SENTINEL - Automatic Dissemination and Discovery of Security Advisories with Web3

Master's Thesis

Aalborg University
Master of Science (MSc) in Software Engineering

Summary

Software supply chain attacks are difficult to protect against, as protection requires a good understanding of the entire software supply chain a system relies on. Furthermore, these attacks are on the rise, with the notorious Log4j vulnerability crisis being an outstanding example of the effects such attacks can cause. As such, security engineers need to keep track of their software and hardware dependencies and any potential vulnerabilities they are affected by. Currently, to keep track of vulnerabilities, security engineers check vulnerability databases such as CVE and NVD or proprietary vendor websites. Consequently, security engineers may have to search multiple sources to perform a complete security analysis and find all potential vulnerabilities relevant for their systems.

In this report, a novel and decentralized system named SENTINEL is proposed. The purpose of SENTINEL is to automate the dissemination and discovery of security advisories using Web3 technologies, in order to improve and streamline the security advisory dissemination pipeline.

SENTINEL consists of a set of Ethereum smart contracts and a user interface for interaction with the system. The security advisories are distributed with IPFS, a decentralized storage system, and security advisories are announced to the Ethereum blockchain in the form of events and transaction logs, making them publicly available. The user interface provides functionality for both discovery and dissemination of security advisories. This is achieved by allowing asset owners to filter security advisories on their dependencies from a SBOM document, and by facilitating announcement of new security advisories for vendors.

A prototype of SENTINEL is implemented and tested to assess the viability of the proposed system. The availability and scalability of the prototype are assessed to measure the practicality of using the system. A cost analysis is performed to examine if the cost of using SENTINEL is reasonable in regard to its improvements to the process of security advisory dissemination and discovery. The cryptographic mechanisms are assessed to investigate if adequate security is provided in cases where confidentiality is required. Finally, a system test is conducted on an

Ethereum testnet to confirm if the required features are present in the system and to show if it works as intended in a real world environment.

Software
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
SENTINEL - Automatic Dissemination and Discovery of Security Advisories with Web3

Theme:
Master's Thesis

Project Period:
Spring Semester 2023

Project Group:
cs-23-ds-10-08

Participant(s):
Jannik Lucas Sommer
Magnus Mølgaard Lund

Supervisor(s):
Michele Albano, mialb@cs.aau.dk
Nicola Cibin, nicolac@cs.aau.dk

Page Numbers: 115

Date of Completion:
June 7, 2023

Abstract:

The prevalence of software supply chain attacks has reached unprecedented levels, primarily due to the increasing reliance on software dependencies and the inherent vulnerabilities they harbor. Currently, vendors share security advisories to centralized databases or proprietary websites, which security engineers have to search manually to find vulnerabilities relevant for their system. 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 report, SENTINEL, a novel solution for automating dissemination and discovery of security advisories using Web3 technologies, is proposed. A system test conducted on the Sepolia Ethereum Testnet confirm that SENTINEL is a functioning solution for securely disseminating and discovering security advisories utilizing a fully decentralized infrastructure.

Preface

Aalborg University, June 7, 2023

Jannik Lucas Sommer
<jsomme18@student.aau.dk>

Magnus Mølgaard Lund
<mlund18@student.aau.dk>

Foreword

This report is the Master's Thesis for the Master program in Software Engineering from Aalborg University. The work of this project is based upon previous work [1] completed on the previous semester. The semester started February 1st 2023, and concluded with the project exam on June 20th.

We would like to show gratitude towards our supervisors Nicola Cibin and Michele Albano for their collaboration and guidance throughout both the pre-specialization project and the Master's Thesis.

The implementation code for this project is open source and can be found at `https://github.com/JannikSommer/SENTINEL`. Note that the repository may change to a different URL in the future.

Reading Guide

As this report is based upon previous work, we assume that the reader has read and understood the concepts introduced in that report. Therefore, we strongly recommend reading the previous work before this report [1].

This report follows the Vancouver referencing method. Citations which refer to multiple paragraphs will be placed at the last paragraph which references the citation. As such, the citations in text will be kept at a minimum.

For an overview of all references cited in the report, see page 83.

Acronyms

CVE Common Vulnerability Enumeration
NVD National Vulnerability Database
SBOM Software Bill Of Materials
DApp Decentralized Application
CSAF Commons Security Advisory Framework
MoSCoW Must have, Should have, Could have, Won't have
CDX CycloneDX
SPDX Software Package Data Exchange
SWID Software Identification Tags
NIST National Institution of Standards and Technologies
IPFS InterPlanetary File System
AS Announcement Service
IIS Identifier Issuer Service
VSC Vendor Smart Contract
PSC Private Smart Contract
IPNS InterPlanetary Name System
CID Content Identifier
DES Data Encryption Standard
AES Advanced Encryption Standard
AES-GCM AES in Galois/Counter Mode
IV Initialization Vector
LTS Long Term Support
SPA Single Page Application
OWASP Open Worldwide Application Security Project
CI/CD Continuous Integration and Continuous Deployment
EVM Ethereum Virtual Machine
SWC Smart contract Weakness Classification
DoS Denial of Service

Contents

Chapter 1

Introduction

According to a survey conducted by The Linux Foundation [2], it was found that 98% of organizations use open source software, and 95% of them express concerns regarding software security. With the increasing reliance on open source in software development, the focus on cybersecurity threats on supply chain attacks has particularly intensified. It is evident that the software supply chain has emerged as a significant attack vector. Notably, the supply chain attack on the SolarWinds Orion framework and the vulnerability associated with Apache Log4j led to two prominent software security crises. The Log4j vulnerability, in particular, had a global impact, affecting over 35,000 Java packages. Moreover, a significant portion of the impacted artifacts consists of transitive dependencies, making patching the vulnerabilities a challenging task. Developers are often required to wait for updates to their dependencies, which can be time-consuming because fixes must propagate through the dependency chain [2, 3].

With the increasing number of publicly disclosed vulnerabilities and the increasing complexity of software systems, it is becoming progressively more difficult to manually track dependencies and any potential vulnerabilities. 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. [4]

Given the discussed problems related to security advisories dissemination and discovery, the following initial problem statement which guides the project design and development is introduced:

"How can mechanisms for vulnerability dissemination and discovery be improved?"

This report is structured as follows. In Chapter 2 previous work regarding the

problem introduced is briefly introduced and explained. Chapter 3 introduces a problem state which expands upon the initial problem introduced in this chapter with more details about the problem explained in previous work. From the problem statement, Chapter 4 details a design of a system, which is then implemented as described in Chapter 5. Chapter 6 investigates different characteristics about the implemented system and if the system conforms the specified requirements. Chapter 8 concludes on the results of the project detailed in this report, and suggestions for future work are presented in Chapter 9.

Chapter 2

Previous Work

The project described in this report is based on previous work [1]. It contained analyses of the problem area and Web3 technologies, which was the basis for a design and proof of concept of a system for automatic dissemination and discovery of security advisories. This chapter will briefly summarize the previous work.

2.1 Summary of Previous Work

Previous work was based on the observation about the state of the industry in regard to how security advisories are disseminated by vendors and discovered by asset owners. This observation is illustrated in Figure 2.1 as the timeframe of concern, where there is a missing link between security advisory publication and discovery. Security engineers spend time on manually searching for security advisories and vulnerabilities which could affect their system. Common Vulnerability Enumeration (CVE) and National Vulnerability Database (NVD) are repositories where security engineers can search for public vulnerabilities. However, these repositories might not be exhaustive, as some vendors use proprietary websites to announce new vulnerabilities in their products. This means that security engineers may need to search multiple sources to exhaust all potentially relevant vulnerability databases.

Furthermore, CVE and NVD are operated by large centralized organizations where bureaucracy can slow down or censor the announcement of new vulnerabilities to the public.

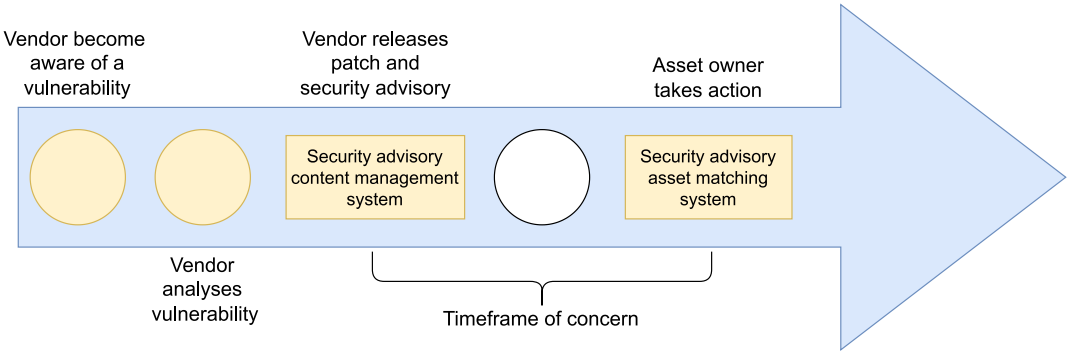

Figure 2.1: Timeframe of concern regarding the security advisory dissemination and discovery pipeline. [1]

From analyzing the industry and an interview with an expert, two distinct use cases for disseminating and discovering security advisories were envisioned. Firstly, the public use case, in which vulnerabilities are shared with the public. However, in some situations, vendors, the party who provides a software component, and asset owners, the party which uses the software component, may have a contract specifying that the vendor must disclose vulnerabilities before it is made public. Then the asset owner can patch the vulnerability as soon as possible and reducing the chances of the vulnerability being exploited. This private use case is useful for critical systems, however, the disclosure is sometimes carried out using an email with a link to a security advisory in cloud storage, which is a manual process.

A set of requirements that could solve these problems with the current dissemination and discovery of security advisories in industry were specified. The focus of these requirements was on automation, unified infrastructure, and decentralization, for both use cases. Furthermore, a proposed system should make use of standards such as Software Bill of Materials (SBOM) to improve automation. From these requirements, an analysis of technologies to meet these requirements was conducted.

With a focus on decentralization, Web3 technologies, were investigated. Several decentralized storage systems and blockchains were investigated to determine which are best suited to build a decentralized application (DApp) that can improve automation of security advisory dissemination.

From the problem and technology analysis, the following problem statement was created: *"How can a single infrastructure for dissemination of security advisories be designed?"* [1]. From this problem statement, a design of such an infrastructure was developed.

The design for both the public and private use cases are based on the same blockchain network. Ethereum was determined to be the most mature blockchain that includes the capabilities for smart contracts that can emit events to the transaction logs. This is useful for announcing security advisories. In the proposed design, vendors use deployed smart contracts to create identifiers for the vulnerabilities, and to announce security advisories. Asset owners search these logs on Ethereum to find security advisory announcements relevant for their system. The actual security advisories are written in a machine-readable format and published to a decentralized storage system, which the asset owners can retrieve it from and parse it to try to match it to any of their dependencies. Asset owners interact with the system from a frontend system, which will perform all computations that are not strictly required to be on the blockchain, in order to reduce the computation cost.

For the private use case, confidentiality of the security advisories is necessary because of the transparent nature of the blockchain. The disclosure agreement between asset owner and vendor is realized with a single smart contract, that can emit events to announce security advisories for the specific asset owner.

A proof of concept was created based on the proposed design. The utilization of the events on smart contracts and retrieval of the data with a frontend component was found to be functional, and the proposed design was therefore considered valid as a solution for the problem statement.

Chapter 3

Problem Statement

In this chapter, a problem statement, based on previous work introduced in Chapter 2, is defined. The problem statement provides the direction of the project. It is presented below, along with its accompanying sub-problem statements.

- How can a prototype system for automatic dissemination and discovery of security advisories be designed, implemented, and tested?
  - How can such a system incorporate decentralization by integration with Web3 technologies?
  - How can such a system use industry standards to increase automation and interoperability?
  - How can such a system disclose vulnerabilities to asset owners without exposing which systems are vulnerable?

To make it easier to refer to the prototype system created from this project, it will be referred to as SENTINEL for the remainder of the report.

Chapter 4

Design

This chapter will include the design of SENTINEL, which aims to solve the issues raised by the problem statement in Chapter 3. In this chapter, concepts and considerations about the design are presented. Based on these, the system architecture and component design of SENTINEL is defined.

The design chapter includes the following sections.

- Event Concept
- User Interaction
- Requirements
- Design Considerations & Assumptions
- Storage System Design
- System Overview
- Smart Contract Design
- Frontend Design

4.1 Event Concept

As smart contracts are isolated in the blockchain they are deployed in, it is not possible for them to send data to any off-chain system. In Ethereum, events can be used to emit data to the transaction logs, which facilitates data transfer from smart contracts to outside the Ethereum network. Another approach would be to store data on the smart contract itself with state variables. However, emitting data is cheaper than storing the data with state variables on a smart contract, which makes it convenient for communicating data from smart contracts to user interfaces, for data that should not be used by smart contracts.

Events are defined in smart contracts, where the properties and the type of those properties are defined. The smart contract can emit the event in smart contract methods. The event data is written to the transaction log when the transaction is mined, and external systems can find the event data in the transaction logs of the mined blocks. [5]

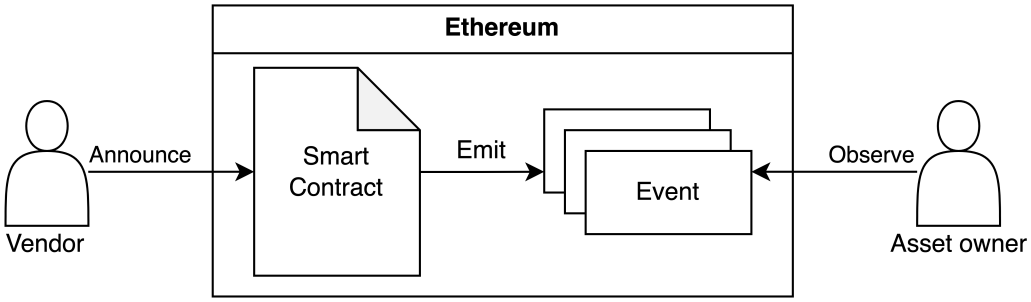

Figure 4.1: High level overview of how the event concept in Ethereum is used.

SENTINEL uses the concept of Ethereum events to announce security advisories to asset owners, as illustrated in Figure 4.1. The security advisory announcements are made public to anyone who listens. As the events are emitted to the transaction logs which are stored in blocks on the blockchain, all security advisory announcements made with SENTINEL are available forever.

4.2 User Interaction

In previous work, a solution for both the public and private vulnerability disclosure use cases was proposed. The proposed solution allows vendors to upload and announce security advisories, which are automatically disseminated to subscribed asset owners. A high level view of the interactions occurring in the platform is illustrated in Figure 4.2. The message chart covers both the public and private use case, however, they require different implementations to fulfill their specific

purposes.

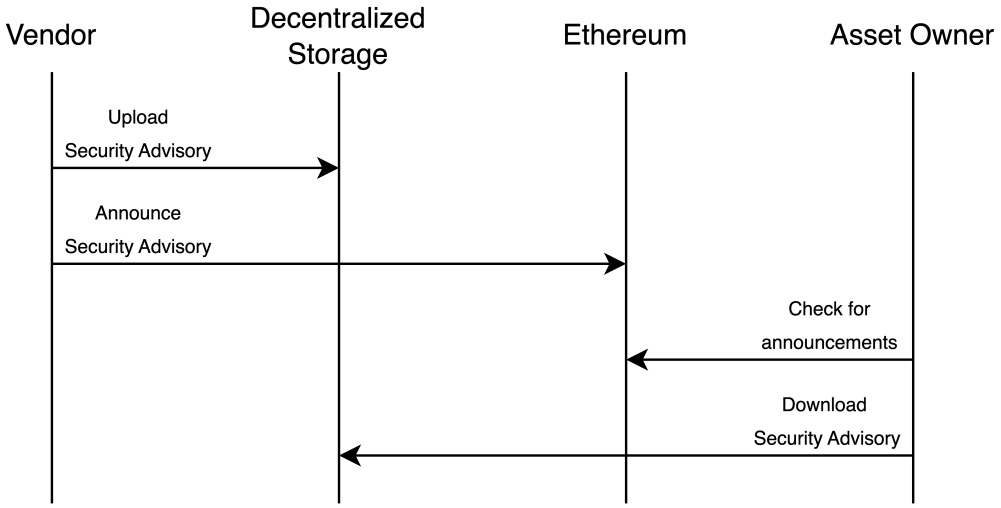

Figure 4.2: Message chart of user interaction with vulnerability disclosure in both public and private use case.

Vendors upload the security advisory for one or more vulnerabilities to a decentralized storage service. The vendor then announces the new public security advisory to the Ethereum blockchain, including the location of the security advisory document. When the event has been emitted, asset owners can retrieve the event data and thereby the security advisory from the decentralized storage. In the private use case, the announcement is still publicly visible, as per the transparency of the blockchain network. However, encryption measures are utilized to ensure confidentiality of the security advisory content. With the security advisory announcements, asset owners can retrieve the security advisory from the decentralized storage service given the storage location from the announcement.

4.3 Requirements

A system to solve the issues raised in Chapter 3 should meet certain requirements. In this section, the criteria and feature requirements that SENTINEL must fulfill are presented. Furthermore, the feature requirements are prioritized for the implementation.

4.3.1 System Requirements

Previous work introduced several requirements for how a system that could improve on existing issues of how security advisories are disseminated and discov-

ered in the current state of the industry. These requirements are reintroduced and expanded in the following list, and in turn they can be broken down into several criteria which SENTINEL should adhere to.

- **Integrity**: Throughout its lifecycle, a security advisory should maintain consistent information, except in cases where updates are necessary. As a result, it should be possible to modify security advisories in order to incorporate new information as it becomes available.
- **Standardization**: In order to facilitate automation and broad adoption, security advisories should adhere to machine-readable standards to facilitate the parsing of information.
- **Unification**: Using a shared infrastructure for storing and distributing security advisories would simplify the process of discovering these documents for asset owners. Additionally, vendors would only need to send the advisories to a single platform to reach all affected asset owners.
- **Decentralization**: By avoiding centralized third-party organizations such as Mitre or National Institute of Standards and Technologies (NIST), the need for time-consuming bureaucracy can be eliminated. Additionally, a decentralized infrastructure is trustless, and asset owners only need to place their trust in the vendors who are publishing security advisories.
- **Non-repudiation**: Asset owners should not be able to repute that a vendor has disclosed a vulnerability in due time, if the security advisory was disclosed in due time. With the immutability of the Ethereum blockchain, it would be easier for a vendor to show asset owners the transaction data that makes the announcement. Thereby, vendors are not accountable for vulnerability exploits that an asset owner has disregarded.
- **Completeness**: Asset owners can have agreements with vendors to receive security advisories before they are made public. Therefore, an infrastructure that supports the dissemination of security advisories should support both public and confidential security advisory announcements.

4.3.2 Requirement Analysis

In order to prioritize features of SENTINEL, a requirement analysis is made using the MoSCoW framework [6]. Thereby, the most important features of the system are developed first, and the less important features are prioritized less. Table 4.1 shows the features of SENTINEL, divided into the four MoSCoW categories. The features are based upon the system requirements specified in Section 4.3.1.

Must have	**Should have**
- Announce public security advisories - Announce private security advisories - Discover security advisories - Load dependencies from SBOM - Filter advisories on dependencies - Present security advisory content - Integrate with storage system - Security advisory identification - Vulnerability identification	- Filter advisories on vendor - Announce updates to advisories - Discover advisory updates - Manage confidential agreements - Persistent storage of settings - Contract deployment - Account management
Could have	**Won't have**
- Support multiple storage systems - Support multiple advisory formats - Support multiple SBOM formats	

Table 4.1: Requirement analysis with the use of the MoSCoW framework.

Must have

The must-have requirements from Table 4.1 are essential for SENTINEL to fulfill its purpose and are non-negotiable. If any of the requirements in the must-have category are not met, the system cannot work as intended. The must-have requirements also outline the features necessary for a minimum viable product.

Announce public & private security advisories: Vendors must be able to use SENTINEL to announce security advisories for both the public and private use case. Smart contracts, that facilitate emitting events to the transaction logs, must be available for vendors to use. In the private use case, asset owners must be able to restrict who can announce security advisories to them.

Discover security advisories: Asset owners must be able to use SENTINEL to discover security advisories that have been announced with SENTINEL. In other words, a frontend component of SENTINEL that allows vendors to discover security advisories announced by vendors, must be created to incorporate automation and usability functionality. Furthermore, the frontend must be connected to the Ethereum network, as it is where the announcements of security advisories are stored.

Load dependencies from SBOM and filter advisories on dependencies: Asset owners must be able to provide a SBOM document to a frontend component from which SENTINEL can load their dependencies, and filter announced security advisories on. This requirement will provide the asset owners with security advi-

sories relevant for their system and reduce the amount of manual labor of filtering security advisories.

Present security advisory content: Asset owners must be able to read security advisory information in a human-readable format, with only relevant information presented. Security advisory documents may contain many vulnerabilities for many products, which necessitates filtering the information on the dependency information provided from SBOM documents, such that only the information relevant to the asset owner is presented.

Integrate with storage system: A frontend component of SENTINEL must integrate with a decentralized storage system. As the security advisories are published to a decentralized storage system, the frontend component must integrate with such a storage system, to upload and download security advisories for vendors and asset owners respectively.

Security advisory & vulnerability identification: In SENTINEL, the announced security advisories and the vulnerabilities within must be identifiable and, in turn, easily distinguishable. Moreover, they must be uniquely identifiable, such that two security advisories or vulnerabilities never have the same identifier. With identification in place, the security advisories and vulnerabilities can be referenced by a unique identifier, and the security advisories can be updated based on the identifier.

Should have

The should-have requirements from Table 4.1 are important but not necessary for the system, and should therefore be prioritized after all must-have requirements are fulfilled. Furthermore, many of these requirements are dependent on the implementations of the must-have requirements.

Filter advisories on vendor: Asset owners should be able to filter the security advisories they are presented based on the vendor who published the security advisory. Due to the permissionless nature of the Ethereum blockchain and the decentralized nature of SENTINEL, there is no restriction on who can announce public security advisories. Therefore, asset owners should be able to specify which vendors they trust to reduce the risk of receiving false information from malicious actors.

Announce and discover updates to advisories: Vendors should be able to announce updates to already announced security advisories. As security advisories

can be updated with more information or modifications, SENTINEL should support vendors with functionality that facilitates announcing updates to published security advisories. By extension, asset owners should also be able to discover these updates to security advisories.

Manage confidential agreements: Asset owners should be able to manage their confidential agreements with a vendor to facilitate the private use case. As such, the frontend component should include functionality for this.

Persistent storage of settings: Asset owners and vendors should be able to store any settings used to interact with SENTINEL persistently, such that the settings are retained with any crashes or applied to a new environment. This requirement only applies to any information not stored on the blockchain.

Contract deployment: Asset owners and vendors should be able to use the frontend component to easily deploy any smart contracts they will use for either the public or private use cases. This will make it easier for the users to get started with SENTINEL, as they do not have to know the contract bytecode that is required for a deployment transaction.

Account management: Asset owners and vendors should be able to store Ethereum account information. This will enable a more streamlined process when they create transactions, as they do not need to copy and paste both the private key for each transaction. This allows asset owners and vendors to multiple accounts more easily.

Could have

The could-have requirements in Table 4.1 are not important for the system outside usability and user satisfaction improvements. Such requirements can be considered if the must-have and should-have requirements have been satisfied.

Support multiple storage systems & standards: SENTINEL could have support for multiple decentralized storage systems, security advisory formats, and SBOM formats. This would give more freedom of choice for the users that may prefer some standards or technologies over others. Furthermore, unlocking users from any specific technology will decrease the bias towards some standard or technology in the design and development.

4.4 Design Considerations & Assumptions

Several considerations and assumptions for the design process will be described in this section. These considerations are the foundation on which the system design and architectural decisions are based upon.

4.4.1 Use of Centralized Services

As SENTINEL is supposed to be in a decentralized environment to eliminate the reliance on trusted third parties, it is important that the design is made with such considerations. In the context of Web3, there are several third party services that can be used to ease integration with blockchain networks and decentralized storage systems. However, these must be avoided in the design and implementation of SENTINEL as these centralized services are subject to censorship and bureaucracy, which is exactly what the decentralization with Web3 is trying to eliminate. Moreover, these centralized organizations could introduce a single point of failure.

4.4.2 Operation Cost

In the design of SENTINEL, the operational cost is considered when choosing between different design choices. Operational cost in this context is the gas fees paid when making transactions to the Ethereum network when interacting with smart contracts. If the different design choices fulfill a requirement evenly, the design choice with the least cost will be chosen to SENTINEL, as it is more attractive for asset owners and vendors. However, in some cases where a design choice is cheaper but more complex, the lesser complex design choice is chosen, if they fulfill the same requirement.

4.4.3 Security Advisory Format

In this project, it is assumed that vendors will disclose vulnerabilities in the form of the Common Security Advisory Framework (CSAF) 2.0 specification [7]. This framework provides concise information in a machine-readable format, which is useful for automation. As such, SENTINEL can parse the security advisory information consistently and display that information to the asset owners.

The CSAF specification allows different product structure definitions. While this supports for a large variety of structures, it is difficult to support all possible structures. Therefore, SENTINEL will only support the product tree structure $vendor \rightarrow product\_name \rightarrow product\_version$. The supported structure can be seen in Listing 4.1. While this structure is restrictive, it is the structure recommended by the OASIS CSAF technical committee [7].

Listing 4.1: Example of the supported structure of CSAF product tree property.

```
"product_tree": {
  "branches": [
    {
      "category": "vendor",
      ...,
      "branches": [
        {
          "category": "product_name",
          ...,
          "branches": [
            {
              "category": "product_version",
              ...,
            }
            ...
```

4.4.4 Software Bill of Materials Format

In previous work, three SBOM formats were introduced, those being CycloneDX [8] (CDX), Software Package Data Exchange [9] (SPDX), and Software Identification (SWID) Tags [10]. Each format contains information about a piece of software, however, for the purpose of SENTINEL only the dependencies from SBOM documents are relevant, as they should be matched with vulnerable products in CSAF security advisories. In theory, all three formats are interoperable and all can be converted into an equivalent version in another format. However, CDX is chosen as the SBOM format for SENTINEL due to the fact that it explicitly conveys dependency trees, which are useful to depict transitive relationships. An example of how transitive dependencies are described in CDX SBOM documents is illustrated in Listing 4.2.

Listing 4.2: Example of how transitive dependencies are described in a CDX SBOM document.

```
"dependencies": [
    {
        "ref": "library-a",
        "dependsOn": [
            "library-b"
        ]
    },
    {
        "ref": "library-b",
```

```
10          "dependsOn": [
11              "library-c",
12              "library-d"
13          ]
14      }
15  ]
```

4.4.5 Product Identification Scheme

The key part of automation in the discovery of relevant security advisories is identification of products. Asset owners should be able to identify which software components they are dependent on, which can be provided from SBOM documents. Vendors should also be capable of identifying which of their products are vulnerable to a specific vulnerability, such that they can disseminate that information concisely. If both parties are able to uniquely identify the products in this manner, then it is possible to match a vendor's vulnerable products to the asset owner's dependencies. However, a major concern in matching products in this manner is that all parties must use the same, or interoperable, product identification scheme.

With the design assuming that security advisories follow CSAF, it is valuable to understand how the framework specifies product identification. There is no single standard which the product identification scheme should follow. This allows vendors to use their preferred schemes for product identification, but it is assumed that vendors will be consistent in their chosen identification scheme. Thereby, vendors are not restricted to a single identification scheme when using SENTINEL, however, they must be sure to use the same scheme for SBOM and security advisory documents.

4.4.6 Encryption

SENTINEL will in some cases be handling confidential data in the form of confidential security advisories. As such, proper precautions and procedures must be taken in order to ensure safe transmission of confidential security advisories from vendor to asset owner. In particular, the advisories should be encrypted to ensure that only the intended recipient can access and read it.

NIST maintains a large repository of guidelines and standards for the use of encryption and how to manage cryptographic keys, which are thoroughly reviewed. SENTINEL must comply with these guidelines where possible to ensure secure data transmission. [11]

4.5 Storage System Design

In SENTINEL, a decentralized storage system is used to store and share the security advisories between asset owners and vendors. This reduces the operation cost of using the system, as storing security advisories on Ethereum is more expensive than decentralized storage services. Choosing the right storage system is important, as it is how vendors relay vulnerability information to asset owners.

4.5.1 Decentralized Storage Systems

In previous work, several decentralized storage systems were investigated and are listed below.

- Hypercore
- Storj
- Sia
- Swarm
- Filecoin (IPFS)
- Arweave

Hypercore, recently renamed to Holepunch, is useful for sharing real-time data, large datasets and streams [12]. This is in contrast to the static nature of CSAF security advisories, which are often smaller files that do not need to be acquired in real-time. Therefore, the use case Holepunch is created for, is not compatible with SENTINEL.

Storj and Sia are similar in service structure and provide users with decentralized personal cloud storage. Both require payment to upload data to their services, which is not in itself problematic for SENTINEL. However, users are billed for the bandwidth they, and others who access their files, use. In a system where the security advisories are publicly available, this can be exploited to make vendors who upload security advisories pay massive bills for the bandwidth used to download their security advisories. Because of this potential exploit, Storj and Sia are not suitable for the public use case. This exploit is not as problematic if Storj or Sia is used in the private use case, but it will require some access control to make sure adversaries cannot access the information in the vendor's personal storage, which increases the complexity. [13, 14]

Swarm suffers from the opposite issue to Storj and Sia as only those who download files are billed for the bandwidth they use, and not the owner of the file. In

Swarm, asset owners need to pay a fee, if they use too much bandwidth when downloading files. This becomes an issue when the content of the security advisory is used to determine if the vulnerability is relevant for an asset owner or not, as they can potentially pay to download files, which they do not need. However, in the private use case, paying to download a security advisory is not an issue, as all security advisories announced are relevant for a specific asset owner. [15]

Filecoin also suffers from the same issue as Swarm, as users always have to pay to download files in that system. [16]

Even though Filecoin is not suitable, the system it is built on top of, is. The InterPlanetary File System (IPFS) allows free upload and download of files. The catch is that data is only available as long as the file is active, that is, the file is being stored on at least one node. Files which are not accessed for a while are removed from the cache and the network. This issue of availability is solved by locally *pinning* files on a node, which causes the node to retain the file regardless of activity. Vendors should use local pinning with their own IPFS node to ensure the availability of their security advisories. [17, 18]

Arweave, much like other system described in this section, requires an upfront fee for storing files on the Arweave network. However, uniquely to Arweave the files are stored permanently on the network after that fee. While this feature is not necessarily useful for SENTINEL, however, there are no other downsides that would make it unsuitable. [19]

	Holepunch	Storj	Sia	Filecoin	IPFS	Swarm	Arweave
Upload	✗	✓	✓	✓	✗	✓	✓
Download	✗	✗	✗	✓	✗	✗	✗
Bandwidth	✗	✓	✓	✗	✗	✓	✗

Table 4.2: Comparison of the decentralized storage services.

The decentralized storage systems introduced in this section are compared in relation to their payment structure in Table 4.2. The storage systems are compared by their payment requirements for uploading and downloading files, and bandwidth for accessing files.

4.5.2 Decentralized Storage System Choice

As this project is a prototype, the chosen decentralized storage is IPFS. It allows for broad experimentation without having any cost, as illustrated in Table 4.2, other than running an IPFS node and interacting with the network. In a production implementation of SENTINEL, it may be advantageous to use other decentralized

storage systems if their characteristics achieve more desirable features. For example, the increased privacy focus in Swarm could make it an attractive decentralized storage system for the private use case.

4.6 System Overview

In this section, an overview of the design of SENTINEL is presented. The architecture is presented first, which describes which components are present and how they are connected to other components and their domains. Specific component interactions in both the public and private use cases are also presented to demonstrate how the components are used.

4.6.1 System Architecture

The architecture of SENTINEL is illustrated in Figure 4.3. It consists of multiple components, spread out on three different platforms. Specifically, the system is composed of a local machine, the Ethereum network, and the IPFS network. The various components are described in detail in the remaining sections in this chapter.

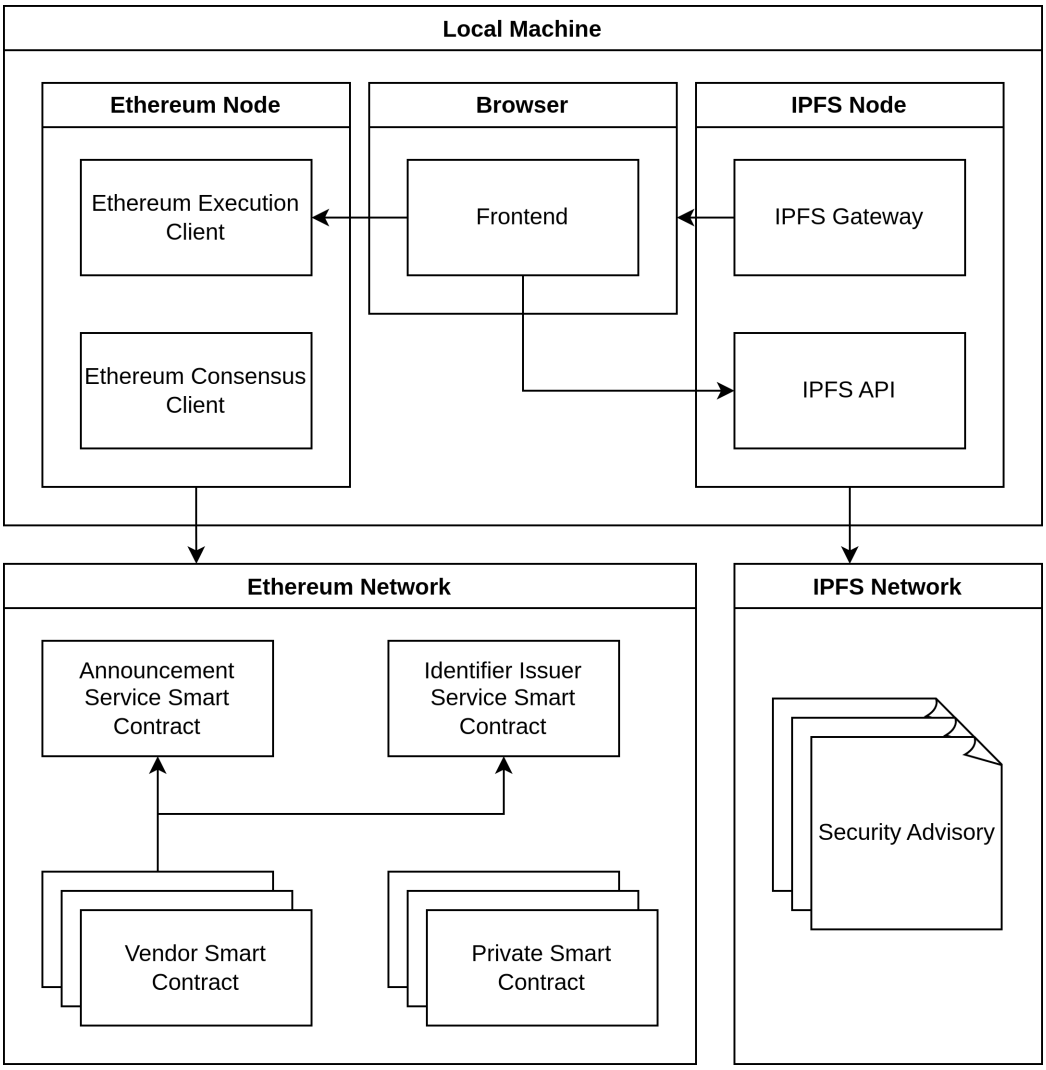

Figure 4.3: The architecture of SENTINEL.

SENTINEL makes use of several smart contracts to execute specific functionality in a decentralized manner. The design of the smart contracts are detailed in Section 4.7. For the public use case, the Announcement Service (AS) smart contract, Identifier Issuer Service (IIS) smart contract, and the Vendor Smart Contract (VSC) are used to announce security advisories. The private use case only makes use of the Private Smart Contract (PSC) to make confidential security advisory announcements.

The local machine component includes an IPFS node which is connected to the IPFS network. The node includes an API which is used by the frontend to interact with the IPFS network. While an in-browser IPFS node is available, due to

certain limitations this setup is not feasible for SENTINEL [20]. Consequently, the users of SENTINEL must have a standalone IPFS node running. Furthermore, as a standalone IPFS node is used it will not shut down when the SENTINEL frontend is not running, and thus continue to be connected to the IPFS network. [21]

The frontend component is the access point for asset owners and vendors to use SENTINEL. Design details about the frontend are introduced in Section 4.8. Functionality for both the public and private use case are implemented into the same frontend, which makes the frontend component more complex but also more complete. The files that make up the frontend component are uploaded to the IPFS network under an InterPlanetary Name System (IPNS) name, such that the frontend is stored in a decentralized manner, with the same IPNS name. Using an IPNS name to access the frontend ensures that the frontend can always be accessed from the same URL. Users use their local IPFS gateway to download the files and run the frontend in a local browser. While it is possible to access the frontend from public gateways, it should only be done with gateways that use subdomain resolution. This is because gateways that use path resolution, will store local storage under the gateway domain which could be exploited by cross-site-scripting. [22]

In order to connect to and read data on the blockchain, an Ethereum node must be present on the local machine. While it is possible to use third-party services for this purpose, it goes against the decentralized nature of SENTINEL, as explained in Section 4.4.1. Therefore, users must have a running, and up-to-date node on their local system. A full node on Ethereum consist of an execution and consensus client. The frontend only connects to the execution client, which stores the state of the blockchain. The consensus client is connected to the Ethereum network, where it receives blocks which are passed to the execution client. While light nodes for Ethereum exist which only holds parts of the blockchain, they are currently under development and therefore not considered viable for SENTINEL. [23]

4.6.2 Public Use Case Component Interactions

The public use case utilizes multiple components to announce new security advisories. An overview of how these components interact with each other in the public use case is illustrated in Figure 4.4.

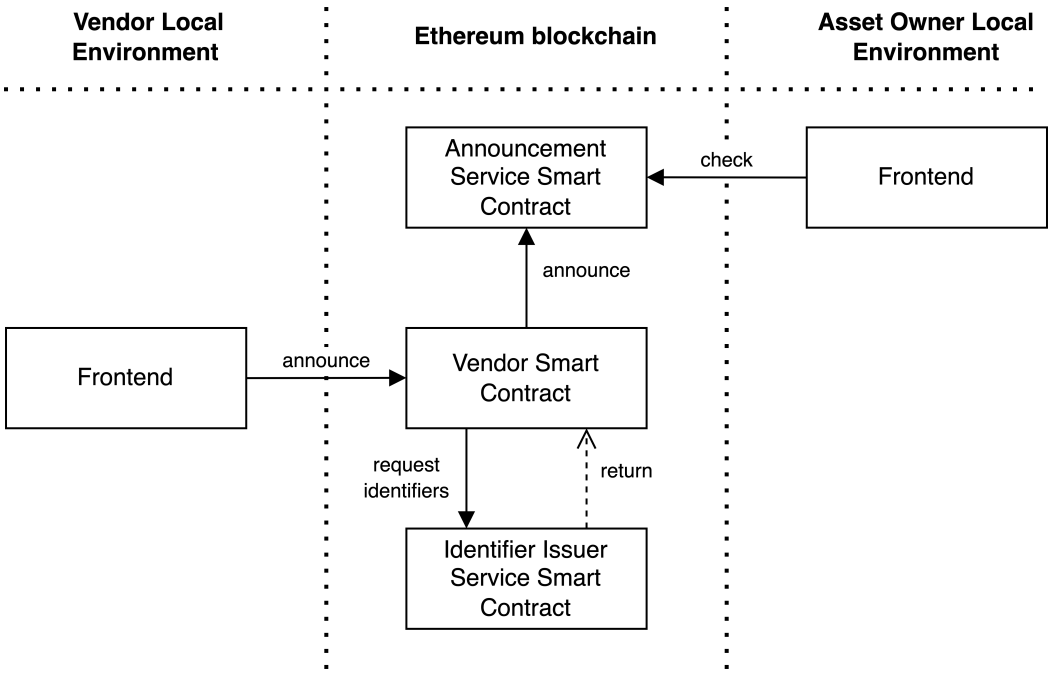

Figure 4.4: Overview of component interactions in the public use case.

First, the vendor announces a security advisory using their VSC. The VCS then requests identifiers for the security advisory from the IIS. Once the identifiers are returned, the VSC forwards the information to the AS, which will emit an event containing the security advisory information. The asset owner's frontend periodically checks for new security advisory announcements and will automatically retrieve the relevant security advisories.

4.6.3 Private Use Case Component Interactions

The private use case requires use of both the frontend component and PSC component in order to announce confidential security advisories. An overview of component interaction in the private use case is illustrated in Figure 4.5.

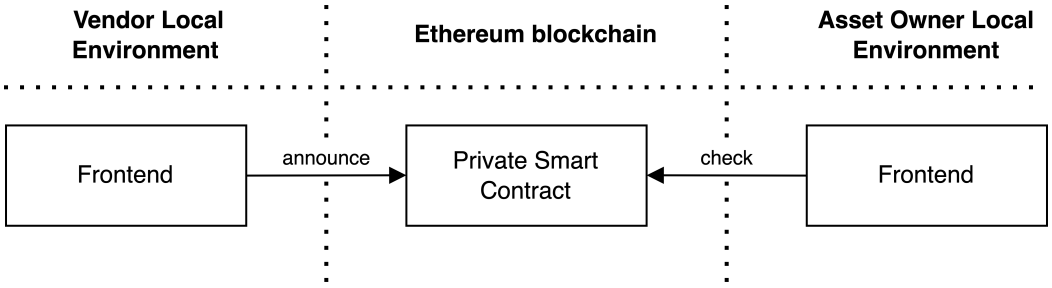

Figure 4.5: Overview of component interactions in the private use case.

The vendor announces a security advisory using the PSC, which is deployed to meet the agreement terms. Upon calling the PSC, an event containing the security advisory information is emitted to the blockchain. Similarly to the public use case, the frontend periodically checks the PSC for security advisory announcements.

4.7 Smart Contract Design

SENTINEL makes use of several smart contracts as a backend functionality, as shown in Section 4.6.1. The smart contracts each have their individual purpose and design, which is described in the following sections.

4.7.1 Announcement Service

The AS is responsible for announcing both new security advisories and updates to already announced advisories. The announcements are created by emitting events to the transaction log, which the frontend component automatically looks for. The service is a smart contract without any ownership to ensure that no organization or individual to controls the service. The design of the AS is illustrated in Figure 4.6.

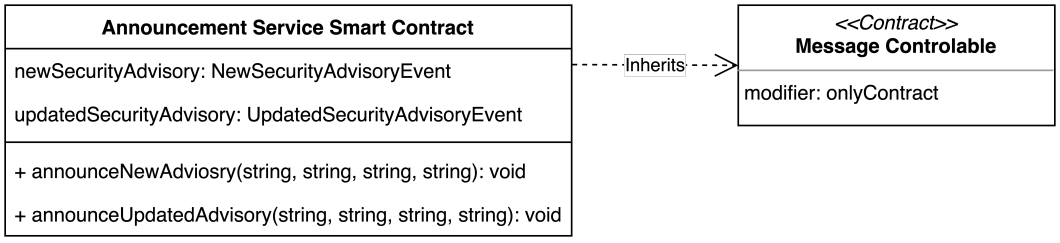

Figure 4.6: Design of the Announcement Service.

The *Message Controllable* contracts that the AS inherits contains a modifier used by methods in the AS smart contract. The modifier ensures that the exposed func-

tions are only called by a smart contract and not an externally owned account. This restriction allows SENTINEL to connect security advisory announcements with information from the announcing VSC.

The AS defines two different events, one for each type of announcement, that is, new and updated security advisory announcements. The event data structure for announcing new security advisories can be seen in Figure 4.7.

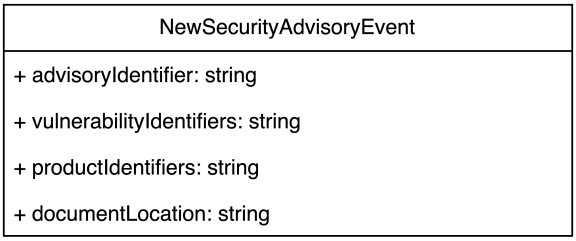

Figure 4.7: Data structure of the event that announces new security advisories.

Each security advisory announced in SENTINEL will have an associated *Advisory Identifier*, which is used to uniquely identify security advisories. Furthermore, it is also used to link updated security advisory announcements to the original security advisory announcement. The identifier is issued by the IIS, which will be described in Section 4.7.2.

The *Vulnerability Identifiers* property is a string with vulnerability identifiers for the vulnerabilities that are found in the associated security advisory. Vulnerability identifiers are issued by the IIS, similar to the advisory identifiers. The identifiers are comma-separated to reduce the character overhead and keep it compact.

Similarly to the *Vulnerability Identifiers* property, the *Product Identifiers* property is also a comma-separated string. However, this property contains the product identifiers of the products which are vulnerable to any of the vulnerabilities in the security advisory. The product identifiers specified in this string are used to identify which products are described as vulnerable in the security advisory, to indicate if an asset owner should be notified of the security advisory or not.

Lastly, the *Document Location* property is used to identify which storage system is used, and an identifier to download the security advisory, separated using front-slash. An example is *"ipfs/cid"*, where *cid* is the IPFS content identifier (CID), which is the identifier used to retrieve the file from the network.

The data structure of the event used to announce updates to security advisories is shown in Figure 4.8. The purpose of this event is to connect already announced security advisories to any updates to security advisories that may be announced later, such that vendors can stay up to date with any information regarding the vulnerabilities found in the security advisory.

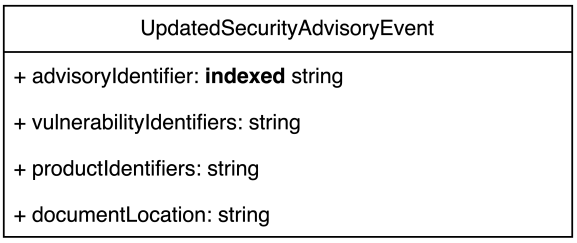

Figure 4.8: Data structure of the event that announces updates to security advisories.

The event data structure is almost identical to the new security advisory announcement data structure, but with a single important difference. The advisory identifier is `indexed`, which means that the property is hashed and can be efficiently filtered on, which makes it easier to filter for updates for only relevant security advisories that match the asset owner's dependencies. All other properties of the updated security advisory event should be used in the same manner as in the new security advisory event data structure.

The AS smart contract also includes two methods that can be called from other smart contracts, with one method for each announcement type. These methods will emit the events and write the data to the transaction logs. The functions will simply take in input data, which will then be emitted without any modifications or authentication.

4.7.2 Identifier Issuer Service

The IIS is the smart contract responsible for generating unique identifiers for advisories and vulnerabilities disclosed using SENTINEL. The identifiers that the IIS issues are similar to how a CVE ID is attributed to a CVE Record. However, the IIS can automatically issue the identifiers following the implementation of the smart contract, whereas CVE IDs have to be manually assigned. [24] The design of the IIS is illustrated in Figure 4.9. The IIS inherits the *MessageControllable* smart contract for the same reasons as for the AS.

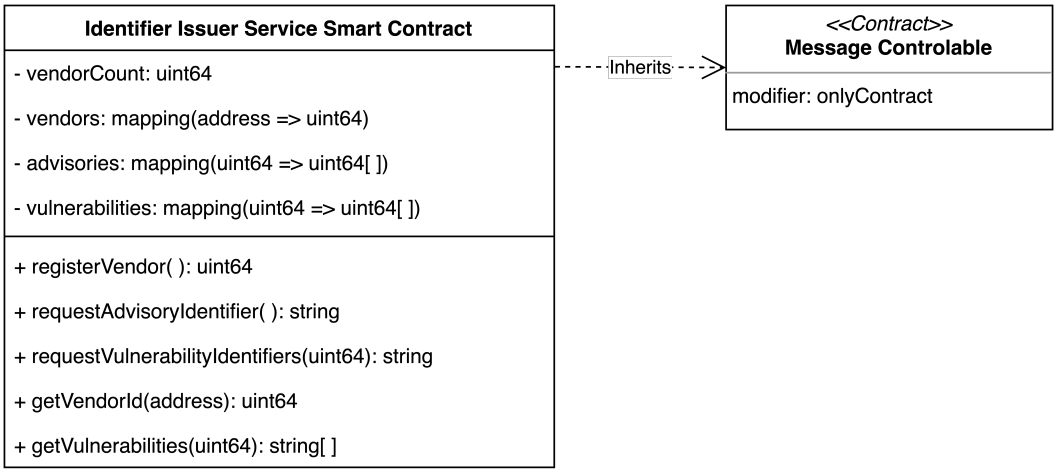

Figure 4.9: Design of the Identifier Issuer Service.

In order to keep track of the amount of vendors registered and to assign identifiers for new vendor registrations, the IIS uses an unsigned 64-bit integer, which allows for $2^{64} - 1$ different vendors to register to the IIS and use SENTINEL. While the size may be excessive for its intended use, it is large enough to facilitate registration for a long period of time. The same argument goes for the size of advisory and vulnerability identifier mappings.

Advisory and vulnerability identifiers issued by the IIS follows a strict format, which begins with the system identifier “SNTL”, which indicates that the identifier was issued by the IIS, similarly to how CVE IDs starts with “CVE”. The system identifier is then concatenated with either a “A” or “V” to indicate the identifier type for advisory and vulnerability identifiers, respectively. Vendor identifiers are related to the address of a smart contract, that requests a unique identifier, and are issued incrementally starting from 1. The last part of the identifier is the advisory or vulnerability number, which is an integer also issued incrementally from 1. All parts of the identifiers are separated with a hyphen. Examples of identifiers, and the different parts that make up the identifiers, are illustrated in Figure 4.10.

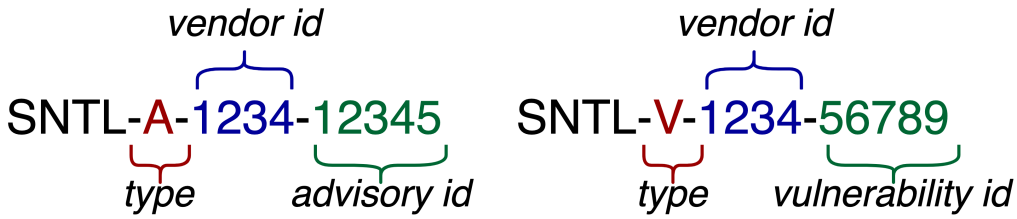

Figure 4.10: Visualization of how identifiers from the IIS are composed.

To facilitate the storage and tracking of the advisory and vulnerability identifiers, the IIS includes mappings of VSC addresses to their unique vendor identifier. Furthermore, mappings from that vendor identifier to the generated advisory and vulnerability identifiers are included. The IIS smart contract 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.

4.7.3 Vendor Smart Contract

VSC is the smart contract which vendors use to interact with SENTINEL in the public use case. Each vendor deploys and uses their own VSC, as the smart contract 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. The VSC smart contract is described in Figure 4.11.

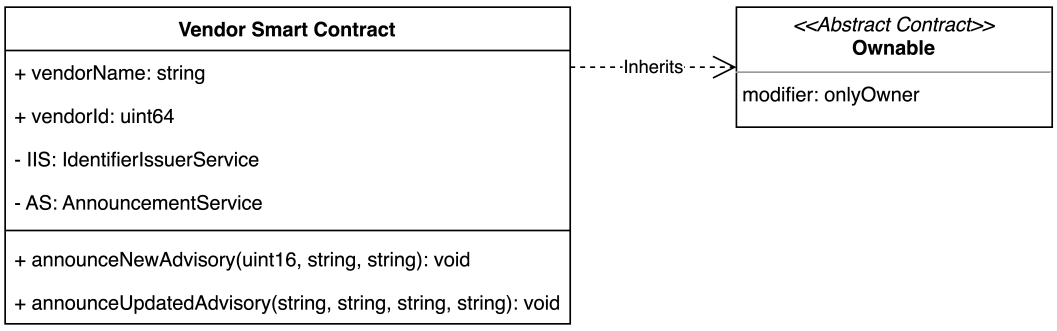

Figure 4.11: Design of the Vendor Smart Contract.

In order to control who can interact with any VSC, an *Ownable* contract is inherited, which can be used to control who has access to methods on a smart

contract with the *onlyOwner* modifier. This assures that only the owner of the VSC can interact with certain methods on the VSC.

The VSC contains four state variables, which are set in the constructor during the creation of the smart contract. Firstly, the vendor provides a name to a *vendorName* state variable, which is used to map from an address to a name when asset owners are accessing security advisories. The VSC also contains instance state variables for the AS and IIS, which are immutable, as the VSC should always use the same services via the same addresses. Lastly, a property for the vendor identifier from the IIS is implemented. All the state variables are assigned in the contract constructor, such that the VSC is ready for use after creation.

In the public use case, there are two different types of announcements. Therefore, the VCS includes methods to accommodate both types of announcements. The methods call the AS to emit events with the input data given from the VSC. For announcing new security advisories, the VCS 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, as there is no need to generate new identifiers.

4.7.4 Private Smart Contract

The PSC is the smart contract on which asset owners receive confidential security advisories from a vendor in the private use case.

The PSC contains essential methods and properties for ensuring disclosure of confidential security advisories. The interface of PSC is illustrated in Figure 4.12.

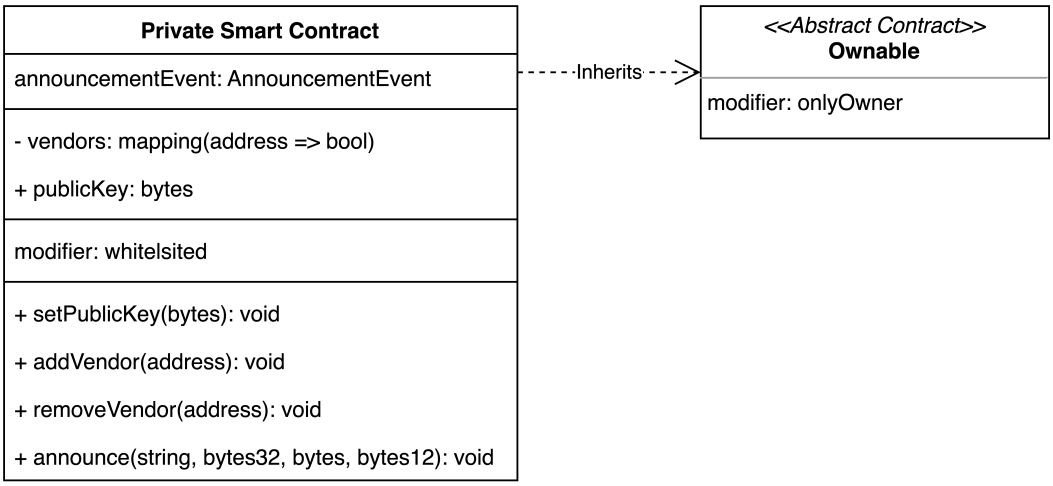

Figure 4.12: Design of the Private Smart Contract.

As with the VSC described in Section 4.7.3, the PSC inherits the *Ownable* contract in order to implement access control. Most of the methods in the PSC are restricted with the *onlyOwner* access modifier to enforce access control.

In addition to the *onlyOwner* access modifier, the PSC contains a custom modifier named *whitelisted*, which restricts access to allow only whitelisted Ethereum addresses. This enables an asset owner to allow only specific whitelisted vendors to announce security advisories on their PSC. The whitelist 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 methods are protected by the *onlyOwner* modifier, such that only the owner of the PSC can whitelist addresses. Each method requires a valid Ethereum address as input.

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. This state variable, called *publicKey* contains a public encryption key used for announcing confidential security advisories. An accompanying method, *setPublicKey* is used to set the *publicKey* state variable. It requires a string of bytes as input. More details on the use of the public key can be found in Section 4.7.5.

To announce a confidential security advisory, whitelisted vendors can call the *announce* method. This method announces an *AnnouncementEvent* which is illustrated in Figure 4.13. This event contains four values used for secure announcements. Firstly, a CID is required to inform the asset owner of the location of the security advisory. Secondly, a cryptographic hash is included to ensure non-repudiation, in the event that either the security advisory is inaccessible. By having a backup of the security advisory, the vendor can show that the cryptographic hash of this security advisory matches the one emitted in the event. Finally, an encryption key and an initialization vector (IV) is included, which allows the asset owner to decrypt the security advisory. The use of the encryption key and initialization vector is described in Section 4.7.5.

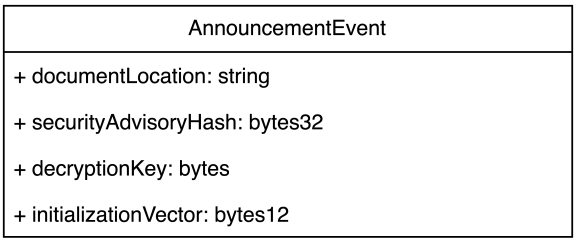

Figure 4.13: Data structure of the event announcing a confidential security advisory.

4.7.5 Security of Confidential Announcements

As the disclosure of private vulnerabilities requires transmission of highly confidential security advisories via the public networks of Ethereum and IPFS, it is pertinent to incorporate security measures to accommodate this in the design.

As mentioned in Section 4.4.6 the design of SENTINEL follows NIST security guidelines where possible. Currently, there are two NIST approved block cipher algorithms: Triple DES and AES. However, Triple DES will be deprecated after 2023 and may only be used when interacting with legacy systems. As such, SENTINEL utilizes AES encryption to encrypt confidential security advisories. [25, 26]

AES is a symmetric key block cipher technique for of encrypting data blocks of 128 bits using key lengths of 128, 192, and 256 bits. AES must be used in a NIST approved operation mode, that is, an algorithm implementing a symmetric block cipher technique. Multiple modes exist, however, SENTINEL is designed with *Galois/Counter Mode* AES-GCM). AES-GCM is an authenticated mode, which means that it protects against *chosen-ciphertext* attacks in which an attacker decrypts arbitrary messages and deduce the key from the result. Of the five currently approved authenticated modes, AES-GCM was chosen as it is already implemented in most modern browsers and can be utilized using the Subtle Crypto library included in JavaScript. [25, 27, 28, 29]

AES-GCM utilizes a randomly generated string of bits called an *initialization vector* to initialize the encryption functions. It is essential that the IV is unique for each message, as a reused IV will compromise the implementation. SENTINEL uses an IV of 96 bits, as recommended by NIST. [30]

In symmetric encryption algorithms, such as AES-GCM, a single private key is utilized for both encryption and decryption. Therefore, it is of utmost importance that only the intended parties are in possession of the key. To ensure that only the intended parties receive the encryption key, it should be distributed using NIST approved key establishment methods. [31]

One such method is a key-transport scheme in which two parties, a recipi-

ent and a sender, establish a key chosen by the sender. A NIST recommended key-transport scheme is KTS-OAEP which utilizes the asymmetric encryption algorithm RSA-OAEP to keep the key encrypted when it is transported. KTS-OAEP is illustrated in Figure 4.14. [32]

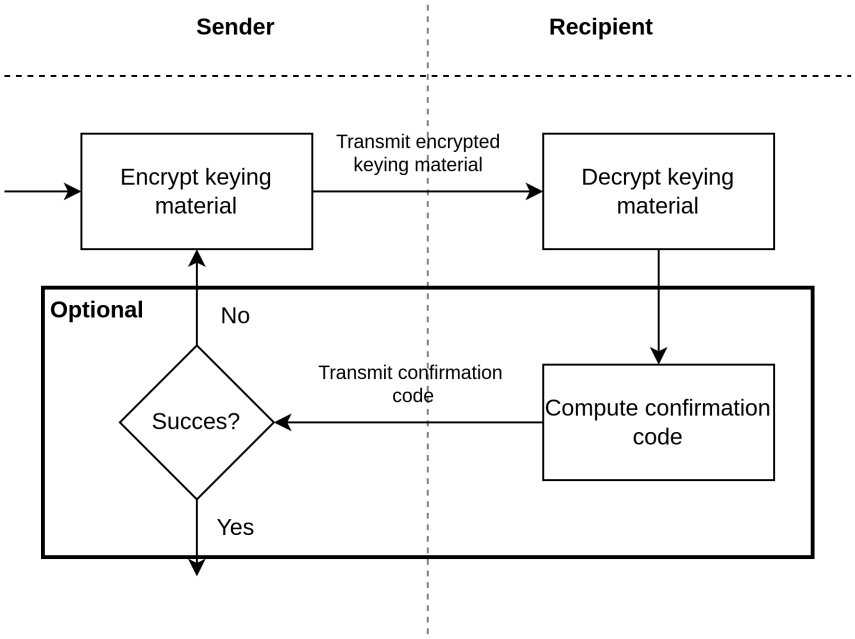

Figure 4.14: The KTS-OAEP key transport scheme.

The KTS-OAEP procedure has few steps. First, the sender encrypts the keying material, that is, the key to be transported, using RSA-OAEP encryption with the recipient's public key. The now encrypted keying material is then transmitted to the recipient, who will decrypt the keying material using RSA-OAEP with their private key. It is possible to add an optional verification step after decryption to notify the sender if the key has been transmitted successfully. [32]

SENTINEL utilizes KTS-OAEP when transporting the AES-GCM keys used to encrypt confidential security advisories. KTS-OAEP was chosen as it is a NIST recommended algorithm with few stages and because Subtle Crypto, and thus most modern browsers, already include an implementation of RSA-OAEP. This makes room for fewer mistakes and vulnerabilities in the implementation. [29]

With the algorithms for secure transmission of confidential security advisories in place, the key sizes are determined. Depending on the security strength, the key sizes can differ significantly. Security strength is a number, usually in the form of n-bits, describing the amount of work it will take to break a cryptographic algorithm. Security strengths of 112 bits and up are approved by NIST, and they

specify that 128, 192, and 256 bits security strengths are secure through 2031 and beyond. [31]

SENTINEL uses the Ethereum blockchain, on which minimizing on-chain storage cost is a priority. As such, AES-GCM using 128-bit keys, corresponding to 128 bits security, is used. As they are both approved and secure through 2031 and beyond, and relatively small, it is an ideal choice of key size. [31]

According to NIST, the security strength of a system is that of the algorithm with the weakest security strength. Following that guideline, the RSA-OAEP keys used in the KTS-OAEP procedure should have at least 128 bits of security strength. This equates to RSA-OAEP keys that are 3072 bits long. [31]

Having determined the algorithms and procedures for securely transmitting confidential security advisories, the entire data flow can now be established. Figure 4.15 illustrates the flow. First the asset owner, as the recipient, generates an RSA-OAEP key pair and publishes the public key to their PSC. Afterward, the vendor, as the sender, can fetch the public key for future key transport. When the vendor intends to announce a confidential security advisory, generate an AES-GCM key and a random IV, which they then use to encrypt the security advisory and upload to IPFS. Following this, the vendor encrypts the AES-GCM key using RSA-OAEP with the key retrieved from the asset owner's PSC. They can then announce an event to the smart contract including the encrypted AES-GCM key, the IV, and other necessary information. Finally, the asset owner, upon discovering the newly emitted event, decrypts the AES-GCM key using their RSA-OAEP private key and in turn uses the AES-GCM key to decrypt the security advisory.

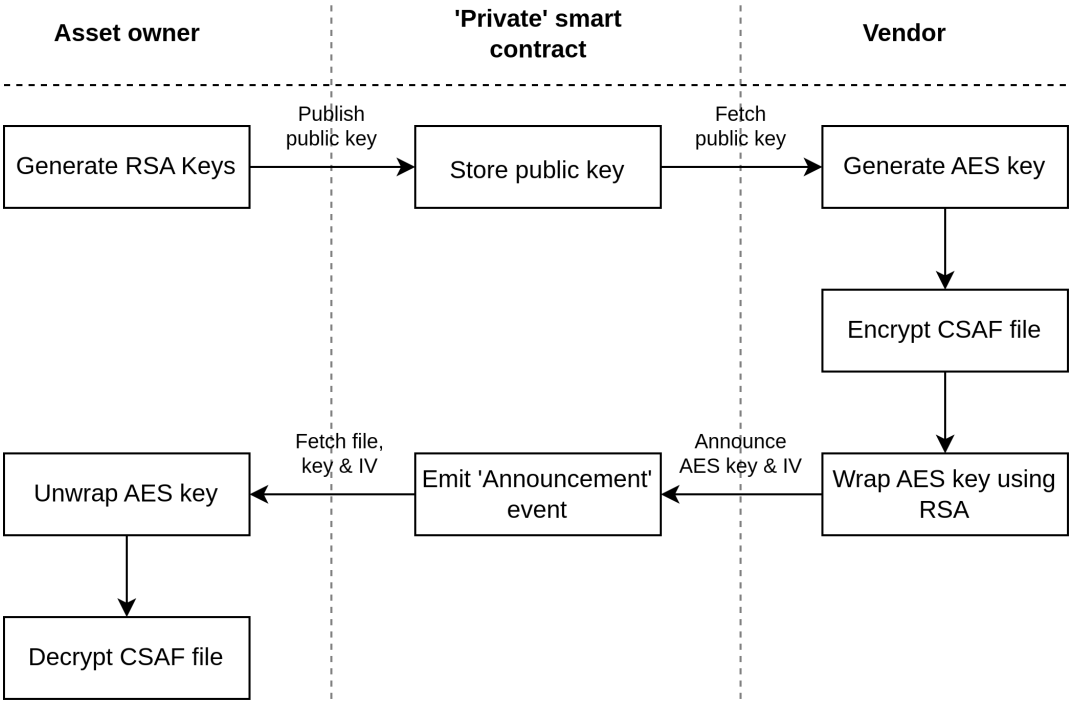

Figure 4.15: The data flow of the private use case, including encryption procedures.

When handling cryptographic keys, several assurances and protection requirements should be upheld in the system using the keys: [31]

- **Confidentiality Protection**: When the private key information is available outside a cryptographic module, confidentiality protection is provided by encryption at an appropriate security strength or by controlling access to the key via physical means.
- **Integrity Protection**: When a key is available outside a cryptographic module, integrity protection is provided by appropriate cryptographic integrity mechanisms, non-cryptographic integrity mechanisms, or physical protection mechanisms.
- **Association Protection**: Association protection is provided for a cryptographic security service by ensuring that the correct keying material is used to protect the correct data in the correct application or equipment.
- **Availability Protection**: Availability protection is provided for all key information that needs to be available beyond its immediate use for protecting data.
- **Assurance of Domain-Parameter and Public Key Validity**: This assurance

provides confidence that the parameters and keys used with cryptographic algorithms are arithmetically correct.

- **Assurance of Private Key Possession**: This assurance provides assurance that the owner of a public key actually possesses the corresponding private key.

Only the confidentiality-, integrity-, association-, and availability-protection are strictly required by NIST. As such, the two remaining assurances will not be prioritized in the prototype of SENTINEL.

4.8 Frontend Design

The frontend is the component which the asset owners and vendors use to interact with SENTINEL. It contains several important interactive interfaces which ease the interactions with the smart contracts and facilitate other functionality such as encryption.

4.8.1 Settings Storage

The asset owners and vendors must provide settings, such as dependencies, to the frontend for it to work properly. These settings are stored in the browsers local storage, where they can be saved and loaded between running sessions of the SENTINEL frontend in a manner that does not require any external database systems. The stored information is listed below.

- Software dependencies
- Ethereum accounts
- Confidential agreements
- Vendor whitelist

As the frontend stores sensitive data, such as Ethereum private keys and RSA-OAEP private keys, in local storage, password encryption is employed to ensure the confidentiality of this data.

When implementing password encryption, multiple requirements must be met to ensure that the password encryption provides the necessary security, according to NIST. As this project describes a prototype, only the strictly necessary requirements, which are listed below, will be considered at this time. [33]

1. Passwords must be at least 8 characters in length.

2. Passwords that are randomly generated by the service must be at least 6 characters in length and use a random number generator recommended by NIST.
3. The password must not be truncated before hashing.
4. No password hints must be available.
5. There must be no composition rules on the password.
6. The password must be compared to compromised passwords, dictionary words, repetitive or sequential characters, and context specific words. If the password matches any of these strings, the password cannot be used.
7. There must be a rate limit for password authentication.
8. Encryption and protected channels must be used to avoid Man-in-the-Middle attacks.
9. A NIST recommended password derivation algorithm must be utilized to protect against offline attacks.

There are multiple password derivation algorithms. The one SENTINEL uses is PBKDF2. PBKDF2 applies a pseudo-random function, such as a cryptographic hashing function, along with a salt, many times in order to derive a string of bytes. These bytes can be used as a cryptographic key for various algorithms. [34]

4.8.2 Vulnerabilities Page

Most importantly, the frontend must contain a page where asset owners can get an overview of the vulnerabilities which could affect their system. A mockup of this page is illustrated in Appendix A.1.

The page consists of a list of vulnerabilities for the asset owner to get a quick overview of the current vulnerabilities with some information about them. Asset owners can select a security advisory announcement to read the security advisory data parsed from CSAF to a human-readable format. This page also contains the updates for specific security advisories and allow asset owners to see all versions of a security advisory relevant to their dependencies.

4.8.3 Accounts Page

The frontend contains a page where asset owners and vendors can store their accounts to be used to create transactions easily. The accounts page is illustrated in Appendix A.2.

Asset owners and vendors can save accounts by filling out the form, which encrypts and saves the account in local storage. All saved accounts are displayed in a table, but without the sensitive private key account information. The saved accounts are available to be used on other pages for creating transactions.

4.8.4 Settings Pages

There are two different settings pages on the frontend, one for each use case. The public settings page, which is for the public use case, is used by the asset owner to upload their dependencies from a CDX SBOM document, and to whitelist VSC addresses to allow security advisory announcements announced with the specified VSC. The mockup for the public settings page is illustrated in Appendix A.3.

The private settings page is used by asset owners to manage the PSC. On this page they can save and see their deployed PSCs, specify which vendors can interact with the smart contract, and update the public key used for encryption of security advisory. Any announcements made to saved PSCs will be displayed on the vulnerability page. A mockup of the private settings page is illustrated in Appendix A.3.

4.8.5 Announcement Pages

The announcement pages are used by vendors to announce security advisories to the public or to a specific vendor. To announce new and updated security advisories to the public, the public announcement page is used, of which a mockup is illustrated in Appendix A.4. The vendor can fill out a form to announce a new security advisory or announce updates to a previously announced security advisory. Vendors can use an account they have set up on the accounts page to create the transaction which calls the announcement method on the specified VSC.

To announce confidential security advisories, the private announcement page is used. Vendors can use this page the same way as the public announcements page, but with the address of a PSC instead of their VSC. A mockup of the private announcements page is illustrated in Appendix A.4.

Chapter 5

Implementation

This chapter will cover the implementation of the SENTINEL prototype. The implementation follows the design from Chapter 4. Additionally, the development environment, integration of external components, and automation pipelines are described.

This chapter includes the following sections.

- Development Environment
- Announcement Service Implementation
- Identifier Issuer Service Implementation
- Vendor Smart Contract Implementation
- Private Smart Contract Implementation
- IPFS & Ethereum Nodes
- Frontend Implementation
- Continuous Integration & Deployment

5.1 Development Environment

SENTINEL consists of two Node.js [35] projects, one for the smart contracts, and one for the frontend. The latest LTS version of Node.js was used for both projects.

For smart contact development, Truffle Suite [36] was utilized to compile, deploy, and test smart contracts on a simulated Ethereum network, enabling fast and practical testing. Furthermore, the Truffle Console allows interaction with deployed smart contracts using built-in libraries.

For frontend development, the React.js [37] framework was used to build and prototype the frontend. Additionally, IPFS Kubo [38], an IPFS client, was instrumental in the implementation and testing of upload and download of security advisories as it allowed access to the IPFS network.

5.1.1 Smart Contract Development

While the immutability of Ethereum provides advantages in some situations, such as non-repudiation, it also creates disadvantages in others. Smart contracts being immutable means that any bugs or vulnerabilities found in deployed code, usually will be hard or impossible to patch.

As such, development and security guidelines from both the Ethereum docs [39] and ConsenSys [40, 41] were followed in order to minimize possible vulnerabilities in SENTINEL's smart contracts. Furthermore, all smart contracts are written in Solidity v0.8.18

5.2 Announcement Service Implementation

As specified in Section 4.7.1, the purpose of the Announcement Service is to emit events that announce security advisories. The service supports both announcements of new security advisories and updates to already announced security advisories.

5.2.1 Announcement Events

The AS specifies two different events to support the announcements of new and updated security advisories. The events implement different data according to the different purposes which they are responsible for.

In Listing 5.1 the event implementation for announcing new security advisories can be seen. Four string properties are used to describe new security advisories, as they are explained in Section 4.7.1

Listing 5.1: New security advisory announcement event implementation in Solidity.

```
event NewSecurityAdvisory(
    string advisoryIdentifier,
    string vulnerabilityIdentifiers,
    string productIdentifiers,
    string documentLocation
);
```

The event representing the updated security advisory announcements can be seen in Listing 5.2. The main difference is the inclusion of the `indexed` keyword to the `advisoryIdentifier` property, which is used to increase search efficiency of events based on the value of that property.

Listing 5.2: Updated security advisory announcement event implementation in Solidity.

```
event UpdatedSecurityAdvisory(
    string indexed advisoryIdentifier,
    string vulnerabilityIdentifiers,
    string productIdentifiers,
    string documentLocation
);
```

5.2.2 Announcement Methods

Two different methods are implemented in the Announcement Service smart contract to facilitate announcements of the two different event types. The methods can be seen in Listing 5.3.

Listing 5.3: Methods in the Announcement Service to announce new and updated security advisories.

```
function announceNewAdvisory(
    string memory advisoryIdentifier,
    string memory vulnerabilityIdentifiers,
    string memory productIdentifiers,
    string memory documentLocation) external onlyContract
{
  emit NewSecurityAdvisory(advisoryIdentifier,
   vulnerabilityIdentifiers, productIdentifiers,
   documentLocation);
}

function announceUpdatedAdvisory(
    string memory advisoryIdentifier,
```

```
12      string memory vulnerabilityIdentifiers,
13      string memory productIdentifiers,
14      string memory documentLocation) external onlyContract
15  {
16    emit UpdatedSecurityAdvisory(advisoryIdentifier,
        vulnerabilityIdentifiers, productIdentifiers,
        documentLocation);
17  }
```

Vendors can only call these methods from a smart contract, as the address of the VSC is used to identify what vendor has initiated the transaction and not from externally owned accounts. To achieve this, the methods implement a modifier called `onlyContract` which can restrict access to methods. The modifier is specified in the `MessageControllable` contract, which is inherited by the AS. The implementation of the modifier can be seen in Listing 5.4.

Listing 5.4: The `MessageControlable` smart contract implementation.

```
1 contract MessageControllable {
2     modifier onlyContract() {
3         require(msg.sender != tx.origin, "Call only
      accessible from smart contract");
4         _;
5     }
6 }
```

Smart contracts cannot create transactions, however, they can create messages to interact with other smart contracts. The `onlyContract` modifier does a simple check to verify whether the transaction and the message are originating from the same address.

5.3 Identifier Issuer Service Implementation

The purpose of the IIS, as explained in Section 4.7.2, is to assign unique identifiers to vendor smart contracts, security advisory announcements, and vulnerabilities. Similar to the AS, the IIS inherits the `MessageControllable` contract, which was previously described in Section 5.2.2.

5.3.1 Identifier Issuer Service State Variables

The IIS smart contract contains four `private` state variables to keep track of all identifiers issued by the service, and to help generate new identifiers when requested. These state variables can be seen in Listing 5.5.

Listing 5.5: State variables of the Identifier Issuer Service.

```
1 uint64 private vendorCount = 0;
2 mapping(address => uint64) private vendors;
3 mapping(uint64 => uint64[]) private advisories;
4 mapping(uint64 => uint64[]) private vulnerabilites;
```

The `vendorCount` state variable is a simple count of how many vendors that have registered to the IIS. It is also used to assign a unique unsigned integer identifier to vendors that register to the service.

The `vendors` state variable is a `mapping` to keep track of the identifiers issued for an `address` of a VSC. The `uint64` identifier assigned to a vendor is used in the `advisories` and `vulnerabilities` state variables to keep track of advisory and vulnerability identifiers generated for the specific vendor.

5.3.2 Register Vendor

Vendors must be registered on the IIS before any security advisories or vulnerability identifiers can be issued for the vendor. In Listing 5.6, the `registerVendor` method can be seen, which is used to assign an identifier to a vendor smart contract.

Listing 5.6: The method in the Identifier Issuer Service that registers new vendors to the service.

```
1 function registerVendor() external onlyContract returns (
    uint64) {
2     require(vendors[msg.sender] == 0, "Vendors can only
    register once"); // Vendor is not registered already
3     vendorCount++;
4     vendors[msg.sender] = vendorCount;
5     return vendorCount;
6 }
```

The method has the `external` and `onlyContract` modifiers, which restricts any callers of the methods only be smart contracts other than the IIS itself. On line 2 is a guard that checks if the `vendors` mapping already contains the address of the message sender, which is the smart contract address. If the vendor is already registered, the transaction will revert. Otherwise, the `vendorCount` is incremented by one and the vendor address is added to the `vendors` mapping with the new current value of the vendor count.

5.3.3 Request Advisory Identifier

The IIS issues advisory identifiers for any new security advisory announcement that the AS announces. However, it is vendor's responsibility to request the ad-

visory identifier before announcing with the AS. The method on the IIS, which generate such an identifier, can be seen in Listing 5.7.

Listing 5.7: The method in the Identifier Issuer Service that generates advisory identifiers.

```
1 function requestAdvisoryIdentifier() external onlyContract
    returns (string memory) {
2     require(vendors[msg.sender] != 0, "Vendor must be
    registered");
3     uint64 vendorId = vendors[msg.sender];
4
5     advisories[vendorId].push(uint64(advisories[vendorId].
    length + 1));
6     uint64 advisoryNumber = uint64(advisories[vendorId].
    length);
7
8     return string.concat("SNTL-A-", Strings.toString(vendorId
    ), "-", Strings.toString(advisoryNumber));
9 }
```

As in the `registerVendor` method, the `requestAdvisoryIdentifier` method also has the `external` and `onlyContract` modifiers. The method starts with a guard checking that the vendor already has been registered, as the vendor identifier is used as a part of the advisory identifier. Next, to generate the new advisory identifier, the `advisories` state variable entry for the specific vendor identifier is appended with the next advisory number. This means that, if a vendor already has had ten advisory identifiers generated, "11" would be appended to the mapping, and be used as the advisory number for the advisory identifier. The complete advisory identifier is created with a concatenation of strings of the `vendorId` and `advisoryNumber` with hyphens. This is accomplished with the built-in `string.concat` function from Solidity, and with `Strings.toString` function from OpenZeppelin Strings utility library [42].

5.3.4 Request Vulnerability Identifiers

The final responsibility of the IIS is to generate unique identifiers for vulnerabilities. As there can be multiple vulnerabilities in a single security advisory, the method in the IIS responsible for generating these identifiers should therefore also support this functionality. The implementation of this functionality can be seen in Listing 5.8.

Listing 5.8: The method in the Identifier Issuer Service that generates vulnerability identifiers.

```
1 function requestVulnerabilityIdentifiers(uint16 count)
    external onlyContract returns (string memory) {
```

```
2     require(vendors[msg.sender] != 0, "Vendor must be
      registered");
3     uint64 vendorId = vendors[msg.sender];
4     string memory ids = "";
5
6     for (uint16 index = 0; index < count; index++) {
7         vulnerabilites[vendorId].push(uint64(vulnerabilites[
      vendorId].length + 1));
8         uint64 vulnerabilityNumber = uint64(vulnerabilites[
      vendorId].length);
9         string memory id = string.concat("SNTL-V-", Strings.
      toString(vendorId), "-", Strings.toString(
      vulnerabilityNumber));
10        if (index != 0)
11            ids = string.concat(ids, ",", id);
12        else
13            ids = string.concat(ids, id);
14    }
15
16    return ids;
17 }
```

The `requestVulnerabilityIdentifiers` works similarly to the method for requesting advisory identifiers. However, to facilitate generation of multiple identifiers in a single call, there are some key changes. Firstly, the method takes an `uint16` variable as input which specifies how many identifiers to generate, up to $2^{16} - 1$. Secondly, the for-loop from line 6 to 14, generates the vulnerability identifiers one at a time, and concatenates the identifier to the `ids` string variable declared on line 4, which is returned once the correct amount of identifiers has been generated.

5.4 Vendor Smart Contract Implementation

As detailed in Section 4.7.3, the VSC is the way for vendors to interact with AS and IIS in SENTINEL for the public use case. It contains predefined state variables and methods which enable correct and valid transactions that announce new and updated security advisories.

5.4.1 Access Control

In order to control who can interact with any VSC, the `Ownable` abstract contract is inherited. The functionality of the inherited contract is used to control who has ac-

cess to methods on a smart contract with an `owner` state variable. It contains methods to update the owner and modifiers to check if the transaction creator has rights to interact with the smart contract methods. The *Ownable* contract implementation is from a collection of open source smart contracts from OpenZeppelin [43] which provides smart contracts for a large variety of purposes. Many of these smart contracts have been trialed and tested by many developers and are implemented in smart contracts on the Ethereum Mainnet.

5.4.2 Vendor Smart Contract State Initialization

In the VCS implementation, there are four state variables, each of these are assigned in the constructor. The definitions of these state variables can be seen in Listing 5.9 and the constructor can be seen in Listing 5.10.

Listing 5.9: State variables in the Vendor smart contract.

```
1 string public vendorName;
2 IdentifierIssuerService private _IIS;
3 AnnouncementService private _AS;
4 uint64 public vendorId;
```

The `vendorName` enables vendors to provide a readable name, which will be displayed on the frontend instead of the smart contract address, to make it easier for asset owners to identify who a security advisory announcement is from. The `_IIS` and `_AS` state variables are contract instances that the VSC uses to send transactions to the IIS and AS smart contracts, respectively. These instance state variables are set in the constructor from the `address` constructor input arguments. Lastly, the `vendorId` is assigned with the identifier returned from the `registerVendor` method from the IIS on line 5 in Listing 5.10.

Listing 5.10: The constructor of the Vendor Smart Contract.

```
1 constructor(string memory name, address asAddress, address
    iisAddress) {
2     vendorName = name;
3     _AS = AnnouncementService(asAddress);
4     _IIS = IdentifierIssuerService(iisAddress);
5     vendorId = _IIS.registerVendor();
6 }
```

5.4.3 New Security Advisory Announcement

To announce new security advisories, the VSC has a method that takes in the necessary arguments and calls the IIS to get identifiers for the announcement. The

implementation of this method can be seen in Listing 5.11.

Listing 5.11: Vendor Smart Contract method to announce new security advisories.

```
function announceNewAdvisory(
    uint16 count,
    string memory productIds,
    string memory location) onlyOwner public {
    _AS.announceNewAdvisory(getAdvisoryId(),
    getVulnerabilityIds(count), productIds, location);
}
```

The announcement method is restricted by the `onlyOwner` modifier, such that only the vendor who owns the smart contract can make announcements from the smart contract and malicious actors cannot impersonate the vendor. The `count` input argument is used to specify how many vulnerability identifiers the IIS should generate for the announcement. The `productId` input argument is a string of product identifiers separated by a comma, that indicates which products are impacted by the vulnerabilities in the security advisory. Lastly, the `location` argument is the IPFS CID of the announced security advisory. Before invoking the announcement method on the AS, the methods to generate advisory and vulnerability identifiers are called.

The VSC implements helper methods that call the IIS and can be seen in Listing 5.12.

Listing 5.12: Helper functions for announcing new security advisories from the Vendor Smart Contract.

```
function getVulnerabilityIds(uint16 count) onlyOwner public
    returns (string memory) {
    require(vendorId != 0, "Function only available with a
    vendor id");
    return _IIS.requestVulnerabilityIdentifiers(count);
}

function getAdvisoryId() onlyOwner public returns (string
    memory) {
    require(vendorId != 0, "Function only available with a
    vendor id");
    return _IIS.requestAdvisoryIdentifier();
}
```

The identifier helper methods both require that the `vendorId` state variable has been assigned a value other than 0, which is fulfilled if the constructor has been executed successfully. The guards ensure that the IIS can generate identifiers for the VSC and return the generated identifier.

5.4.4 Updated Security Advisory Announcement

The VSC implements a method to announce updated security advisories as well. This method takes several arguments as input and calls the AS directly without any identifier generation from the IIS. The method can be seen in Listing 5.13.

Listing 5.13: Vendor Smart Contract method to announce updated security advisories.

```
function announceUpdatedAdvisory(
    string memory advisoryId,
    string memory vulnerabilityIds,
    string memory productId,
    string memory location) onlyOwner public {
    _AS.announceUpdatedAdvisory(advisoryId, vulnerabilityIds,
    productId, location);
}
```

Similarly to the `announceNewAdvisory` method, the `announceUpdatedAdvisory` method restricts access with the `onlyOnwer` modifier. The `advisoryId` and `vulnerabilityIds` are given as input to this method as the security advisory, to be updated, already exists. The `productId` and `location` arguments are handled identically as in the method that announces new security advisories.

5.5 Private Smart Contract Implementation

The PSC is the smart contract that assists an asset owner and a vendor in the announcement of confidential security advisories in the private use case. The implementation of this smart contract follows the design in Section 4.7.4.

5.5.1 Vendor Whitelisting

The PSC implements a whitelist to creation of announcements strictly to whitelisted addresses. To implement this, the PSC contains a state variable to store the whitelist and methods to manage the whitelist, which can be seen in Listing 5.14.

Listing 5.14: Whitelist state variable and management methods on the Private Smart Contract.

```
mapping(address => bool) private vendors;

function addVendor(address vendor) public onlyOwner {
    require(vendor != address(0), "Address 0 is not
    whitelistable");
    require(!vendors[vendor], "Address is already whitelisted
    ");
    vendors[vendor] = true;
```

```
7 }
8
9 function removeVendor(address vendor) public onlyOwner {
10     require(vendor != address(0), "Address 0 is not
      whitelistable");
11     require(vendors[vendor], "Address is not whitelisted");
12     vendors[vendor] = false;
13 }
```

The whitelist is implemented as a `mapping` from an address to a Boolean which indicates if the vendor is whitelisted or not. This means that only vendor addresses in the mapping that maps to `true` are considered whitelisted. On lines 3 to 7, the method to whitelist vendors can be seen, where the method is protected by two guards to cover whitelisting the *zero address* and already whitelisted addresses. If both guards are passed, the vendor address is added to the mapping with `true` as value. If a vendor has to be removed from the whitelist, they are blacklisted in the method on line 9 to 13. The method also checks that the address is not the zero address and that the vendor address is already whitelisted. The vendor is blacklisted by mapping the address to `false`.

The `whitelisted` modifier in the PSC checks if the caller address is whitelisted or not, and only proceeds if it is. The modifier implementation can be seen in Listing 5.15.

Listing 5.15: The `whitelisted` modifier on the Private Smart Contract.

```
1 modifier whitelisted() {
2     require(vendors[msg.sender], "Caller is not whitelisted")
      ;
3     _;
4 }
```

5.5.2 Encryption Key

An important part of announcing confidential security advisories is encryption. As specified in the design in Section 4.7.5, the vendors use a public RSA-OAEP key to encrypt the AES-GCM key, which the security advisory is encrypted with. The PSC provides a state variable for this public RSA-OAEP key and a method to update it, which can be seen in Listing 5.16.

Listing 5.16: The `publicKey` state variable and the related setter function on the Private Smart Contract.

```
1 bytes public publicKey;
2
```

```
3 function setPublicKey(bytes memory pKey) public onlyOwner {
4     publicKey = pKey;
5 }
```

The `publicKey` state variable is a dynamically-sized byte array in which the public RSA-OAEP key is stored such that vendors can retrieve it when they should encrypt the key they encrypted the security advisory with. Asset owners update the key by calling the `setPublicKey` method with the key as input. The public key state variable is accessible for everyone at all times from the PSC address when they need to encrypt data for the asset owner.

5.5.3 Confidential Announcement

In order to implement the announcement functionality on the PSC, an event declaration and a method which emits this event, is introduced. The implementation of these can be seen in Listing 5.17.

Listing 5.17: Event declaration and announcement method on the Private Smart Contract.

```
1 event Announcement(
2     string location,
3     bytes32 hash,
4     bytes decryptionKey,
5     bytes12 iv
6 );
7
8 function announce(string memory location, bytes32 hash, bytes
      memory decryptionKey, bytes12 iv) external whitelisted {
9     emit Announcement(location, hash, dKey, iv);
10 }
```

In the `announce` method, an instance of the previously declared `Announcement` event is emitted. The input arguments are the IPFS CID of the encrypted security advisory, a hash of the security advisory, the encrypted AES-GCM key, and the initialization vector. With this data, the asset owner is able to retrieve the encrypted security advisory, decrypt it with the use of the private RSA-OAEP key, AES-GCM decryption key and the initialization vector. The hash is used to ensure non-repudiation, as described in Section 4.7.4.

The use of this method is restricted by the `whitelisted` modifier to only allow access to whitelisted vendors, such that the event is not emitted by a malicious actor.

5.6 IPFS & Ethereum Nodes

The IPFS and Ethereum nodes are essential parts of SENTINEL, as illustrated in Section 4.6.1. As there are many existing software solutions for these nodes, and implementation of such nodes falls outside the scope of this project, third-party nodes are used. The chosen solutions are covered in this section.

5.6.1 IPFS Node Integration

Kubo [38] is used as IPFS node on the local machine of asset owners and vendors. Kubo is created by Protocol Labs [44] which is the creator of IPFS. This node contains the core IPFS functionality to interact with the IPFS network, it includes an API to interact with the node, and has a gateway allowing the user can access content on IPFS. All parts are vital in the use of SENTINEL, as the API is used from the frontend to upload and download security advisories, and the gateway is used to host the frontend itself in a local environment.

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. Because the access to this API is restricted by default, it is necessary to configure the IPFS node to allow API calls from the frontend. The API configuration can be changed from the terminal on the localhost or via the web user interface that the Kubo provides. To configure the API via the terminal, the command in Listing 5.18 is used, where `FRONTEND_URL` is the URL on which the frontend is running.

Listing 5.18: Kubo API configuration bash command.

```
1 ipfs config --json API.HTTPHeaders.Access-Control-Allow-
    Origin "[\"<FRONTEND_URL>\"]"
```

5.6.2 Ethereum Node Integration

As a part of the design from Section 4.6.1, the frontend connects to the Ethereum network through an Ethereum node's execution client. However, the full node must also contain a consensus client as it is responsible for receiving blocks from the network and pass them to the execution client which keeps the locally stored blockchain up to date. [23]

Geth [45] is created by the Ethereum Foundation as an official execution client for Ethereum networks, written in the Go programming language. It is the most used execution client, which is the reason for it to be chosen as the execution client in the Ethereum node setup in this project [46]. Geth exposes a JSON-RPC server

on `http://localhost:8545` which the frontend can query to get data from the blockchain.

Lighthouse [47] is one of the two most used consensus clients [48], which are responsible for the proof-of-stake consensus protocol in combination with other validation nodes. Lighthouse validates blocks from other consensus nodes with the blockchain data stored on Geth. This will ensure that the locally stored blockchain is synchronized with the state of the blockchain network and that the SENTINEL frontend has access to all blocks from the past and in the future.

5.7 Frontend Implementation

The frontend is an important component for SENTINEL, as it is how the asset owners and vendors interact and use the system. Important implementation details about the frontend are described in this section.

5.7.1 Framework

React.js is chosen as the frontend framework for the development of the frontend to SENTINEL. This framework is chosen because it is the most popular JavaScript frontend framework, according to the 2022 developer survey conducted by Stack Overflow [49]. With React.js being the most popular, it is reasonable to assume more support is available for issues regarding Web3 frontend development and frontend development in general.

The SENTINEL frontend is implemented as a Single Page Application (SPA). With the use of SPA, the frontend is loaded entirely on the first request, and will not have to request resources when navigating between pages.

5.7.2 Components

React components are the building blocks that make up the frontend. The components include the actual UI elements which a user can see and interact with, and contain functionality to change the state of the component. The frontend is built of components in a tree-like structure, with a root component called `Index.js`, where other components branch from.

Screenshots of the frontend UI can be seen in Appendix B.

To simplify the implementation and focus on functionality, UI components are built with Bootstrap React components [50]. Bootstrap is an open source library with HTML, CSS, and JavaScript for a collection of components. Bootstrap React

is a version of Bootstrap ported to React by a GitHub community. The components have standardized themes and can be implemented directly into the React components to build the UI. Using Bootstrap saves development time that would otherwise be used to create these UI components.

5.7.3 Web3.js Integration

In order to read the events and other smart contract data from the Ethereum blockchain, the frontend uses the Web3.js [51] JavaScript library. The library is an abstraction over the Ethereum JSON-RPC API and makes it easier to implement interactions with blockchain with the connected client in JavaScript code.

The Web3 client is used in the `Web3Gateway` class, which is a wrapping class that contains the necessary Web3 functionality for the SENTINEL frontend. This class includes fields to store the Web3.js API client instance, events from the AS, event subscriptions, and methods to create transactions for the smart contracts. The `Web3Gateway` integrates with the Geth execution client via `ws://localhost:8546`. A web socket connection is used instead of HTTP, as subscribing to events is not supported for HTTP connections.

One instance of `Web3Gateway` is stored in an outer component, where it is kept active regardless of routing. It is used to keep the state of event subscriptions and already discovered events, such that an asset owner does not have to wait for the frontend to search through all announcements every time they access the vulnerabilities page.

5.7.4 IPFS API Integration

The frontend integrates with the local IPFS node's API, which should be exposed on `http://localhost:5001`. The `ipfs-http-client` npm [52] package is used to create the API client to the IPFS node. The IPFS client is created in an outer component, where it remains usable on all frontend pages. The function creating the IPFS client can be seen in Listing 5.19.

Listing 5.19: IPFS API client loading function in the frontend.

```
1 async function loadIpfs() {
2     var ipfsClient = create({
3         host: "127.0.0.1",
4         port: 5001,
5         protocol: "http"
6     });
7     setIpfs(ipfsClient);
8 }
```

When the API client is done loading it is passed to the child components where it is used to download security advisories from on the vulnerabilities page, or to upload security advisories to IPFS on the announcement pages. Passing the client to child components ensures that the client is initialized with the correct configuration to all components it is passed to.

On the vulnerabilities page, security advisories are downloaded from IPFS upon request and automatically parsed from CSAF document to a human-readable format. The information parsed from the CSAF document is pruned, such that only relevant vulnerabilities are displayed.

5.7.5 User Settings

In SENTINEL, settings such as dependencies and account information are stored in the browser's local storage. This ensures that settings are persisted between sessions and are easily readable on the various pages on the frontend. Data in local storage that can be considered sensitive is encrypted, as explained in Section 5.7.6.

A part of the settings stored in local storage is the dependencies of the asset owner. This information is parsed from a CDX SBOM document, and the product identifiers for the dependencies are saved in local storage.

5.7.6 Password Encryption

In order to encrypt the data in local storage, the SENTINEL frontend uses PBKDF2 to derive an AES-GCM key, which is used to encrypt and decrypt the data. The responsible function can be seen in Listing 5.20

Listing 5.20: The `deriveAesKey` function from the frontend. The function derives an AES-GCM key from a provided password string.

```
1  static async deriveAesKey(password, salt) {
2      const keyMaterial = await this.computeKeyMaterial(
       password);
3      const aesParams = {
4          name: "AES-GCM",
5          length: 256
6      }
7
8      const derivationParams = {
9          name: "PBKDF2",
10         salt: salt,
```

```
11          iterations: 600000,
12          hash: "SHA-256"
13      }
14
15      return window.crypto.subtle.deriveKey(
16          derivationParams,
17          keyMaterial,
18          aesParams,
19          true,
20          ["decrypt", "encrypt"]
21      );
22  }
```

The key derivation method from SubtleCrypto [53] is used to derive an AES-GCM *cryptokey* from a password string. The password is converted to bytes and passed to the `deriveKey` method along with the PBKDF2 parameters and AES-GCM parameters.

To ensure that the PBKDF2 algorithm provides adequate security, several configuration considerations must be made. PBKDF2 applies a pseudo-random function, such as a cryptographic hash function, along with a salt, to some data and repeats this many times to produce a key. As such, the choice of hash function, salt size, and iteration count is important. [33, 34]

The hash function should be approved by NIST and should match the security strength of the key that is derived. For local storage encryption, 256 bit AES-GCM encryption is used, as in contrast to the encryption process in Section 4.7.5, there are no storage limitations to consider. As such, the SHA-256 hash function is chosen, as it matches the security strength of 256 bits when used in key derivation functions. [33, 54]

It is recommended that the salt size is at least 16 bytes long and is generated by a NIST approved random number generator. The SENTINEL frontend implementation utilizes the random number generator in the *WebCrypto* module [55] of JavaScript to generate a 16 byte salt. [33, 34]

Finally, the number of iterations used in PBKDF2 is recommended to be as high as possible while still being tolerable for the application. In their password guidelines, NIST recommends at least 10,000 iterations, while OWASP recommends 600,000 when using SHA-256. In the frontend, as seen in the `derivationParams` in Listing 5.20, 600,000 iterations are used. [33, 34, 56]

5.7.7 Confidential Advisory Process

In Section 4.7.5 the process of transmitting confidential advisories securely was described. This process is simplified in the implementation from a user standpoint.

On the *Confidential Settings page* the asset owner can generate an RSA-OAEP key pair. The public key is automatically uploaded to the specified PSC, thus making it accessible to others. The private key is encrypted and saved to local storage for decryption.

When the vendor intends to announce a confidential security advisory, they can fill out the form on the *Confidential Announcements page*. Upon submitting the form, the provided security advisory is encrypted using AES-GCM and uploaded to IPFS. After a successful upload, a CID, corresponding to the location of the advisory, is returned. This CID, along with the IV, a SHA-256 hash of the security advisory, and the AES-GCM private key encrypted using the asset owner's RSA-OAEP public key, are emitted as an event on the Ethereum blockchain using the PSC.

The asset owner, having added their PSC to their settings, will automatically retrieve new events related to that smart contract. The advisory is automatically downloaded from IPFS, decrypted using the corresponding RSA-OAEP private key from local storage, and displayed to the asset owner on the *Vulnerabilities page*.

5.8 Continuous Integration & Deployment

Continuous integration was used during development to automate code analysis and testing, thus adding additional quality control to new code. Continuous deployment was used to automatically update the frontend when new changes were made. The various workflows utilized for this are described in this section.

5.8.1 GitHub Workflows

As the code is saved on GitHub [57], the CI/CD pipeline can be automated with the use of GitHub Workflows. A workflow is a set of jobs that describe what actions should be made for that specific task specified by a YAML file. Workflows can be triggered on specific events, such as commits to a pull request or when code is committed to the main branch. This makes it useful to check for breaking changes committed to a pull request and to deploy new versions when released. [58]

5.8.2 Dependabot Updates

Dependabot is used to automatically check for security patches and version updates to dependencies of a GitHub repository. It can automatically create pull requests and security alerts if updates or patches are available. [59]

In the SENTINEL repository, Dependabot keeps the GitHub-Actions, utilized in the workflows, up to date. The script can be seen in Appendix C.1. Additionally, Dependabot also automatically creates security alerts for patches to vulnerable npm packages used in the repository.

5.8.3 Smart Contract Analysis

In order to catch known vulnerable smart contract patterns, Slither is utilized. Slither is a static analysis tool for solidity code, which finds vulnerabilities and optimizations in smart contracts. [60, 61]

To run Slither in a GitHub workflow, *slither-action* is used, which enables Slither analysis against the repository in workflows. slither-action allows GitHub Code Scanning integration, which will create security alerts on the GitHub repository if the analysis detects any vulnerabilities. The script can be seen in Appendix C.2. [62]

5.8.4 Integration Tests

During the development, the implementation code for the smart contracts and the frontend was checked against a certain set of unit- and integration tests to make sure that any changes would not break the code. The smart contract tests are run with Truffle on any commits to a pull request.

As the frontend tests utilize an Ethereum blockchain connection, the test command for npm is customized. The `npm test` command that runs frontend tests automatically create a local Ethereum blockchain instance with Ganache [63], run the frontend test, and then remove the blockchain instance when the tests are completed. With this setup, the tests can be run locally and in GitHub actions.

5.8.5 Frontend Deployment

As mentioned in Section 4.6.1, the frontend is distributed via the IPFS network under an IPNS name, such that users can access it from their local IPFS node. Uploading the frontend build to the network is done using a GitHub workflow, which can be found in Appendix C.5. The workflow is triggered on commits to the main branch, which contains the code of the latest version of the SENTINEL frontend. The workflow builds the frontend and uploads the build artifacts to an IPFS node with the script in Listing C.6. The frontend build is uploaded to IPNS on the specified IPFS node.

Chapter 6

Test & Assessment

To ensure that the implementation detailed in Chapter 5 conforms to the requirements specified in Section 4.3, several tests are performed. Additionally, important characteristics such as security, scalability, availability, and cost of SENTINEL are assessed in the chapter.

This chapter includes the following sections.

- Scalability Assessment
- Cost Assessment
- Availability Assessment
- Security Assessment
- Unit & Integration Testing
- System Testing

6.1 Scalability Assessment

An important factor of SENTINEL's usability is scalability. It is important that SENTINEL and by extension the Ethereum network can support the amount of transactions that users of SENTINEL will create when interacting with the system. This section investigates if CVE's current security advisory publication rate is supported by SENTINEL and the Ethereum network.

6.1.1 Publication Rate of Security Advisories

Calculating the publication rate of all security advisories is an impossible task because security advisories can be shared in confidentiality, and public security advisories can be published many places. Therefore, the calculated publication rate is not of all security advisories published everywhere. However, as SENTINEL is supposed to be a decentralized alternative to centralized organizations, it is useful to look at the publication rate of security advisories on CVE and NVD as an estimate.

Table 6.1 shows the published security advisory statistics from 2016 to 2022 for both CVE and NVD. The data shows a trend of increasing number of published security advisories each year. The statistics are naturally similar for CVE and NVD, as NVD is basing their vulnerability database on the content of CVE. [64, 65]

CVE statistics							
Year	2022	2021	2020	2019	2018	2017	2016
Published	25,059	20,161	18,375	17,308	16,512	14,645	6,457
Daily average	68.65	55.24	50.34	47.42	45.24	40.12	17.69
Increase	24.29%	9.72%	6.16%	4.82%	11.27%	126.80%	−0.57%
NVD statistics							
Year	2022	2021	2020	2019	2018	2017	2016
Published	25,101	20,158	18,350	17,305	16,509	14,644	6,447
Daily average	68.77	55.23	50.27	47.41	42.49	40.12	17.66
Increase	24.52%	9.85%	5.49%	4.82%	12.74%	127.14%	−0.62%

Table 6.1: CVE and NVD security advisories statistics from 2016 to 2022. [64, 65]

The data from 2022 is used as the target, that is, SENTINEL, and by extension the Ethereum network, should at least be able to support this amount of security advisory publications. However, as the trend is showing that the target will likely increase in the future, it would be favorable for SENTINEL to support more than what was disclosed in 2022, to continue to be usable in the future. Table 6.2 shows a breakdown of the 2022 targets for SENTINEL to achieve.

Interval	Number of public advisories
Yearly	25,101
Monthly	2,092
Weekly	483
Daily	69
Hourly	3

Table 6.2: Security advisory publication targets that SENTINEL should achieve, broken down into different intervals.

6.1.2 Ethereum Transaction Capacity

Since The Merge on September 15th 2022 to April 25th 2023 there have been an average of 1,061,576 transactions per day on the Ethereum blockchain. However, there has been an average of 165,167 pending transactions which has never been cleared entirely [66, 67]. This means that anyone who wishes to make a transaction has to *compete* with other users to have their transaction selected for the next blocks. To make sure that a transaction will be mined the shortest time possible, the user can provide a higher *priority fee*, and pay more gas for the transaction, to incentivize miners to choose that specific transaction for the next block.

6.1.3 Scalability Results

Assuming the number of transactions is 1,049,678 for a given day, and the number of daily security advisories published is 69, the amount of announcement transactions is only 0.000657% of all transactions on the Ethereum network for that day. However, this number only takes the transactions that announce new public security advisories into account. Other transactions, such as contract deployment, announcement for updates to public security advisories, and all transactions in the private use case are not covered by that estimate. Even still, asset owners and vendors are able to use SENTINEL to announce security advisories without overwhelming the Ethereum network.

Another point is that the SENTINEL users can pay higher gas fees to increase the chances of having their transaction chosen for a block. In contrast, if users are not willing to pay the necessary gas fees, discussed in Section 6.2, then their transaction can potentially be pending indefinitely. Users can use Layer 2 rollups [68], where transactions are executed off chain and bundled together in a single Ethereum transaction, to minimize the transaction cost and increasing the transaction throughput of the network. However, this comes at a cost as the Layer 2 operators often are centralized services. It should be noted that there are upgrades planned for the Ethereum network which will increase its scalability. [68]

6.2 Cost Assessment

As transactions in the Ethereum network cost money and these transactions are essential for SENTINEL to function, it is in order to investigate the operation cost of SENTINEL. The cost is measured in gas, as converting to a fiat currency is troublesome due to fluctuating exchange rate. However, it should be noted that the actual gas used will also vary depending on the input data, and thus the cost assessment in this section should not be taken as concrete values but only as approximate values instead.

The formula for calculating the transaction cost is:

$$units\,of\,gas\,used * (base\,fee + priority\,fee),$$

where *base fee* is provided by the protocol and the *priority fee* is decided by the transaction creator [69].

6.2.1 Methodology

The gas cost is calculated by making transactions to a locally run Ganache blockchain instance, where gas used for the transactions is recorded from the transaction receipts. The Ganache blockchain uses the Petersburg Ethereum Virtual Machine (EVM) version, which means the transaction cost may be different on blockchains with a different EVM version.

In addition to the cost provided by the transaction receipts, the gas-estimate function from the Web3.js [51] JavaScript library is run on all transactions. This data is not presented for conciseness and readability reasons; however, it was found that the estimation and the recorded gas are accurate most of the time.

6.2.2 Deployment Cost

Smart contracts must be deployed to the network for both the public and private use cases. The deployment cost of SENTINEL smart contracts can be seen in Table 6.3. It should be noted that the cost of the VSC deployment can vary depending on the input variables for the constructor.

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 6.3: SENTINEL smart contract deployment gas cost.

6.2.3 Interaction Cost

Another element leading to the total cost is the Ethereum transactions made after deployment to interact with the smart contracts. These transactions are fundamental functionality for the system. To collect data on the cost, the transactions are created through the frontend, and are recorded to calculate how much gas each transaction consumes. Transactions can change the state of the blockchain, and depending on smart contract memory state, more storage could be allocated to store new data. Therefore, the transactions are created ten times in sequence with the same input data to more accurately capture the cost over a larger sample size.

Table 6.4 shows the cost of different transactions in SENTINEL as recorded by the transaction receipts in the frontend. The data shows that for announcing new security advisories, vendors can expect that the gas cost will differ. This is due to the identifier generation from the IIS, which allocates memory to store the identifiers in the smart contract memory.

	Transaction 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 6.4: Gas cost of transaction for the public use case of SENTINEL.

Table 6.5 shows the cost of the transactions related to the private use case of SENTINEL. The biggest cost is saving the RSA-OAEP public key in the smart contract memory.

Transaction	Execution		
	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 6.5: Cost comparison of private use case transactions in SENTINEL.

6.2.4 Cost Calculation

The cost of different blockchain transactions in SENTINEL can be seen in Table 6.6. The transactions are grouped by their respective use case. In situations where the gas for a transaction is not constant, the average cost over ten transactions is used instead, apart from the operation of updating the public key. Here, the gas cost for the first transaction is used to show an upper bound of cost, as this is when the storage space is allocated.

From The Merge on September 15th 2022 to April 25th 2023, the average gas price is 25.63 Gwei and the Ether exchange rate is $1,499. These prices are assumed for the conversion in Table 6.6.

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 6.6: Cost calculations for different blockchain specific operations of SENTINEL. Gas price assumed to 25.63 Gwei and one Ether is $1,499.

The cost in Table 6.6 shows a large up-front payment to deploy the smart contracts for the respective use cases, and the announcement of security advisories is much lower in comparison. A caveat in the private use case is that most transactions are part of the deployment of the smart contract and setup. This means that the cost of deploying and setup in the private use case is actually $57.7.

6.3 Availability Assessment

In this section, the availability of security advisories published with SENTINEL is assessed. Specifically, the time for security advisory announcements to be discoverable and the time for security advisory uploaded to IPFS is investigated, as both of these are necessary for asset owners to become aware of potential vulnerabilities.

6.3.1 Event Data Availability

No data have been published yet on transaction confirmation time after The Merge in September 2022. As such, it is difficult to give an estimate on how long a vendor will have to wait, before their announcements are stored in the blockchain. However, Etherscan's Gas Tracker [70] shows real-time estimates on the size of the transaction fee which users should use, in order to be competitive in the miner transaction selection. Observations on this page show, that paying near the highest gas fees gives an estimated confirmation time of about 30 seconds, while paying a lower priority fee offers an estimated confirmation time of around 3 minutes. These timings assume that the base fee matches the required gas determined by the Ethereum protocol.

When the transaction is first mined, it must wait for approximately 15 minutes for the block to be *finalized*, which means that the block cannot be changed without burning at least 33% of the total staked Ether, the cryptocurrency of Ethereum. This means that asset owners and vendors should wait approximately 15 minutes after the transaction is mined, before they should consider it valid. [71]

With this knowledge about confirmation and finality time, vendors can expect their announcements to be available after 0.5 to 3 minutes and stored indefinitely after about 15.5 to 18 minutes. However, this assumes that vendors pay competitive gas fees and that miners will select the transaction to include it in a block.

6.3.2 Security Advisory Availability

The availability of security advisories uploaded to IPFS is an important part of the general availability of SENTINEL, because if the security advisories are not available, the asset owners cannot take action against potential vulnerabilities. Therefore, the availability of security advisories uploaded to IPFS, as is designed for SENTINEL, is assessed.

In this availability assessment, two different IPFS nodes with different participation time in the IPFS network are used: a well established node with three months of participation and a new node with no previous participation. With this

setup, the difference between newly and well established nodes is investigated. The nodes are kept connected to the IPFS network for the duration of the assessment. Each node is given a similar CSAF document [72] with modification to ensure different IPFS content identifiers. With the security advisories uploaded and pinned to the nodes, they are accessed through the `https://ipfs.io/ipfs/` public gateway at different intervals to check if the file is accessible from the gateway. A document is considered inaccessible if the gateway suffers a timeout when trying to access the document.

For the well established node, the security advisory document was available practically immediately, which was not the case for the newly established node, where the document was available within 30 minutes. When the security advisories became available from the public gateway, they remained available for the remainder of the requests because they are pinned and never removed from the nodes by the garbage collector. The availability of security advisories with IPFS is therefore acceptable for the use of SENTINEL, where security advisories are not required to be available immediately.

6.4 Security Assessment

In Chapter 4, multiple security requirements were presented. In this section, SENTINEL is assessed in relation to the requirements and smart contract security guidelines.

6.4.1 Smart Contract Security

As mentioned in Section 5.1.1 development guidelines from Ethereum and ConsenSys were followed during development. These guidelines focus on prevention of common patterns and mistakes that make smart contracts vulnerable. Known attacks are also considered in the smart contract security assessment.

Known Attacks

Multiple known smart contract attacks have been explored in relation to SENTINEL, however, most are not applicable as the features they depend on are not found in SENTINEL. Several relevant attacks and vulnerabilities which can directly affect SENTINEL are discussed below. These known vulnerabilities are classified in Smart contract Weakness Classification (SWC) where each weakness has a unique identifier. [73]

SWC-107, known as reentrancy [74] is a potentially dangerous vulnerability that can occur when smart contracts make external calls. In a reentrancy attack, a

malicious contract initiates a callback to the calling contract prior to the completion of the first function invocation and before the state of the calling contract gets updated. However, it can be avoided by using the *checks-effects-interactions* pattern, as well as, by using locks. [74, 39]

In SENTINEL, the checks-effects-interactions pattern was followed where applicable. Additionally, only four external calls are made in SENTINEL smart contracts, all in the VSC. The calls are made to the AS and the IIS, which both are trusted smart contracts. In the case that they were malicious smart contracts, a reentrancy attack would not be able to manipulate the state of the VSC, as no state variables are modified after the external calls.

Another common attack is SWC-114 [75], also known as frontrunning, in which an attacker manipulates the order of transactions to their gain. Because asset owners explicitly have to whitelist trusted addresses, frontrunning is not impactful regarding security advisory announcements, as the attacker is not trusted and their announcements would be ignored. As such, manipulating the order of announcements would not provide any advantage.

SENTINEL, is vulnerable to SWC-128, Denial of Service (DoS) attacks with block gas limit [76], however, the impact of such an attack is difficult to determine. The fact that vendors would be unable to announce security advisories is detrimental to the entire security advisory dissemination process. Nonetheless, it is not critical that advisories are announced the instant it is possible. As such, an attacker would likely have to keep up the DoS attack for an extended period of time for it to have a noticeable effect, all while expending large amounts of gas for the attack.

Another possible DoS attack is that an attacker could continue to register vendors and saturate the vendor registry in the IIS. If 2^{64} vendors are registered, new vendors cannot register, thus leaving them excluded from the service. This would, however, be very expensive, as the call to register a vendor must be received from a smart contract, and each smart contract can only register once. As such, it is reasonable to assume that it is infeasible for an attacker to register enough vendors to deny usage of the IIS.

Analysis Tools

In addition to following the guidelines and checking common vulnerabilities, it is recommended by both the Ethereum Foundation and ConsenSys to utilize analysis tools to check smart contracts for vulnerabilities. For these purposes, Slither [60] and Mythril [77] were utilized. Slither is a static analysis tool for Solidity code, which finds vulnerabilities and optimizations in smart contracts. Mythril is a tool

for security analysis of smart contracts. It utilizes symbolic execution, Satisfiability Modulo Theories solving, and taint analysis to find vulnerabilities. [60, 61, 77]

The analysis results only found one potential vulnerability, which is the *SWC-115* vulnerability. This relates to the use of `tx.origin` in authorization, as potentially malicious code can be executed if it is used in authorization mechanisms. In SENTINEL, however, `tx.origin` is used to ensure that only smart contracts can call specific functions. It is not used to authorize an address, thus not creating the SWC-115 vulnerability. As such, the tools returned a false positive, in this case.

6.4.2 Cryptographic Key Management

In Section 4.7.5 several protection requirements were presented, of which four were strictly required when handling cryptographic keys. SENTINELS compliance with the four strictly required requirements is assessed in this section.

Confidentiality Protection

In SENTINEL, confidentiality protection should be provided in two cases.

1. When the AES-GCM private key is transported from the vendor to the asset owner.
2. When the RSA-OAEP private key is stored in the browsers local storage.

In case 1, the AES-GCM private key is transmitted using KTS-OAEP, which is a NIST approved key transportation method. When KTS-OAEP is employed in SENTINEL, 3072 bit RSA-OAEP keys are used. This translates to 128 bits security strength, which matches the security strength of the AES-GCM private key. As such, confidentiality is protected in this case. [31]

In case 2, the RSA-OAEP private keys are encrypted using AES-GCM encryption when stored locally. The AES-GCM keys are 256 bits long, thus having 256 bits of security strength, which exceeds the 128 bits of the RSA-OAEP key. The AES-GCM key is generated using PBKDF2, which is a NIST approved key derivation algorithm. SENTINEL uses the SHA-256 cryptographic hash function in its PBKDF2 configuration, which results in a security strength of 256 bits, thus matching the AES-GCM security strength.

Additionally, as mentioned in Section 4.8.1 and Section 5.7.6, proper guidelines and configurations are followed to provide secure key derivation. As such, confidentiality protection is ensured in case 2.

Integrity Protection

Integrity protection should be provided by SENTINEL in three cases.

1. When an AES-GCM key is transmitted via an event.
2. When an RSA-OAEP public key is written to a PSC.
3. When an RSA-OAEP private key is saved to local storage.

NIST's key management guide specifies that integrity of a transmitted cryptographic key can be verified if the recipient of the key can perform the intended cryptographic operation correctly using the key. This method assumes that protection of the key is in place to protect against key manipulation from physical attacks, where an attacker modifies the key via physical access to the machine, on which it is stored. In all the abovementioned cases this means that if the asset owner, whom is the recipient of the keys, can decrypt the encrypted security advisory and wrapped AES-GCM key respectively, the integrity can be verified. [31]

It is worth noting that integrity verification of the RSA-OAEP public key is a process consisting of the recipient manually informing the sender that the key is invalid and as a result is inefficient. Providing a built-in integrity mechanism in the PSC could improve the process.

Physical protection in case 1 and 2 is not as important because the keys are stored on the Ethereum blockchain. As such, if an attacker were to modify these keys physically they would have to access to a majority of the computers validating the blockchain, which is infeasible for the Ethereum blockchain due to its size, thus providing integrity protection.

Regarding case 3, which is stored locally on the asset owner's machine, physical protection is the responsibility of the asset owner. It is assumed that the asset owner has taken precautions, such as authentication and disk encryption, to secure the machine from physical attacks.

Association Protection

Association protection should be provided in four different cases in SENTINEL.

1. When an AES-GCM key is emitted in an event.
2. When the RSA-OAEP public key is stored on a PSC.
3. When the RSA-OAEP private key is stored locally.
4. When the IV used to encrypt locally stored data is stored locally.

In case 1, association protection is provided by including the location to retrieve the ciphertext and the related IV. The sender, who is also authorized to use the key, is known via the address of the transaction creator.

In case 2 association protection is achieved as the key-pair owner is known to be the owner of the PSC which stores the public key. This party is also in pocession of the private key. Furthermore, the usage of the RSA-OAEP public key is specified in SENTINEL, both in the smart contracts and on the frontend.

In case 3, the association protection is achieved as the RSA-OAEP private key is stored locally along with the necessary information. This information includes the address of the smart contract, on which the related public key is stored. The key-pair owner is known implicitly, as this is the user of the system. Usage of the key is specified in the SENTINEL frontend.

Finally, in case 4, association protection is provided by storing the IV related to each distinct message alongside the ciphertext.

Availability Protection

Availability protection can be provided by storing backups of keys in key archives or other backup systems. NIST recommends that use of backup systems for cryptographic keys is determined based on the needs of the application, however, it is not strictly required to back up any keys according to their guidelines. [31]

The SENTINEL prototype does not implement any backup storage options for cryptographic keys, as it was omitted due to requirements prioritization.

6.4.3 Password Encryption Assessment

In Section 4.8.1 multiple requirements for password encryption from NIST were introduced. The requirements are specified as strictly necessary, however, four of them are not fulfilled in SENTINEL.

The requirement for randomly generated passwords is not fulfilled, as SENTINEL does not provide any service to automatically generate passwords or pin codes. As such, this requirement is irrelevant.

The requirement regarding password comparison to compromised passwords, dictionary word and sequential characters, has not been fulfilled due to time constraints. Because this project outlines a prototype, this requirement was not prioritized as the focus of development was on feature implementation.

The requirements regarding rate-limiting and protected communication channels are irrelevant to SENTINEL, as the frontend is run locally in the browser. If

a rate limit was introduced, a malicious user could simply change or remove it by modifying the JavaScript. Encryption and protected communication is irrelevant, as the password never leaves the local machine. As such, these requirements have not been implemented.

6.5 Unit & Integration Testing

This section describes how the different parts of the system were tested during development. The tests cover all different components, that is, smart contracts and frontend in both public and private use cases.

6.5.1 Unit Testing

For unit testing, the Node.js module *Mocha* was utilized for both frontend and smart contracts. Mocha was especially practical for testing smart contracts, as it is already integrated with Truffle. As such, Truffle's features such as smart contract deployment to Ganache, smart contract calls and transactions are available to use in Mocha unit tests. Additionally, *Truffle-Assertions*, an add-on package to Mocha, was used to allow more detailed checks of reverts.

In smart contracts, it is important to avoid bugs as they can be hard, if not impossible, to patch once the smart contract is deployed. Therefore, when unit testing smart contracts, it is especially important to cover as many cases as possible. The smart contracts in SENTINEL have unit tests to cover all the functionality of the smart contracts.

The SENTINEL frontend primarily consists of React.js components, in which most of the functionality is used to change the user interface. However, unit tests are created for standalone JavaScript components such as the CSAF parser and the Web3Gateway.

6.5.2 Integration Testing

Integration testing was utilized during and at the end of development to test the SENTINEL components in integration. This ensures that the interfaces between the components are working as intended and that any breaking changes introduced in one component is caught before it breaks other components.

In order to test the integration between the frontend and the smart contract, the frontend must integrate with an Ethereum network. During development, the

Ethereum integration was tested using a local blockchain instance where the smart contracts were deployed to. As mentioned in Section 5.8.4, the `npm test` command automatically creates a blockchain instance which the frontend integration tests interact with.

The Ethereum integration tests ensures that the methods on the smart contracts exist, can be called with specific input data, and execute as expected without reverting.

Integration with IPFS was tested using a local Kubo IPFS node. With this node, the file upload and download from the frontend could be tested for both use cases. This ensures that the frontend can integrate with the IPFS network as it was designed to.

6.6 System Testing

A system test is conducted to ensure that all components, that is, the frontend, smart contracts, IPFS node, and Ethereum node, integrate and achieve the desired functionality when combined. In this test, all integrations from the frontend to the individual components and frontend functionalities are tested.

6.6.1 Test Procedure & Setup

To test the two distinct use cases in SENTINEL, the following procedure is used. Each use case is divided into a set of features that must work as intended in order for the use case to satisfy the system requirements. Each feature has unique success criteria which is verified using data collected from the blockchain, as well as, state changes on the frontend. Based on the collected data, the system is evaluated depending on the collective success criteria of all features.

For the system test, SENTINEL is deployed in the Sepolia Ethereum testnet [78]. This includes deployment of an AS, an IIS, a VSC, and a PSC as they are described in Chapter 4 and Chapter 5. A custom script is developed to deploy the smart contracts to the testnet. The SENTINEL smart contracts are deployed to the following addresses:

- **Announcement Service:** `0xbdBc312f3dc75a6D47D7Eaa7E6a4BBFbb07f09fc`
- **Identifier Issuer Service**: `0x577a791f4033F7905b822664ff0E1a74dbe5EF70`
- **Vendor smart contract**: `0xF472cebcd32953E165eD35B51708a796EEA76A34`
- **Private smart contract**: `0xFfb2234E55D1D238fE8b80Ef6e4f435AC89c375d`

All the smart contracts have been deployed from the same account address (`0x84Ed2d4aF7C11E637Beab9F08677937B7994c07E`) and the system test is performed from the same address. All transactions are public and can be browsed at `https://sepolia.etherscan.io`

The frontend is published to IPFS where it is downloaded and accessed from a local IPFS gateway and utilizes the corresponding IPFS API to retrieve data from IPFS, as explained in Section 4.6.1 and Section 5.6.1.

During the system test, success and failure data is recorded. For integrations with Sepolia, this data is the receipts of the transactions that are executed and added to a block in the case of success and reverted in the case of failure. For IPFS integrations, recorded data is the accessibility and integrity of the security advisories. Recorded data for the frontend specific functionality is the success of storing data in the browser's local storage and the encryption and decryption of stored data.

The plan for the system test is located in Appendix D.

6.6.2 Test Results

As the system test is used to test the integration of components and the capabilities of the entire system, the results from the test is evaluated by which requirements from Section 4.3.2 have been fulfilled. Table 6.7 shows the requirements with a symbol to indicate the level of fulfillment. Each requirement can either be fully, partially, or not fulfilled as indicated with ✓, ≈, and × respectively.

Must have	**Should have**
✓ Announce public security advisories	✓ Filter advisories on vendor
✓ Announce private security advisories	✓ Announce updates to advisories
✓ Discover security advisories	✓ Discover advisory updates
✓ Load dependencies from SBOM	✓ Manage confidential agreements
✓ Filter advisories on dependencies	× Persistent storage of settings
✓ Present security advisory content	≈ Contract deployment
✓ Integrate with storage system	≈ Account management
✓ Security advisory identification	
✓ Vulnerability identification	
Could have	**Won't have**
× Support multiple storage systems	
× Support multiple advisory formats	
× Support multiple SBOM formats	

Table 6.7: Requirement fulfillment based on the system test.

All must have requirements are fulfilled, alongside a small majority of the should have requirements. As such, it can be concluded that the components have been successfully implemented and integrated.

Even though the system test shows that the SENTINEL prototype implemented in this report fulfills the majority of the requirements created, there are issues as well. Foremost, during the system test, little guidance were given to the tester from the frontend, which made it difficult to navigate unless they have had training. This resulted in errors being made, even though the tester was well versed in the system. Shown in Figure 6.1 are 4 out of 22 transactions made during the system test, where two of them are reverts, meaning some error was made when creating the transactions. This proves that, while SENTINEL is functioning as a prototype, more work should be made to help users interact with the system successfully.

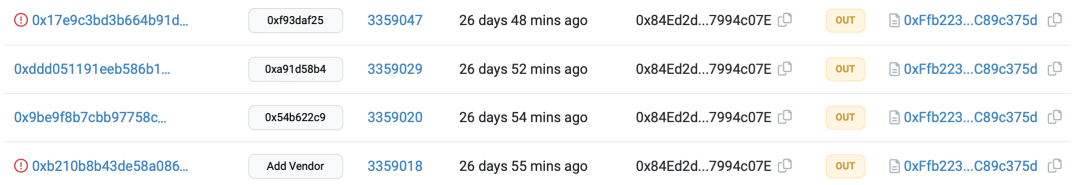

Figure 6.1: Two reverting transactions created during system testing.

Chapter 7

Discussion

In this chapter, several interesting topics raised during and from this project are discussed. The discussion covers many parts of the development, such as design decisions and the development process.

7.1 Previous Work

In previous work [1] a design for SENTINEL was proposed alongside a proof of concept. However, the design for SENTINEL has undergone significant changes when more knowledge on DApp development was acquired. Even still, the proposed design from previous work was vital in the development of SENTINEL, as it was a starting point from which the different components were based and expanded upon. Moreover, the design created in this project also underwent several iterations with definitive modifications.

Furthermore, the investigation into the problem area and technologies in previous work was fundamental for the design decisions made in the project, which solidifies the importance of the previous work even further.

7.2 Non-repudiation

As mentioned in Section 4.3.1, non-repudiation as a characteristic is a system requirement which SENTINEL should adhere to. Non-repudiation is introduced in SENTINEL by using the Ethereum blockchain to announce security advisories, and store the announcement indefinitely. This is useful for vendors to prove that they have disclosed some vulnerability in a timely manner.

However, another part of non-repudiation is the availability of security advisories documents. IPFS as a decentralized data storage system does not provide non-repudiation with the setup of SENTINEL explained in Section 4.6.1. There-

fore, vendors cannot prove the accessibility of security advisories that they have announced.

Non-repudiation for both the announcement and accessibility of security advisories is useful for vendors if they are ever required by law to prove they have disclosed a vulnerability in due time. If SENTINEL should be adopted for this, another storage system that provide proof-of-storage should be used instead of a self-hosted IPFS node. For example, Swarm [15] provides functionality for long term storage, where storage nodes provide continuous proofs that the security advisory is available.

7.3 Frontend Distribution

In Section 4.6.1 it is described that the SENTINEL frontend is accessible via IPNS such that users are able to access the frontend from the same IPNS name all the time, which is not possible with standard IPFS content, if the frontend code should ever change. In order to achieve this, an IPFS node must publish the IPNS name. This creates a centralized component, which is undesirable for SENTINEL. Therefore, it may be preferred to distribute the frontend as *ordinary* IPFS content and advertise the new content identifier after updates. Even still, the frontend has to be uploaded to a IPFS node to be distributed in the IPFS network.

7.4 Development Process

During the development of SENTINEL, the process defined in Appendix E was followed. As specified, the process follows Agile principles. Due to inexperience with DApp development, Agile would allow changes to the design if new information came to light, without large repercussions and time loss. It would also enable modifications and optimizations to the process through retrospectives, if something did not work as expected.

In Appendix E.5 two timelines are illustrated: one described the planned time allocation, and one depicting the actual time allocation. As illustrated, significantly more time was spent on implementation than anticipated. This stems from the many technologies, such as React.js, Ethereum, IPFS, and Solidity, which took longer to learn than anticipated.

It was, however, possible to reduce time for testing, as much of the planning and tooling for it was handled quickly, which allowed the sprints to be finished at the original goal.

7.5 Security Concerns

In Section 4.7.5 the process and methods used to keep confidential security advisories encrypted are described. Among these details, it is specified that a security strength of 128-bits is used to ensure confidentiality. While this is an acceptable security strength according to NIST's guidelines regarding both the strength and the timeframe in which it will be secure, it was chosen because it was the cheapest option that was approved by NIST, as storage on the Ethereum blockchain can be expensive.

However, when handling confidential and potentially highly critical security advisories, having the highest possible security strength might be preferable to saving money on storage. If AES-GCM with 256-bit keys, and therefore 256-bit security strength, were to be used instead, it would also require larger RSA-OAEP keys. The size of the RSA-OAEP keys would increase from 3072 bits to 15360 bits, which in turn will cause a significant increase in gas usage.

In Section 6.4.3 several requirements for secure password encryption, which are omitted from SENTINEL, are described. Two of these requirements could result in considerably reduced security of the password encryption used to protect locally stored data. Firstly, if SENTINEL could generate random passwords for the user, passwords would likely have a higher entropy. Secondly, if SENTINEL could check for compromised password, dictionary words, sequential characters, when users create a password they will be more secure against dictionary attacks [79].

7.6 Ethereum Considerations

As mentioned in Section 6.4.1, SENTINEL is, as many other DApps, vulnerable to Denial of Service attacks. As such, attackers can extend the time for security advisories to be announced to the network. However, if such an attack is executed, the attackers would have to keep the DoS attack going for a long time for the attack to have any significant impact, as it is not detrimental if the publication of a security advisory is delayed by a short period. Furthermore, the cost of the attack will increase exponentially due to the design of the Ethereum protocol [69]. As such, a DoS attack against SENTINEL is possible, however, to have any significant effect, adversaries would have to provide an extreme amount of Ether at their disposal.

As detailed in Section 6.1, Ethereum transactions made by SENTINEL do not overwhelm the network. However, the assessment only considers historical data from The Merge in September 2022. If the network sees an increase in traffic compared to this data, the gas fees will increase and asset owners and vendors will

have to pay more to use SENTINEL, which could disincentivize its use. Furthermore, an increase in the exchange rate regardless of network traffic could increase the cost of transactions when users have to purchase Ether.

In the future, Ethereum is aiming to be able to handle more than 15 transactions per second with upgrades to the network [68], which makes it cheaper to use and more scalable. However, until such upgrades are introduced, users of SENTINEL introduce a minute traffic load to the Ethereum network.

7.7 System Test Result

The system test in Section 6.6 showcased that most of the desired requirements and functionality from Section 4.3.2 were successfully implemented in the SENTINEL prototype. However, the system test also demonstrated that the system is fragile in that transaction reverts can easily happen with small errors. With the restricted time for the development of the frontend in this project, the focus was on development of features and not on error handling and usability, which causes the frontend to be fragile.

Chapter 8

Conclusion

In this project, a novel system to improve the dissemination and discovery of security advisories was designed, implemented, and tested. The system, named SENTINEL, makes use of the Ethereum blockchain and smart contracts to automate and decentralize the process of announcing security advisories, which are distributed via IPFS. Additionally, a web-based graphical user interface has been developed to ease user interactions with these web3 technologies and to improve automation. With SENTINEL, asset owners can automatically discover vulnerabilities for their software dependencies, and vendors can disclose vulnerabilities in both public and confidential contexts.

With the integration to the Ethereum blockchain, assessments made show that SENTINEL has a nearly negligible impact on the blockchain network traffic and that the transactions announcing new security advisories are economically viable. Furthermore, the assessments illustrated that security advisories are quickly available such that asset owners are expeditiously aware of any vulnerabilities in their dependencies.

A security assessment of SENTINEL showed that, the smart contracts developed for SENTINEL are secure against common and known attacks, and the processes that SENTINEL use are secure such that confidential information is not leaked.

A system test of SENTINEL, conducted on an Ethereum testnet, proves that SENTINEL works in the intended environment and fulfill most requirements envisioned for SENTINEL. Furthermore, the system tests shows that asset owners and vendors can use SENTINEL to disseminate and discover security advisories in an efficient and economically viable manner.

Chapter 9

Future Work

Several ideas for expansion and improvement of the SENTINEL prototype are detailed in this chapter. These ideas are suggestions of what would be interesting additions to SENTINEL.

9.1 System Features

As outlined in Section 6.6, there are some features from the requirement analysis that have not been implemented in the prototype for this project. A natural next step for SENTINEL would be to implement the missing features. Furthermore, it would be interesting to gather information from actual asset owners and vendors to investigate if more system requirements should be specified and implemented. As such, SENTINEL could include more functionality which the users find useful.

9.2 Security Advisory Formats

In this project, CSAF v2.0 [7] was chosen as the only format supported for security advisories in the SENTINEL prototype. Furthermore, as explained in Section 4.4.3, only a specific structure of the product tree in CSAF documents is supported. This restriction can disincentivize some vendors from using SENTINEL if the format and structure does not suit them. As such, it would add value to certain vendors if SENTINEL supported the entirety of the CSAF specification and other security advisory formats.

9.3 Encryption Strength

As mentioned in Section 7.5, while confidential advisories currently are locked to a security strength of 128-bits, some users may prefer to use stronger encryption,

such as AES with 256-bit keys. For SENTINEL to support this, only minor modifications would be required on the frontend, allowing users to choose between different strengths. The PSC technically already supports this, as the state variable storing both the RSA-OAEP key and the event data property storing the AES-GCM key are dynamically allocated.

9.4 Extensibility

To enable further extension to SENTINEL, an additional data field could be added to every announcement event. This field could include metadata or data required for new features on the frontend, while not requiring further changes to the smart contracts. Such changes could include new or custom encryption schemes, quality of life features, or simply metadata for further automation.

9.5 Frontend Usability

From the system test in Section 6.6 it was concluded that the frontend needs more error handling to increase usability and reduce the chance of errors. The frontend should handle potential errors before a transaction is created, such that the user funds are not wasted on a transaction that will revert. Conducting a usability test can help pinpoint flaws in the user interface and error-prone parts of the frontend where the error handling functionality should be implemented.

9.6 Automatic Vendor Whitelisting

Currently, asset owners must manually whitelist vendor addresses which they want to receive public security announcements from. This process can be error-prone, and adversaries could potentially be whitelisted by mistake. A solution could be to provide the addresses that should be whitelisted in some formal document, which the asset owners can inject into the SENTINEL frontend. One such approach is to include the addresses in SBOM documents, which can be parsed into the frontend and whitelist vendors automatically.

Bibliography

[1] Jannik Lucas Sommer and Magnus Mølgaard Lund. *Automating Dissemination and Discovery of Security Advisories on Web3: Design and Proof of Concept*. 2023. URL: https://projekter.aau.dk/projekter/da/studentthesis/automating-dissemination-and-discovery-of-security-advisories-on-web3-design-and-proof-of-concept(d1fb7370-0020-4ee7-8018-1d067a8b90f1).html.

[2] Stephen Hendrick and Jim Zemlin. *The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness*. 2022. URL: https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness.

[3] James Wetter and Nicky Ringland. *Understanding the Impact of Apache Log4j Vulnerability*. 2021. URL: https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html (visited on 11/20/2022).

[4] Roshni R Ramnani, Karthik Shivaram, and Shubhashis Sengupta. "Semi-automated information extraction from unstructured threat advisories". In: *Proceedings of the 10th Innovations in Software Engineering Conference*. 2017, pp. 181–187.

[5] ConsenSys. *A Guide to Events and Logs in Ethereum Smart Contracts*. URL: https://consensys.net/blog/developers/guide-to-events-and-logs-in-ethereum-smart-contracts/ (visited on 05/18/2023).

[6] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach*. USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 020162432X.

[7] OASIS CSAF Technical Committee. *Common Security Advisory Framework Version 2.0*. 2022. URL: https://docs.oasis-open.org/csaf/csaf/v2.0/csaf-v2.0.html.

[8] OWASP Foundation. *CycloneDX*. URL: https://cyclonedx.org (visited on 05/23/2023).

[9] SPDX Workgroup. *Software Package Data Exchange*. URL: https://spdx.dev/ (visited on 05/23/2023).

[10] National Institute of Standards and Technology. *Software Identification (SWID) Tagging*. URL: https://csrc.nist.gov/projects/Software-Identification-SWID (visited on 05/23/2023).

[11] National Institute of Standards and Technology. *Cryptographic Standards and Guidelines*. URL: https://csrc.nist.gov/Projects/cryptographic-standards-and-guidelines (visited on 05/01/2023).

[12] Holepunch. *Hypercore*. URL: https://docs.holepunch.to/building-blocks/hypercore (visited on 03/24/2023).

[13] Storj. *Pricing*. URL: https://www.storj.io/pricing (visited on 03/24/2023).

[14] Sia Foundation. *About Renting on Sia*. URL: https://docs.sia.tech/renting/about-renting (visited on 03/24/2023).

[15] Viktor Trón. *The book of Swarm*. 2020. URL: https://www.ethswarm.org/The-Book-of-Swarm.pdf.

[16] Filecoin. *Retrieval market*. URL: https://docs.filecoin.io/basics/what-is-filecoin/retrieval-market/ (visited on 03/24/2023).

[17] Protocol Labs. *Pin files using IPFS*. URL: https://docs.ipfs.tech/how-to/pin-files/ (visited on 03/24/2023).

[18] Protocol Labs. *Working with pinning services*. URL: https://docs.ipfs.tech/how-to/work-with-pinning-services/ (visited on 03/24/2023).

[19] ArWiki. *What is Arweave?* URL: https://arwiki.wiki/ (visited on 05/05/2023).

[20] IPFS GitHub Community. *Using JS IPFS in the Browser*. URL: https://github.com/ipfs/js-ipfs/blob/master/docs/BROWSERS.md#limitations-of-the-browser-context (visited on 05/03/2023).

[21] IPFS. *Nodes*. URL: https://docs.ipfs.tech/concepts/nodes/#nodes (visited on 05/01/2023).

[22] Inc. Protocol Labs. *Address IPFS on the web*. URL: https://docs.ipfs.tech/how-to/address-ipfs-on-web/#dweb-addressing-in-brief (visited on 05/19/2023).

[23] Ethereum Foundation. *Nodes and clients*. URL: https://ethereum.org/en/developers/docs/nodes-and-clients/ (visited on 05/01/2023).

[24] The MITRE Corporation. *Process*. 2022. URL: https://www.cve.org/About/Process#CVERecordLifecycle (visited on 11/01/2022).

[25] National Institute of Standards and Technology. *Block Cipher Techniques*. URL: https://csrc.nist.gov/projects/block-cipher-techniques (visited on 05/02/2023).

[26] E. Barker and A. Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Tech. rep. NIST Special Publication (SP) 800-131A, Rev. 2. National Institute of Standards and Technology, 2019. DOI: 10.6028/NIST.SP.800-131Ar2.

[27] National Institute of Standards and Technology. *ADVANCED ENCRYPTION STANDARD (AES)*. Tech. rep. NIST Federal Information Processing Standards Publication (FIPS) 197. National Institute of Standards and Technology, 2001. DOI: 10.6028/NIST.FIPS.197.

[28] National Institute of Standards and Technology. *Block Cipher Modes*. URL: https://csrc.nist.gov/projects/block-cipher-techniques/bcm (visited on 05/04/2023).

[29] Mozilla. *SubtleCrypto: encrypt() method*. URL: https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/encrypt (visited on 05/04/2023).

[30] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. rep. NIST Special Publication (SP) 800-38D. National Institute of Standards and Technology, 2007. DOI: /10.6028/NIST.SP.800-38D.

[31] E. Barker et al. *Recommendation for Cryptographic Key Generation*. Tech. rep. NIST Special Publication (SP) 800-57, Part 1, Rev. 5. National Institute of Standards and Technology, 2020. DOI: /10.6028/NIST.SP.800-57pt1r5.

[32] E. Barker et al. *Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography*. Tech. rep. NIST Special Publication (SP) 800-56B, Rev. 2. National Institute of Standards and Technology, 2019. DOI: /10.6028/NIST.SP.800-56Br2.

[33] P. A. Grassi et al. *Digital Identity Guidelines - Authentication and Lifecycle Management*. Tech. rep. NIST Special Publication (SP) 800-63B. National Institute of Standards and Technology, 2020. DOI: /10.6028/NIST.SP.800-63b.

[34] M. S. Turan et al. *Recommendation for Password-Based Key Derivation*. Tech. rep. NIST Special Publication (SP) 800-132. National Institute of Standards and Technology, 2010. DOI: /10.6028/NIST.SP.800-132.

[35] OpenJS Foundation. *Node.js*. URL: https://nodejs.org/en (visited on 05/23/2023).

[36] Truffle Suite. *Truffle Suite*. URL: https://github.com/trufflesuite (visited on 03/10/2023).

[37] Meta Open Source. *React*. URL: https://react.dev (visited on 05/23/2023).

[38] Protocol Labs. *Kubo*. URL: https://github.com/ipfs/kubo (visited on 05/11/2023).

[39] Ethereum Foundation. *Smart contract security*. URL: https://ethereum.org/en/developers/docs/smart-contracts/security/ (visited on 05/12/2023).

[40] ConsenSys. *Ethereum Best Practices - General Philosophy*. URL: https://consensys.github.io/smart-contract-best-practices/general-philosophy/ (visited on 05/12/2023).

[41] ConsenSys. *Ethereum Best Practices - Development Recommendations*. URL: https://consensys.github.io/smart-contract-best-practices/development-recommendations/ (visited on 05/12/2023).

[42] OpenZeppelin. *Strings.sol*. URL: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Strings.sol (visited on 05/07/2023).

[43] OpenZeppelin. *The standard for secure blockchain applications*. URL: https://www.openzeppelin.com (visited on 05/05/2023).

[44] Inc. Protocol Labs. *Protocol Labs*. URL: https://protocol.ai (visited on 05/18/2023).

[45] Ethereum Foundation. *go-ethereum*. URL: https://geth.ethereum.org (visited on 05/15/2023).

[46] Ethereum Foundation. *go-ethereum*. URL: https://geth.ethereum.org/docs (visited on 05/15/2023).

[47] Sigma Prime. *Lighthouse GitHub*. URL: https://github.com/sigp/lighthouse (visited on 05/15/2023).

[48] Ether Alpha. *Client Diversity*. URL: https://clientdiversity.org (visited on 05/15/2023).

[49] Stack Overflow. *2022 Developer Survey*. URL: https://survey.stackoverflow.co/2022/#technology (visited on 04/25/2023).

[50] Bootstrap React Community. *Bootstrap React*. URL: https://react-bootstrap.github.io (visited on 05/18/2023).

[51] Ethereum. *Web3.js Documentation*. URL: https://web3js.readthedocs.io/en/v1.8.1/ (visited on 01/08/2023).

[52] Inc. npm. *npm*. URL: https://www.npmjs.com/ (visited on 05/23/2023).

[53] Mozilla Foundation. *SubtleCrypto: deriveKey() method*. URL: https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/deriveKey (visited on 05/18/2023).

[54] Dang. Q. *Recommendation for Applications Using Approved Hash Algorithms*. Tech. rep. NIST Special Publication (SP) 800-107, Rev. 1. National Institute of Standards and Technology, 2012. DOI: /10.6028/NIST.SP.800-107r1.

[55] Mozilla Foundation. *Crypto: getRandomValues() method*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues (visited on 05/18/2023).

[56] OWASP CheatSheets Series Team. *Password Storage Cheat Sheet*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (visited on 05/09/2023).

[57] Inc. GitHub. *GitHub*. URL: https://github.com (visited on 05/15/2023).

[58] Inc. GitHub. *About workflows*. URL: https://docs.github.com/en/actions/using-workflows/about-workflows (visited on 05/15/2023).

[59] Inc. GitHub. *Dependabot*. URL: https://github.com/dependabot/dependabot-core (visited on 05/18/2023).

[60] Trail of Bits. *Slither, the Solidity source analyzer*. URL: https://github.com/crytic/slither (visited on 05/18/2023).

[61] J. Feist, G. Grieco, and A. Groce. "Slither: A Static Analysis Framework For Smart Contracts". In: (2019). arXiv: 1908.09878.

[62] Trail of Bits. *Slither Action*. URL: https://github.com/marketplace/actions/slither-action (visited on 05/18/2023).

[63] Truffle Suite. *Ganache*. URL: https://github.com/trufflesuite/ganache (visited on 03/10/2023).

[64] The MITRE Corporation. *Metrics*. URL: https://www.cve.org/About/Metrics (visited on 04/18/2023).

[65] The MITRE Corporation. *Metrics*. URL: https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all&isCpeNameSearch=false (visited on 04/18/2023).

[66] Etherscan. *Ethereum Daily Transactions Chart*. URL: https://etherscan.io/chart/tx (visited on 04/18/2023).

[67] Etherscan. *Ethereum Network Pending Transactions Chart*. URL: https://etherscan.io/chart/pendingtx (visited on 04/18/2023).

[68] Ethereum Foundation. *Scaling Ethereum*. URL: https://ethereum.org/en/roadmap/scaling/ (visited on 04/18/2023).

[69] Ethereum Foundation. *GAS AND FEES*. URL: https://ethereum.org/en/developers/docs/gas/ (visited on 04/18/2023).

[70] Etherscan. *GAS AND FEES*. URL: https://etherscan.io/gastracker (visited on 04/18/2023).

[71] Ethereum Foundation. *Single slot finality*. URL: https://ethereum.org/ph/roadmap/single-slot-finality/ (visited on 05/31/2023).

[72] OASIS CSAF TC. *bsi-2022-0001.json*. URL: https://github.com/oasis-tcs/csaf/blob/master/csaf_2.0/examples/csaf/cisco-sa-20180328-smi2.json (visited on 05/23/2023).

[73] Smart Contract Security. *Smart Contract Weakness Classification Registry*. URL: https://github.com/SmartContractSecurity/SWC-registry/ (visited on 06/07/2023).

[74] SmartContractSecurity. *SWC-107*. URL: https://swcregistry.io/docs/SWC-107 (visited on 05/24/2023).

[75] SmartContractSecurity. *SWC-114*. URL: https://swcregistry.io/docs/SWC-114 (visited on 05/30/2023).

[76] SmartContractSecurity. *SWC-128*. URL: https://swcregistry.io/docs/SWC-128 (visited on 05/25/2023).

[77] ConsenSys. *Mythril*. URL: https://github.com/ConsenSys/mythril (visited on 05/23/2023).

[78] Ehtereum Foundation. *Sepolia Resources*. URL: https://sepolia.dev (visited on 04/25/2023).

[79] L. Bošnjak, J. Sreš, and B. Brumen. "Brute-force and dictionary attack on hashed real-world passwords". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 1161–1166. DOI: 10.23919/MIPRO.2018.8400211.

Appendix A

Frontend Mockups

This appendix chapter includes mockups of the various frontend pages in SENTINEL.

A.1 Vulnerabilities Page Mockup

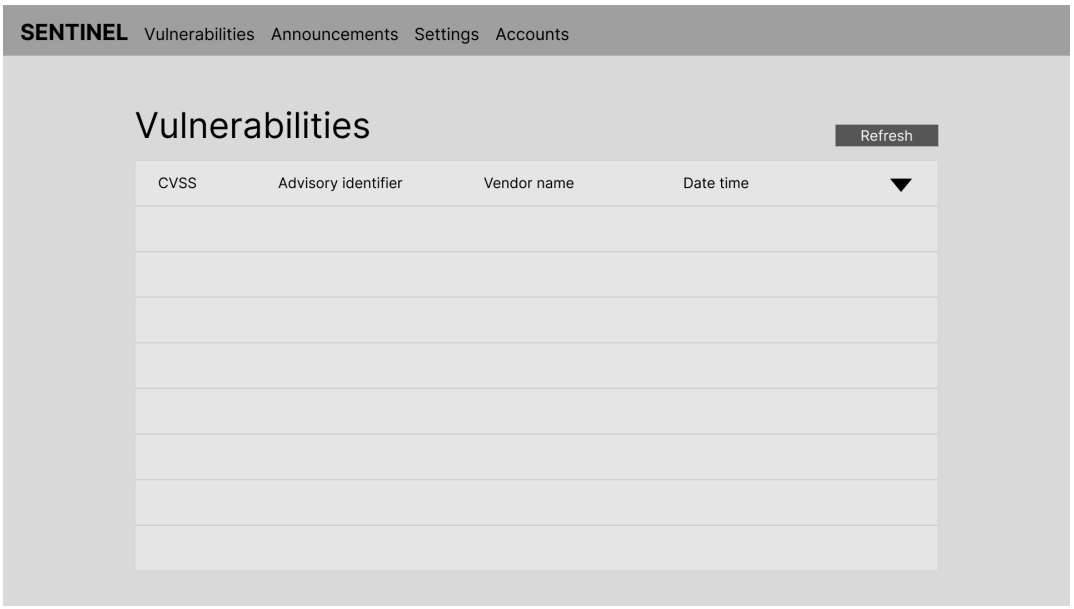

Figure A.1: Mockup of the vulnerabilities page on the frontend.

A.2 Accounts Page Mockup

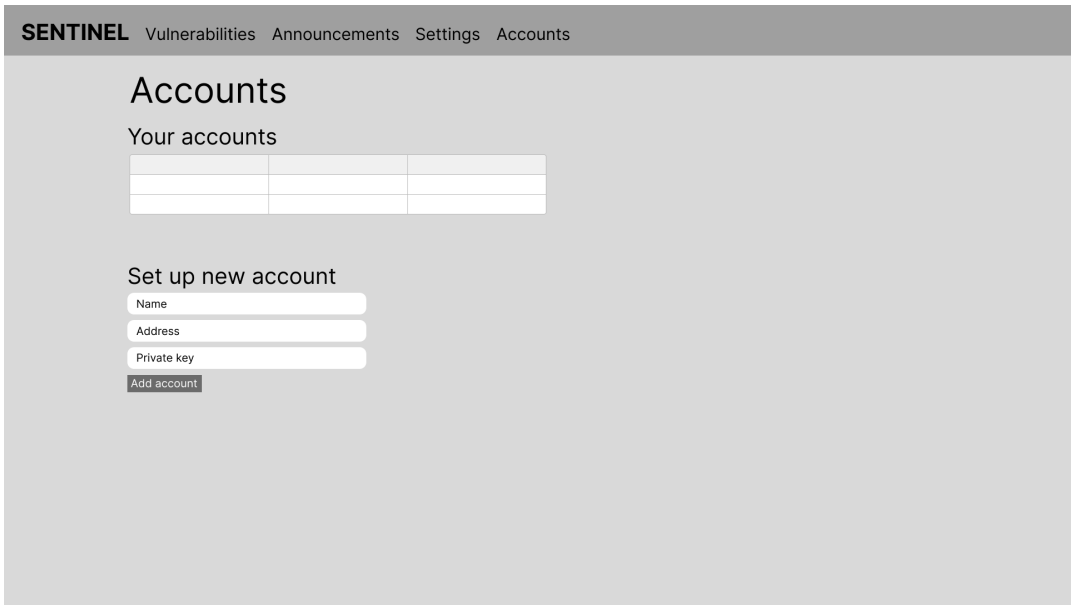

Figure A.2: Mockup of the accounts page on the frontend.

A.3 Settings Pages Mockup

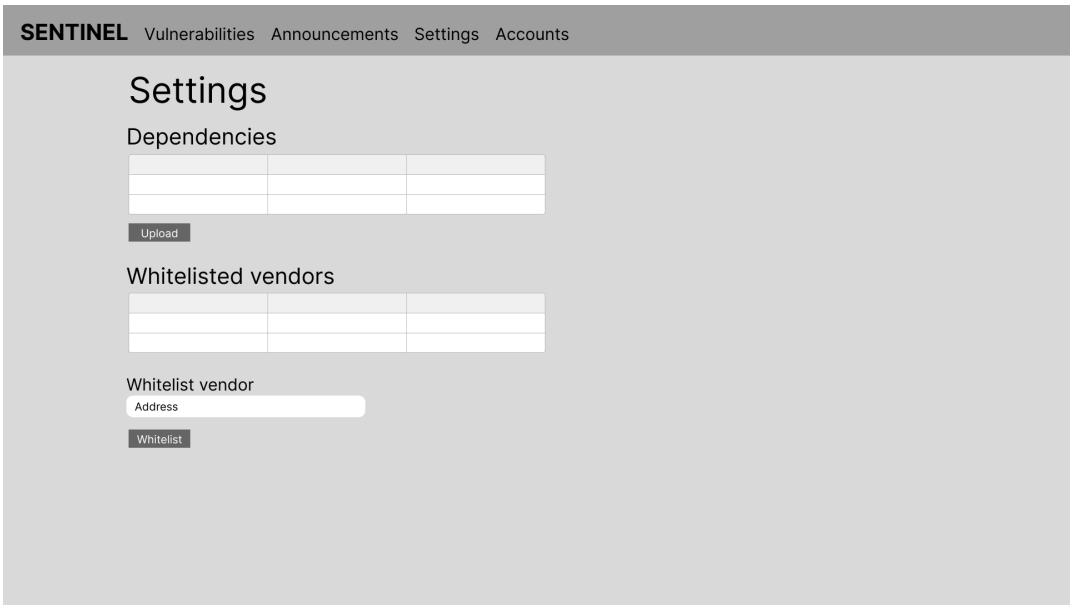

Figure A.3: Mockup of the settings page for the public use case on the frontend.

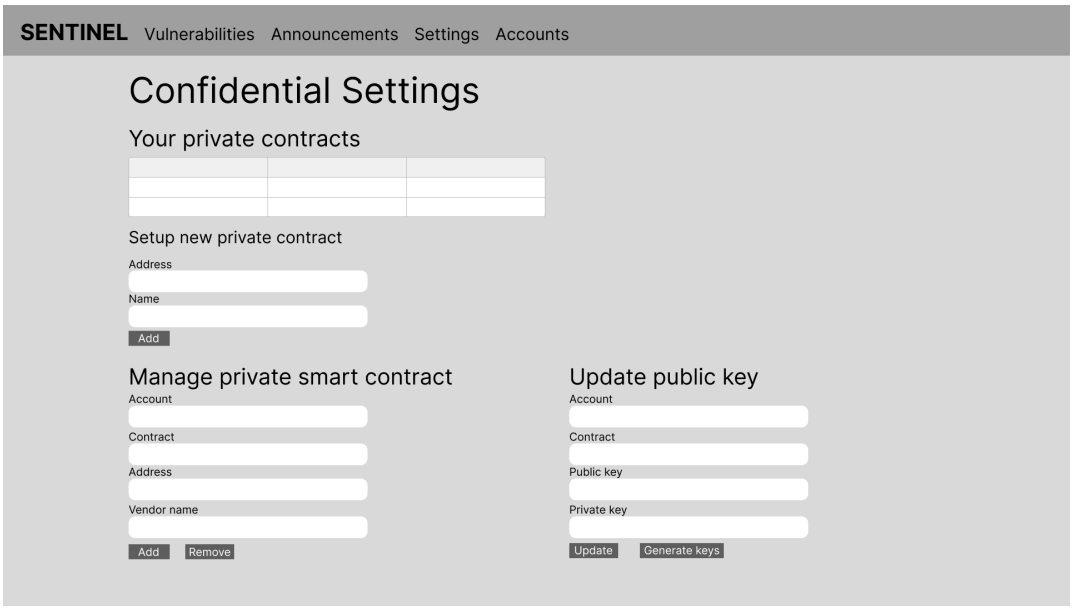

Figure A.4: Mockup of the settings page for the private use case on the frontend.

A.4 Announcement Pages Mockup

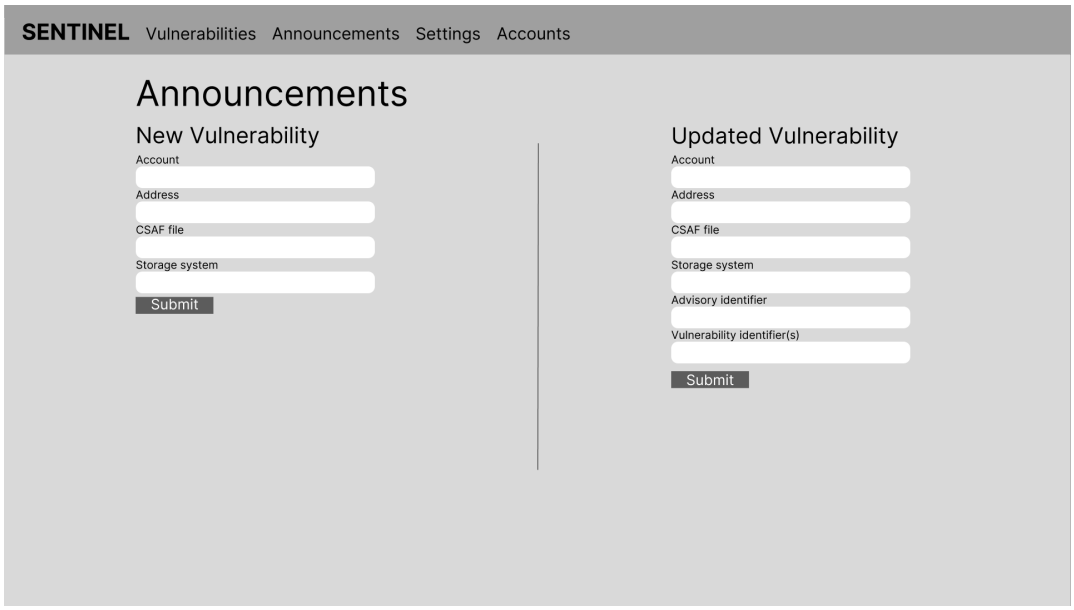

Figure A.5: Mockup of the announcement page for the public use case on the frontend.

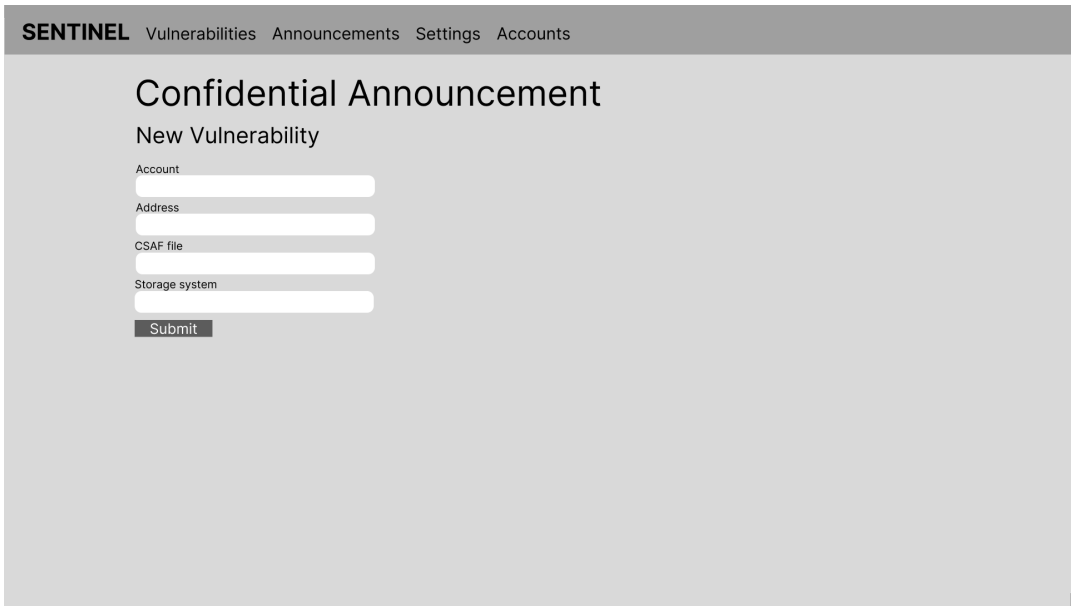

Figure A.6: Mockup of the announcement page for the private use case on the frontend.

Appendix B

Frontend Screenshots

This appendix chapter contains screenshots of the various pages of the SENTINEL frontend.

B.1 Vulnerabilities Page Screenshots

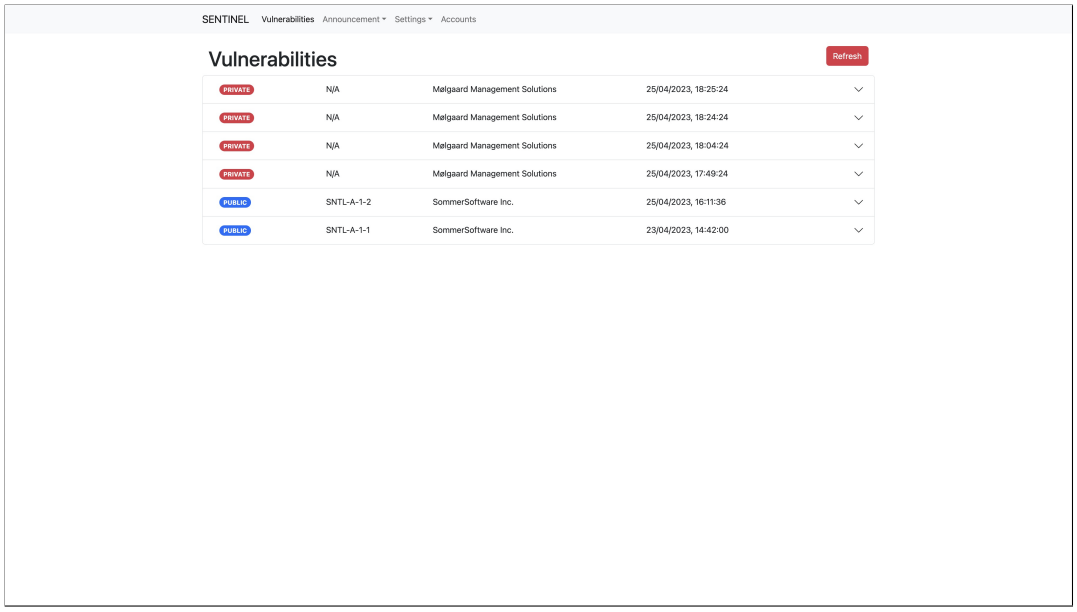

Figure B.1: Screenshot of the vulnerabilities page on the SENTINEL frontend.

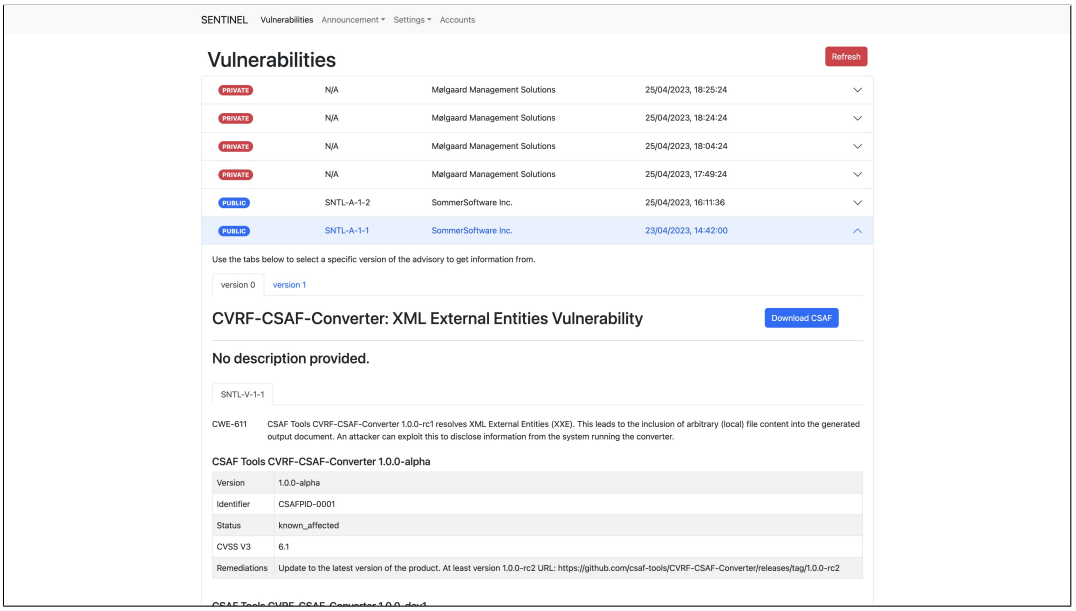

Figure B.2: Screenshot of the vulnerabilities page on the SENTINEL frontend with a public security advisory selected and detailed.

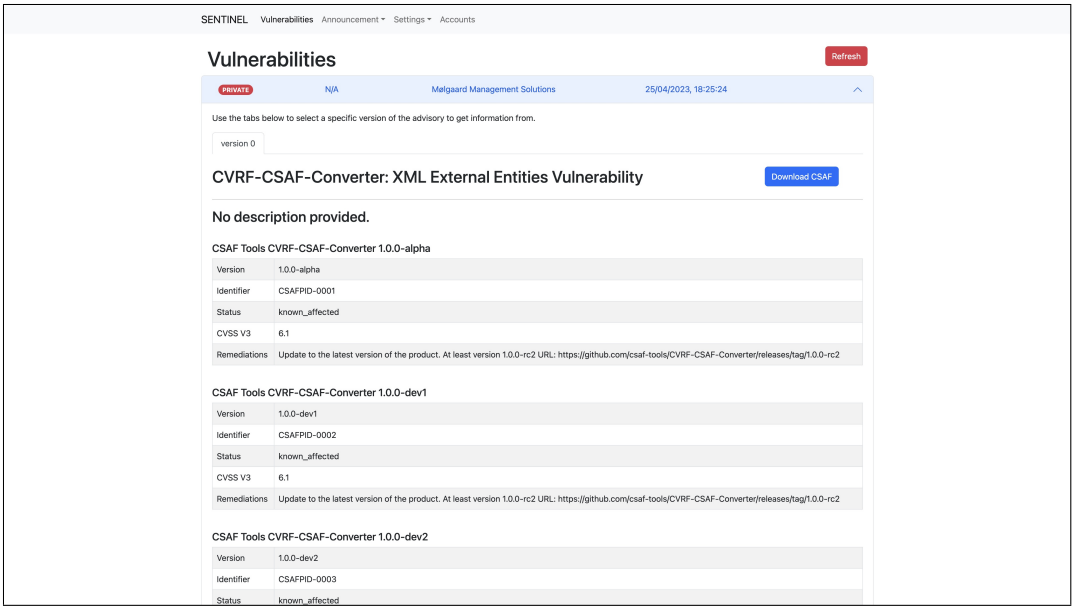

Figure B.3: Screenshot of the vulnerabilities page on the SENTINEL frontend with a confidential security advisory selected and detailed.

B.2 Accounts Page Screenshots

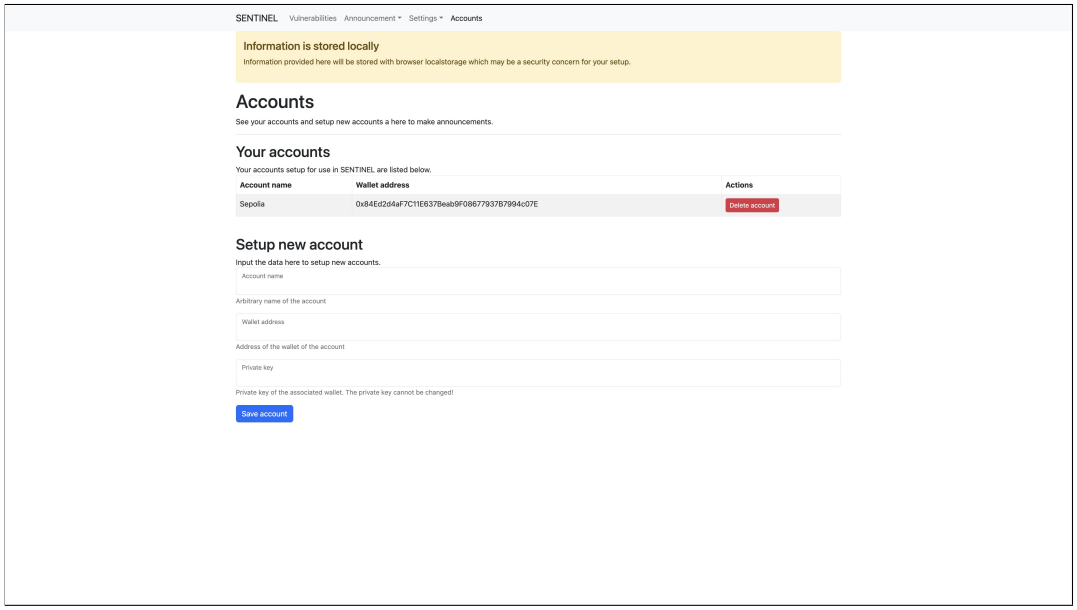

Figure B.4: Screenshot of the accounts page on the SENTINEL frontend.

B.3 Settings Pages Screenshots

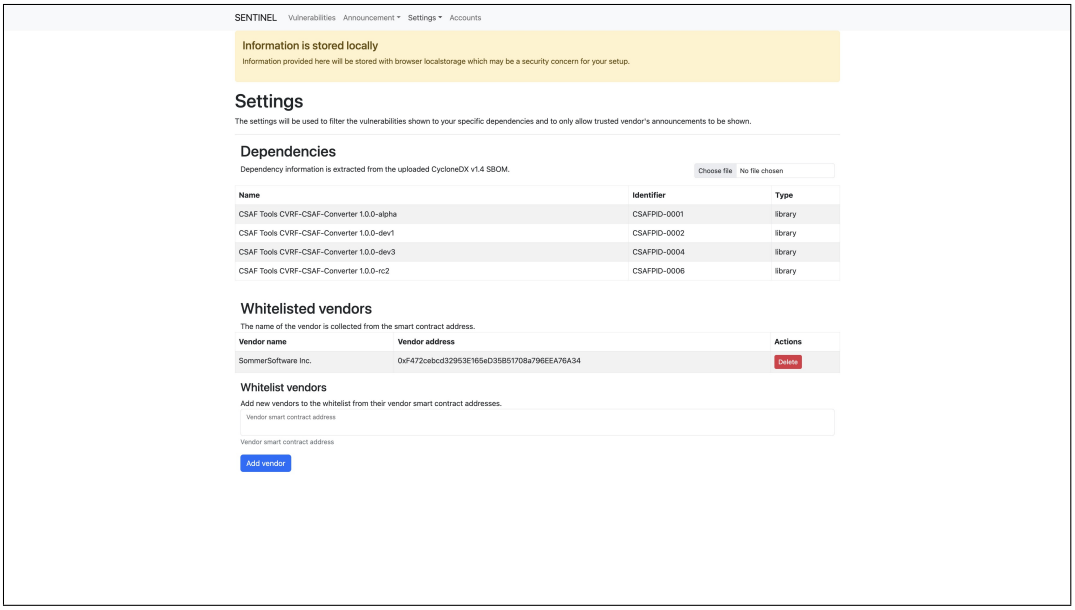

Figure B.5: Screenshot of the public settings page on the SENTINEL frontend.

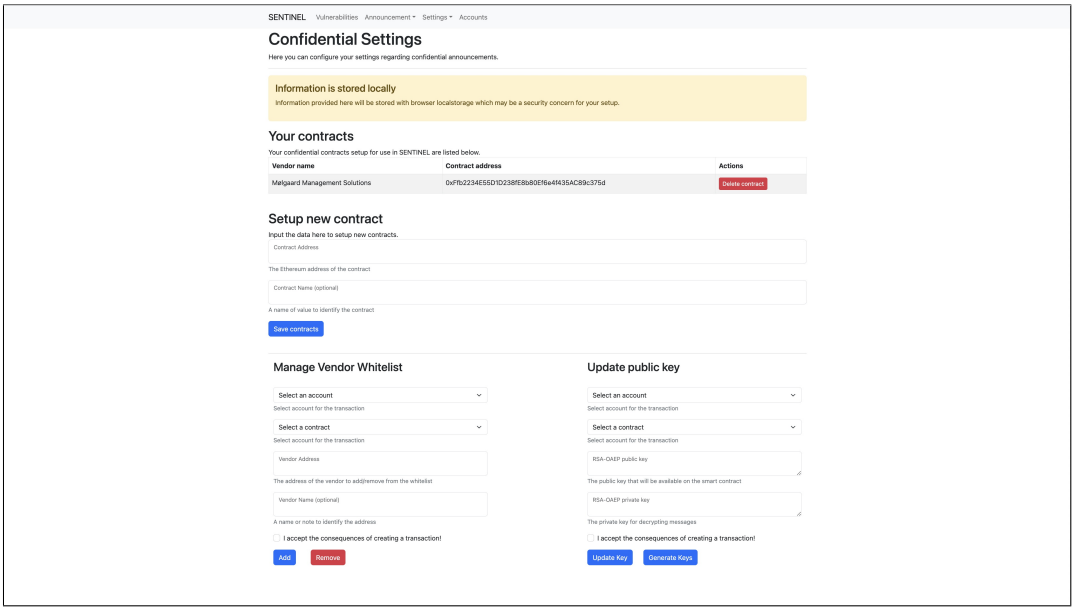

Figure B.6: Screenshot of the confidential settings page on the SENTINEL frontend.

B.4 Announcement Pages Screenshots

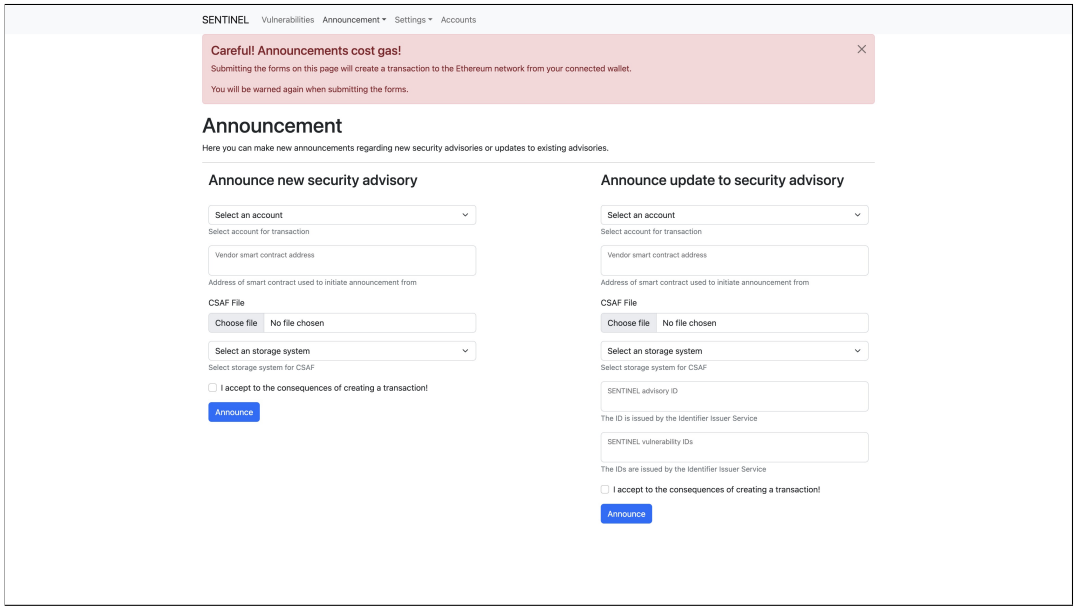

Figure B.7: Screenshot of the public announcements page on the SENTINEL frontend.

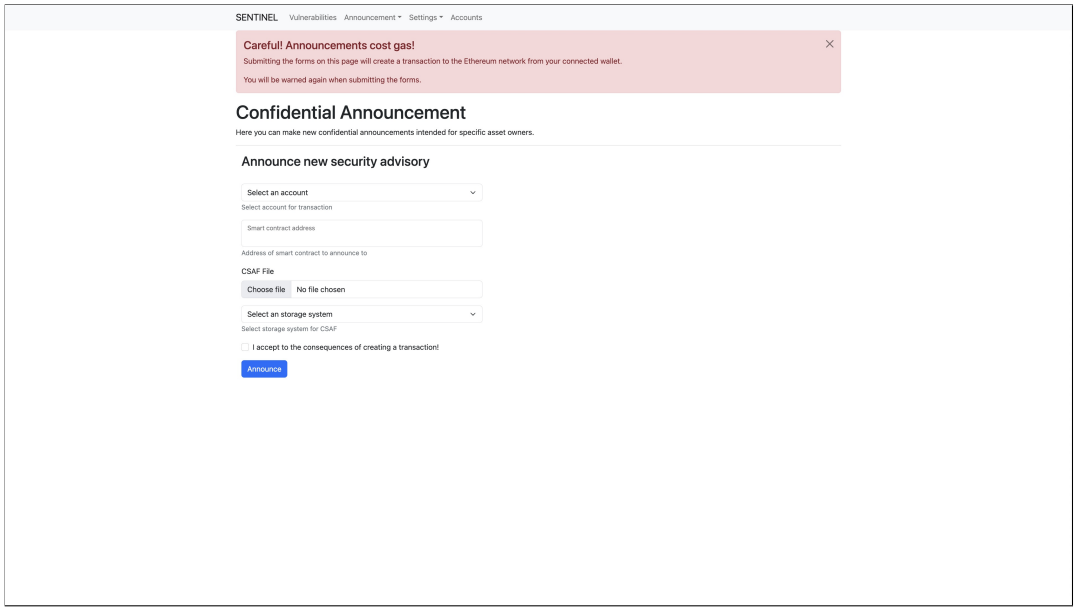

Figure B.8: Screenshot of the confidential announcements page on the SENTINEL frontend.

Appendix C

GitHub Workflows

This appendix chapter contains the GitHub Workflows utilized for continuous integration and continuous deployment pipelines.

C.1 Dependabot Update Script

Listing C.1: Dependabot workflow to keep GitHub Actions up to date.

```yaml
version: 2
updates:
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "daily"
```

C.2 Slither Smart Contract Analysis

Listing C.2: GitHub workflow file to run smart contract test with Truffle.

```yaml
name: Slither Analysis
on: pull_request
jobs:
  analyze:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup NodeJS 16
        uses: actions/setup-node@v3
```

```
10          with:
11            node-version: 16.x
12        - name: Install Truffle
13          run: npm install truffle@5.7.6 -g
14        - name: Install Dependencies
15          run: npm install
16        - name: Truffle Compile
17          run: truffle compile
18        - name: Run Slither Analysis
19          uses: crytic/slither-action@v0.3.0
20          id: slither
21          with:
22            sarif: results.sarif
23            fail-on: none
24            slither-args: '--ignore-compile --filter-paths
    openzeppelin'
25        - name: Upload SARIF file
26          uses: github/codeql-action/upload-sarif@v2
27          with:
28            sarif_file: ${{ steps.slither.outputs.sarif }}
```

C.3 Truffle Smart Contract Test

Listing C.3: GitHub workflow file to run smart contract test with Truffle.

```
1  name: Truffle Build
2  on:
3    pull_request:
4  jobs:
5    build:
6      runs-on: ubuntu-latest
7      steps:
8      - uses: actions/checkout@v3
9      - name: Setup NodeJS 16
10       uses: actions/setup-node@v3
11       with:
12         node-version: 16.x
13     - name: Show NodeJS version
14       run: npm --version
15     - name: Install Truffle
16       run: npm install truffle@5.7.6 -g
```

```
17      - name: Install Truffle Dependencies
18        run: npm install
19      - name: Run Truffle Test
20        run: truffle test
```

C.4 Frontend Integration Test

Listing C.4: GitHub workflow file to run frontend integration tests.

```
1  name: Frontend Tests
2  on:
3    pull_request:
4  jobs:
5    test:
6      runs-on: ubuntu-latest
7      steps:
8      - uses: actions/checkout@v3
9      - name: Setup NodeJS 16
10       uses: actions/setup-node@v3
11       with:
12         node-version: 16.x
13     - name: Show NodeJS version
14       run: npm --version
15     - name: Install Dependencies
16       run: npm install --prefix ./frontend
17     - name: Install Ganache CLI
18       run: npm install -g ganache
19     - name: Run Tests
20       run: npm test --prefix ./frontend
```

C.5 Frontend IPFS Deployment

Listing C.5: GitHub workflow file to deploy the frontend to IPFS.

```
1  name: Frontend IPFS deployment
2  on:
3    push:
4      branches:
5        - main
6  jobs:
```

```
7   deploy:
8     runs-on: ubuntu-latest
9     steps:
10      - uses: actions/checkout@v3
11      - name: Setup NodeJS 16
12        uses: actions/setup-node@v3
13        with:
14          node-version: 16.x
15      - name: Install frontend dependencies
16        run: npm install --prefix ./frontend
17      - name: Build frontend
18        run: npm run build --prefix ./frontend
19      - name: Install deployment dependencies
20        run: npm install --prefix ./.github/scripts/
    deployment
21      - name: Publish to IPFS
22        run: node ./.github//scripts/deployment/publish.js $
    {{ secrets.IPFS_HOST }} ./frontend/build
```

Listing C.6: publish.js Javascript code used in the frontend IPFS deployment workflow.

```
1 import {create, globSource} from "ipfs-http-client";
2
3 const ipfs = await create({url: process.argv[2]});
4 console.log("IPFS node connection established");
5
6 //options specific to globSource
7 const globSourceOptions = { recursive: true };
8
9 //example options to pass to IPFS
10 const options = { wrapWithDirectory: true, timeout: 60000 };
11
12 var results = [];
13 for await (const file of ipfs.addAll(globSource(process.argv
    [3].toString(), "**/*", globSourceOptions), options)) {
14     results.push(file);
15 }
16 console.log("IPFS directory/files added to host node.");
17
18 for await (let result of results) {
19     if (result.path === '') { // this is the wrapping
```

```
    directory
20          const res = await ipfs.name.publish("/ipfs/" + result
    .cid.toString(), {key: "self"});
21          console.log("IPNS key updated.");
22          console.log(`https://gateway.ipfs.io/ipns/${res.name
    }`);
23      }
24  }
```

Appendix D

System Test Plan

This appendix chapter contains the test plan used in system testing. The test plan is divided into two use cases, namely the public use case and the private use case. Each use case covers different features and thereby require different test to verify.

D.1 Common Features

These features are used in both the public and private use case.

Choose password	
Description	A user of SENTINEL must create a password the first time they visit the webpage.
How to conduct	Load the webpage with no data in local storage. When prompted for a password, type in a password.
Success criteria	The password modal will close, and the information is reflected in local storage.

Login with password	
Description	A user of SENTINEL will be prompted to log in to the webpage with their chosen password. The users should provide the correct password.
How to conduct	Open the webpage and type in the password.
Success criteria	The password modal will close, and the user can interact with the system and settings which have been encrypted with the password.

Settings: Upload dependencies from SBOM	
Description	An asset owner should be able to upload their dependencies from a CycloneDX v1.4 SBOM document on the settings page.
How to conduct	Navigate to the settings page. Find and click on the file upload button under "Dependencies". Choose the right SBOM document and click "Upload".
Success criteria	The uploaded and parsing of the SBOM document is successful if the UI is updated with the dependencies are shown in a table under "dependencies" and is reflected in local storage.

Accounts: Add new account	
Description	A user of SENTINEL should be able to set up an account that can be used to make transactions from.
How to conduct	Navigate to the "Accounts" page and locate the "Setup new account" section. Fill out the form with the required information and press "Save account".
Success criteria	The table of accounts is updated with the new account, the information is reflected in local storage, and the account can be chosen on other pages to make transactions from.

Accounts: Remove account	
Description	A user of SENTINEL should be able to remove an account when they would no longer use it.
How to conduct	Navigate to the "Accounts" page, locate the account that should be removed, and press "Delete account".
Success criteria	The table of accounts is updated without the removed account, the information is reflected in local storage, and the account can no longer be chosen on other pages to make transactions from.

Vulnerabilities: Refresh vulnerabilities	
Description	An asset owner should be able to manually refresh the vulnerabilities found on the vulnerabilities page.
How to conduct	Navigate to the "Vulnerabilities" page and press "Refresh".
Success criteria	Assuming that any vulnerability announcements are displayed, the page will clear of all vulnerabilities and repopulate with the relevant vulnerability announcements.

Vulnerabilities: Download security advisory	
Description	An asset owner should be able to download the security advisory to find details not presented by SENTINEL.
How to conduct	Navigate to the "Vulnerabilities" page and select a vulnerability announcement. In the vulnerability details, locate the "Download CSAF" button and press it.
Success criteria	The selected security advisory will be downloaded as a CSAF v2.0 document.

D.2 Public Use Case

These features are used in the public use case.

Settings: Add vendor to whitelist	
Description	Asset owners must whitelist vendor smart contracts on the settings page in order to have their security advisories displayed on the vulnerabilities page.
How to conduct	Navigate to the "Settings" menu and choose "General" and locate the "Whitelisted vendors" section. Fill out the form to add a new vendor to the whitelist and press the "Add" button.
Success criteria	The vendor smart contract is whitelisted when it is added to the list of whitelisted vendors in the "Whitelisted vendors" section, the information is reflected in local storage, and the vendor's security advisories are discoverable on the "Vulnerabilities" page.

Settings: Remove vendor from whitelist	
Description	Asset owners can remove vendors from the whitelist when they no longer want to subscribe to their security advisory announcements.
How to conduct	Navigate to the "Settings" menu and choose "General". Locate the "Whitelisted vendors" section and find the whitelisted vendor to remove and press the "Delete" button.
Success criteria	The vendor smart contract is no longer whitelisted when the table of whitelisted vendors is updated with the removed vendor no longer present, the information is reflected in local storage, and the vendor's security advisories are no longer discoverable on the "Vulnerabilities" page.

Announcement: Announce new public security advisory	
Description	A vendor should be able to announce a new security advisory in the form of a CSAF v2.0 document.
How to conduct	Navigate to the "Announcement" menu and choose "Public". Fill in required information in the "Announce new security advisory" form. Press "Announce" and agree to the consequences.
Success criteria	A receipt is displayed upon announcing the security advisory, a "NewSecurityAdvisory" event is emitted on the Ethereum network, and the security advisory can be accessed on IPFS using the given CID.

Announcement: Announce updated public security advisory	
Description	A vendor should be able to announce a security advisory update in the form of a CSAF v2.0 document that is linked to an existing advisory identifier and vulnerability identifier(s).
How to conduct	Navigate to the "Announcement" menu and choose "Public". Fill in required information in the "Announce updated security advisory" form. Press "Announce" and agree to the consequences.
Success criteria	A receipt is displayed upon announcing the security advisory, a "UpdatedSecurityAdvisory" event is emitted on the Ethereum network, and the security advisory can be accessed on IPFS using the given CID.

Vulnerabilities: Discover and retrieve relevant public security advisories	
Description	An asset owner should be able to discover relevant security advisories based on their dependencies and vendor whitelist. Additionally, the asset owner should be able to retrieve and read the security advisories.
How to conduct	Assuming proper setup of general settings and an announced advisory. Navigate to the "Vulnerabilities" page. Click on the advisory on the page.
Success criteria	An advisory entry can be seen on the "Vulnerabilities" page showing the expected advisory identifier, vendor name, and announcement date. CSAF v2.0 information is displayed when expanding the advisory entry.

D.3 Private Use Case

These features are only related to the private use case.

Confidential settings: Setup private smart contract	
Description	Asset owners should be able to set up private smart contracts if they want to receive security advisories from in the confidential settings page.
How to conduct	Navigate to the confidential settings page and locate the "Setup new contract" section. Fill out the form and press "Save contract" button.
Success criteria	The private smart contract added when the table of confidential contracts is updated, the information is reflected in local storage, and any advisories announced from the private smart contract are displayed on the "Vulnerabilities" page.

Confidential settings: Remove private smart contract	
Description	Asset owners should be able to remove private smart contracts from their settings when they no longer want security advisory announcements from that smart contract.
How to conduct	Navigate to the "Confidential" menu and choose "Confidential". Locate the private smart contract in the table under "Your contracts" section and press "Delete contract".
Success criteria	The private smart contract is removed when the table is updated with the removed contract no longer present, the information is reflected in local storage, and any advisories announced from the private smart contract are no longer displayed on the "Vulnerabilities" page.

Confidential settings: Add vendor to whitelist	
Description	Asset owners should be able to whitelist specific addresses that can interact with the private smart contract for a vendor to announce security advisories on that particular smart contract.
How to conduct	Navigate to the "Settings" menu and select "Confidential". Fill out the form under the "Manage Vendor Whitelist" section and press "Add".
Success criteria	A transaction receipt will be shown, and the address will be added to a list of whitelisted vendors on the smart contract.

Confidential settings: Remove vendor from whitelist	
Description	Asset owners should be able to remove addresses from the whitelist on the private smart contract when a vendor should no longer announce security advisories to it, from that specific address.
How to conduct	Navigate to the "Settings" menu and select "Confidential". Fill out the form under the "Manage Vendor Whitelist" section and press "Remove".
Success criteria	A transaction receipt will be shown, and the address will be blacklisted on the private smart contract.

Confidential Settings: Update RSA-OAEP keys	
Description	An asset owner should be able to generate and update RSA-OAEP key pairs related to a specific private smart contract.
How to conduct	Navigate to the "Settings" menu and select "Confidential". Fill out the "Update public key" form. Click "Generate Keys". Click Update.
Success criteria	A receipt is displayed upon updating the keys, the public key is written to the "publicKey" state variable on the private smart contract, and the private key is written to local storage.

Announcement: Announce confidential security advisory	
Description	A vendor should be able to announce a confidential security advisory to a specific asset owner in the form of a CSAF v2.0 document.
How to conduct	Navigate to the "Announcement" menu and choose "Confidential". Fill in required information in the "Announce new security advisory" form. Press "Announce" and agree to the consequences.
Success criteria	A receipt is displayed upon announcing the security advisory, a "ConfidentialSecurityAdvisory" event is emitted on the Ethereum network, and the security advisory can be accessed on IPFS using the given CID only by the asset owner.

Vulnerabilities: Discover and retrieve relevant confidential security advisories	
Description	An asset owner should be able to discover relevant confidential security advisories from vendors. Additionally, the asset owner should be able to retrieve and read the security advisories.
How to conduct	Assuming proper setup of confidential settings and an announced advisory. Navigate to the "Vulnerabilities" page. Click on the advisory on the page.
Success criteria	An advisory entry can be seen on the "Vulnerabilities" page showing the expected advisory identifier, vendor name, and announcement date. CSAF v2.0 information is displayed when expanding the advisory entry.

Appendix E

Development Process

Due to the nature of the project and the lack of experience for the specific development domain, the process of the project will be AGILE. This allows the development of the project to be flexible for the large unknown problem domain of smart contract development.

E.1 The Backlog

The development is divided into backlog items. These items are broken down to small objectives which are prioritized for different sprints, based on the sprint goal. Backlog items are both implementation features for the development of the SENTINEL components, and report sections.

The backlog is used to get an overview of remaining work, both the overall project and current sprint, and to track what work has been completed in the individual sprints.

E.2 Iterations

We want to develop in sprints to keep the iterations short, but also long enough to have substantial work. Each sprint will consist of two working weeks excluding the weekends, for a total of 10 working days for each sprint. At the start of each sprint, sprint planning will take place, where the backlog items will be refined and selected for the upcoming sprint. When the sprint ends, the sprint review and retrospective will begin. In the review, the work of the last sprint will be evaluated. The overall process will be evaluated in the retrospective, where any issues will be raised, and a potential solution will be suggested.

E.3 Peer-reviews

Everything will be peer-reviewed to ensure the quality is high. Several methods will be employed in this matter. Pull-requests will be used as a means to ensure that each piece of new code is reviewed at least once by another developer. Pair programming will be employed during implementation of smart contracts due to our inexperience with Solidity and smart contract development. Regarding the report, each new section is also peer-reviewed and revised in order to ensure a consistent quality.

E.4 Documentation

Sprint reviews and retrospectives will be documented to monitor progress. Additionally, daily logs will be recorded for future recollection and understanding. These will be compiled into a single document for each sprint to describe it in detail.

E.5 Time Allocation

In the beginning of the project, most of the time will be spent with implementation and iterating upon the system. This will help the development in the right direction as we get more knowledge of the domain and can make better design and implementation as soon as possible.

The design is largely based on previous work, but will be iterated on during the implementation as changes are necessary. The planned time allocation for development is illustrated in Figure E.1.

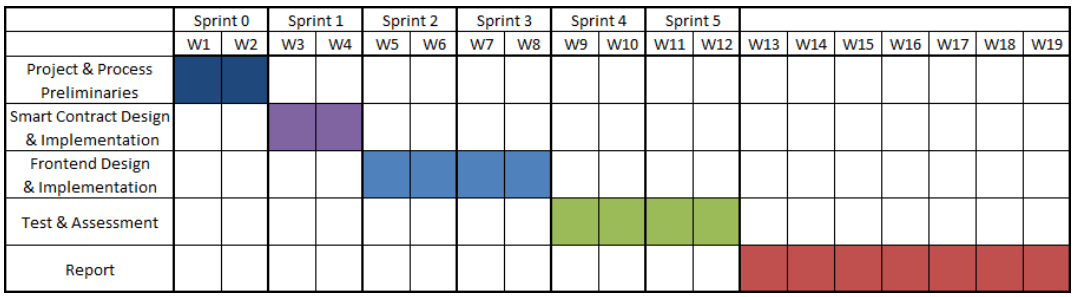

Figure E.1: A Gantt chart of the planned project time allocation.

In Figure E.2 a timeline of the project time allocation is illustrated. This timeline depicts the actual time allocation in the project.

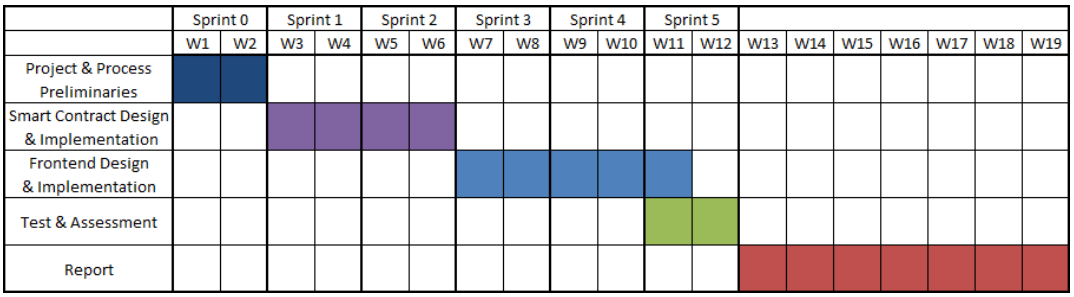

Figure E.2: A Gantt chart of the actual project time allocation.