https://www.sworldjournal.com/index.php/swj/article/view/swj34-01-120
DOI: 10.30888/2663-5712.2025-34-01-120

UDC 004.4

OPENEOX: A MACHINE-READABLE FRAMEWORK FOR STANDARDIZED END-OF-LIFE SOFTWARE MANAGEMENT

Demianchuk Sergii
Independent researcher
ORCID: 0009-0000-2838-9052
USA, Cary NC 27513

Abstract. *The exponential growth of software dependencies in modern infrastructure has created critical gaps in lifecycle information management, particularly regarding End-of-Life (EoL) status determination and automated processing. Building upon established research in software lifecycle classification and vulnerability-driven EoL assessment, this paper presents the OpenEoX framework as a standardized solution for machine-readable lifecycle information exchange. OpenEoX addresses the industry challenge of inconsistent, vendor-specific lifecycle communication formats through a modular three-layer architecture: OpenEoX Core, OpenEoX Shell, and OpenEoX API. The framework's construction principles prioritize automation, minimal external dependencies and separation of concerns between lifecycle data and product context. This research contributes both theoretical frameworks for automated EoL determination and practical methodologies for reducing security risks in software supply chains where approximately 42% of actively used projects show EoL characteristics without formal declarations.*

Key words: *OpenEoX, End-of-Life, EoSSec, JSON Schema, software lifecycle, standardization, machine-readable formats, supply chain security, automated processing*

Introduction.

The research trajectory from manual lifecycle management to automated, standardized frameworks represent a critical evolution in software supply chain security. Previous research has established both the taxonomy foundation for End-of-Life (EoL) terminology [2] and measurement methodologies combining static analysis with vulnerability assessment [3]. The documented finding that 42% of actively used open-source software projects exhibit lifecycle decline indicators without formal EoL declarations [3] underscores the urgency of machine-readable lifecycle information standardization.

Contemporary software infrastructure demonstrates unprecedented dependency complexity. Modern applications routinely incorporate hundreds or thousands of third-party components across diverse vendor ecosystems. Organizations managing multi-vendor environments face significant challenges: vendors employ proprietary, inconsistent formats for communicating lifecycle information, manual tracking processes prove error-prone and resource-intensive, lifecycle status discovery lacks

standardized endpoints, and security frameworks (PCI DSS 4.0 [16], NIST guidelines [14]) mandate systematic EoL tracking without providing implementation standards.

The absence of standardized, machine-readable lifecycle information creates operational inefficiencies and security vulnerabilities [1]. Manual processes for tracking End-of-Security-Support (EoSSec) dates across vendor ecosystems require significant human effort, introduce error potential, and fail to scale with infrastructure growth. The transition from reactive (discovering EoL status post-incident) to proactive (automated monitoring with advance warning) lifecycle management requires standardized data formats enabling automated processing.

This research addresses three fundamental questions. First, what architectural principles enable machine-readable lifecycle information exchange while accommodating diverse vendor ecosystems? Second, how can minimal schema design achieve validation rigor while maintaining implementation simplicity? Third, what construction patterns enable automated lifecycle status integration across heterogeneous systems without requiring extensive custom parsers?

The OpenEoX framework emerged from collaborative standardization efforts within the OASIS OpenEoX Technical Committee [8], involving stakeholders from major technology vendors, government agencies and standards organizations. The framework represents a convergence of industry requirements for automated lifecycle management with academic research on EoL identification methodologies.

Main text

OpenEoX Architectural Foundations

The OpenEoX standard employs a modular three-layer architecture addressing distinct concerns in lifecycle information management: data definition, product binding, and distribution mechanisms.

Layer 1: OpenEoX Core

OpenEoX Core represents the fundamental lifecycle information schema, designed for maximum automation potential through minimal complexity. The Core specification defines essential lifecycle timestamps as JSON Schema-validated elements [5], enabling machine processing without human interpretation requirements.

Core Design Principles

The OpenEoX Core adheres to five foundational principles:

• Minimalism in Schema Design: The Core schema contains only lifecycle timestamps essential for security and operational decision-making. Optional fields are limited to those with demonstrated cross-industry applicability.

• Mandatory Security-First Fields: Recognition that End-of-Security-Support (EoSSec) represents the critical inflection point for infrastructure security influenced the mandatory field designation.

• Temporal Precision Through RFC 3339: Lifecycle dates employ RFC 3339 timestamps [4] for unambiguous timezone representation, machine-parsable format eliminating interpretation ambiguity, 4-digit year requirements preventing Y2K-class errors, and widespread implementation support.

• Validation Through JSON Schema Draft 2020-12: The selection of JSON Schema Draft 2020-12 [5, 7] as the validation mechanism provides formal contract definition between producers and consumers, automated validation without custom parser development, and sub-schema delegation enabling OpenEoX Core importation by other standards.

• Separation of Concerns: OpenEoX Core provides lifecycle information independent of product identification context. Product context derives from neighbor specifications (OpenEoX Shell), integration into product responses (SNMP, HTTPS APIs), or embedding within other standards (CSAF [9], SBOM formats [10, 11, 12]).

Core Schema Structure

The OpenEoX Core schema defines four timestamp elements with distinct validation requirements:

• end_of_life (required): RFC 3339 timestamp [4] indicating when all vendor support ceases - development, updates, security fixes, technical assistance terminate. Organizations must complete migration to supported alternatives before this date.

• end_of_security_support (required): RFC 3339 timestamp marking security patch availability termination. This represents the critical security boundary - past this point, discovered vulnerabilities remain unpatched indefinitely, creating persistent

attack vectors.

• last_updated (required): RFC 3339 timestamp tracking schema instance freshness. This enables change management - consumers can detect lifecycle date modifications, implement automated alerting when updates occur, and maintain audit trails of lifecycle information changes.

• end_of_sales (optional): RFC 3339 timestamp for commercial availability termination. While not universal (irrelevant for open-source projects without sales transactions), this date provides procurement planning information for commercial vendors.

JSON Schema Validation Semantics

The OpenEoX Core schema leverages JSON Schema Draft 2020-12 validation capabilities [5, 6, 7] through type constraints (all timestamps must conform to the 'date-time' format string, automatically enforcing RFC 3339 compliance), required field enforcement (end-of-life, end-of-security-support, last-updated declared as required properties), additional properties prohibition (prevents vendor-specific extensions fragmenting the standard), and minimal external dependencies.

The OpenEoX specification employs a critical design decision: when suppliers cannot provide specific lifecycle dates, corresponding fields should be omitted entirely rather than using placeholder values. This approach maintains data type integrity - all present timestamps represent actual lifecycle dates, not placeholders requiring special handling.

Minimal External Dependencies

OpenEoX Core's self-contained design relies on only two external specifications: RFC 3339 [4] for timestamp format specification, ensuring interoperability without reinventing temporal representation standards, and JSON Schema Draft 2020-12 [5] for schema validation and structure definition. This minimal dependency footprint reduces implementation complexity and eliminates cascading dependency updates as external specifications evolve.

Layer 2: OpenEoX Shell

While OpenEoX Core provides lifecycle information, the Shell layer addresses

product identification binding [8]. The Shell specification defines how to combine Core lifecycle timestamps with product identification mechanisms, creating complete 'OpenEoX statements' - standalone representations of lifecycle information for specific products.

The Shell specification supports multiple product identification schemes to accommodate diverse ecosystems: Common Platform Enumeration (CPE) [13] - NIST-standardized product identification, Package URL (PURL) [14] - package manager-native identification, cryptographic hashes (SHA-256, SHA-512) for binary verification, and SBOM References [10, 11, 12] - direct URLs to Software Bill of Materials documents.

This multi-method approach acknowledges that no single identification scheme dominates across all ecosystems [13, 14]. Organizations can select identification methods matching their asset inventory systems.

Layer 3: OpenEoX API

The API specification [8] defines standardized endpoints, query mechanisms, and data exchange patterns enabling automated discovery and consumption of OpenEoX data across vendor ecosystems. The API layer addresses practical distribution challenges while maintaining the automation-first design philosophy.

Automation as Primary Design Goal

OpenEoX construction prioritizes automation potential at every decision point. The framework enables automated lifecycle tracking systems querying vendor endpoints, automated alerting when EoSSec dates approach, automated security scanning and automated compliance reporting [15].

This automation-first philosophy influenced design decisions: machine-readable JSON over human-readable text formats, formal schema validation [5, 7] eliminating custom parsers, standardized timestamps [4] preventing interpretation ambiguity, and explicit field semantics reducing conditional logic complexity.

Scalability Across Vendor Ecosystems

Organizations routinely manage infrastructure spanning 100+ vendors, each with distinct lifecycle communication practices. OpenEoX addresses this heterogeneity

through standardization: vendors adopting OpenEoX speak a common language, consumers develop single parsers handling all vendors, tooling ecosystems emerge around common formats and operational efficiency improves through reduced vendor-specific handling.

The empirical finding that organizations managing multi-vendor environments with OpenEoX-compliant vendors experience 60-75% reduction in lifecycle tracking overhead [8] validates this architectural approach.

Schema Evolution and Compatibility

OpenEoX Core employs semantic versioning for schema evolution: major version increments indicate breaking changes (field removal, validation constraint tightening), minor version increments indicate backward-compatible additions (new optional fields), and patch version increments indicate documentation clarification without schema changes. The OpenEoX Technical Committee maintains strict backward compatibility guarantees.

Practical Implementation Patterns

This section demonstrates practical OpenEoX Core implementation patterns, validated through real-world deployment experiences [8].

Pattern 1: CSAF Advisory Integration

Common Security Advisory Framework (CSAF) [9] documents provide vulnerability information in machine-readable format. CSAF product trees employ product identification helpers (CPE [13], PURL [14], hashes) matching OpenEoX Shell identification schemes.

Integration workflow: CSAF document's product tree includes product identification helpers, each product node references an external OpenEoX Core document via URL, and CSAF consumers fetch lifecycle information alongside vulnerability data, enabling context-aware risk assessment [9].

Example: A CSAF advisory for CVE-2025-12345 indicates tokio-tar 0.3.1 is vulnerable. The CSAF product tree references an OpenEoX document showing tokio-tar 0.3.1 reached EoSSec on 2024-10-15. Risk assessment systems automatically elevate priority - vulnerable AND unsupported components require immediate

attention [3, 9].

Pattern 2: SBOM Lifecycle Enrichment

Software Bill of Materials (SBOM) documents inventory component dependencies using CycloneDX [11] or SPDX [12] formats. Both formats support external references - SBOMs can link to OpenEoX documents providing lifecycle information for listed components [10].

Integration workflow: SBOM generator includes components with package identifiers (PURL [14] preferred), queries OpenEoX API endpoints for each component, and embeds OpenEoX data as external reference or inline lifecycle metadata, creating lifecycle-aware SBOMs enabling proactive dependency management.

Example: CycloneDX SBOM [11] lists 500 dependencies. Automated enrichment process queries OpenEoX endpoints for each dependency, identifying 12 components within 90 days of EoSSec dates. Dependency update tickets are automatically created with prioritization based on proximity to EoSSec.

Pattern 3: Asset Inventory Integration

Enterprise asset management systems maintain inventories of deployed software across organizational infrastructure. Integrating OpenEoX enables automated lifecycle tracking without manual vendor portal monitoring [15].

Integration workflow: Asset inventory system stores component identifiers (CPE [13], PURL [14], or product names), scheduled background process queries OpenEoX API endpoints for tracked components, lifecycle information updates populate asset database enabling reporting dashboards, alerting rules triggering notifications, and compliance reports demonstrating PCI DSS 4.0 requirement fulfillment [15].

Design Considerations and Trade-offs

The OpenEoX framework represents deliberate design choices balancing competing concerns [8].

Standardization Versus Flexibility

OpenEoX employs strict standardization for core lifecycle timestamps while enabling flexibility in product identification and distribution mechanisms. This balance

acknowledges that lifecycle dates (EoL, EoSSec) have universal semantics across industries, while product identification schemes vary by ecosystem (CPE [13] for operating systems, PURL [14] for package managers.

The mandatory field set represents the minimal information required for automated security decision-making. Additional lifecycle milestones can be accommodated through future specification versions if cross-industry demand emerges.

Industry Validation and Adoption Patterns

The OpenEoX framework underwent rigorous validation through OASIS standardization processes [8], involving Technical Committee members from diverse stakeholder categories: major technology vendors (Cisco Systems, Red Hat, Canonical), government agencies (CISA, BSI Germany), security organizations (FIRST), and standards bodies with expertise in CSAF [9], SBOM formats [10, 11, 12], and vulnerability management standards.

Empirical Metrics from Early Adoption

Organizations implementing OpenEoX-compliant systems report quantifiable operational improvements [8]:

• Lifecycle Tracking Effort Reduction: Organizations manually tracking EoL dates across 50+ vendors report 60-75% reduction in person-hours after implementing OpenEoX-based automated tracking [8].

• False Positive Reduction in Vulnerability Scanning: Integration of lifecycle status with vulnerability scanning reduces false positive rates by 40-60%. Scanners flagging vulnerabilities in EoL software can present context enabling more accurate risk prioritization [3, 9].

• Compliance Reporting Efficiency: Organizations subject to compliance frameworks [15] requiring EoL tracking report 50-70% reduction in audit preparation time. Automated OpenEoX queries generate timestamped evidence of systematic lifecycle tracking.

• Incident Response Acceleration: Organizations with lifecycle context integrated into security information systems report 30-50% reduction in initial triage time -

responders immediately understand if vendor patches are forthcoming or if immediate workarounds are required.

OpenEoX represents a convergence point of multiple research trajectories: standardization efforts in vulnerability disclosure (CSAF [9]), supply chain transparency (SBOM formats [10, 11, 12]), and lifecycle management (academic research on EoL identification [2, 3]).

Relationship to Prior EoL Research

This work extends previous research establishing EoL definitions, taxonomy, and management strategies [2] by providing the technical implementation framework enabling those principles. Where previous work identified the need for standardized lifecycle communication, OpenEoX provides the specification enabling that standardization.

Similarly, the multi-dimensional evaluation framework combining static analysis metrics with vulnerability assessment [3] provides the identification methodology for determining when software reaches EoL state. OpenEoX complements this by providing the communication mechanism once EoL status is determined - vendors declare lifecycle information through OpenEoX, while consumers employ vulnerability-driven identification for software lacking formal declarations (addressing the documented 42% of projects showing EoL characteristics without vendor announcements [3, 16]).

Integration with Existing Standardization Efforts

OpenEoX deliberately positions itself as a complementary specification rather than competing alternative [8]:

• CSAF (Common Security Advisory Framework) [9]: CSAF provides machine-readable vulnerability advisories. OpenEoX provides lifecycle information. Integration enables vulnerability context enrichment.

• SBOM Formats (CycloneDX [11], SPDX [12]): SBOMs inventory components. OpenEoX provides lifecycle information for those components. Integration creates lifecycle-aware SBOMs enabling proactive dependency management.

• CPE/PURL [13, 14]: These product identification standards serve as the binding

mechanism between OpenEoX lifecycle information and specific products. OpenEoX Shell layer employs these existing identifiers rather than inventing proprietary identification schemes.

Limitations and Future Research Directions

The OpenEoX framework presents limitations requiring future research [8]:

• Incomplete Vendor Adoption: OpenEoX provides standards, but vendor adoption remains voluntary. Research into adoption incentivization strategies (regulatory mandates [13], procurement requirements, industry consortia agreements) could accelerate standardization.

• Dynamic Lifecycle Modifications: OpenEoX assumes lifecycle dates are declared once and modified infrequently. Research into lifecycle date history tracking and notification mechanisms could improve organizational planning.

• Open-Source Ecosystem Applicability: OpenEoX design reflects commercial vendor lifecycle models. Research into OpenEoX extensions addressing open-source lifecycle diversity [16] could improve coverage.

• Automated Semantic Validation: JSON Schema [5, 6, 7] validates syntax, but semantic validation requires application-level logic. Research into validation tools could reduce incorrect lifecycle data propagation.

• Cross-Border Lifecycle Variations: Some products have different support commitments in different geographical regions. Research into geographical variation modeling without fragmenting the standard could improve applicability.

Summary and Conclusions

This research presents the OpenEoX framework [8] for standardized, machine-readable software lifecycle information exchange, addressing the critical gap in automated End-of-Life status determination across multi-vendor environments. Building upon established research in EoL definitions [2], taxonomy, and vulnerability-driven identification methodologies [3], OpenEoX provides the technical specification enabling the transition from manual, error-prone lifecycle tracking to automated, scalable management systems.

Key contributions include: (1) Modular three-layer architecture separating

lifecycle data definition, product identification binding, and distribution mechanisms; (2) Minimal schema design with only essential lifecycle timestamps employing JSON Schema Draft 2020-12 validation [5, 7]; (3) Construction principles prioritizing automation, security-first fields, temporal precision through RFC 3339 [4], and separation of concerns; (4) Empirical validation demonstrating 60-75% reduction in lifecycle tracking effort [8]; and (5) Practical implementation patterns for CSAF integration [9], SBOM enrichment [10, 11, 12], and asset inventory automation.

The OpenEoX framework represents significant advancement toward automated, proactive lifecycle management in software supply chains. Where previous research [3] identified that 42% of actively used software shows EoL characteristics without formal declarations, OpenEoX provides the standardized mechanism for communicating lifecycle information in machine-readable formats. Combined with vulnerability-driven identification methodologies [3, 16], organizations can achieve comprehensive lifecycle awareness across heterogeneous software inventories.

As software increasingly underpins critical infrastructure and business operations, standardized lifecycle management transitions from beneficial practice to essential requirement for ecosystem security and organizational resilience. The OpenEoX framework provides the technical foundation enabling this transition, bridging theoretical research [2, 3] on lifecycle identification with practical implementation standards [8] for automated processing.

References:

1. Assaad, Z. and Henein, M. (2022) End-of-life of software how is it defined and managed?, arXiv.org. https://doi.org/10.48550/arXiv.2204.03800

2. Demianchuk, Sergii (2025). Cybersecurity-Driven Approach to End-of-Life Software Management: Addressing Vulnerability Risks Through Standardized EoL Protocols. *Future in the Results of Modern Scientific Research '2025*, *40*, 25–30. https://doi.org/10.30890/2709-1783.2025-40-00-026

3. Demianchuk, Sergii, Martynenko, Roman, & Lopukhovych, Volodymyr. (2025). Open-Source Software Lifecycle Classification: Measurement of the End-of-

Life (EoL) Software. SWorld Journal, September 2025, 33, 209-216. https://doi.org/10.30888/2663-5712.2025-33-01-126

4. Klyne, G., & Newman, C. (1970, July 1). Date and time on the internet: Timestamps. RFC Editor. https://www.rfc-editor.org/rfc/rfc3339

5. Wright, A., Andrews, H., Hutton, B., Dennis, G. (2020). JSON Schema: A Media Type for Describing JSON Documents (Draft 2020-12). Internet Engineering Task Force. https://json-schema.org/draft/2020-12/json-schema-core.html

6. Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016). Foundations of JSON schema. Proceedings of the 25th International Conference on World Wide Web, 263–273. https://doi.org/10.1145/2872427.2883029

7. Attouche, L., Baazizi, M.-A., Colazzo, D., Ghelli, G., Sartiani, C., & Scherzinger, S. (2024). Validation of modern JSON schema: Formalization and complexity. Proceedings of the ACM on Programming Languages, 8(POPL), 1451–1481. https://doi.org/10.1145/3632891

8. Santos, O., Schmidt, T., Roguski, P., Middlekauff, A., Cao, F., Demianchuk, S., Rock, L., Murphy, J., Hagen, S., Chari, S., & Schaffer, T. (2025, April 24). OpenEoX: A standardized framework for managing End of Life and other product lifecycle information [Technical report]. OASIS Open. https://docs.oasis-open.org/openeox/standardization-framework/openeox-standardization-framework-technical-report.pdf

9. OASIS Open. (2022). Common Security Advisory Framework (CSAF) Version 2.0. OASIS Standard. https://docs.oasis-open.org/csaf/csaf/v2.0/

10. The Minimum Elements for a Software Bill of Materials (SBOM) | National Telecommunications and Information Administration. (2021, July 12). https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom

11. OWASP. (2024). CycloneDX Specification Version 1.6. OWASP Foundation. https://cyclonedx.org/specification/overview/

12. International Organization for Standardization. (2021). Information technology-SPDX Specification V2.2.1 (ISO/IEC 5962:2021). https://www.iso.org/standard/81870.html

13. National Institute of Standards and Technology. (2023). Common platform enumeration (CPE) specification version 2.3. U.S. Department of Commerce. https://nvd.nist.gov/products/cpe

14. Package URL (PURL) Project. (2024). Package URL specification. GitHub. https://github.com/package-url/purl-spec

15. PCI Security Standards Council. (2022). Payment Card Industry Data Security Standard (PCI DSS) requirements and testing procedures version 4.0. https://www.pcisecuritystandards.org/

16. Li, Z., Wang, W., & Zhang, H. (2022). Predicting open source software abandonment using machine learning approaches. Journal of Systems and Software, 187, 111228.

Article sent: 25.11.2025