, internal review board, or procurement team brings policies, thresholds, and risk frameworks of its own to bear on the facts in an AIBOM. Those judgments are downstream. If the AIBOM had already encoded one set of those judgments, the artifact would be useful only within that regime. By staying at the level of fact and provenance, the AIBOM serves as the common substrate that many regimes can build on.

That common substrate is not optional. The AIBOM is what makes those judgments falsifiable. Without it, a risk assessment is an unaudited assertion. With it, every claim in the assessment can be traced back to a node, a flow, an evidence reference, or a declared completeness boundary.

How an AIBOM is written down, for readers going deeper

The conceptual material in this chapter is what a non-implementer needs. For producers who will actually emit an AIBOM, one further point is worth knowing.

The aspects discussed in this chapter are expressible natively in CycloneDX 2.0[4]. The graph lives in a blueprint: components are the nodes, flows are the directed data movements, and zones and boundaries carry the trust context. The self-descriptions map onto the blueprint's declared model types and scope, with the completeness claim carried by the document's composition declarations. Evidence attaches through declarations that reference any node or flow by its identifier. There is no separate AIBOM file format, no AI-specific extension schema, and no compatibility encoding to choose between: an AIBOM is a structured document that uses these standard constructs to describe an AI system.

[4] https://cyclonedx.org/news/cyclonedx-v2.0-coming-soon/

The identifiers do more than internal bookkeeping. Every component, flow, zone, boundary, and claim in the document carries a stable reference, and the standard defines its threat, risk, and control models as siblings of the blueprint that point into the same graph: a threat references the boundary its attack path crosses, a control references the flow it protects, a vulnerability analysis references the components it affects. None of them re-describe the system. This is the structural reason the AIBOM can stay free of verdicts while remaining the substrate every verdict is traceable to - remove the AIBOM and the downstream analyses have nothing to point at.

A producer can still choose how richly to populate the document. A minimal AIBOM - components, flows, and the three self-descriptions - is fully conforming and can be a complete producer assertion. A richly evidenced one adds declarations, compositions, and formulation records for machine-readable evidence and process lineage. Richness is not an assurance level. A complete and well-evidenced minimal AIBOM is more useful than an incomplete rich one. Confidence in an AIBOM comes from completeness, provenance, evidence quality, and the producer's track record, not from which constructs it uses.

6. AIBOM for Agentic AI Systems

Agentic AI systems introduce a new level of complexity to AI supply chains. Unlike traditional AI systems that operate as relatively static pipelines, agentic systems are dynamic, composable, and often autonomous. They interact with external tools, invoke other agents, adapt behavior at runtime, and rely on evolving ecosystems of services and protocols.

A traditional AIBOM approach focuses on static model, dataset, and dependency documentation. Whereas AIBOM for agentic systems must account for dynamic dependencies, runtime interactions, and orchestration logic that together define how these systems behave in production. This section outlines how AIBOM practices extend to agentic architectures, with emphasis on multi-agent relationships, tool ecosystems, and external connectivity..

Multi-Agent Dependencies

Agentic systems frequently consist of multiple cooperating agents, each of which may have distinct roles such as planning, execution, retrieval, validation, or orchestration. These agents may:

- Be built on different models (e.g., GPT-based planner + retrieval agent + tool executor).
- Be owned by different teams or vendors.
- Communicate via structured protocols or informal message passing.

In this context, an AIBOM must explicitly capture:

- Agent identities and roles (planner, executor, critic, retriever, governor, router, etc.).
- Underlying model dependencies for each agent (base model, fine-tunes, adapters).
- Interaction patterns (hierarchical, peer-to-peer, event-driven, hub-and-spoke).
- Control-flow orchestration (who invokes whom, under what conditions, with what escalation paths).

This is critical because risk is no longer isolated to a single model or service. A vulnerability or failure in one agent can propagate across the system. For example:

- A compromised retrieval agent can poison downstream reasoning and recommendations.
- A hallucinating planner can trigger unsafe tool usage or high-risk workflows.
- A misconfigured guardrail agent can fail to block harmful or non-compliant outputs.

Accordingly, AIBOM should represent agentic systems as graphs of interdependent components rather than linear pipelines. The BOM becomes a graph of agents, models, and orchestration edges, enabling impact analysis when one node or edge changes.

Associated Tools

Agentic AI systems derive much of their power from tool use. Tools may include:

- APIs (search, weather, finance, HR, ticketing, etc.).
- Code execution environments and function runners.
- Databases and vector stores. Memory and state
- Internal enterprise services and microservices.
- File systems or document repositories.

These tools are not merely integrations, they are active extensions of the agent's capabilities, often invoked autonomously and at high frequency.

An AIBOM for agentic systems must document, at minimum:

- Tool identity and function (what the tool does, what domain it touches).
- Invocation mechanisms (function calling, HTTP APIs, plugins, framework-native tools).
- Access scope and permissions (read-only vs read/write vs admin; data and resource scope).
- Authentication methods and secrets handling (API keys, OAuth, workload identity, secret stores).
- Rate limits, quotas, and operational constraints (per-agent and global).
- Tool definition version or hash (the exact schema and description the agent was approved to use).

From a security and governance perspective, tools represent a major expansion of the attack surface:

- Prompt injection attacks may manipulate tool usage and cause harmful actions.
- Over-permissioned tools may expose sensitive data or enable lateral movement.
- External APIs may introduce data exfiltration, integrity, or availability risks.
- A tool's definition and behavior can change after review, so approval must bind to a specific definition version rather than a tool name.

Capturing tool dependencies in AIBOM enables:

- Fine-grained access audits (which agents can invoke which tools, with what permissions).
- Tool-level risk assessment (criticality, blast radius, and compensating controls).
- Faster incident response when a tool or connector is compromised (who uses it, where, and how).

Third-Party Connectors

Modern agentic systems often rely on third-party connectors to integrate with external platforms such as:

- SaaS applications (e.g., collaboration platforms, CRM, productivity suites, and ITSM tools such as Slack, Salesforce, Google Workspace, and ServiceNow).
- Data providers (financial feeds, legal databases, healthcare APIs).
- Cloud services and AI platforms (storage, analytics, model APIs).

These connectors introduce supply-chain dependencies beyond traditional software libraries and models, including:

- Vendor-controlled APIs and SLAs.
- Opaque data processing pipelines and transformations.
- External identity and access management systems and policy engines.

AIBOM should therefore capture:

From a security and governance perspective, connectors carry risks that are largely outside the organization's direct control:

- OAuth scopes granted at connector setup may be far broader than the use case requires, giving agents standing access to entire workspaces or record sets.
- Vendors may change API behavior, data handling or sub-processors without notifying downstream AIBOM holders.
- Data may cross residency or sovereignty boundaries through sub-processors that were never disclosed.
- Connector provider and service details (vendor identity, product, region, deployment model).
- Data exchanged (input/output classification, data categories, sensitivity levels).
- Authentication and authorization models (OAuth flows, API keys, service accounts, delegated access).
- Data residency, sovereignty, and compliance considerations.
- Contractual, licensing, and usage constraints that may affect operation or data handling.

A key challenge is that visibility into third-party systems is often limited. Organizations should:

- Require AIBOM-like disclosures or security attestations from vendors where possible.
- Document assumed trust boundaries and residual risks explicitly.
- Track known gaps in transparency, including unknown sub-processors or opaque processing steps.

This ensures that even when full provenance or internal detail is unavailable, risk is explicitly acknowledged, recorded, and managed within the AIBOM framework.

MCP/A2A Ecosystem Tracking

Emerging protocols such as Model Context Protocol (MCP)[5] and Agent-to-Agent (A2A)[6] communication frameworks are rapidly shaping how agentic systems interoperate. These ecosystems enable:

- Standardized tool discovery and invocation across platforms.
- Inter-agent communication across organizational or tenant boundaries.
- Dynamic composition of capabilities at runtime without code deployment.

While powerful, they also introduce highly dynamic and potentially ephemeral dependencies that are difficult to track using traditional BOM methods.

To remain relevant, AIBOM must evolve to include:

- Protocol usage (e.g., MCP, A2A, other agent interoperability standards).
- Registered endpoints and service providers participating in these ecosystems.
- Dynamic capability discovery mechanisms (registries, catalogs, marketplaces).
- Session-level or context-level dependencies, not just static configuration.
- MCP: server identity, transport, version pinning, tool definition hashes and granted scopes.
- A2A: Agent Cards which advertise an agent's identity, capabilities and authentication requirements and which serve directly as an AIBOM input.

For example:

- An agent may discover and invoke a tool at runtime via MCP, without prior static configuration.
- Two agents from different vendors may collaborate via A2A channels, exchanging tasks and results.
- Available capabilities and policies may change without any application redeployment.

To address this, AIBOM implementations should:

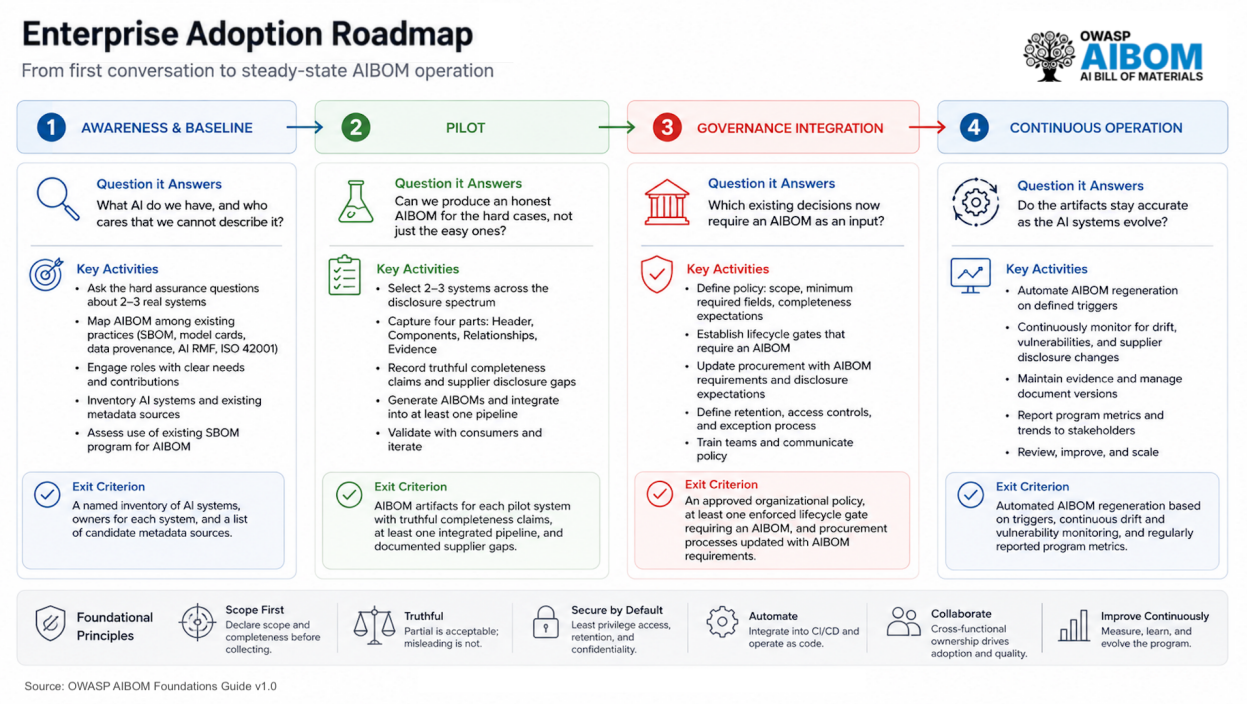
- Incorporate runtime telemetry, logs, and traces as sources of truth for actual dependencies.
- Track capability registries, discovery endpoints, and marketplaces as first-class components.
- Maintain time-bound records of interactions and dependencies (who talked to whom, when, and via what protocol).

This shifts AIBOM from a purely static artifact to a living representation of system behavior over time, blending design-time configuration with runtime observations.

[5] MCP: https://modelcontextprotocol.io/specification/2026-07-28
[6] A2A: https://a2a-protocol.org/latest/

7. Enterprise Adoption Roadmap

Producing a single AIBOM is an engineering task, and the artifact lifecycle described earlier — plan, produce, validate, publish, operate, audit and retire — is the shape that task takes. Doing it for every AI system in an enterprise, repeatably, with the results actually consumed in decisions, is an organizational change programme. This chapter sets out the path from the first conversation to steady-state operation.



Source: OWASP AIBOM Foundations Guide v1.0

The four phases below are ordered by dependency. Each answers one question and ends in a condition you can check.

Phase	Question it Answers	Exit Criterion
Awareness and Baseline	What AI do we have, and who cares that we cannot describe it?	A named inventory of AI systems, an identified owner for each system, and a list of candidate metadata sources.
Pilot	Can we produce an honest AIBOM for the hard cases, not just the easy ones?	AIBOM artifacts generated for each pilot system with truthful completeness claims, at least one integrated pipeline, and documented supplier disclosure gaps.
Governance Integration	Which existing decisions now require an AIBOM as an input?	An approved organizational policy, at least one enforced lifecycle gate requiring an AIBOM, and procurement processes updated with AIBOM requirements.
Continuous Operation	Do the artifacts stay accurate as the AI systems evolve?	Automated AIBOM regeneration based on defined triggers, continuous drift and vulnerability monitoring, and regularly reported program metrics.

Phase 1 — Awareness and Baseline

The case for starting does not need to be rebuilt here: the security, regulatory, and governance drivers set out earlier in this document are the argument. What this phase adds is specificity. The assurance questions an organization cannot currently answer — which model version is in production, what it was trained or fine-tuned on, who the upstream supplier is, and what its licence permits — should be asked out loud about two or three real systems. The inability to answer them is the most effective sponsorship argument available, and it is more persuasive than any framework citation.

Awareness also means placing AIBOM correctly among practices the organization may already run. An SBOM inventories software components and remains necessary; an AIBOM extends the same discipline to models, datasets, prompts and guardrails, and the services in the request path. A model card documents one model's intended use and performance; an AIBOM carries that content as evidence attached to a component, inside a versioned document with a declared scope. Data provenance work supplies the dataset lineage an AIBOM records. Established AI risk-management and management-

system frameworks call for structured documentation of system characteristics without specifying its form; the AIBOM is a form that satisfies the call. AIBOM replaces none of these and duplicates none of them.

Engagement should be role-specific from the outset, because misalignment here produces fragmented pilots that never reach governance.

Function	What it Needs from AIBOM	What it Contributes
Security Architecture and Red Team	AI-specific attack surface information mapped to components.	Threat modeling scope and definition of the security questions the AIBOM must answer.
Legal and IP Counsel	Model and dataset license terms, usage rights, and applicable restrictions.	Interpretation of legal obligations, licensing requirements, and intellectual property compliance.
Compliance and Audit	Evidence supporting AI transparency, traceability, and regulatory obligations.	Retention policies, governance controls, audit requirements, and evidence expectations.
Platform Engineering and MLOps	Automation requirements, metadata collection mechanisms, and pipeline integration points.	Implementation of automated AIBOM generation, CI/CD integration, and ongoing maintenance.
AI and ML Teams	Recognition that lineage capture is an integral part of AI development.	Training data lineage, base model provenance, fine-tuning history, evaluation results, and model development metadata.

Baseline discovery closes the phase: inventory every AI system in production or active development — internally trained models, fine-tuned variants, externally procured model APIs, and AI features inside procured SaaS; identify the metadata that already exists in model cards, training run logs, dataset registries, and dependency manifests; and assess whether an existing SBOM programme can carry AIBOM work rather than a new one being stood up beside it.

Phase 2 — Pilot

Pick two or three systems that sit at different points on the disclosure spectrum, because the difficulty of authoring an AIBOM is determined almost entirely by how much the producer is permitted to know. An internally trained and deployed model gives full metadata access and is the tractable starting point. An internally fine-tuned variant of an external base model forces the pilot to record two provenance chains, one of which is someone else's, and is the most common enterprise case. A hosted model accessed by API tests whether the organization can rely on supplier disclosure and reveals exactly where that disclosure stops. An AI feature inside a procured SaaS application establishes the floor of achievable coverage and, in doing so, produces the most useful procurement input of the whole pilot.

Capture is where the pilot is most often mishandled, because teams collect fields without deciding what the document claims. Structure the work around the four parts described earlier:

The header. Declare the scope before collecting anything: which system, which version, which deployment, which interfaces are in and out - then record a completeness claim that is honestly derived from what was actually reachable. A partial claim on a black-box SaaS feature is a correct artifact. A complete claim on the same system is a false one.

The components. Model identity, version, architecture, and base model reference; datasets with version, source, collection method, licence, and known limitations; the tools, orchestrators, and connectors the system calls; prompts and guardrail policies; the runtime and its dependencies, cross-referenced to existing SBOM records rather than re-inventoried.

The flows. What data moves between those components, and across which boundaries — the part no field checklist produces, and the part incident responders use first.

The evidence. Hashes for weight files and dataset snapshots, signatures, model cards, evaluation results, and the accountable authorship, review, and approval record that makes the document reviewable rather than merely present.

On automation, the pilot's question is not which format to use — that is settled elsewhere in this document — but where each field comes from. Extract from build and training pipelines wherever the information already exists there, connect to whatever experiment tracking the organization runs, and be explicit about which fields must be hand-authored and who owns them. Resist committing the programme to a platform on pilot evidence alone.

Authoring belongs to the team closest to each system, with review and approval by a designated governance or security function. Centralizing authoring in a team without direct system knowledge is the failure mode this ordering exists to prevent.

Phase 3 — Governance Integration

Integration means the AIBOM becomes a required input to decisions the organization already makes. Creating a parallel AIBOM process is the reason most programmes stall after the pilot.

Existing Process	What the AIBOM Supplies	Where it Binds
Model Risk Management	A comprehensive inventory of AI components, dependencies, and supporting metadata required for risk assessment.	Mandatory input to every model risk assessment and review.
AI Development Lifecycle	Declared system scope, provenance, lineage, and evaluation evidence captured at each lifecycle stage.	Required gate for pre-training approval, pre-deployment review, and post-incident assessment.
Procurement and Third-Party Risk	Supplier-declared provenance information, software and model dependencies, and explicit identification of missing or incomplete disclosures.	Minimum supplier disclosure requirement during evaluation of high-risk vendors and AI systems.
Security Design Review	AI components, data flows, dependencies, and trust boundaries required for threat modeling and architecture review.	Standard input for AI security architecture and design reviews.
Regulatory Readiness and Audit	Version-controlled, traceable evidence mapped to applicable regulatory, governance, and compliance obligations.	Retained governance artifact provided during regulatory examinations, audits, and compliance assessments.

A formal policy makes this enforceable. It should define the minimum usable AIBOM per risk tier, authoring and review responsibilities, retention periods, update triggers, and the escalation path when a supplier will not disclose. The artifact itself must be treated as a controlled document: version-controlled, signed, with reviewer identity, review date, and review scope captured, held to the same retention and access controls as model validation records.

Phase 4 — Continuous Operation

An AIBOM describes a system at a moment. The systems change continuously, and an artifact that no longer matches the deployment is worse than none, because it invites confidence it cannot support.

Regeneration should be triggered by retraining or dataset refresh; adapter replacement or fine-tuning parameter change; dataset version changes, including incremental ones; upstream model or alias changes made by a supplier; dependency upgrades in the inference runtime or preprocessing path; guardrail and policy pack changes; disclosed vulnerabilities affecting a tracked component; and any change to intended use, deployment scope, or access boundaries. Supplier-side changes deserve particular attention, because they are the trigger class that fires without any local release.

Operationally this means AIBOM generation embedded in training pipelines so artifacts are produced on successful build; validation checks in deployment gates so incomplete or stale records block promotion; component records fed into the same vulnerability correlation workflows the SBOM programme already uses, rather than a second silo; and integrity verification so post-generation modification is detectable. Ownership divides along existing lines: ML engineering and platform teams update, security operations triages alerts, the governance function reports, and third-party risk tracks supplier disclosure.

Programme health is worth measuring on a small number of indicators: coverage, the share of production AI systems with a current artifact; freshness, the lag between a trigger event and the corresponding update; field population against the minimum usable AIBOM, segmented by risk tier; vulnerability response, time from disclosure to assessed impact; and supplier compliance, the share of contracted AI suppliers meeting the minimum disclosure standard. These measure the programme, not the systems it documents. A fully populated record of a badly governed system is still just a fully populated record, and no metric here is a rating of the AI system or of the AIBOM. Surface them where AI risk is already reported so the same evidence serves regulatory readiness, internal audit, and executive reporting without being prepared three times.

ACKNOWLEDGEMENTS

Contributors

Yuvaraj Govindarajulu (Workstream Lead, Project Co-Lead)	Protectt.ai, India
Aruneesh Salhotra (Project Lead)	Founder/CEO, Stealth Startup, USA
Martin Espinoza	Fabletics
Van Lindberg	Model Monster, USA
Tanmay Ambegaokar	Onix, USA
Bakul Singhal	Steve Madden, USA
L.Frances. Domingo	
Anmol Kumar	Stealth Startup, USA
Binal Patel	SISA, India
Ranjan Goel	NuGuard AI, USA
Sapna Paul	Dayforce, USA
Behnaz Karimi	OWASP, Germany
John Cavanaugh	ProCap360, USA
Yash Omer	Bajaj Financial Services, India
Pratibha Aphale	Goldman Sachs, USA
Deba Mishra	EY, USA
Karen Bennet	
Chetan Velicheti	

Project Sponsors

We appreciate our Project Sponsors whose funding contributions help advance the objectives of the OWASP AIBOM Project and cover the operational costs in addition to the resources provided by the OWASP Foundation.

Sponsors receive recognition for their contributions across our materials and web properties. All materials the project produces are community-developed and community-driven, and are released under open-source and Creative Commons licenses. To learn more about becoming a sponsor and support the project, please visit the Sponsorship section of our website at https://owaspaibom.org/sponsorship/.

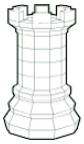
Project Gold Sponsors

ThreatReaper

Project Silver Sponsors

MANIFEST

Codacy



Prediction Guard

OWASP AIBOM Foundations Guide

SEPTEMBER 2026

Version 1.0 (EN)