A Method and Platform for Security Advisory Dissemination Leveraging Web3 Technologies

Cibin, Nicola; Sommer, Jannik Lucas; Lund, Magnus Mølgard; Albano, Michele

Published in:
Proceedings of 6th IEEE International Conference on Blockchain

DOI (link to publication from Publisher):
10.1109/Blockchain60715.2023.00050

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

A Method and Platform for Security Advisory Dissemination Leveraging Web3 Technologies

Jannik Lucas Sommer, Magnus Mølgaard Lund, Nicola Cibin, Michele Albano
Department of Computer Science Aalborg University, 9220 Aalborg, Denmark
jannik.sommer@sommerhouse.net, themlund98@gmail.com, nicolac@cs.aau.dk, mialb@cs.aau.dk

***Abstract*—The frequency of software supply chain attacks has reached unprecedented levels, primarily due to the increasing reliance on huge numbers of software and hardware dependencies, and the inherent vulnerabilities they harbor. Currently, vendors providing these software and hardware components share security advisories to centralized databases or post them on proprietary websites, which security engineers have to search manually to find vulnerabilities relevant for their systems. Furthermore, the security advisories often do not follow a standard machine-readable format, which results in the engineers having to manually analyze the documents. In this paper, *SENTINEL*, a novel solution for automating dissemination and discovery of security advisories leveraging Web3 technologies, is presented. In particular, the Ethereum blockchain is used by vendors to notify asset owners of novel vulnerabilities in their systems in a reliable and accountable manner. Evaluation tests conducted on the Ethereum Sepolia Testnet confirm that our proposal is a functional and functioning solution for securely disseminating and discovering security advisories utilizing a fully decentralized infrastructure. *SENTINEL*'s source code is released as open source software on GitHub.**

***Index Terms*—Blockchain, Distributed Storage, Security Advisories, SBOM, CSAF**

Acronyms	
AS	Announcement Service
CDX	CycloneDX
CID	Content IDentifier
CPE	Common Platform Enumeration
CSAF	Common Security Advisory Framework
CVE	Common Vulnerability Enumeration
CVSS	Common Vulnerability Scoring System
dApp	Decentralized Application
EVM	Ethereum Virtual Machine
GUI	Graphical User Interface
IIS	Identifier Issuer Service
IPFS	InterPlenary File System
IPNS	InterPlenary Name System
IV	Initialization Vector
NVD	National Vulnerability Database
PoS	Proof-of-Stake
PoW	Proof-of-Work
PSC	Private Smart Contract
SBOM	Software Bill Of Materials
SC	Smart Contract
SPDX	Software Package Data eXchange
SWID	Software Identification tags
VEX	Vulnerability Exploitability eXchange
VSC	Vendor Smart Contract

I. Introduction

Software supply chain attacks are difficult to protect against, as protection requires a deep understanding of the entire software supply chain a system relies on. Given that the number of software dependencies in a software can vary from tens to hundreds, and even to thousands if indirect dependencies are kept into consideration, it is getting increasingly arduous for security engineers to keep track of their dependencies and any potential vulnerability they could be affected by. Indeed, software supply chain attacks are on the rise [1], with the notorious Log4j vulnerability crisis being an outstanding example of the effects such attacks can cause [2]. In the Log4j case, it was estimated that more than 35,000 Java packages were affected, the majority of which were artifacts with deep transitive dependencies which are difficult to detect and patch [3]. Currently, to keep track of vulnerabilities, security engineers have to periodically check vulnerability databases such as CVE [4] and NVD [5] or proprietary vendor websites, leading to a dispersion of information on many different sources. Moreover, the task of analyzing unstructured and non-standard security advisories documents for vulnerability details and remediation strategies further increases the amount of manual work required to keep systems secure [6].

In this paper, a novel and decentralized system named *SENTINEL* is presented. The purpose of the proposed solution is to automate the dissemination and discovery of security advisories using Web3 technologies (i.e., blockchain and distributed storage system), in order to improve and streamline the security advisory dissemination pipeline. The proposed system addresses two different use cases: the *public use case*, in which security advisories are meant to be publicly available and easily accessible by anyone, and the *private use case*, concerning the confidential communication of a new possible threat between a vendor and an asset owner who signed a confidential disclosure agreement. *SENTINEL* consists of a set of Ethereum Smart Contracts and a web-based Graphical User Interface (GUI) for user interaction. The security advisories, following the Common Security Advisory Framework (CSAF) specification [7], are distributed through InterPlanetary File System (IPFS), and their availability is announced through the Ethereum blockchain in the form of events and transaction logs, making them publicly available and making the publisher accountable for vulnerability disclosure. The GUI provides functionalities for both discovery and dissemination of security advisories; this is achieved by allowing asset owners to filter security advisories based on their dependencies from the imported Software Bill Of Materials (SBOM) documents, and by facilitating announcements of new security advisories for vendors. Further, the GUI allows a user-friendly interaction

with the whole infrastructure and tracking of the security advisories of user's interest. *SENTINEL*'s source code is publicly available as open source software on GitHub [8].

The remainder of the paper is structured as follows. Section II presents a background on standards for SBOM, VEX, and CSAF documents, and on the Ethereum blockchain, whereas Section III provides an overview of existing solutions using blockchain to track software dependencies. Then, our proposed system architecture is described, and subsequently its uses cases are described in Sections IV and V, respectively. *SENTINEL*'s performance assessment results are discussed in Section VI. Finally, Section VII concludes the paper summarizing the interesting features of the proposed system and proposing future work.

II. Background

In this section, the SBOM documents used for tracking software and hardware dependencies, as well the VEX and CSAF standards for vulnerability disclosure are introduced. These standardized formats are adopted in *SENTINEL*, and they are of fundamental importance because their specification provides machine-readable documents allowing the automation of the vulnerability disclosure and management processes. Then, the Ethereum blockchain, which our system relies on, is briefly described too.

A. SBOM

A Software Bill of Materials (SBOM) is a formal machine-readable inventory of a piece of software. It contains details and supply chain relationships of components used to build the final software. The details should be comprehensive enough to allow for identification of individual components, and it typically includes information about the supplier, a unique software identifier, software version, etc [9]. Baseline attributes used for identification, as well as, an example of dependency graph are shown in Figure 1.

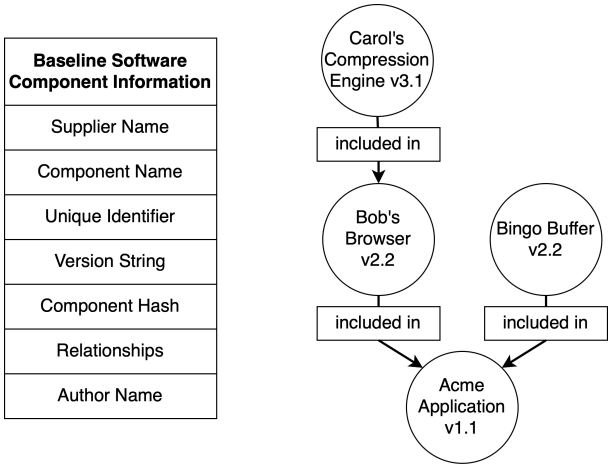

Fig. 1. Baseline information of software components (left), and an example of component relationship description (right) [10]

As discussed in [11], the adoption of SBOM to track the software dependencies leads to a number of benefits like easier management of (1) dependencies, (2) vulnerabilities, (3) risk and (4) licensing. Currently, three key SBOMs formats are adopted in the industry, namely, Software Package Data eXchange (SPDX) [12], CycloneDX (CDX) [13], and Software Identification (SWID) Tags [14]. It is important to note that even though these three formats can be used to represent a SBOM document, they were created at different times for different purposes, thus, the provided crucial information overlap between the different formats, but not all field values can be directly mapped between the formats.

B. VEX

The Vulnerability Exploitability eXchange (VEX) is a security advisory format providing information about the impact of a given vulnerability and recommendations to address the threat posed to the affected system in a machine-readable format [11]. To relate vulnerability and product information, a VEX document must include the four following sections:

- VEX metadata: Identifiers for the VEX format and the document, authors and their role, and timestamp.
- Product details: Product(s) or product families identifiers.
- Vulnerability details: Reported vulnerability identifier and description.
- Product status details: The product status in relation to the stated vulnerability, which can be one of the following: *Not Affected* if no actions are needed, as the vulnerability does not affect the product; *Affected* when the product is affected by the vulnerability, and recommended actions to address the issue are provided in the VEX document; *Fixed* if the mentioned product versions are not affected anymore because the vulnerability has been patched; *Under Investigation* in case the impact of the vulnerability on the product is still unknown, and an update on the status is meant to be provided later.

As the fields above only describe the minimum required elements, software vendors can include more information as they find suitable. Such information could be Common Vulnerability Scoring System (CVSS) scores to state the criticality of the vulnerability, or other non-standard information [15].

C. CSAF

The most popular and industry-wise recognized implementation of VEX is the Common Security Advisory Framework (CSAF) [7]; CSAF documents follow the VEX standard and augment it with additional information which allow for more detailed and complete specification. Different industry standards are adopted to describe the various CSAF documents fields; for instance, Common Platform Enumeration (CPE) identifiers are used to identify systems, and CVSS is used to score the criticality of vulnerabilities. Different tools to support the creation, management, and consumption of CSAF documents have been proposed; still, a common infrastructure to distribute these documents is missing.

D. Ethereum

Ethereum blockchain was launched in 2015, and it is at time of writing the second most popular blockchain, only second to Bitcoin. In contrast to Bitcoin, Ethereum allows users to write and run applications, named Smart Contracts (SC), on the blockchain; more specifically, these programs are executed by the validators on the Ethereum Virtual Machine (EVM) [16]. As of September 2022, Ethereum went through *The Merge*, where the Ethereum Mainnet went from using Proof-of-Work (PoW) to Proof-of-Stake (PoS) consensus algorithm, thus increasing the blockchain scalability and reducing its energy usage [17].

III. State of the Art

In this section, selected existing and under development solutions making use of blockchain technologies to keep track of software dependencies are presented. Whereas all the projects discussed in the following aim at providing a solution to keep track of software dependencies using the SBOM standard and blockchain technologies, our proposal moves the state of the art a few steps forward given that it does not only take care of tracking the dependencies tree but it also provides a comprehensive pipeline for vulnerability disclosure and management.

A. Ortelius

The Ortelius project [18], funded by The Linux Foundation, aims at providing a catalog designed for tracking and versioning microservices software dependencies. The project purpose is to allow developers to easily view their applications' SBOMs, related vulnerabilities, service dependencies, and inventories based on their application version. To keep track of all of these microservices' information, Ortelius creates *snapshots* whose versions are stored immutably on the blockchain. In this way it is possible to track and audit all changes to applications based on updates to microservices [19].

B. D-SBOM

The D-SBOM research project aims to increase IoT device security by providing a clear picture of all installed software in a system. The project is being researched by Asvin GmbH, a company operating in the IoT and cybersecurity fields, and is funded by the European Union's NGI TruBlo Initiative [20]. Asvin proposes to use a decentralized version of a SBOM document by utilizing blockchain technology that, in a secure and trusted way, distributes all the required software to the various actors throughout the supply chain. By allowing IoT vendors to keep track of installed software on their devices, they are able to continuously monitor for CVEs and plan risk mitigation accordingly [20].

C. Software Transparency Foundation

The Software Transparency Foundation is an organization focused on providing transparency to the software supply chain by using existing SBOM standards. By introducing tooling for generation, notarizing, relating, and validation of SBOMs, this initiative seeks to facilitate broader use of SBOMs and thus reduce the effort required to ensure the security of open source related software [21]. The blockchain is used for notarizing and relating SBOMs; by uploading cryptographic checksums of SBOMs together with timestamps and relationship to preceding SBOMs to the blockchain, the ledger functions as a decentralized validation entity allowing asset owners to validate their SBOMs integrity and enabling tracing of all components without any depth limit [22].

IV. Platform Architecture

In this section, *SENTINEL*'s overall architecture and its constituent modules are presented. The system consists of the web-based GUI (which is referred to as "Frontend" in the following), the IPFS and the Ethereum nodes instances, which have to be executed on the user local machine, the four SCs to be deployed on the Ethereum blockchain, and the IPFS network which is used to store and retrieve the published security advisories. These modules are depicted in Figure 2, in which there is also described where each of them is deployed and executed, and they are discussed in detail in the following sections.

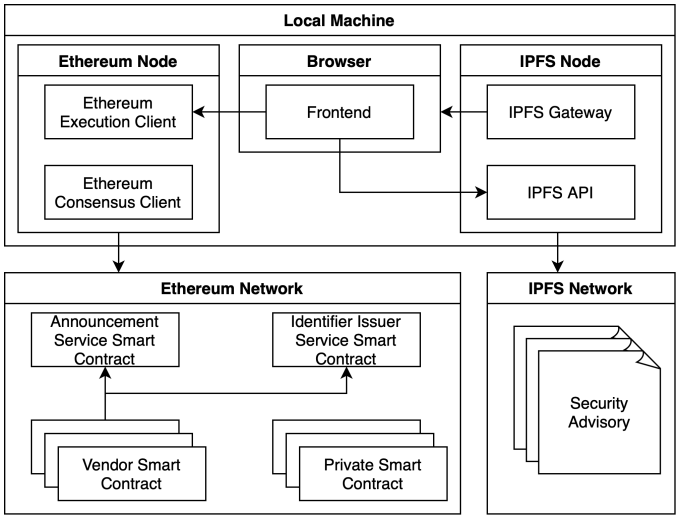

Fig. 2. The architecture of *SENTINEL*.

A. Local Machine

1) Ethereum Node: A local execution client is deployed in the local machine to allow the Frontend to connect to the Ethereum network; the chosen one is Geth [23], given that it is the most popular within the Ethereum community. It exposes a JSON-RPC server on the local machine which the Frontend can query to get data from the blockchain. However, a consensus client needs to be run too in order to receive the blocks from the network and pass them to the execution client which keeps the locally stored blockchain up to date. The most popular consensus client is chosen, namely Lighthouse [24], and it is responsible for the PoS consensus protocol in combination with other validation nodes;

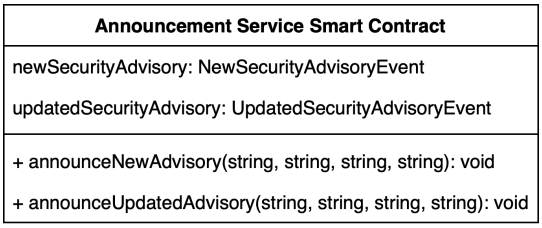

Fig. 3. Announcement Service SC specification.

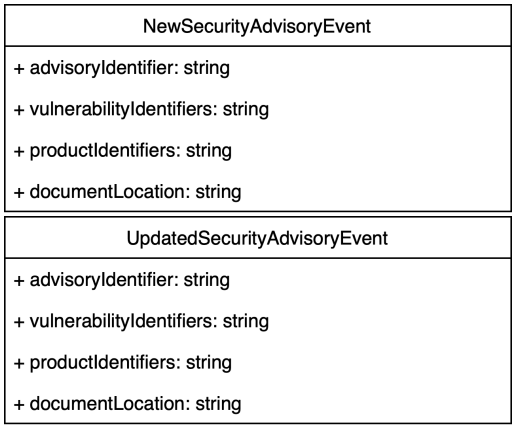

Fig. 4. Events specification for Advisory Announcement (on the top) and Advisory Update (on the bottom).

furthermore, it validates blocks from other consensus nodes with the blockchain data stored in Geth.

2) IPFS Node: Kubo [25] is used as IPFS node on asset owners' and vendors' local machines. Kubo is developed by Protocol Labs [26]. This node contains the core IPFS functionalities to interact with the IPFS network; it includes an API to interact with the node, and has a gateway allowing the user to access content on IPFS. Asset owners and vendors interact with the API on the IPFS node from the Frontend, to upload and download security advisories from the IPFS network.

3) Frontend: It consists of a web-based GUI which allows vendors and asset owners to easily interact with other *SENTINEL*'s modules. It has been developed in React.js as a Single Page Application, and, to follow the Web3 decentralization principle, all the files constituting the interface are uploaded to the IPFS network under a InterPlenary Name System (IPNS) name, such that they can always be accessed at the same URL. Through the GUI, asset owners can import their dependencies by loading one or multiple SBOM files, which are automatically parsed by the system; after that, the system automatically checks for security advisories which are of user interest. Then, once that a new or updated advisory has been discovered, it is automatically downloaded and parsed, and the user can for example inspect the contained information. On the opposite side, vendors can use the GUI to announce public and private security advisories, as will be discussed in Section V.

B. Smart Contracts

In the following, the four SCs needed to operate the *SENTINEL* dApp are described in detail.

1) Announcement Service: The Announcement Service (AS) SC is used by any vendor registered in the system to announce both new security advisories and updates to already announced advisories. These functionalities are made possible by exposing two public functions, namely *announceNewAdvisory* and *announceUpdatedAdvisory*, which in turn emit two events, *newSecurityAdvisory* and *updateSecurityAdvisory*, in order to notify the asset owners on the availability of new security advisories documents. These functions and events specifications can be found in Figures 3 and 4.

Each security advisory announced in *SENTINEL* is associated to an *Advisory Identifier* which is used to uniquely refer to it. Furthermore, it is also used to link updated security advisory announcements to the original advisory announcement. The identifiers are issued by the Identifier Issuer Service (ISS), which is discussed in Section IV-B2. The *Vulnerability Identifiers* property is a string with vulnerability identifiers for the (one or multiple) vulnerabilities reported in the security advisory. Similarly to the *Advisory Identifiers*, the *Vulnerability Identifiers* are issued by the IIS, and are comma-separated to reduce the character overhead and keep the whole field as compact as possible. Similarly to the previous one, the *Product Identifiers* property is a comma-separated string; however, this property contains the identifiers of the products which are vulnerable to any of the vulnerabilities reported in the advisory. This field is of fundamental importance given that it allows asset owners to infer if they could be affected by the announced vulnerability.

2) Identifier Issuer Service: The Identifier Issuer Service (ISS) SC is responsible for generating unique identifiers for advisories and vulnerabilities disclosed using *SENTINEL*. The issued identifiers are similar to how CVE IDs are attributed to CVE records [27]. However, the IIS automatically keeps track and issues the identifiers following the SC implementation, whereas CVE IDs have to be manually assigned.

Advisories and vulnerability identifiers issued by the IIS are generated according to the following string format:

- Platform identifier: the "STNL" string is prepended to indicate that the identifier has been issued by the *SENTINEL* IIS.
- ID's type: whether the ID refers to an Advisory or to a Vulnerability, the "A" or "V" letter is concatenated, respectively.
- Vendor's ID: internal vendor identifier related to the Vendor Smart Contract. Vendor identifiers are issued following the vendor registration in the platform, and are issued incrementally starting from 1.
- Advisory/Vulnerability number: refers to the specific advisory or vulnerability published by the vendor.

All parts of the identifiers are separated with a hyphen, and an example of these IDs is depicted in Figure 5.

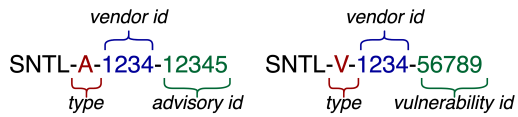

Fig. 5. Example of Advisory Identifier (on the left) and of Vulnerability Identifier (on the right) issued by the IIS.

Identifier Issuer Service Smart Contract
- vendorCount: uint64
- vendors: mapping(address => uint64)
- advisories: mapping(uint64 => uint64[ ])
- vulnerabilities: mapping(uint64 => uint64[ ])
+ registerVendor( ): uint64
+ requestAdvisoryIdentifier( ): string
+ requestVulnerabilityIdentifiers(uint64): string
+ getVendorId(address): uint64
+ getVulnerabilities(uint64): string[ ]

Fig. 6. Identifier Issuer Service SC specification.

To facilitate the storage and tracking of the advisory and vulnerability identifiers, the IIS includes mappings of Vendor Smart Contracts addresses to their unique vendor identifier. Furthermore, mappings from that vendor identifier to the generated advisory and vulnerability identifiers are included. The IIS SC also implements functions that vendors can call to register and receive a vendor identifier, and to generate new advisory and vulnerability identifiers, based on the structure as explained in this section. In Figure 6, the ISS SC specification is depicted.

3) Vendor Smart Contract: Vendor Smart Contract (VSC) is used by vendors to interact with *SENTINEL* in the public use case, and its specification is depicted in Figure 7. Each vendor deploys and uses its own VSC, as the SC represents the vendor. The VSC contains key state variables and methods that are necessary for the vendors to correctly interact with the AS and IIS described in the previous sections. Indeed, during SC deployment, the vendor provides its official name to be used and mapped to its VSC's address, and the IIS SC's and AS SC's addresses to be used during security advisories announcements. Further, the vendor ID assigned by the ISS is stored.

In the public use case, there are two different types of announcements, namely the *new advisory publishing* and the *advisory update*. Therefore, the VSC includes methods to accommodate both types of announcements. The methods interact with the AS SC to emit events with the input data given from the VSC. For announcing new security advisories, the VSC calls the IIS to get advisory and vulnerability identifiers before calling the AS and emitting the event. To announce updates to security advisories, the VSC calls the AS directly,

Vendor Smart Contract
+ vendorName: string
+ vendorId: uint64
- IIS: IdentifierIssuerService
- AS: AnnouncementService
+ announceNewAdvisory(uint16, string, string): void
+ announceUpdatedAdvisory(string, string, string, string): void

Fig. 7. Vendor Smart Contract specification.

as there is no need to generate new identifiers.

4) Private Smart Contract: The Private Smart Contract (PSC) is the SC on which asset owners receive confidential security advisories from a vendor in the private use case, and it is deployed by the asset owner. The PSC contains essential methods and properties for ensuring disclosure of confidential security advisories, and its specification is illustrated in Figure 8. In the PSC a custom modifier named *whitelisted* is used to restrict access to only *whitelisted* vendors, i.e., vendors who signed a confidential disclosure agreement with the asset owner. This ensures that the PSC is not flooded with events from malicious actors. To complement this modifier, two methods, *addVendor* and *removeVendor*, are used. As the names suggest, these methods allow the asset owner to add or remove vendors from the whitelist. The whitelisted addresses are stored in the vendor state variable as a mapping from address to boolean. This allows a constant time lookup to determine if a vendor is whitelisted or not. The PSC implements another state variable in addition to the whitelist, namely *publicKey*, which contains the public key to be used for announcing confidential security advisories. An accompanying method, *setPublicKey*, is used to set the *publicKey* state variable.

To announce a confidential security advisory, whitelisted vendors can call the *announce* method, which in turns emits the *AnnouncementEvent* which is illustrated in Figure 9. This event contains the following four fields used for confidential announcements:

- Document Location: Content IDentifier (CID) referencing to the document location on IPFS.
- Security Advisory Hash: the hash of the security advisory is included to ensure document integrity and non-repudiability in the event that the security advisory is no more accessible; by having a backup of the security advisory, the vendor can show that the cryptographic hash of the security advisory matches the one emitted in the event.
- Decryption Key and Initialization Vector (IV): Information needed to decrypt the encrypted security advisory document. Further details are provided in Section V-B.

V. Use Cases

As briefly discussed in the introduction, the platform is designed to support two different use cases, for the *public*

Private Smart Contract
announcementEvent: AnnouncementEvent
- vendors: mapping(address => bool) + publicKey: bytes
modifier: whitelisted
+ setPublicKey(bytes): void + addVendor(address): void + removeVendor(address): void + announce(string, bytes32, bytes, bytes12): void

Fig. 8. Private Smart Contract specification.

AnnouncementEvent
+ documentLocation: string + securityAdvisoryHash: bytes32 + decryptionKey: bytes + initializationVector: bytes12

Fig. 9. Confidential Announcement event specification.

and *private* disclosure of vulnerabilities. In the next two subsections these two use cases and the respective interaction processes are discussed.

A. Public Vulnerability Disclosure

The Public Vulnerability Disclosure use case regards the public disclosure of security issues found in a software or hardware component which needs to be notified to a wide audience of asset owners. To publish a new security advisory, the vendor first announces it using the VSC. The VSC then automatically retrieves from the IIS the advisory and vulnerability identifiers to be used, and once the identifiers are returned, they are forwarded by the VSC to the AS, which finally emits an event containing the security advisory information. The Frontend module automatically takes care of periodically checking whether a new advisory of interest of the asset owner is available, and if it is the case, the user is notified about the possible security threat, and the advisory is retrieved and made accessible. This process is also depicted in Figure 10.

B. Private Vulnerability Disclosure

The Private Vulnerability Disclosure use case addresses the issue of confidential disclosure of vulnerabilities by vendors; indeed, it often happens that vendors and asset owners have private agreements which assure the user that as soon as a new vulnerability in one of their dependencies has been discovered, they are timely and confidentially notified by the vendor on the issue. In order to guarantee state of the art security policies, NIST directives are followed on the encryption of

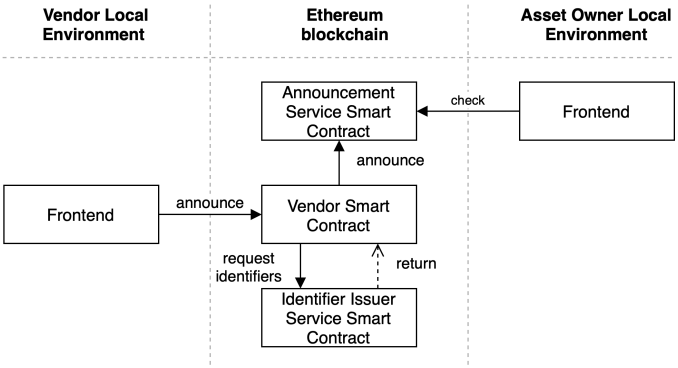

Fig. 10. Overview of modules interactions in the public use case.

the advisories and on distribution of the decrypting keys [28]. To support this use case, the asset owner needs to deploy the PSC, generate a new RSA-OAEP key pair, and publish the public key in the PSC. When the vendor intends to announce a confidential security advisory, it generates an AES-GCM key and a random Initialization Vector (IV), which is used to encrypt the security advisory; then the advisory is published on IPFS in encrypted form. Following this, the vendor encrypts the AES-GCM key with the key retrieved from the asset owner's PSC using RSA-OAEP. The vendor can then announce an event through the PSC including the encrypted AES-GCM key, the IV, and the other necessary information presented in Section IV-B4. Finally, the asset owner, upon automatically discovering the newly emitted event through the GUI, decrypts the AES-GCM key using their RSA-OAEP private key and in turn uses the AES-GCM key to decrypt the security advisory. This process is depicted in detail and from a high level point of view in Figures 11 and 12, respectively.

VI. Performance Assessment

To evaluate the adoption feasibility of the proposed platform, *SENTINEL*'s SCs have been deployed on the Ethereum Sepolia testnet, and its scalability, usage cost, and availability are assessed.

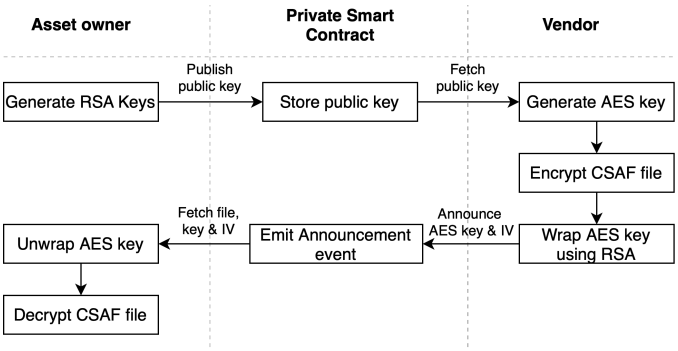

Fig. 11. The data flow of the private use case, including encryption procedures.

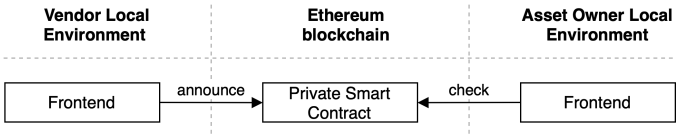

Fig. 12. Overview of modules interactions in the private use case.

A. Scalability

It is required that *SENTINEL*, and by extension the Ethereum network, can support the amount of transactions that users will generate when interacting with the system. Since September 15th 2022 (date of The Merge) to April 25th 2023 there have been an average of 1,061,572 transactions per day on the Ethereum Mainnet. To estimate the impact that our dApp would have on the Ethereum blockchain, the number of security advisories published yearly by CVE and NVD are taken into consideration [29], [30]. According to the data from 2019 to 2022, 55 new security advisories are published on average each day, meaning that the amount of announcement transactions would be only 0.000657% of all the transactions on the Ethereum network per day on average, which is a negligible quantity of further traffic that the network can handle without any Quality of Service degradation.

B. Cost

As executing a transaction in the Ethereum network requires the transaction issuer to pay a fee to the validators for the transaction to be executed and added to a block, the cost of using *SENTINEL* is investigated. The cost is measured in gas units, as converting to a fiat currency is not a reliable metric given the fluctuating exchange rate. In Eq. 1, the formula to calculate the transaction cost is presented, where *baseFee* is a hardcoded value associated to each *EVM OPCODE* and the *priorityFee* is decided by the transaction issuer [31].

$$txCost = gasUnits \cdot (baseFee + priorityFee) \quad (1)$$

1) Deployment Cost: SCs must be deployed in the blockchain for both the public and private use cases. In Table I the transaction costs are presented; it should be noted that these are one-off interactions needed only at SCs deployment.

Contract	Deployment Cost
Announcement Service	404,628 gas
Identifier Issuer Service	1,328,841 gas
Vendor Smart Contract	1,098,681 gas
Private Smart Contract	1,124,256 gas

TABLE I
SENTINEL SMART CONTRACT DEPLOYMENT GAS COST.

2) Interactions Cost: The transactions discussed in the following are the ones needed to be issued in order to make use of the functionalities offered by the platform. These include security advisories publication and update, and the various operations required by the private use case, which are: public key update, vendor whitelisting, vendor removal from whitelist, and confidential announcement. Given that transactions change the state of the blockchain, and depending on SC memory state, more storage could be needed to be allocated to store new data, the transaction cost can slightly vary from one execution to another; for this reason, during the performance assessment, the transactions are issued ten times in sequence with the same input data to more accurately capture the execution cost; these results are reported in tables II and III.

	Execution Cost		
Transaction	**min**	**max**	**avg**
New announcement	88,903 gas	148,803 gas	100,900 gas
Update announcement	42,744 gas	42,744 gas	42,744 gas

TABLE II
GAS COST OF TRANSACTION FOR THE PUBLIC USE CASE OF *SENTINEL*.

	Execution Cost		
Transaction	**min**	**max**	**avg**
Update public key	45,192	332,884	73,961
Add to whitelist	44,790	44,790	44,790
Remove from whitelist	14,905	14,905	14,905
Announcement	40,423	40,423	40,423

TABLE III
COST COMPARISON OF PRIVATE USE CASE TRANSACTIONS IN *SENTINEL*.

3) Cost Considerations: For the sake of providing a more appreciable cost estimation of operating the system, the average gas price and Ether-USD exchange rate which has been observed from September 2022 to April 2023 are used to compute the actual cost in fiat currency. The former one has been set to 25.63 Gwei, whereas the latter one to $1,499. In Table IV the results are presented for each interaction discussed in the previous subsections.

Public use case			
Action	**Gas units**	**Ether**	**USD**
Vendor deployment	1,098,681 gas	0.028159194 ETH	$42.21
New announcement	100,900 gas	0.002586067 ETH	$3.88
Update announcement	42,744 gas	0.001095528 ETH	$1.64
Private use case			
Action	**Gas units**	**Ether**	**USD**
Private deployment	1,124,256 gas	0.02881468128 ETH	$43.19
Update public key	332,884 gas	0.00853181692 ETH	$12.79
Add to whitelist	44,790 gas	0.00114796770 ETH	$1.72
Remove from whitelist	14,905 gas	0.00038201515 ETH	$0.57
Private announcement	40,423 gas	0.00103604149 ETH	$1.55

TABLE IV
COST CALCULATIONS FOR DIFFERENT BLOCKCHAIN SPECIFIC OPERATIONS OF *SENTINEL*. GAS PRICE ASSUMED TO 25.63 GWEI AND ETH-USD EXCHANGE RATE IS SET TO $1,499.

It can be noted that a relatively large up-front payment to deploy the SCs is required to deploy the SCs required for each use case, whereas the cost of further interactions is much lower. It can be argued that these costs are reasonable and affordable for both vendors and asset owners.

C. Availability

Timely and continuous availability of security advisories is fundamental to allow asset owners to react when a new

vulnerability is disclosed; for this reason, the time for security advisory announcement to be retrievable through the blockchain and the time for security advisory documents to be uploaded to IPFS are analyzed, as both of these are necessary for asset owners to become aware of potential vulnerabilities. Even though no official data is currently available regarding the transaction confirmation time after The Merge, Etherscan's Gas Tracker [32] provides accurate real-time estimates on suggested transaction fee and expected confirmation time. Etherscan insights show that paying near the highest gas fee gives an estimated confirmation time of around 30 seconds, increasing to 3 minutes if the lower priority fee is paid. Moreover, a 15 minutes time span should be waited for the block to be considered actually finalized and accepted by a wide majority of the validators. With this knowledge about confirmation and finality time, vendors can expect their announcements to be available to users after 0.5 to 3 minutes and stored indefinitely after about 15.5 to 18 minutes. For what concerns the availability of the security advisory documents on IPFS, tests have been performed using two different IPFS nodes: a well established node, connected to a p2p network for three months, and a new node with no previous connections. During the tests it was observed that the documents were available practically immediately when using the well-established node, and after 30 minutes when using the other one.

VII. Conclusions and Future Work

In this paper, *SENTINEL*, a platform based on Web3 technologies meant to automate the dissemination and discovery of security advisories is presented. Thereafter introducing the various standards and solutions for software dependencies and vulnerability tracking, and discussing the state of the art, our platform architecture is discussed in detail. The considered two use cases, namely, the public one and the private one, aim to address different needs coming from the industry when it comes to vulnerability disclosure and management. Finally, *SENTINEL* performances are assessed in terms of scalability, operation cost, and availability; these tests performed on the Ethereum Sepolia testnet confirm the operability of the whole system, and pave the path to a real testing and adoption in the industry. Indeed, the envisioned next steps include the collaboration with vendors and asset owners to evaluate the infrastructure performances and usability in a real deployment scenario.

Acknowledgment

Research funded in part by the European Union through the Horizon 2020 project FEVER (grant agreement 864537).

References

[1] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*. Springer, 2020, pp. 23–43.

[2] (2021) Apache log4j security vulnerabilities. [Online]. Available: https://logging.apache.org/log4j/2.x/security.html

[3] J. Wetter and N. Ringland. (2021) Understanding the impact of apache log4j vulnerability. [Online]. Available: https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html

[4] Common vulnerability enumeration (cve) webpage. [Online]. Available: https://www.cve.org

[5] NIST. National vulnerability database (nvd). [Online]. Available: https://nvd.nist.gov

[6] R. R. Ramnani, K. Shivaram, and S. Sengupta, "Semi-automated information extraction from unstructured threat advisories," in *Proceedings of the 10th Innovations in Software Engineering Conference*. ACM, 2017, pp. 181–187.

[7] OASIS CSAF Technical Committee, "Common security advisory framework version 2.0," 2022. [Online]. Available: https://docs.oasis-open.org/csaf/csaf/v2.0/csaf-v2.0.html

[8] Sentinel's public github repository. [Online]. Available: https://github.com/JannikSommer/SENTINEL

[9] National Telecommunication and Information Administration. Sbom at a glance. [Online]. Available: https://ntia.gov/files/ntia/publications/sbom_at_a_glance_apr2021.pdf

[10] ——. Software bill of materials. [Online]. Available: https://www.ntia.gov/files/ntia/publications/sbom_overview_20200818.pdf

[11] N. Zahan, E. Lin, M. Tamanna, W. Enck, and L. Williams, "Software bills of materials are required. are we there yet?" *IEEE Security & Privacy*, vol. 21, no. 2, pp. 82–88, 2023.

[12] Linux Foundation and its Contributors. A common software package data exchange format, version 2.0. [Online]. Available: https://spdx.org/sites/spdx/files/SPDX-2.0.pdf

[13] OWASP Foundation. Cyclonedx specification. [Online]. Available: https://cyclonedx.org/specification/overview/

[14] D. Waltermire, B. A. Cheikes, L. Feldman, D. Waltermire, and G. Witte, *Guidelines for the creation of interoperable software identification (SWID) tags*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[15] Allan Friedman et al. Vulnerability exploitability exchange (vex) – use cases. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/VEX_Use_Cases_Aprill2022.pdf

[16] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[17] E. Kapengut and B. Mizrach, "An event study of the ethereum transition to proof-of-stake," *Commodities*, vol. 2, no. 2, pp. 96–110, 2023.

[18] Ortelius. Ortelius website. [Online]. Available: https://ortelius.io/

[19] ——. la-sbom-ledger. [Online]. Available: https://github.com/ortelius/la-sbom-ledger

[20] Trublo. D-sbom - distributed software bill of materials. [Online]. Available: https://www. trublo.eu/D-SBOM/

[21] Software Transparency Foundation. Software transparency foundation charter. [Online]. Available: https://st.foundation/stf_charter.pdf

[22] Software Transparency Foundation Mission. Software transparency foundation charter. [Online]. Available: https://st.foundation/mission

[23] Ethereum Foundation. go-ethereum. [Online]. Available: https://geth.ethereum.org

[24] Sigma Prime. Lighthouse github. [Online]. Available: https://github.com/sigp/lighthouse

[25] Protocol Labs. Kubo. [Online]. Available: https://github.com/ipfs/kubo

[26] Protocol labs. [Online]. Available: https://protocol.ai

[27] The MITRE Corporation. Cve numbering authority (cna) rules. [Online]. Available: https://www.cve.org/ResourcesSupport/AllResources/CNARules

[28] E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon, "Recommendation for pair-wise key establishment using integer factorization cryptography," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-56B, Rev. 2, 2019.

[29] The MITRE Corporation. Metrics. [Online]. Available: https://www.cve.org/About/Metrics

[30] NIST. Metrics. [Online]. Available: https://nvd.nist.gov/vuln/search

[31] Ethereum Foundation. Gas and fees. [Online]. Available: https://ethereum.org/en/developers/docs/gas/

[32] Etherscan. Estimated transaction fees and confirmation time. [Online]. Available: https://etherscan.io/gastracker