Security Document

Security document of the Care web application

Name: Victoria C. A. Fong

Year: 2022

Class: S-A-RB-CMK 4

Introduction

This document summarizes the security aspects that have been applied to the Care web application. In the document you will come across the security requirements, misuse cases, and how the set security requirements have been integrated into the application for a more secure platform.

Table of Contents

Security Requirements

Context

The security requirements have been inspired and based on the ASVS requirements. Not only do I investigate which security requirements best fit the Care project but also create user stories and misuse diagrams based on these requirements and how attackers can attack the system.

#	Description	Applying
1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development.	By making a checklist for adding security in each development lifecycle (analysis, design, implementation, testing) and applying this can secure software development lifecycle be addressed at each development stage.
1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.	Creating for example misuse diagrams of attackers.
1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.	Researching and verifying which authentication mechanisms have secure authentication and can be extended to stronger authentications.
1.14.4	Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.	Using git and integrating automated security tests for integration and building.
6.3.2	Verify that random GUIDs are created using the GUID v4 algorithm, and a Cryptographically secure Pseudo-random Number Generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.	Implementing that when GUIDs are made are they made with using GUID v4 algorithm.
8.3.2	Verify that users have a method to remove or export their data on demand.	Implementing methods where users can remove their data on demand.
14.1.1	Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.	Implementing secure and repeated build and deployment process in the CI /CD automation pipeline.

User stories & Misuse cases

#	Cases	User Story / Misuse Case
1.1.1	As a developer, I have added security practices in each stage of the software development lifecycle to be able to build a secure system.	User Story
1.1.1	As an attacker, I can find a non-secure development of the system where I can misuse the system.	Misuse Case
1.1.2	As a developer, I create and model designs of threats to be able to identify them and plan how to prevent these threats.	User Story
1.1.2	As an attacker, I can try to attack the system at every new release to see if there are not any secure implementations.	Misuse Case
1.2.2	As a developer, I have implemented authentication in APIs and not the components of the application to be to make sure the correct access is given to each user.	User Story
1.2.2	As an attacker, I am attacking the application components for vulnerabilities to see what access I need to get information for personal misuse.	Misuse Case
1.2.3	As a developer, I have chosen an authentication mechanism that is secure and can be extended to strong authentication to prevent easily breaches by attackers.	User Story
1.2.3	As an attacker, I have found out that the system does not use a strong authentication system where I can abuse and breach user accounts and the system.	Misuse Case
1.14.4	As a developer, I have created security tests and implemented in CI/CD pipeline whenever a project is built to reassure security of the code.	User Story
6.3.2	As a developer, I have used the GUID v4 algorithm to generate random GUIDs for more secure and not predictable GUIDs	User Story
8.3.2	As a developer, I have implemented methods where users can remove their own data.	User Story
8.3.2	As a user, I am able to remove private data I do not want to show or change my personal data.	User Story
14.1.1	As a developer, I have implemented in the CI / CD deployment pipeline secure and repeated procedures for building and deploying the application so that it can be build and deployed in a secure manner.	User Story

Adapting security to the project

In this section are the security requirement actions addressed to show proof of a more secure application created.

1.1.1 Verify the use of a secure software development lifecycle that addresses security in all stages of development

Applying: By making a checklist for adding security in each development lifecycle (analysis, design, implementation, testing) and applying this can secure software development lifecycle be addressed at each development stage.

The software development life cycle (SDLC) is a framework of collecting the best practices in adding security at each stage of the cycle which are the:

- Planning and Requirements stage
- Architecture and Design stage
- Development Stage
- Testing Stage

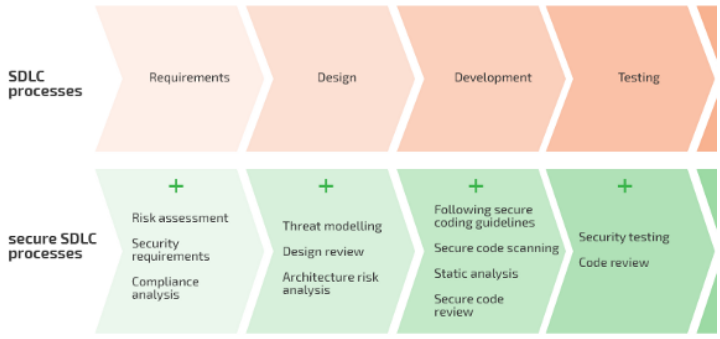

Image 1: *"Software Development lifecycle processes"*

Checklist

	Set checklist	More that can be added
Planning and requirements	• Security Requirements • Misuse Cases	○ Risk assessment ○ Compliance analysis
Design	• Threat modeling	○ Design review ○ Architecture risk analysis
Development	• Secure code scanning • Static analysis	○ Following secure coding guidelines ○ Secure code review
Testing	• Security testing	○ Code review

The above checklist describes what we will be implementing into the software and what can be added if there is more time or for a next iteration of the cycle due to the agile method approach used for the project.

Planning and Requirements stage

Requirements have been set up for the application to set as goal to how the application should function. Based on these requirements I have create security requirements and misuse cases to address threats that can occur based on the set requirements.
Setting up security requirements will address security vulnerabilities in the rest of the SDLC.

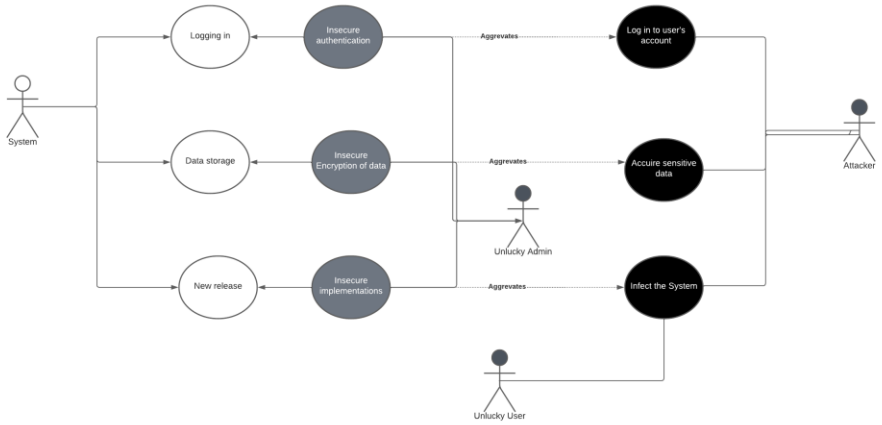

Image 2: *"Threat / misuse case model diagram based on the created misuse stories"*

Design Stage

Based on the set security requirements and misuse cases that have been set up has a threat model been designed (image 2). The threat model helps to identify, communicate, and understand the threats within application.

Development / Testing Stages

For security we will be looking into security testing. Security testing examines if the data and resources of the system are protected from possible intruders and seeks out vulnerabilities of the system.
For testing there are two options we are looking at which is

- **Static application security testing (SAST):** is a testing procedure which looks at the source code byte code or application binaries for any indication of security vulnerabilities.
- **Dynamic application security testing (DAST):** is a testing procedure which creates attacks against a web application and analyses the application's reactions. Based on their reactions the test determines whether it is vulnerable.

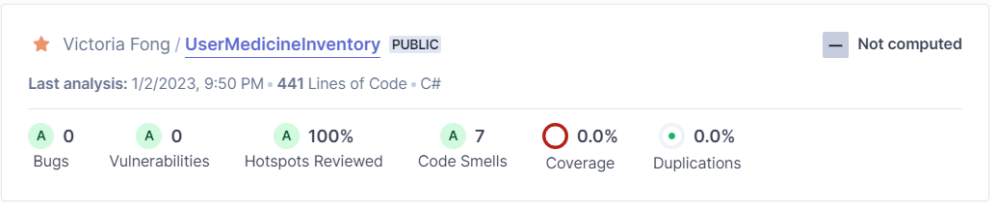

Image 3: *"Added the repository to where one of the services are running to Sonar Cloud"*

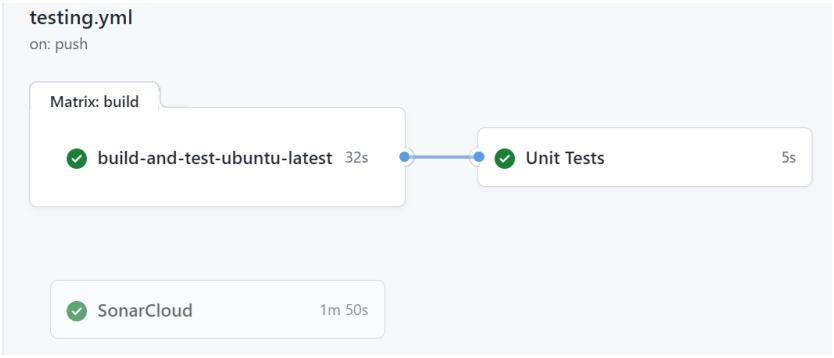

Image 4: *"Pipeline of when a pull request is made where Sonar Cloud has been added"*

We have added SAST to the application pipeline to perform security testing into looking for any security vulnerabilities with the help of Sonar Cloud. Sonar Cloud automatically review static analysis of code to detect bugs, code smell and security vulnerabilities.

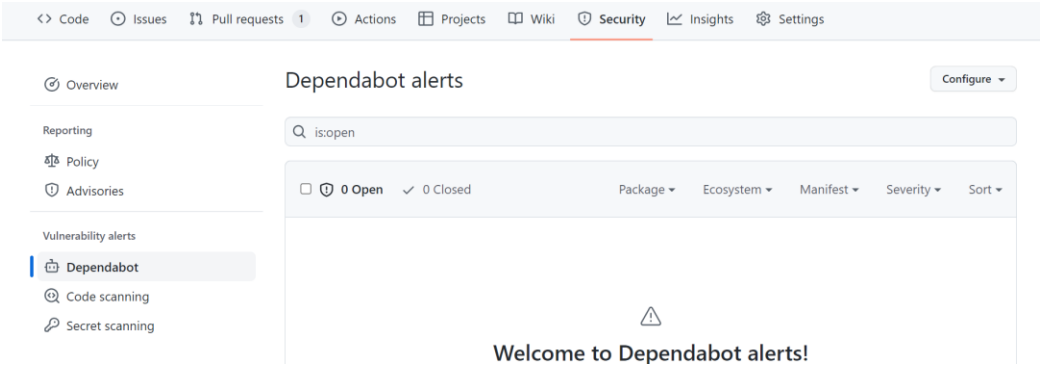

Image 5: *"Dependabot configured into the repository"*

We have decided not to add Dynamic application security testing due to time constraints, but we have added Dependabot to look at any library vulnerabilities of the system.
The approach used falls under both the development and testing cycle for the reason that the static testing used is for both static analysis and testing of the system.

1.1.2 Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.

Applying: Creating for example misuse diagrams of attackers.

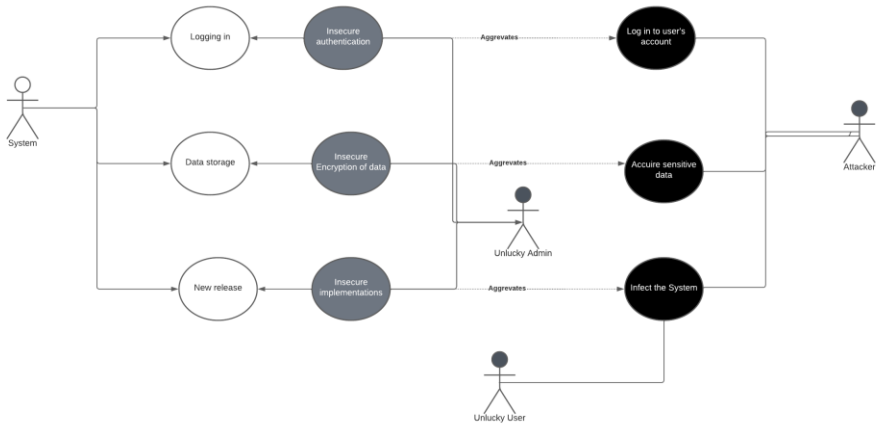

Image 6: *"Threat / misuse case model diagram based on the created misuse stories"*

Base on the set misuse cases I have designed a misuse diagram of how attackers can attack the vulnerabilities of the system.

1.2.3 Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.

Applying: Researching and verifying which authentication mechanisms have secure authentication and can be extended to stronger authentications.

From research and coming to a decision on which authentication platform I found best I chose to use Azure Active Directory.
This is for the reason that the application already runs on azure Kubernetes where I would not have to keep up with different features from another platform.

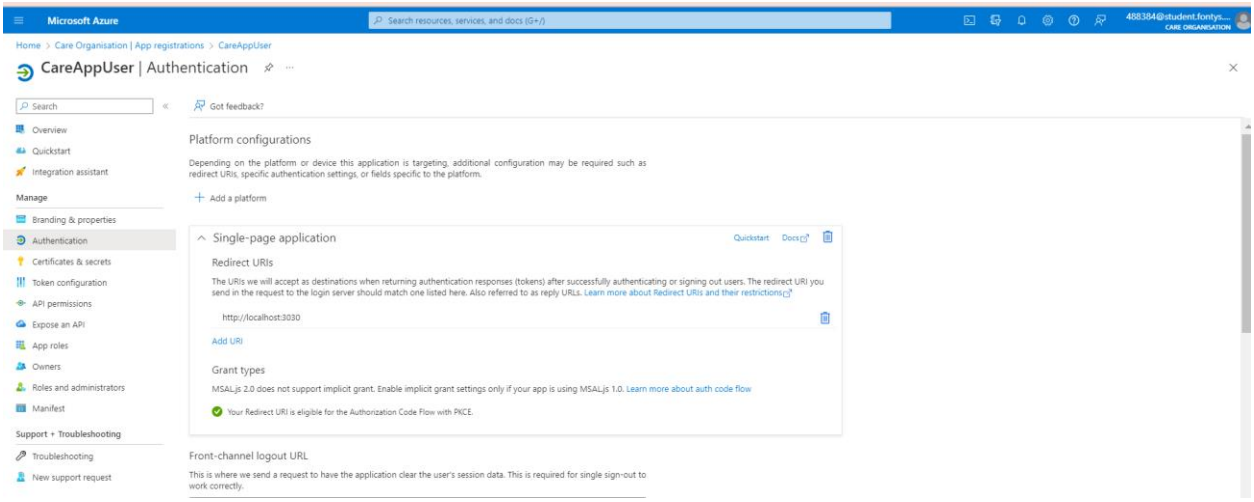

Image 7: *"Azure Active Directory Authentication added to application as Single-page application"*

Further, I have chosen this platform for the reason that you can set up a basic security login for development and easily add more secure authentication along the way which would include personalize tokens that can be used within the system making calls safer and security for each user.

1.14.4 Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.

Applying: Using git and integrating automated security tests for integration and building.

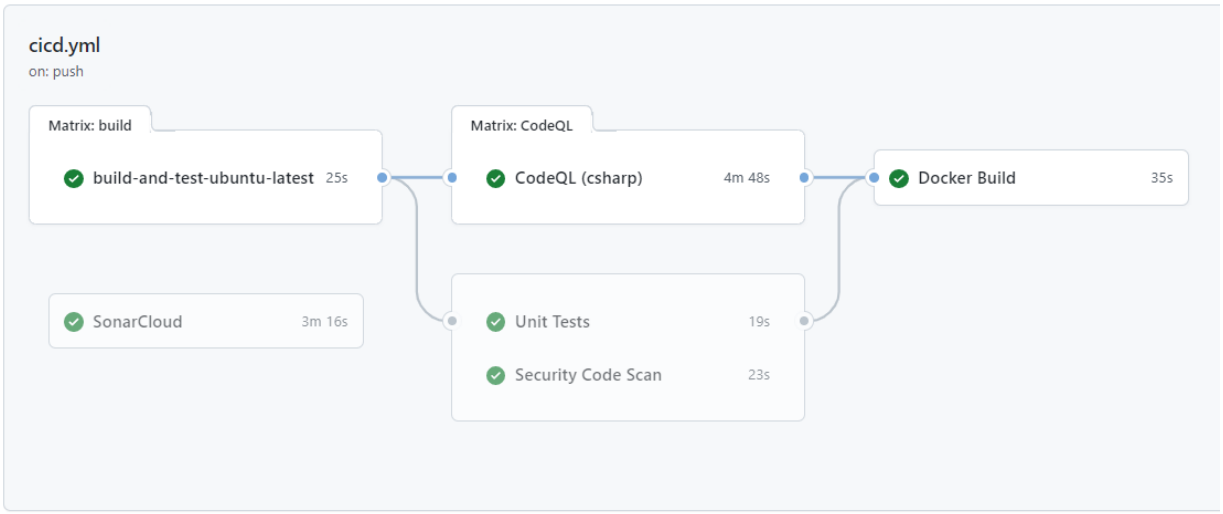

Image 8: *"Pipeline of when a pull request with SAST tools"*

For the application and based on the checklist designed for the development and testing stages, the GitHub pipeline that has been setup for the Care application has added integration for building, testing the functionality of the code and testing for any vulnerabilities of the code.
Based on the security aspect, the pipeline has Sonar Cloud added for static application security testing and Dependabot for searching for dependency vulnerabilities

Further, for a more secure automated pipeline, the GitHub action Security Code Static Code analyzer for .Net has been added to the pipeline to detect Security vulnerabilities patterns:

- SQL Injection
- Cross-Site Scripting
- Cross Site Request Forgery
- XML eXternal Entity Injection
- And more..

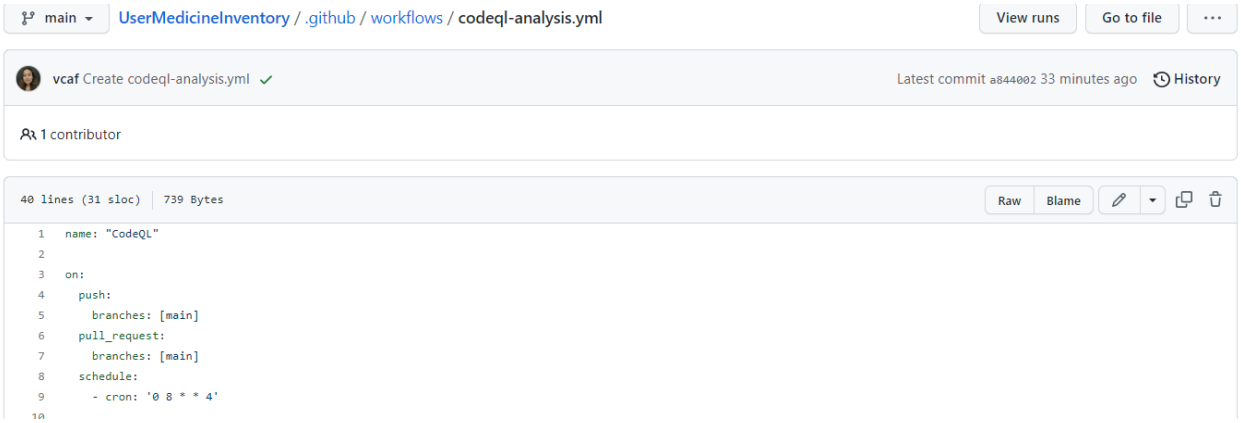

Image 9: *"Pipeline of when a pull request is made where Sonar Cloud has been added"*

Lastly, I have also integrated CodeQL to the pipeline and also to run weekly as a SAST tool as seen in image 9. In short, CodeQL is used for security checks such as looking for any security vulnerabilities, bugs and other errors that can occur to queries against the database.

6.3.2 Verify that random GUIDs are created using the GUID v4 algorithm, and a Cryptographically secure Pseudo-random Number Generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.

Applying: Implementing that when GUIDs are made are they made with using GUID v4 algorithm.

The GUID v4 algorithm that is used to generate a unique id for users and medicines is not meant for cryptographic purposes for the reason of its predictable bit pattern.
For a more cryptographic purpose I would use the RandomNumberGenerator static method class. The class provides a random number generator suitable for cryptographic use.

8.3.2 Verify that users have a method to remove or export their data on demand.

Applying: Implementing methods where users can remove their data on demand.

```
//Delete medication
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteAsync(Guid id)
    {
      var userMedicine = await _userMedicineInventoryService.DeleteAsync(id);


      if (userMedicine == null) return NotFound();


      return NoContent();
    }
```

Image 10: *"Endpoint for users to be able to remove their own registered medicine to the application"*

As seen in image 10 is the endpoint for users to be able to instantly delete their registered medicine and personal data belonging to their medicine. This shows how the user has control over their data as can also be seen in image 11.

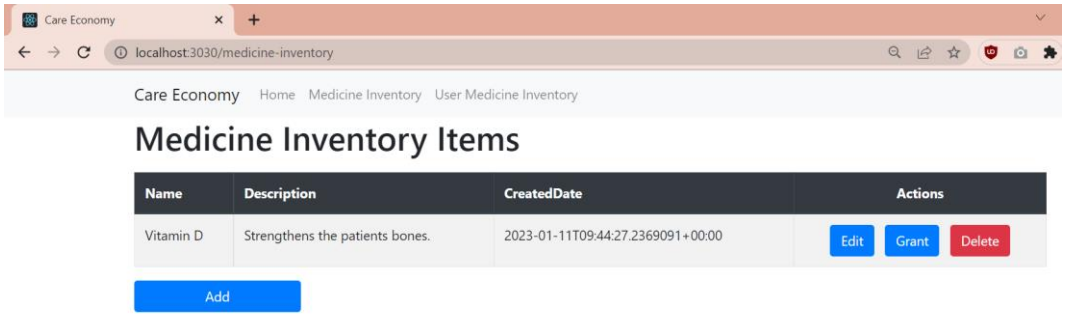

Image 11: *"Frontend option for the user to delete their registered medicine and personal data"*

14.1.1 Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.

Applying: Implementing secure and repeated build and deployment process in the CI /CD automation pipeline.

As indicated in section 1.14.4 to how I set up the pipeline, it can be justified that the ci/cd for the project is securely configured with adding secure actions such as Sonar Cloud, Dependatbot and two SAST tools (Security Code Static Code Analyzer for .Net and CodeQL).

Source

OWASP_Application_Security_Verification_Standard_4.0. (n.d.). Retrieved March 1, 2019, from https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf

Jaiswal, N. (2022, January 5). *Listicle for secure Software Development Life Cycle(SDLC)*. Medium. https://medium.com/@jaisnain/listicle-for-secure-software-development-life-cycle-sdlc-b5913f60c24f

Miller, A. (2022, November 1). *Secure SDLC | Secure Software Development Life Cycle*. Snyk. https://snyk.io/learn/secure-sdlc/

Quickstart: Register an app in the Microsoft identity platform - Microsoft Entra. (2022, November 14). Microsoft Learn. https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app

Quickstart: Register and expose a web API - Microsoft Entra. (2022, October 12). Microsoft Learn. https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-configure-app-expose-web-apis

Redirect URI (reply URL) restrictions - Microsoft Entra. (2022, August 27). Microsoft Learn. https://learn.microsoft.com/en-us/azure/active-directory/develop/reply-url

The Azure Cave. (2021, January 16). *Tutorial Add Azure AD login to a React app* [Video]. YouTube. https://www.youtube.com/watch?v=4pH5spE2Yss