In-Depth: Component Analysis and Vulnerability Management for Applications and Containers

Dylan Butler Parry (20099082)

BSC (HONS) COMPUTER FORENSICS AND SECURITY
Supervisor: Lucy White

Academic Integrity Declaration

I certify that this assignment is all my own original work and contains no plagiarism. By submitting this assignment, I agree to the following terms:

Any text, diagrams or other material copied from other sources (including, but not limited to, books, journals, and the Internet) have been clearly acknowledged and referenced as such in the text. These details are then confirmed by a fuller reference in the bibliography.

I have read the sections on referencing and plagiarism in the handbook or in the SETU Plagiarism policy and I understand that only assignments which are free of plagiarism will be awarded marks.

I further understand that SETU has a plagiarism policy which can lead to the suspension or permanent expulsion of students in serious cases.

Student: Dylan Butler Parry

Table of Contents

Abstract

This paper explores the modern-day supply chain, specifically investigating the vulnerabilities that can be introduced through it, how to best detect these weaknesses, and then how to use this information to better protect an organisation. By addressing the gaps in existing solutions, this paper presents a full-stack application that uses SBOM generation, vulnerability detection and vulnerability management into a unified platform. Presenting a user-friendly interface in conjunction with real-time alerting and effectively portraying the data, this solution aims to reduce response times to vulnerabilities and offer a proactive approach to mitigating these vulnerabilities and protecting organisations.

Investigation

Description

The aim of this project is to develop a full-stack application which can be used by application developers to improve supply chain security in their applications. This application will be able to be used to detect all components (and subcomponents) within supplied source code, find any associated vulnerabilities and weaknesses, and portray them to the user in a way that they can best utilise the information. It will be a combination of a software bill of materials (SBOM) tool and vulnerability management dashboard, allowing the user to keep track of components and vulnerabilities on a per-application basis.

Objectives

This project has several objectives which are required to be met to provide the intended functionality. It should be able to parse source code, which will be used to determine the list of third-party components embedded within. These components, for example, would be libraries being imported through node package manager or pip. These components are then stored within a database for the associated version of the application. For each new version of the application, the parser will need to be ran again as components or subcomponents may have updated their versions, or new libraries may have been brought into the application.

In-depth will then need to regularly check for any associated vulnerabilities within these components. Using databases of known vulnerabilities, the components can be cross referenced against these databases to determine if the user supplied application contains these vulnerabilities. In-depth will then notify the user (via email or SMS) should any new vulnerabilities arise. Notifications will also be severity based, for example, if a new vulnerability had a CVSS score indicating it is critical, notifications would be sent to multiple people to allow remediation and mitigation efforts to begin sooner.

This project will also require a front-end which clearly shows the user which components their application is using, and where their supply chain weaknesses are; A dashboard for showing all applications that a user/organisation is monitoring, allowing them to drill-down into each application to see all components. These components can then be filtered based on criteria, such as if they are no longer detected within the application source code, if they have associated known vulnerabilities, and where they are being pulled in from (pip, NPM, Unix binaries). The user can then drill down even further into each component to see if there are any

known vulnerabilities associated with that version of their component – which will then highlight to the user the severity and impact of having that component within their application.

Why SBOM and the Supply Chain?

The supply chain is increasingly becoming a major component in software development – vulnerabilities within are arising more frequently and with greater consequences than in the past, being highlighted by the attacks on Log4j and XZ (for different reasons respectively). SBOM is also now a requirement for all software supplied to federal agencies within the US, and all vendors supplying software to a federal agency must procure an SBOM for that software (U.S. Government, 2021).

Log4j, or the attack known as "Log4Shell", had a high impact due to its extensive usage within millions of packages globally directly, or indirectly as a transitive dependency, while allowing attackers to create a remote shell through a low complexity attack (NCSC Ireland, 2021). While developers wrote the code for their applications, the vulnerability was introduced through a trusted third-party. At the time, usage of SBOM data would have been even more infrequent than now due to the lack of tooling, meaning many organisations were faced with the task of retroactively finding this component to patch it out of systems.

XZ highlights a different kind of weakness within the supply chain that we should be aware of and should monitor for attacks. In the case of XZ, it's an open-source Unix binary used for compression and decompression, is preinstalled on several distributions of Linux, while also being embedded within many Docker containers. In this case, an attacker exploited the trust relationship instead of the code; Since the XZ is open-source and has an old, but stable codebase, this left few maintainers working on it. An attacker was able to push a backdoor into this codebase, which was then installed onto systems silently, leading to the CVE-2024-3094 vulnerability with a CVSS of 10 (NIST, 2024).

In the realm of tooling for this kind of vulnerability detection and management, there is a lack of a full-stack tooling where all corners are covered; There are several open-source solutions that solve the problem of detection, and then portraying that data to its users, but there are no managed services available to solve this problem. Using this approach, higher levels of third-party integrations could be achieved as well as streamlining the workflow for organisations. Notably, using a managed service would close the gap between detection and response lowering it through real-time alerting, and concise component tracking.

Current Solutions

SBOM Data

Currently, there are several ways that an organisation can begin to gather and process SBOM data to harness some value from it, though many of the components are often isolated and don't provide a full management interface for the data. Popular methods of gathering the SBOM data include the following tools:

1. Anchore/Syft
2. OWASP Dependency-Check
3. Microsoft/sbom-tool

These tools will allow a user to generate the list of components and subcomponents within their software but leave it up to the user as to how they want to process the data. These tools can export the data in several formats, but the popular choices are CycloneDX and Json, and typically contain the following information:

Field	Description
Type	The type of component (e.g. library, framework, operating system)
Name	The name of the component (e.g. numpy, XZ)
Version	The version of the component
Description	A short description of the component
License	The licenses associated with the component (e.g. MIT, GPL-3.0)
Purl (Package URL)	A URL used to identify the unique component (name, version, source)
Hashes	Cryptographic hashes used to verify the integrity of the component

CycloneDX is a format provided by OWASP, and is being used to standardise the Bill of Materials format, and has a range of different use cases from Software Bill of Materials, to Machine Learning Bill of Materials (OWASP, 2024); Essentially all uses cases that utilise a supply chain are covered by CycloneDX. The available documentation covers each of the fields, as the table above only covers some of the primary ones seen, but the list is comprehensive (OWASP, 2024).

Processing the Data

Currently, once the user or organisation has created their SBOM file (for each application), they would be required to process these BOM files to check for any known vulnerabilities. The tools listed in the previous section are used only for the generation of the data, but do not check for vulnerabilities. This would require a user or organisation to then take this data and perform the processing themselves through a different tool, or by manually going through each component. Unless the user is using a managed solution, they will have to periodically run this processing again to determine if a new vulnerability has arisen within their BOM. In most cases, users would then be required to use a tool such as:

1. Anchore/Grype
2. OWASP Dependency Track
3. Quay/Clair

This is an extra step, and in all cases, there are features which I consider to be important that are missing, for example, with Grype and Clair we do not get real-time alerting without manually integrating these tools with the users CI/CD pipeline alongside other tools. This is another layer of complexity that will lead to slower detection times and increased operational costs, which will lead to inefficiencies for users trying to manage their SBOMs.

Current Workflows

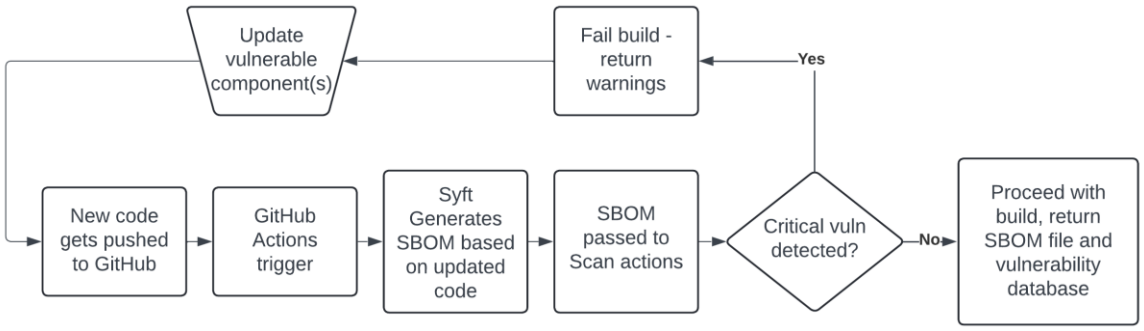

Figure 1 - Current implementation of SBOM workflow

Vulnerability and Component Management

Management of the vulnerabilities and components effectively is a critical part of maintaining a secure supply chain. Once the data has been processed, the vulnerabilities must be identified, tracked and addressed continuously, but the current SBOM ecosystem highlights several issues regarding the management of this data.

Currently, the tooling is siloed – the tooling is operating independently, causing a requirement to integrate many tools to perform a single process, and the same can be said here. The SBOM data can be integrated into several vulnerability management tools and dashboards, but this is adding yet another layer to the unnecessary complexities within SBOM.

Opportunities for Improvement

1. Fragmentation, and the integration of vulnerability scanning with SBOM management

The current ecosystem is too fragmented, with users being required to transition between multiple platforms or processes. An integrated solution that combines the steps of SBOM generation, vulnerability scanning, and vulnerability management into a single process would streamline workflows.

2. Real-time monitoring and alerting

While real-time monitoring and alerting can be enabled by using third-party solutions (i.e. ServiceNow, Dependency Track), this isn't inherent to the SBOM process. By combining the vulnerability management component into a single platform, vulnerability response and remediation times could be further reduced.

3. Enhancing Automation

The current manual process of generation, processing and re-scanning SBOMs is causing a bottleneck, and opportunities exist to streamline these steps into an automated workflow. By using automated scanning, remediation suggestions and CI/CD integrations, this process could be automated to a further degree than is currently on offer.

4. Standardisation and interoperability

A managed platform for SBOM data would ensure that the latest formats and technologies are being used consistently. If the same formats were being used across the board, interoperability could be ensured between services. For example, if an organisation already has a platform in place for vulnerability management, it should be possible to export the vulnerability details to that platform.

Functionality

Proposed Workflow

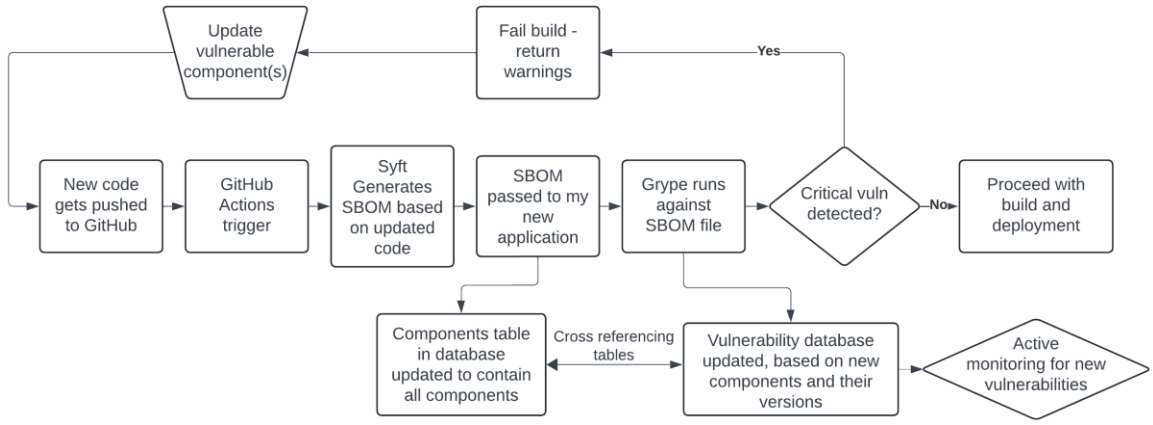

Figure 2 - Proposed SBOM workflow - showing increased cohesion between tooling

Requirements

Considering what's currently available on the market, and how we can improve or build upon it, a suitable application must comply with the following requirements.

Core Requirements

1. SBOM generation and parsing
2. Sign in/Sign out system
3. Vulnerability detection
4. Real-time monitoring and alerting
5. Dashboard and reporting
6. Vulnerability management controls
7. CI/CD Integration capabilities

Non-Functional Requirements

Scalability

The application must allow for increasing and decreasing workloads over time, ensuring that the application always remains responsive.

Automation and Efficiency

The platform should minimise manual input through the automation of repetitive tasks, such as generation and scanning of SBOM data. The process of pushing the code and having a vulnerability report should be seamless.

Security

User authentication and authorisation should be implemented in a robust manner. The data being processed is potentially sensitive and should be protected through a security in depth approach. The application should use TLS where possible for all communications. BOMs and vulnerability reports should be stored securely, and limit which users can access those resources.

Technology Stack

Front-End

React.js

React will be used to design the front-end, as it's a modern and highly capable framework for building a responsive and modular application, like the one I've envisioned. Individual components can be built, and then slotted into a dashboard experience, allowing for the user to tailor their dashboard to their desires. React also has access to various libraries that can be used to perform many functions to enhance the user experience, while also reducing development time and costs.

Back-End

Node.js and Express

Node, paired with Express is in my opinion, the perfect choice for building this application. Node allows for generating dynamic content, based on which user is currently signed in, while Express can be used to handle all API requests, for example, the application will be required to transmit data to the processing server – which can be easily implemented through Expresses routing. These are both highly popular frameworks, used frequently by many developers, reducing the risk of creating a vulnerability within the context of the webserver.

AWS Ecosystem

Amazon DynamoDB

I believe DynamoDB to be a suitable choice for this application. Considering the nature of our data – a Json format, though the fields are not inherently all required, the data can be considered unstructured. DynamoDB allows for a schema-less design, meaning that items are allowed to have different attributes within the same table; This provides us with the flexibility required, as the SBOM standard is likely to change over time. Using DynamoDB would allow for modifications to how the SBOM data is handled, while not requiring a database overhaul in the event of the standard changing.

DynamoDB is scalable, allowing for the application to always remain responsive, even in the event of many read/write operations taking place at any moment. Notably, since DynamoDB is a managed service provided by AWS, it reduces overhead during development, as the database can be easily linked up from an external point of view. It's also a part of the AWS ecosystem, allowing for interoperability with other services such as Lambda and SNS/SQS – enabling real-time alerting more easily. It also comes with the usual security controls that you would expect from an AWS service.

Elastic Compute (EC2)

The platform will require the use of third-party open-source software that can be used to process the application source code, the containers, or the SBOM files. Using an EC2 system from AWS will be a scalable and performant means of processing the data, as well as hosting the webserver. The key considerations that made me choose EC2 include the high customisability of the instances, allowing me to tailor the server to my specific needs, the high scalability using auto-scaling groups, and the security controls which will allow me to isolate the processing server from the front-end.

Syft and Grype

Both Syft and Grype are used as part of the SBOM generation and processing methodology and are popular open-source tools used to create the SBOM, while creating detailed lists of vulnerabilities based on the SBOM itself. These tools would be installed on an EC2 instance; Once the instance receives either application code, a container, or an SBOM file, it would process accordingly and return the results to the relevant database – allowing the user to view and manage this data.

Architecture

The following architecture diagram was created using draw.io. I specifically chose to use draw.io for this purpose instead of Lucidchart as it has many of the AWS icons built-in and allows for finer grained controls when placing different objects within the diagram.

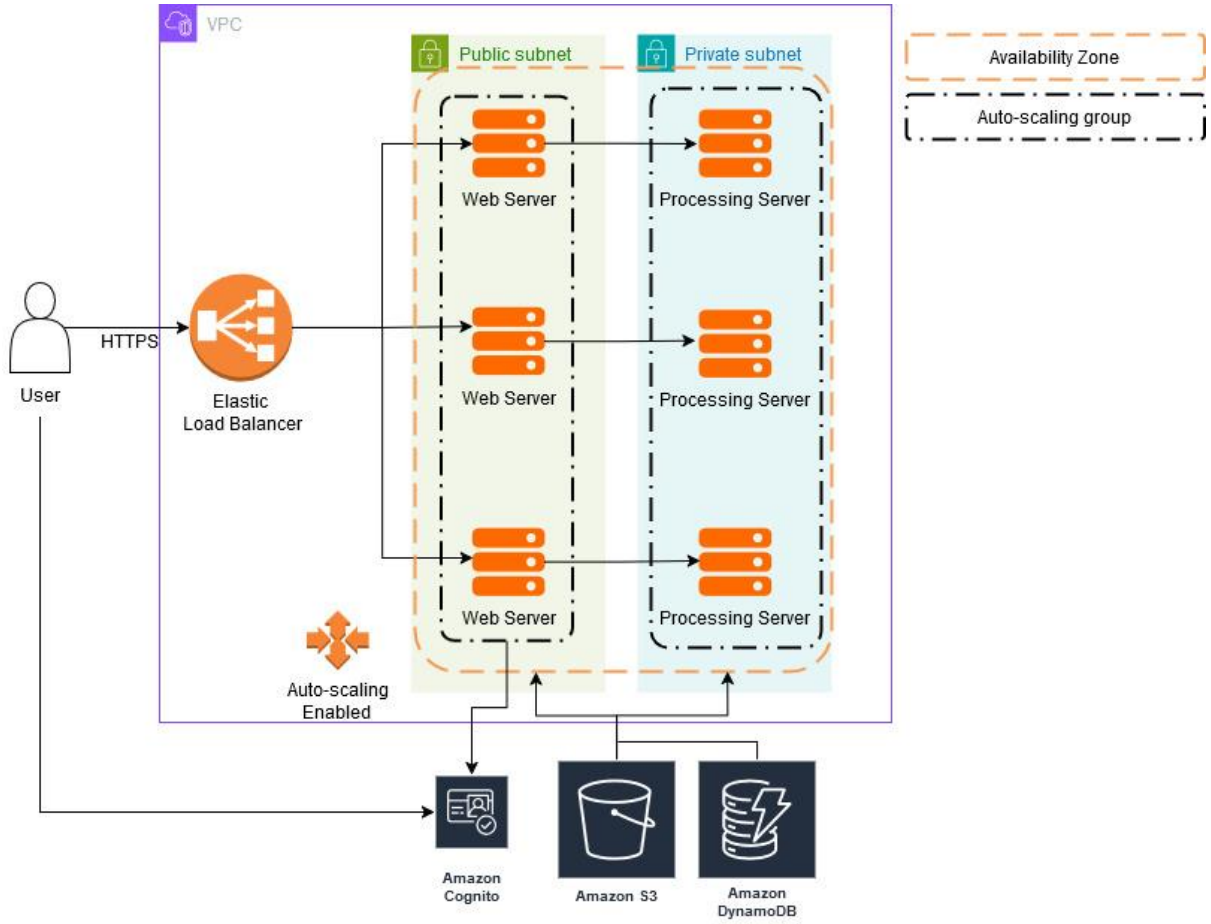

Figure 3 - AWS architecture diagram

Single Availability Zone

I've chosen here to only use a single availability zone, even though this isn't considered best practice. For the purposes of building this platform, it will be restricted to a single AZ; Retrospectively making the application work across multiple AZs is an easy task, but for the design and testing phases, simplicity would be preferred.

DynamoDB and S3

Both elements of infrastructure will be required to communicate with both the web application servers, as well as the processing servers – hence why they are laid out as above.

Private Subnetting

The processing servers should not be exposed to the internet, and users should be unaware of their existence. The processing servers could be triggered directly from S3 – in the event that an SBOM is uploaded directly to the S3 bucket, otherwise the only communication to these servers should be coming from the web servers, requesting that a new SBOM file be generated, or a vulnerability report be generated.

Database Design

SBOM Table

The SBOM table is required to store metadata and organisational details about each SBOM. It acts as a central repository for tracking the creation, source, and ownership of each SBOM.

Key Purposes:

1. Metadata Storage

Stores information about the SBOM, such as the type of source file being processed, the status of processing, and the files' location in storage.

2. Organisational Ownership

Links each SBOM to its relevant organisation through an organisationId field, enabling multi-tenancy where users from different organisations can only access their own data.

3. User and Process Tracking

Tracking of each user that submitted the SBOM and uses timestamps to keep tracking of when it was created and updated, providing accountability.

4. Integration Point

The SBOM table will act as a bridge between the users, and the detailed vulnerability reports provided within the SBOMVulnerability table.

Fields/Attributes

Attribute	Purpose
sbomId (String)	Primary key – Unique identifier for each SBOM file
organisationId (String)	Identifier for the organisation that the SBOM file belongs to
userId (String)	ID of the user that generated or uploaded the SBOM file
createdAt (ISO 8601 String)	Timestamp of when the SBOM was created
Status (String)	Status of the SBOM processing (pending, completed, failed)
sourceType (String)	Type of input (source code, container, SBOM file)
filePath (String)	S3 path of where the SBOM file is stored

SBOM-Vulnerability Table

The SBOMVulnerability table is designed to store the detailed vulnerability reports for each SBOM. Each record will correspond with a specific vulnerability detected during the analysis of an SBOM.

Key Purposes:

1. Vulnerability Tracking

Records each vulnerability, including descriptions and severity levels, and if there is an available fix that can be applied; This will work across each SBOM.

2. Querying Vulnerabilities

Enables easy querying to fetch vulnerabilities for a specific SBOM using the sbomId as the partition key.

3. Security Analysis

Potential to provide detailed insights into each application, highlighting risks associated with each application/container.

4. Reporting and Visualisation

Supports the generation of vulnerability reports, which can be displayed on dashboards. This will offer full transparency into the security posture of an organisation's software.

Fields/Attributes

Attribute	Purpose
sbomId (String)	Primary key – Unique identifier for each SBOM file
vulnId (String)	Sort key – Unique identifier for the vulnerability
description (String)	Description of the vulnerability
severity (String)	Severity level (low, medium, high, critical)
fixAvailable (Boolean)	Indicating if a fix is available or not

Relationship Between Tables

The SBOM table acts as a repository for all SBOMs in the system, while the vulnerability table references each vulnerability detected within each SBOM. Having the data stored and structured this way enables the following workflow, through the relationship:

1. Submission

A user uploads as SBOM, the file is stored on within an S3 Bucket, and an entry is created in the SBOM table which references the fields found within the file.

2. Processing

Grype will be used to process the SBOM file, referencing the filePath parameter of the SBOM table entry.

3. Analysis and Action

After generation of the vulnerability report, the vulnerability table will be updated to contain the results stored within the report. The user can now query the vulnerability database through the web interface.

ER Diagram

While I don't plan on using a relational database model, using an ER diagram is still an effective approach to demonstrate how the data stored may have relationships between them. In this

case, the diagram was designed using Lucidchart (Lucidchart, 2025) and the data is modelled as follows:

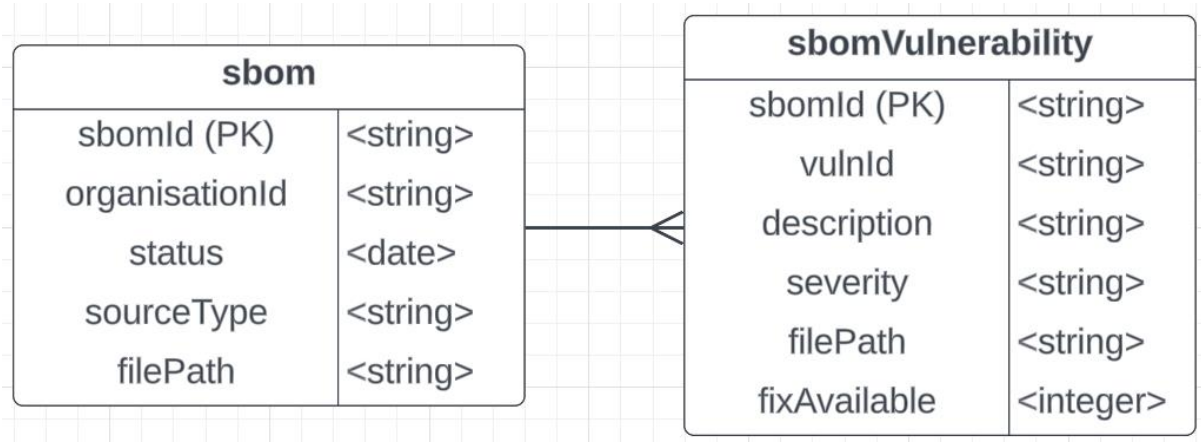

Figure 4 - Diagram showing how the data model is expected to be

The database design does not need to be over-complicated – each step of the process generates a single file, meaning we are only required to store the metadata associated with each file, simplifying the database significantly. If I was required to parse the SBOM/report files and build my own database models from scratch, this would be a much more complex task.

Risk Assessment

Identification and Assessment

Deciding to attempt to build a web application with layers like described comes with inherent risks regarding complexity and time management, while also introducing risks from within the technologies being used. The following risks were identified to potentially have an impact on this project.

Complexity

The use of multiple AWS services integrated with each other would provide the functionality required but introduces many components which must work seamlessly together. This complexity can greatly increase development time and increase the risk of the application having failures. These services often provide fail-safes, for example, encryption at rest and auto-scaling groups which may aid in mitigating the risks. Interfacing the application with the backend doesn't add a significant amount of complexity, as these isolated systems will essentially be communicating through API calls.

Unauthorised Data Access

There is a risk with storing sensitive user data that it may be accidentally exposed to an attacker, or they may be able to gain access to the data. Users will be prevented from directly being able to query the database, and instead be given a fixed set of queries that can be made. The storage bucket being used to store the SBOM data must also be protected from unauthorised access; Using a combination of Security Groups to limit communications to the bucket, and strong input validation on the filePath attribute within the SBOM table should aid in mitigation.

Cross-Site Scripting

The risk of XSS should be considered within a web application. Users will be supplying their own SBOM files in certain use cases, in which it may contain malicious code that would execute on unsuspecting users' browsers. Using popular and known safe tooling such as React and Node will aid in mitigating the risk of reflected XSS, though input validation should still be performed on all user supplied input.

Misconfigured AWS Resources

Introducing a misconfiguration in an AWS resource potentially has the impact of exposing sensitive data to the public. Exposing an incorrect resource would potentially allow an attacker to gain access to backend servers and take control of them, or to access the database directly. The likelihood of a misconfiguring a resource is high, but this is avoidable by taking appropriate precautions when building the infrastructure.

Mitigation

Risk Identified	Impact	Likelihood	Mitigation Measure
Complexity	High	Medium	Test components before building the application. Be prepared to potentially need to use a different tool/service. Use an Agile methodology to provide value in phases.
Unauthorised Data Access	Medium	Low	Design databases and buckets using known best practices. Use AWS Cognito for robust authentication and authorisation. Encrypt sensitive data at rest, and in transit.
Cross-Site Scripting	Medium	Low	Use Reacts built in protections against XSS. Implement strong Content Security Policies. Sanitise all user-generated content before rendering.
Misconfigurations	High	High	Use AWS Config to monitor and enforce configurations. Use strict Security Groups and VPC segregation where possible/required. Use IAM roles with principle of least privilege

Security

This application must conform with the latest best practices when it comes to security, as the nature of the data is potentially highly sensitive. While the purpose of SBOM is to increase transparency into the components within an application, this doesn't mean that an organisation or individual is required to publish that data – potentially highlighting vulnerabilities to attackers. All sensitive data stored within the databases should be encrypted at rest, meaning should an attacker gain access to the database, it's unlikely they will gain any insights or value from the data.

Authentication will be handled by AWS Cognito, which provides a robust and scalable identity management service. This will simply the development process for this platform, while also adhering to the best practices for security in terms of authentication. Cognito notably will

enable flows for sign up, sign in, multi-factor authentication, and secure token-based authentication. Cognito and API Gateway can also be paired together, to authenticate incoming API requests via Cognito User Pools – restricting the data that a user can access.

All processes where there is user supplied input should be validated and controlled as to prevent a malicious actor from gaining unauthorised access to the webserver, or from injecting malicious code that may cause cross-site scripting. From this position, it would increase the likelihood that an attacker would be able to gain access to the database or the SBOM processing server. The webserver is the primary server that needs to be protected due to this.

Where possible, assets are to be segregated on the network. A user should not be capable of sending a request directly to the processing server or directly to the database. The user should only be capable of interacting with the web server. Using VPCs and security groups to segregate the webserver from the processing server will be possible, limiting where connections can be made from.

The platform will be assessed for security vulnerabilities through dynamic analysis prior to making the application publicly available. This will involve a sprint phase dedicated to testing the core functionality using penetration testing methodologies, ensuring that some of the more common vulnerabilities found within the OWASP Top 10 (OWASP, 2025) are not present within the application. Some areas that will be covered here are:

1. Broken Access Control
2. Injection (Stored and reflected XSS, code injection)
3. Insecure Design
4. Security Misconfigurations
5. Vulnerable and Outdated Components

Performing this assessment will require a suite of tools designed to perform this function:

Burpsuite

Using Burpsuite will allow for direct testing of the application functionality, allowing me to modify the requests to determine if the application/server is communicating in an insecure manner. Notably, Burpsuite will aid in testing each of the above vulnerabilities in some way, such as identifying a misconfigured CSP or outdated component.

Being cautious about broken access control, the use of the autorize (Barak Tawily, AppSec Labs, 2024) BApp extension will enable for easier testing of broken access control. It allows the assessor to attempt to access the same resource with multiple authentication tokens simultaneously, highlighting any access control issues.

Nmap

Nmap will be used to ensure that both the frontend and backend servers are not exposing any surface area that is not required.

Sqlmap

Using sqlmap in conjunction with manual SQL injection testing will ensure full coverage for all queries to the database.

Qualys SSL Server Testing

Qualys SSL Test (Qualys SSL Labs, 2025) is a free online service provided by Qualys SSL Labs to evaluate the security of a websites SSL/TLS configuration. Using this service, any potential

vulnerabilities related to weak cryptographic libraries, or a TLS misconfiguration will immediately be highlighted. The detailed reports provided give an overview, and suggestions as to how the SSL/TLS implementation can be improved.

Development Methodology

For the development of this project, the use of Agile will be employed. More specifically, the Scrum framework will be used with iterative sprints. Agile is often used within development teams and is widely recognised as an effective approach for application development.

Why Agile?

Agile was chosen as it allows for flexibility during development based on feedback (and testing), and immediate delivery of value to customers – each component can be developed individually, and then in later sprints, they can be orchestrated together. The key reasons for choosing Agile were because of its adoption within the industry, its iterative development process, and for its ability to allow for changes without derailing development.

Why Scrum?

Scrum is a part of the Agile framework, meaning it aligns with the same goals as Agile, but adds further focus on iterative development. Scrum will be used to incorporate sprints, allowing me to break down each large task into smaller, more manageable chunks that can be slotted into a timeline. The key benefits I would like to gain from using Scrum are the incremental development using sprints and the ability to prioritise tasks – the most critical components can be completed first, adding immediate value to the application.

Implementation Plan

The development will be broken up into one-week sprints. At the start of each sprint, the task will be subdivided into smaller components that are more tangible. In a team, daily stand-ups would take place; I believe there is still value in doing these stand-ups even when working alone, as they allow for time to think about the task at hand and how to best approach it. The key questions to be addressed within these stand-ups will be:

1. What was accomplished yesterday (or during last day)?
2. What is planned for today?
3. What is holding development back?

At the end of each sprint, a review will take place ensuring that the quality of work is up to par with what's expected for the project. For the retrospectives, the primary goal is to identify the following:

1. What went well during the sprint?
2. What challenges was I faced with, and how were they addressed?
3. Are there opportunities to improve this in future?

This methodology will be enabled using Trello, a tool used to manage the feature backlog, assign priority levels, and track progress. Trello was used during the design phase of this report and has been a useful tool for deciding how to structure and architect this platform. It was also a highly useful tool for keeping minutes with my project supervisor, enabling collaboration, as well as allowing me to highlight any actionable items that were brought up during the meetings.

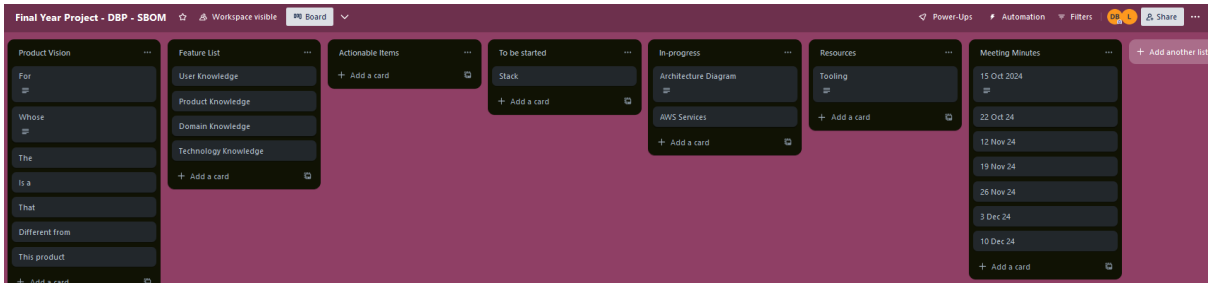

Figure 5 - Trello board; For design phase of project

Project Timeline

Gannt Chart

Semester 1

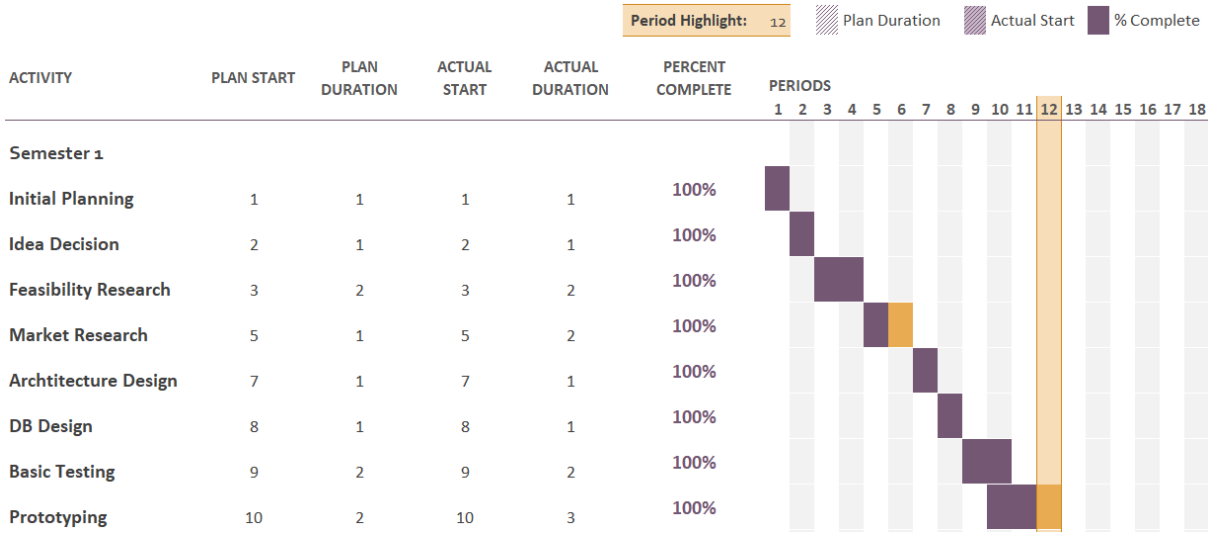

Figure 6 - Gannt chart, used to track progress over semester 1

The Gannt chart is a useful tool for tracking progress over a period, and highlights where issues have arisen. For semester one, most of the targets were achieved on time except for market research and prototyping; The delays here were because of assignment deadlines eating up time. Though there were delays, all targets for semester one was hit.

Activity	Purpose	Target Met?
Initial Planning	Propose idea to supervisor, receive feedback on where to further research. Consider other options	Yes
Idea Decision	Finalise idea, submit proposal, complete ethics checklist	Yes
Feasibility Research	Research open-source tooling that currently exists, research if it would be possible to integrate with a web application	Yes
Market Research	Research for existing SBOM solutions, managed platforms that enable the proposed new workflow	Delayed

Architecture Design	Learn about the services that will be required when building the platform. Consider AWS ecosystem further	Yes
DB Design	Consider the best way to structure the data that is required to be stored, how to manage user accounts	Yes
Basic Testing	Test the open-source tooling in their environments	Yes
Prototyping	Build wireframes and prototypes, showing what's envisioned for the application layout	Delayed

Semester 2

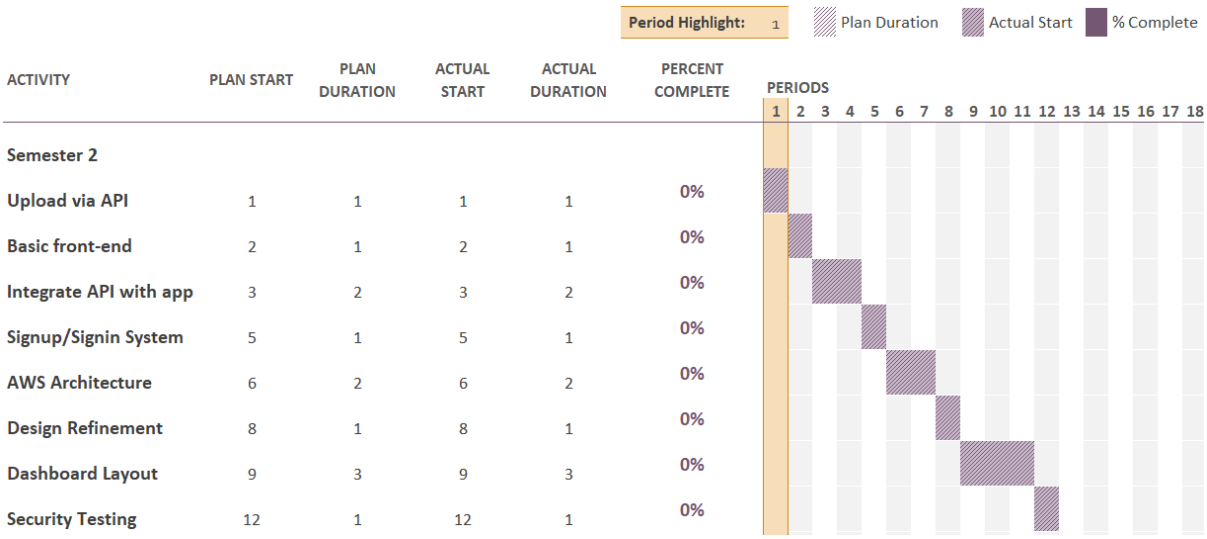

Figure 7 – Gannt chart, used to predict goals and targets for the implementation phases during semester 2

The Gannt chart above shows the expected timeline for the implementation phases of this project. I believe this to be a realistic timeline, and each goal will be attainable within their allocated timeslots; It's likely that some sprints will be completed sooner than others, leaving room to start early on some areas, or to refine other areas.

Testing

To determine if it would be possible to generate the data required by the processing server, testing the components within that environment was a necessary task. To goals during the testing phase were as follows:

1. Build an EC2 instance with the required tools, appropriate configuration
2. Generate an SBOM from user supplied application code
3. Generate a vulnerability report from the new SBOM file

EC2 Instance Configuration Details

The EC2 instance processing server has several configurations that must be applied for core functionality. SSH must be enabled for remote access from my network address, and the device storage must be increased (30gb in this case, up from 8gb). Syft and Grype were shown to be effective tools in performing the required functions but could not be installed through the package manager; It was found possible to install them through the install scripts located within the Syft and Grype Github projects – this is even the recommended way of installing.

```
[ec2-user@ip-172-31-39-148 ~]$ syft --version
syft 1.18.1
[ec2-user@ip-172-31-39-148 ~]$ grype --version
grype 0.86.1
[ec2-user@ip-172-31-39-148 ~]$
```

Figure 8 - Syft and Grype both successfully installed on an EC2 instance

Generating the SBOM

This is where the issue related to storage space was encountered – Syft will generate a relatively large file, and it was not found possible to complete the SBOM generation without increasing the storage space. Another issue arose, in that it was found that the system could be overwhelmed by attempting to download a large container from DockerHub; Notably, this could leave the system vulnerable to an accidental denial-of-service if a user attempts to scan a large container.

Example project code was downloaded to the EC2 instance – a web application created during my module in Web Application Development 2. I felt like this was a good example to use, as there were several direct dependencies and transitive dependencies implemented in this project that could be investigated.

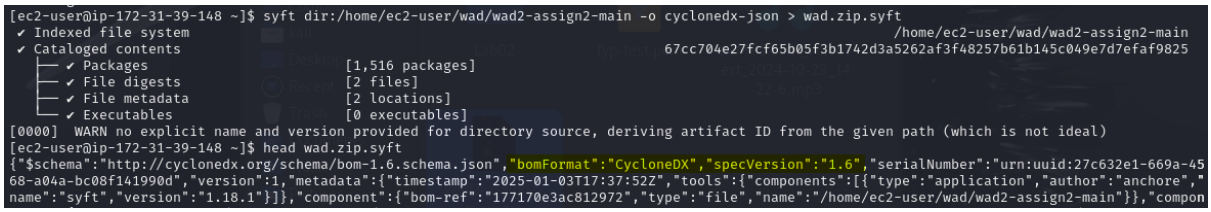

Figure 9 - SBOM for supplied application code generated successfully

Highlighted above is the format and specification version for the format used to store the SBOM file – CycloneDX. This data is essentially in Json format, except the attributes are specified within the CycloneDX documentation. Syft allows for outputting components directly to Json format, but since CycloneDX is already a Json format except with included transparency and accountability features, it was chosen to be used.

Vulnerability Scanning

During the vulnerability scan phase of testing, I encountered an issue regarding the size of the /tmp directory on an EC2 instance. Grype was able to download the vulnerability database but was unable to verify due to running out of storage space (specifically within the /tmp directory). I was unable to continue testing within the context of an EC2 instance due to this error – but a solution is currently being worked on. Everything indicates that Grype should be able to produce the required vulnerability report, but this misconfiguration is currently a roadblock.

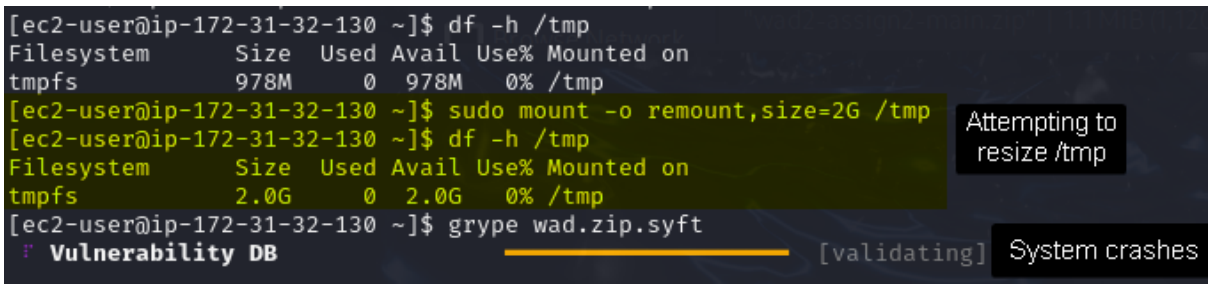

Figure 10 – EC2 instance failing to scan for vulnerabilities within the components due to storage space within /tmp

Continuing the test on my local machine, until the issue regarding storage space is resolved, I was able to produce the following vulnerability report:

```
┌──(kali㉿kali)-[~]
└─$ cat wad.zip.syft| grype
NAME                    INSTALLED  FIXED-IN  TYPE  VULNERABILITY        SEVERITY
@adobe/css-tools        4.3.1      4.3.2     npm   GHSA-prr3-c3m5-p7q2  Medium
@babel/traverse         7.23.0     7.23.2    npm   GHSA-67hx-6×53-jw92  Critical
@grpc/grpc-js           1.9.12     1.9.15    npm   GHSA-7v5v-9h63-cj86  Medium
body-parser             1.20.1     1.20.3    npm   GHSA-qwcr-r2fm-qrc7  High
braces                  3.0.2      3.0.3     npm   GHSA-grv7-fg5c-xmjg  High
cookie                  0.5.0      0.7.0     npm   GHSA-pxg6-pf52-xh8x  Low
cross-spawn             7.0.3      7.0.5     npm   GHSA-3xgq-45jj-v275  High
ejs                     3.1.9      3.1.10    npm   GHSA-ghr5-ch3p-vcr6  Medium
express                 4.18.2     4.19.2    npm   GHSA-rv95-896h-c2vc  Medium
express                 4.18.2     4.20.0    npm   GHSA-qw6h-vgh9-j6wx  Low
firebase                10.7.1     10.9.0    npm   GHSA-3wf4-68gx-mph8  Medium
follow-redirects        1.15.3     1.15.4    npm   GHSA-jchw-25xp-jwwc  Medium
follow-redirects        1.15.3     1.15.6    npm   GHSA-cxjh-pqwp-8mfp  Medium
http-proxy-middleware   2.0.6      2.0.7     npm   GHSA-c7qv-q95q-8v27  High
ip                      2.0.0                npm   GHSA-2p57-rm9w-gvfp  High
ip                      2.0.0      2.0.1     npm   GHSA-78xj-cgh5-2h22  Low
markdown-to-jsx         7.3.2      7.4.0     npm   GHSA-4wx3-54gh-9fr9  Medium
micromatch              4.0.5      4.0.8     npm   GHSA-952p-6rrq-rcjv  Medium
mongoose                8.0.3      8.8.3     npm   GHSA-m7xq-9374-9rvx  High
```

Figure 11 - Output from Grype, indicating several vulnerabilities have been detected within multiple dependencies and transitive dependencies

This data can then be outputted to a file with an attached organisation identifier, which will then be displayed to the user.

Prototypes

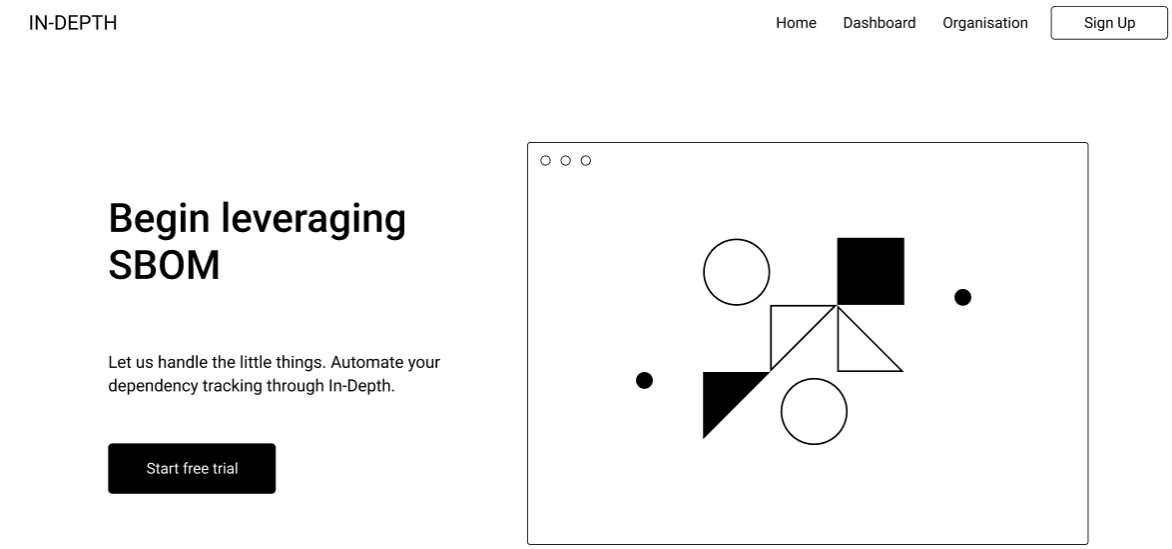

Figure 12 - Prototype of landing page

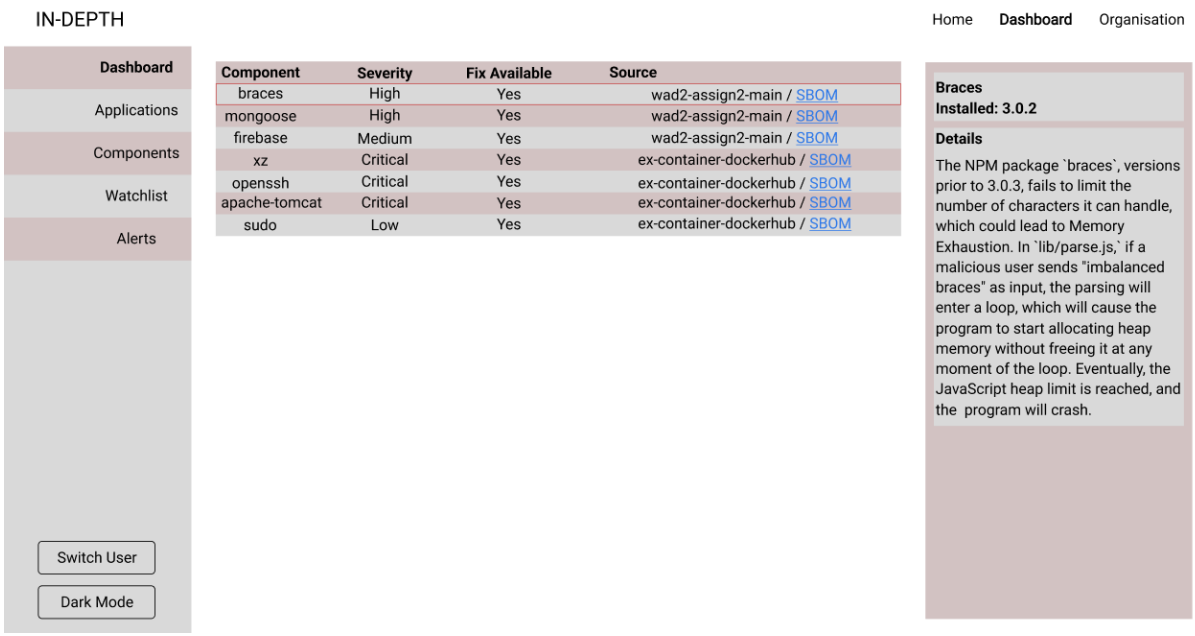

Figure 13 - Prototype of dashboard interface

References

Anchore, 2024. *anchore/grype*. [Online]
Available at: https://github.com/anchore/grype
[Accessed 2 January 2025].

Anchore, 2024. *anchore/syft*. [Online]
Available at: https://github.com/anchore/syft
[Accessed 31 December 2024].

Barak Tawily, AppSec Labs, 2024. *Autorize*. [Online]
Available at: https://portswigger.net/bappstore/f9bbac8c4acf4aefa4d7dc92a991af2f
[Accessed 2 January 2025].

Lucidchart, 2025. *Lucidchart*. [Online]
Available at: https://www.lucidchart.com/pages/
[Accessed 2 January 2025].

NCSC Ireland, 2021. *CVE-2021-44228 - Vulnerable Log4j Server*. [Online]
Available at: https://www.ncsc.gov.ie/emailsfrom/Shadowserver/CVE/CVE-2021-44228/
[Accessed 30 December 2024].

NIST, 2024. *CVE-2024-4068 Detail*. [Online]
Available at: https://nvd.nist.gov/vuln/detail/CVE-2024-4068
[Accessed 3 January 2025].

NIST, 2024. *National Vulnerability Database - CVE-2024-3094 Detail*. [Online]
Available at: https://nvd.nist.gov/vuln/detail/cve-2024-3094
[Accessed 30 December 2024].

OWASP, 2024. *CycloneDX v1.6 JSON Reference*. [Online]
Available at: https://cyclonedx.org/docs/1.6/json/
[Accessed 30 December 2024].

OWASP, 2024. *CycloneDX: The International Standard for Bill of Materials (ECMA-424)*. [Online]
Available at: https://cyclonedx.org/
[Accessed 30 December 2024].

OWASP, 2025. *OWASP Top Ten*. [Online]
Available at: https://owasp.org/www-project-top-ten/
[Accessed 2 January 2025].

Qualys SSL Labs, 2025. *SSL Server Test*. [Online]
Available at: https://www.ssllabs.com/ssltest/
[Accessed 2 January 2025].

U.S. Government, 2021. *Executive Order on Improving the Nation's Cybersecurity*. [Online]
Available at: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/
[Accessed 31 December 2024].