Supply-Chain-Aligned Software Auditing and Usage Justification via Distributed Ledgers

Oliver Stengele, Jan Droll, Hannes Hartenstein
Institute of Information Security and Dependability (KASTEL)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{oliver.stengele,jan.droll,hannes.hartenstein}@kit.edu

***Abstract*—We present a concept to document audit results of software artifacts, like compliance with standards or specifications, and usage justifications for dependencies as human-readable and machine-evaluable policies in software bills of materials (SBOMs). To ensure that auditing results are readily available and reusable, we anchor policy-augmented SBOMs on a distributed ledger. We perform an empiric evaluation of the latency and costs involved in writing and reading anchoring data to and from Ethereum. Reading latency is mainly determined by the degree of optimization of the Ethereum node being queried, but remains mostly below 40 seconds.**

***Index Terms*—software supply chain security, usage justification, distributed ledgers**

I. Introduction

In recent years, attackers have seized the opportunity to infiltrate lucrative targets at the end of software supply chains (SSCs) by compromising easier targets upstream to cause damages in the millions of dollars [1]. As a result of both external legislative requirements, like the EU Cyber Resilience Act [2] and Executive Order 14028 [3], and an internal need to better manage risks in the face of growing complexity of SSCs, the concept of software bills of materials (SBOMs) was developed to make supply chains more comprehensible. As software artifacts are integrated along a supply chain, their SBOMs are nested and passed along such that the SBOMs of end products are a comprehensive inventory based on their multi-step creation. Put simply, SBOMs are a cooperative best effort approach to make software supply chains more comprehensible for the purpose of risk management and improved security in the long run. However, the distribution and retrieval of SBOMs is still an area of active experimentation and research [4].

Distributed ledgers such as Ethereum were conceived and designed to be reliable, decentralized data propagation systems to support the consensus-based extension of their respective blockchains. Using distributed ledgers as a broadcasting mechanism for SBOMs has recently been explored by Xia *et al.* [5]. Through the use of distributed ledgers, it can be ensured that if an SBOM for a particular software artifact was published, any interested party can observe and retrieve it.

The reliable visibility of SBOMs afforded through the use of distributed ledgers presents an opportunity to augment SSCs in another way: transparently documenting both audit results of software artifacts and the decisions of software producers to rely on those results when selecting software dependencies. With the term "usage justification", we describe the rationale behind the selection and use of software dependencies regarding the requirements that each dependency fulfills and which evidence was used to substantiate each decision. With the concept presented in this work, auditing results of increasingly complex software components and their use in justifying decisions by software producers can be recorded in SBOMs and built upon to better understand the trust assumptions inherent in SSCs that are currently rather implicit.

Our contributions in this paper are twofold:

1) A concept for augmenting SBOMs with evidence and usage justification policies using the SecPAL language [6].
2) An empirical evaluation of Ethereum as an underlying broadcast and transparency mechanism regarding the costs and latency of writing SBOM-anchoring references on-chain and the latency of reading those references.

We provide an overview of SBOMs at the example of CycloneDX and policy specification languages at the example of SecPAL in Section II. In Section III, we describe the concept for augmenting SBOMs with evidence and usage justification policies using an Ethereum consensus client as an illustrative example. We report the results of our empirical evaluation of Ethereum in Section IV. We discuss our findings, related, and future work, and conclude the paper in Section V.

II. Background

We give a brief overview on Software Bills of Materials (SBOMs) at the example of CycloneDX and the policy specification language SecPAL. For the distribution of SBOMs, we employ the InterPlanetary File System (IPFS) [7], which provides a means to reference, distribute, and obtain arbitrarily large files via constant-sized strings called content identifiers (CIDs). Since the link between a CID and the data it references is immutable and integrity-protecting, anchoring SBOMs on a DLT reduces to publishing its CID mapped to the hash of its software artifact in an easily observable manner.

A. Software Bills of Materials: CycloneDX

CycloneDX [8] is a standard framework for bills of materials [9] established and maintained by the Open Worldwide Application Security Project (OWASP) Foundation. We limit our scope to *software* bills of materials in CycloneDX. At the

most fundamental level, CycloneDX prescribes a nested object model to record and share relevant aspects of software artifacts. The main benefit is to enable the compatibility of tools to generate, ingest, display, analyze, and manage SBOMs. Each CycloneDX SBOM is identified via an issuer-defined unique serial number and version, is constructed from a selection of object types, and can be cryptographically signed. The most relevant object types are *Components* and *Dependencies*. *Declarations* were added with CycloneDX 1.6 to document the conformance to standards in the creation of a target artifact or as part of its properties. While Declarations as part of SBOMs are machine-readable according to the OWASP Foundation, examples given in the corresponding publication [10] suggest that they are not machine-evaluable. The aim of our concept is to enable machine-evaluability of statements about software artifacts and usage justifications based thereon through the use of a policy specification language like SecPAL. In a sense, declarations in the CycloneDX object model provide a location in SBOMs for our concept to slot in neatly.

B. Policy Specification Language: SecPAL

SecPAL is a declarative authorization language for multi-party settings developed by Becker, Fournet, and Gordon [6]. SecPAL enables the specification of fine-grained access control across trust domains using a human-readable English-like syntax based on *verb phrases*. We use `typewriter` to designate actors, *italics* to designate roles, **bold** for verbs and key words, and sans serif for identifiable objects in SecPAL policies. For example, "`Alice` **says** `Bob` **can read** FileA" is an access policy stated by the *principal* `Alice` giving `Bob` the right to read FileA. SecPAL policies are signed by their principal to ensure authenticity and tamper-proofness. Similarly, principals are able to delegate the right to make certain statements using the verb "**can say$_x$**" where x can either be 0, meaning the recipient of the right cannot delegate it further, or ∞ to allow re-delegation. For the remainder of this work, we only use "**can say$_0$**". If FileA is stored on `FileServer`, which also enforces access control, a policy like "`FileServer` **says** `Alice` **can say$_0$** y **can read** FileA" specifies that Alice is able to grant others (y is a placeholder) access to FileA. When `Bob` wants to access FileA via `FileServer`, he would attach `Alice`'s earlier policy that grants him the right and `FileServer`, according to the above policy, would honor `Alice`'s policy and allow access.

SecPAL also supports the use of roles to make delegations more flexible. For example, with policies like "`Alice` **says** *Researcher* **can read** FileA" and "`FileServer` **says** `EmployeeDB` **can say$_0$** z **is a** *Researcher*", `Alice` would enable a group of her colleagues to access FileA while tasking `EmployeeDB` with adding new colleagues to the role.

We propose to employ SecPAL as part of SBOMs to *justify* past access control decisions. It is this retrospective justification perspective that we take advantage of.

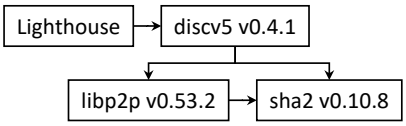

Fig. 1. A simplified excerpt of the dependency graph of the Ethereum consensus client Lighthouse.

`BSI` **says** `SecLab` **is a** *Hash Auditor* **if**
CurrentTime $<$ 2027-12-31

Listing 1: Accreditation policy for a German security lab.

`SecLab` **says** sha2 v0.10.8 **complies with** NIST FIPS 180-4

Listing 2: Evidence policy created after an audit.

III. Augmenting SBOMs with Usage Justification Policies

We present a concept of augmenting SBOMs with evidence and usage justification policies using SecPAL. A *usage justification policy* (UJP) for a given software artifact states requirements that its subcomponents have to fulfill and which entities are trusted to assert the necessary properties. UJPs are created by the same entity that produces the corresponding target artifact. Meanwhile, *evidence policies* are created to document the results of software audits or reviews by the entities performing these tasks. UJPs and evidence policies form a cohesive argument for each subcomponent regarding what requirements it needs to fulfill and who is trusted to state that those requirements are fulfilled.

We employ a semi-synthetic example based on the Ethereum consensus client Lighthouse, the relevant excerpt of its dependency graph is depicted in Fig. 1. Note that the three Rust crates are maintained by different parties: **discv5** implements the protocol by the same name and, like the Lighthouse client, is maintained by Sigma Prime. **libp2p** is an open source networking library for P2P systems. It is maintained by the same open source community that develops the corresponding specification. **sha2** is an implementation of the SHA-2 hash function family maintained by the group "RustCrypto", that also maintains many other Rust implementations of cryptographic primitives. While the software components and their dependencies are real, the following policies are fictional.

We assume that institutions or individuals seek and obtain accreditation in their respective areas of expertise from government ministries. In this example, the IT security firm `SecLab` has obtained a 3-year accreditation as an auditor for cryptographic hash functions from the German Federal Office for Information Security (`BSI`) depicted in Lst. 1. Recall that SecPAL policies are signed by their issuing principal, the `BSI` in this case. After being accredited, `SecLab` would store the above policy locally to include it as part of audit results.

Next, `SecLab` audits the sha2 crate and produces the evidence policy depicted in Lst. 2. `SecLab` then passes this evidence policy along to the maintainers of sha2 along with their accreditation policy (Lst. 1) for inclusion in the SBOM of the audited version.

```
libP2P Project says libp2p v0.53.2 complies with libP2P Spec
```

Listing 3: Self-asserted evidence of the libp2p project.

```
Sigma Prime says discv5 v0.4.1 complies with DiscV5 Protocol
```

Listing 4: Self-asserted evidence of the discv5 implementation.

```
Sigma Prime says
  BSI can say_0 w is a Hash Auditor
  Hash Auditor can say_0 x complies with NIST FIPS 180-4
  libP2P Project can say_0 y complies with libP2P Spec
  Sigma Prime can say_0 z complies with DiscV5 Protocol
  Lighthouse fulfills requirements if
    sha2 v0.10.8 complies with NIST FIPS 180-4
    libp2p v0.53.2 complies with libP2P Spec
    discv5 v0.4.1 complies with DiscV5 Protocol
```

Listing 5: Usage justification policy for Lighthouse.

Rather than an external audit, it is also conceivable that the maintainers of components audit their own implementation and attest to certain properties. For example, the libp2p project maintains both a language-agnostic specification and several implementations in different programming languages. The `libP2P Project` can attest that a particular implementation conforms to their specification with a policy as depicted in Lst. 3, which would then be included in the SBOM of libp2p. Likewise, the evidence policy in Lst. 4 would be included in the SBOM of discv5. Note the different objects between Lsts. 2 to 4 that discern which property is being attested.

Lastly, in preparation for a build of Lighthouse, Sigma Prime, as its maintainer, would compose a UJP as depicted in Lst. 5 to explicate both the requirements it places on the used subcomponents as well as which parties it relied on to fulfill those requirements. For each party, the principal of the UJP specifies which statement the party is trusted to make. That includes the designation of roles, as is the case with the `BSI` and the *Hash Auditor*, as well as specific statements, like compliance with certain standards, by specific parties or roles.

The second half of UJPs specifies requirements that must be fulfilled by the evidence supplied in SBOMs of subcomponents in conjunction with stated trust assumptions from the former half. Note the flexibility afforded through the use of roles: Sigma Prime did not have to specify an exact source for the evidence about the sha2 crate but merely that it trusts the BSI to assign the respective role diligently and that it trusts anyone with that role to provide the necessary evidence.

It is also important to highlight the ability to transparently reuse audit results at the example of sha2 from above. Just like the Lighthouse consensus client as a whole, so too would the SBOMs of all of its subcomponents carry their own UJP, particularly libp2p and discv5, that could both require sha2 to be audited for standards-compliance. If both maintainers trust the source of the audit results, then they can state so clearly in their respective UJP and include the same evidence policy (Lst. 5). Alternatively, if multiple institutions audit the same component, multiple evidence policies can be included in the corresponding SBOM and dependent maintainers can chose which source to rely on.

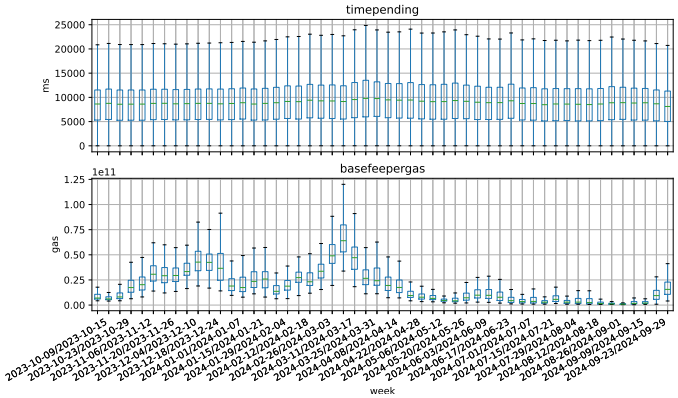

Fig. 2. Time pending (top) and basefeepergas (bottom) distribution of public Ethereum transactions. Basefee represents blockspace utilization.

IV. DLTs as Reliable Broadcast Mechanisms

Using a DLT as communication infrastructure for the described concept aims to provide focus and persistence. *Focus*: Actively participating and passively observing parties know how and where to obtain information pertaining to SBOMs. *Persistence*: SBOMs and their contents, once anchored, cannot be taken back by their authors or censored by malicious parties. Together, focus and persistence encourage the thorough creation of SBOMs, as they are easily scrutinized and hard to repudiate. We examine Ethereum as it is currently one of the most popular and economically secure blockchains. To characterize it as a broadcast channel, we determine its write and read latencies as well as the costs of writes.

Transactions in Ethereum are generally broadcast[1] to the P2P network of validators and spend some time in the *public mempool* until a block proposer includes them in a newly created block. We consider a transaction *confirmed* once it appears in a block. We call the time between a transaction first appearing in the P2P network and its inclusion in a block its *time pending*. To cope with the strict 12s block time, the aforementioned P2P network of validators ensures that messages, like pending transactions, are propagated quickly[2].

We evaluate the time pending of transactions on Ethereum using the Blocknative Mempool Archive [11]. Blocknative operated three globally distributed Ethereum nodes that logged changes to their mempools. In Fig. 2, we depict the distribution of time pending of public Ethereum transactions grouped by week from 2023-10 to 2024-09. As a proxy for blockspace demand, Fig. 2 shows the corresponding distribution of the basefee. Despite the varying influx of transactions, we observe a stable time pending for public transactions. Generally, more than 75% of public transactions are included in a block within 15s of their first detection on the Ethereum P2P network.

From the ways to store arbitrary data on Ethereum [12], we chose events to anchor SBOMs for three reasons: First,

[1]We exclude *private transactions* that circumvent the public mempool: https://www.blocknative.com/blog/ethereum-private-transactions-the-flippening

[2]98% of nodes receive messages within 4s of initial submission: https://ethresear.ch/t/gossipsub-message-propagation-latency/19982

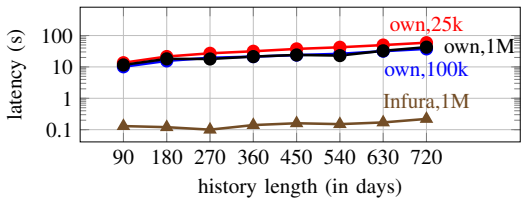

Fig. 3. Logarithmically scaled plot of read latency of various nodes and chunk sizes (see labels) on the Ethereum mainnet.

anchoring data need only be written but never read in any subsequent transaction. Second, events provide a convenient way to monitor and search blocks for specific events and their data. Third, at the time, events were the cheapest option.

To determine the read latency of Ethereum, we synchronized our own full node using geth v1.13.15. When queried for an event in a range of blocks, Ethereum nodes use a Bloom filter that is part of each block header to determine which set of transactions to examine closer for said events. For reasons explained below, we did not have to write any events to the Ethereum blockchain to measure read latency. We queried our node for nonexistent events in the past 3 to 36 months in three chunk sizes and recorded the response time. For comparison, we also queried the Infura API [13] for the same events and timed the results. Due to rate limiting, we could only perform the aforementioned measurements on Infura with a 1M block chunk size. Figure 3 depicts the results of these measurements.

The Bloom filter currently used in Ethereum is of a fixed size and its false positive rate (FPR) thus increases with the number of included events. Due to the number of events increasing over time, the FPR has grown to the point where Infura, as an example, implemented its own indexing for improved scalability and performance [14]. The result of this approach is visible in Fig. 3, especially compared to our own node using the conventional approach with a much lower request load than Infura. Due to the high FPR and the subsequent additional work required, whether or not the events of a a query were present in the requested time span has little to no effect on the time the query takes to complete.

The costs for broadcasting SBOM references can be determined empirically using a local development environment or an online tool like the RemixIDE. To publish SBOM references as Ethereum events, we deploy a smart contract that merely defines the event signature and offers a public function to emit said event. Each event consists of three values: the address that sent the transaction, i.e., the source of the SBOM; the hash of the artifact the SBOM describes; and an IPFS CID of the SBOM itself. The first two values are *indexed*, so that queries for specific events can be narrowed down to specific SBOM authors and artifacts. Deploying the smart contract costs 179915 gas, but it only needs to be deployed once. Using a 256-bit hash fingerprint to identify the software artifact and a 46 character string, the shortest and cheapest option, for the CID reference, a transaction to broadcast an SBOM reference costs 26210 gas.

V. Related Work and Conclusion

Ferraiuolo *et al.* [15] were among the first to highlight the applicability of authorization logic policies to SSC security, albeit by deliberately avoiding the use of SBOMs that have since become more prevalent and practical. Similarly, they also propose to establish a bespoke transparency log rather than building upon existing public DLTs and making use of their economic security and global distribution. Hassanshahi *et al.* [16] build upon the ideas of Ferraiuolo *et al.* to construct Macaron, "A flexible and extensible framework for checking and validating policies for software artifacts". In contrast to the concept proposed in this work, policies in Macaron mostly serve internal purposes of each supplier, unlike our concept that aims to extend the scope of information passed along SSCs. Xia et al. [5] recently described an approach to SBOM sharing with a hybrid blockchain/off-chain construction similar to the presented concept. Their approach is supplier-focused whereas we propose an artifact-focused extension to SSC practices. Since both the evaluation of software development practices and subsequent accreditation of producers as well as the evaluation of artifacts regarding compliance to standards or specifications are distinctly useful, future work could aim to combine both approaches into a comprehensive system.

Challenges in managing the growing complexity of SSCs have given rise to both directed attacks [17] and unintended mistakes [18]. While code repositories, pull requests, and increasingly robust dependency management have enabled and accelerated the cooperative creation of software, similar tools for a cooperative approach to check quality and compliance are only now starting to gain traction. Additionally, decentralized systems have matured to the point where approaches can be conceived that do not solely depend on centralized providers. In the presented concept, the work done to audit software components is broadly visible and transparently reusable.

The Ethereum community currently seeks to improve transaction throughput by decoupling global consensus from execution in a layered construction dubbed "rollups" [19]. The concept presented in this work can be framed as a *data-centric rollup*, in contrast to the predominantly asset-centric rollups in use today. Instead of every party anchoring UJP-augmented SBOMs individually, multiple parties can form a consortium to bundle IPFS CIDs of their SBOMs and anchor them in a single Ethereum transaction. Since both data- and asset-centric rollups share the need to reliably publish small fragments of data, the presented concept stands to benefit from architectural scalability improvements to the Ethereum ecosystem.

Designing a reliable way to revoke previously published evidence policies remains for future work. Similarly, a way to finely control access to SBOM contents to protect trade secrets in commercial contexts is relevant to both the concept presented in this work as well as SBOM adoption in general.

Acknowledgement. This work was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) as part of the project Software-Defined Car (SofDCar).

REFERENCES

[1] "2023 Software Supply Chain Attack Report," Cybersecurity Ventures, Tech. Rep., 2023. [Online]. Available: https://go.snyk.io/2023-supply-chain-attacks-report.html

[2] European Commission. (2022) Cyber resilience act. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act

[3] Executive Office of the President. (2021) Exec. order 14028. [Online]. Available: https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

[4] "SBOM Sharing Roles and Considerations," Cybersecurity & Infrastructure Security Agency, Tech. Rep., Mar. 2024. [Online]. Available: https://www.cisa.gov/resources-tools/resources/sbom-sharing-roles-and-considerations

[5] B. Xia, D. Zhang, Y. Liu, Q. Lu, Z. Xing, and L. Zhu, "Trust in Software Supply Chains: Blockchain-Enabled SBOM and the AIBOM Future," in *Proc. 2024 ACM/IEEE 4th Intl. Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM 2nd Intl. Workshop on Software Vulnerability*, ser. EnCyCriS/SVM '24. New York, NY, USA: ACM, 2024, p. 12–19. [Online]. Available: https://doi.org/10.1145/3643662.3643957

[6] M. Y. Becker, C. Fournet, and A. D. Gordon, "SecPAL: Design and semantics of a decentralized authorization language," *Journal of Computer Security*, vol. 18, no. 4, pp. 619–665, Jun. 2010.

[7] J. Benet, "IPFS - content addressed, versioned, P2P file system," 2014. [Online]. Available: http://arxiv.org/abs/1407.3561v1

[8] OWASP Foundation. Cyclonedx. [Online]. Available: https://cyclonedx.org

[9] ——. (2024) Authoritative guide to SBOM. [Online]. Available: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-SBOM-en.pdf

[10] ——. (2024) Authoritative guide to attestations. [Online]. Available: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-Attestations-en.pdf

[11] Blocknative. Mempool archive. [Online]. Available: https://www.blocknative.com

[12] P. Kostamis, A. Sendros, and P. Efraimidis, "Exploring Ethereum's Data Stores: A Cost and Performance Comparison," in *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2021, pp. 53–60. [Online]. Available: https://ieeexplore.ieee.org/document/9569804/

[13] Infura. Ethereum API. [Online]. Available: https://www.infura.io

[14] ——. (2019) Introducing faster Ethereum logs and events. [Online]. Available: https://www.infura.io/blog/post/faster-logs-and-events-e43e2fa13773

[15] A. Ferraiuolo, R. Behjati, T. Santoro, and B. Laurie, "Policy transparency: Authorization logic meets general transparency to prove software supply chain integrity," in *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. ACM, 2022.

[16] B. Hassanshahi, T. N. Mai, A. Michael, B. Selwyn-Smith, S. Bates, and P. Krishnan, "Macaron: A Logic-based Framework for Software Supply Chain Security Assurance," in *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. Copenhagen Denmark: ACM, Nov. 2023, pp. 29–37.

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

[18] G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? How vulnerable dependencies affect open-source projects," *Empirical Software Engineering*, vol. 26, no. 4, p. 59, 2021. [Online]. Available: https://link.springer.com/10.1007/s10664-021-09959-3

[19] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain Scaling Using Rollups: A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 93 039–93 054, 2022.