Utilizing GitHub's Built-in Security Features and GitHub Actions to Automate SAST and Deployment for EJS Web Applications

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor thesis

Degree Programme in Computer Applications
Autumn, 2023

Rachmad Fauzan

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Computer Applications — Abstract

Author Rachmad Fauzan — Year 2023

Subject Utilizing GitHub's Built-in Security Features and GitHub Actions to Automate SAST and Deployment for EJS Web Applications

Supervisors Tommi Lahti, Deepak KC

ABSTRACT

This thesis explores GitHub's security features and GitHub Actions for automating Static Application Security Testing (SAST) processes and Embedded JavaScript (EJS) web application deployment. It addresses research questions regarding the topic, guided by Tommi Lahti and Deepak KC. The thesis involves explaining SAST and EJS concepts, discussing the example project, and utilizing GitHub's features.

The primary research method combines literature review, hands-on implementation, and analysis. GitHub's security features and Actions effectively automate SAST processes for EJS web applications, including code linting, secret scanning, dependency scanning, and code scanning. GitHub Actions automate EJS web application deployment to GitHub Pages.

Recommendations include organizations adopting GitHub's features and Actions to enhance EJS web application security and deployment. Automating SAST processes improves security, while GitHub Actions streamline deployment for efficient updates on GitHub Pages.

This thesis contributes to understanding GitHub's features and Actions for SAST automation and deployment, highlighting automation's benefits in development, security, and deployment. Future work may explore additional GitHub features and Actions, expand to other web frameworks, and investigate external tool integration with GitHub Actions.

Keywords static analysis security testing automation, deployment automation, GitHub Actions utilization, DevOps, CI/CD

Pages 37 pages including 2 pages of annexes

Glossary

CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
EJS	Embedded JavaScript
HTML	HyperText Markup Language for web pages
SAST	Static Application Security Testing
URL	Uniform Resource Locator

Contents

Figures and tables

Program codes

Annexes

1 Introduction

As the web applications grow, it can be difficult to manage the software development lifecycle, particularly when it comes to ensuring that code is thoroughly tested and deployable. Keeping up with this routine becomes more difficult as the project expands. Automating this process with GitHub's built-in security features and Actions offers a potential solution.

The objective of this thesis is to explore the use of GitHub's features and Actions to automate Static Application Security Testing (SAST) and deployment processes for an Embedded JavaScript (EJS) web application. The thesis will concentrate specifically on automating code linting, secret scanning, dependency scanning, and code scanning as part of the SAST process, as well as generating static HTML files from EJS then deploying it to the GitHub Pages. By automating these repetitive duties, developers can devote their time to writing high-quality code and adding new features to the application, as opposed to performing time-consuming and tedious manual processes.

Notably, this thesis will not provide a detailed explanation of GitHub, EJS or SAST but will presume that the reader has a basic understanding of these topics. The thesis will instead provide a brief introduction for the tools and technologies utilized in this study, including GitHub, GitHub Actions, GitHub Workflows, GitHub Pages, software deployment, and the various components of SAST. In addition, the thesis will investigate the advantages and challenges of automating SAST and deployment processes for an EJS web application using GitHub built-in security features and GitHub Actions.

The research questions that guide this thesis are:

- How can GitHub's built-in security features and GitHub Actions be utilized to automate SAST processes for an EJS web application, including code linting, secret scanning, dependency scanning, and code scanning?
- How can GitHub Actions be utilized to automate the deployment of an EJS web application to GitHub Pages, including generating static HTML files from EJS?
- What are the advantages and challenges of using GitHub's built-in security features and GitHub Actions to automate SAST and deployment processes for an EJS web application?

2 Tools and terminologies

This chapter introduces the terminologies, tools, and technologies that are used on the practical part of this thesis.

2.1 GitHub

GitHub is a web-based platform that enables developers and teams to collaborate on projects using Git, a version control system. It is an ecosystem that includes storage, social networking, and collaboration tools that enable individual developers to contribute to numerous teams and projects. GitHub is used for version control, which is a method for managing different code base versions. It enables multiple developers to collaborate on multiple files and features within the same application while avoiding major conflicts caused by conflicting code or different versions on each developer's local machine. For a single project, GitHub offers public repositories that anyone can create. However, users can pay a small fee to keep their projects private. (Horst, 2023)

2.1.1 GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that automates build, testing, and deployment pipelines. It enables users to create workflows which is a collection of one or more jobs that are executed when a specific event occurs, such as a push to a repository or the creation of a pull request. When a new Issue is created in the repository, for instance, a workflow can automatically add labels to it. (GitHub, 2023g)

Workflows can be executed on Linux, Windows, or macOS virtual machines provided by GitHub. In addition, self-hosted runners can also be installed in personal data centers or cloud infrastructures. (GitHub, 2023g)

2.1.2 GitHub Pages

GitHub Pages is a service that GitHub provides for free for public repositories to host static HTML, CSS, and JavaScript websites. By default, the site will be published publicly to github.io. To publish

a website from a private repository and/or to publish it to a custom domain it requires additional steps and/or subscription to GitHub Enterprise. (GitHub, 2023b)

2.2 EJS

Embedded JavaScript (EJS) is an Apache version 2 licensed templating language used to render JavaScript code with extended EJS features, syntaxes and/or tags to HTML. It enables developers to construct reusable HTML templates with dynamic data that can be rendered client- or server-side. EJS is frequently employed by web development frameworks such as Node.js and Express.js to construct dynamic web pages and web applications. (EJS, 2023)

2.3 EJS portfolio application

Is a web-based application created by the author of this thesis to showcase his professional skills and experiences. The application utilizes EJS and Bootstrap frameworks to develop an interactive and responsive interface for users to access the author's portfolio information. The application is designed to promote the author's capabilities to potential employers or clients by providing detailed information about his background, work experience, and skills.

2.4 SAST

Static Application Security Testing (SAST) is a technique in information security testing that analyzes source code or a compiled version of code to help identify security vulnerabilities. SAST tools can detect issues during software development and provide feedback to save time and effort. It can detect certain well-known vulnerabilities, such as buffer overflows and SQL injection flaws, and can scale well by running repeatedly. There are many SAST tools available which may vary in features and functionalities. (OWASP, 2023b)

2.4.1 Linting

Lint or linter is a type of SAST tool that is specifically used to find stylistic or programming errors. So, linting is a process of the linter analyzing code to find stylistic or programming errors. Depending on the programming language and the tool used, there are some conventions that

standardize rules and styling that can be followed to unify the code styling or pattern within the software development organization or project. (Ogut, 2021)

2.4.2 Secret scanning

It is arguable whether secret scanning is a type of SAST technique or process. However, secret scanning is a process to find secrets like passwords, tokens, API keys, and other forms of secrets within files. It will protect the developer or organization from secret disclosures that could result in a great loss. (Nightfall, 2023)

2.4.3 Dependency scanning

Dependency scanning is a process of analyzing source code to identify libraries or dependencies used by a project and their versions. It will then compare the found dependencies with the Common Platform Enumeration (CPE) identifier for the provided dependency. When the dependency and the CPE match, it will generate a report that contains a link to the associated Common Vulnerabilities and Exposures (CVE). (OWASP, 2023a)

2.4.4 Vulnerability scanning

Vulnerability scanning is a security practice used to identify weaknesses or vulnerabilities in computer systems, networks, or applications. It involves using automated tools or software to scan for known security flaws and misconfigurations that could potentially be exploited by attackers. Vulnerability scanners typically search for common vulnerabilities such as unpatched software, weak passwords, open ports, and insecure network configurations. The purpose of vulnerability scanning is to proactively identify security weaknesses and take appropriate measures to mitigate them, such as applying patches, fixing configurations, or implementing additional security controls. (Tech Target, 2023)

2.5 Software deployment

Software deployment is the procedure of providing users with new or updated software. Typical activities include provisioning environments, installing, and testing software, and monitoring the

health and efficacy of newly deployed environments on an ongoing basis. Software deployment involves deploying an application with additional functionality, problem fixes, or security patches, whereas software release is the iterative process of developing an application. (Codefresh, 2023)

3 Work management

This chapter will present briefly about the methodology used for managing tasks during the development of this thesis. This thesis uses Kanban as the methodology and GitHub Projects as the tool to visualize it.

3.1 Kanban

Kanban is a visual workflow management methodology that was first used in the manufacturing sector and has now been adopted by several industries, including software development. With the help of visualizing the flow of work, restricting the amount of work in progress, and continuously refining procedures, kanban strives to optimize workflow. Kanban's fundamental ideas include visualizing work, capping work-in-progress, regulating flow, making process policies explicit, putting feedback loops in place, improving collectively, and trying new things. Because of its adaptability and flexibility, Kanban is a work-management strategy that can be used in a variety of settings and sectors. (Kanbanize, 2023)

3.2 GitHub Projects

On GitHub, Projects is a flexible tool for organizing and tracking work. It allows users to create and customize multiple views, visualize work with configurable charts, and add custom fields to track metadata specific to their team. It has three different view types which are table view, board view (Kanban) and Roadmap view. Views and charts are automatically updated as a result of information syncing as changes are made to the project. Custom fields can be used to add metadata to issues, pull requests, and draft issues to build a richer view of item attributes. (GitHub, 2023c)

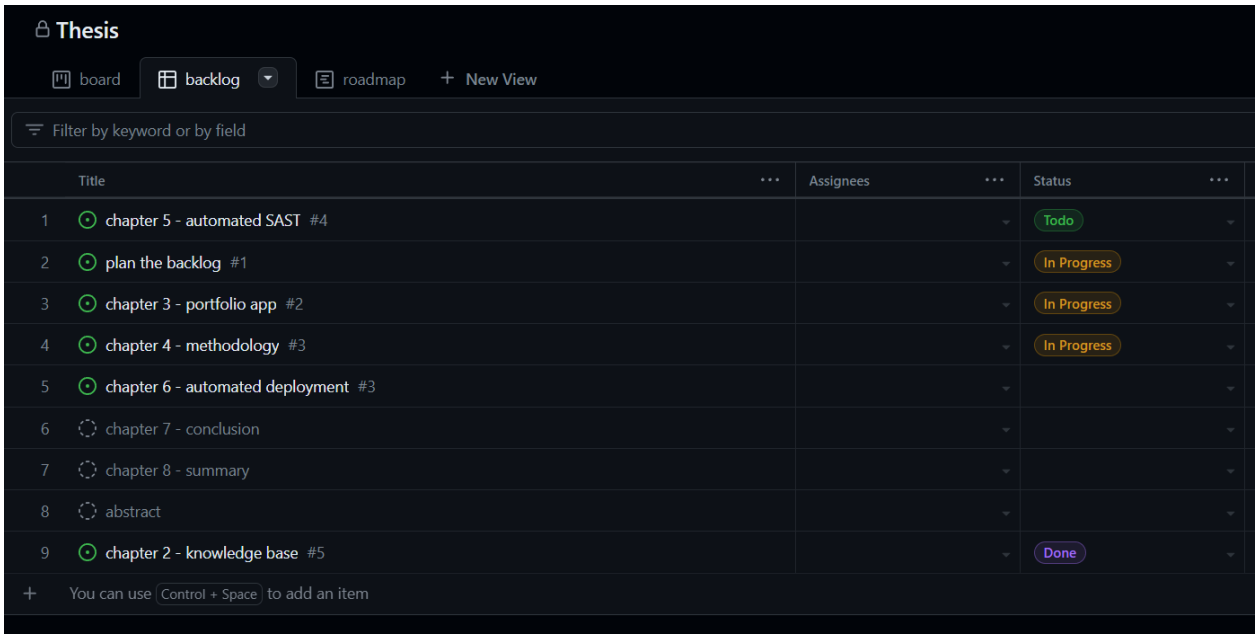
3.3 Planning and managing work with GitHub Projects

The author utilizes 3 different Projects' views to help organizing and tracking the tasks. Each of these views has its own use case and functionality that will be demonstrated within the next three sub-sections.

3.3.1 Table view

Table view Is utilized by the author of this thesis to effectively create and manage tasks. As shown in Figure 1, a comprehensive list of tasks is presented in a clear and organized manner by this view.

Figure 1 GitHub Projects table view



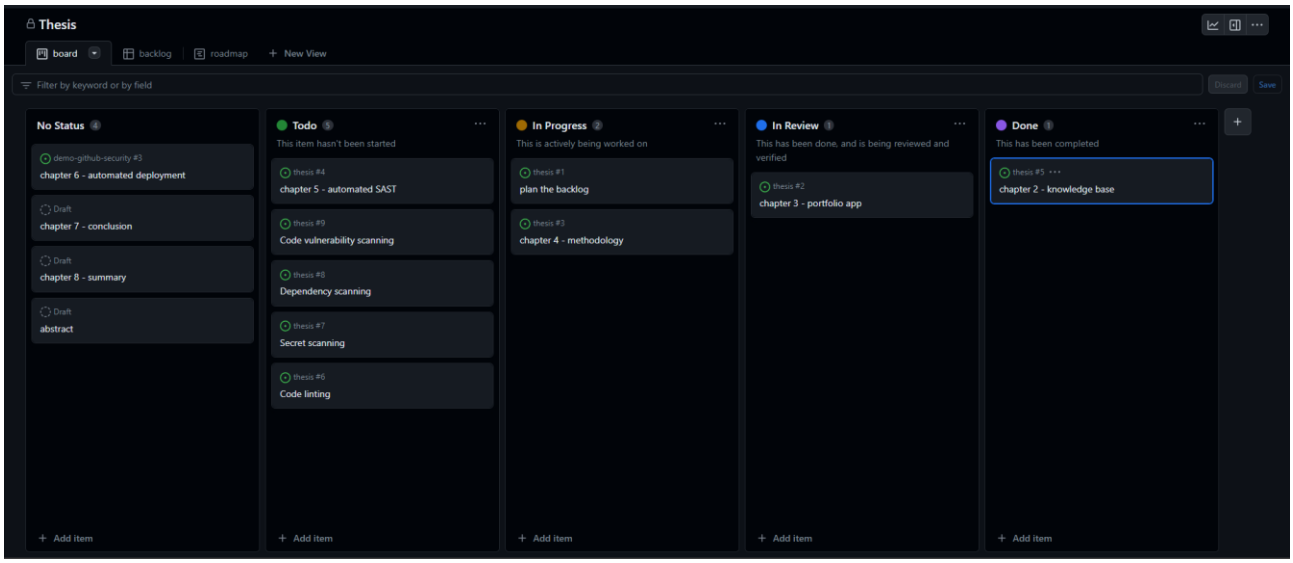
The default display includes the status of each task, which helps the author avoid creating duplicate tasks inadvertently. Furthermore, the table view allows for customization of the column fields to meet specific needs. For instance, one can replace the default "Assignees" column with "Target Date" or any other column that suits their requirements. Although only a few fields are initially available, users can create custom fields using supported field types, such as text, number, date, single select, and iteration.

3.3.2 Board view (Kanban)

GitHub Project Board view, also known as a Kanban board is a customizable tool that helps the author of this thesis to manage and track work items by displaying their status. This allows for easy prioritization, assignment, and progress tracking towards project goals. Tasks can be moved between columns, which represent different stages of the workflow or project.

Figure 2 shows the look of the board view, which consists of multiple columns and each column has multiple items.

Figure 2 GitHub Projects board view

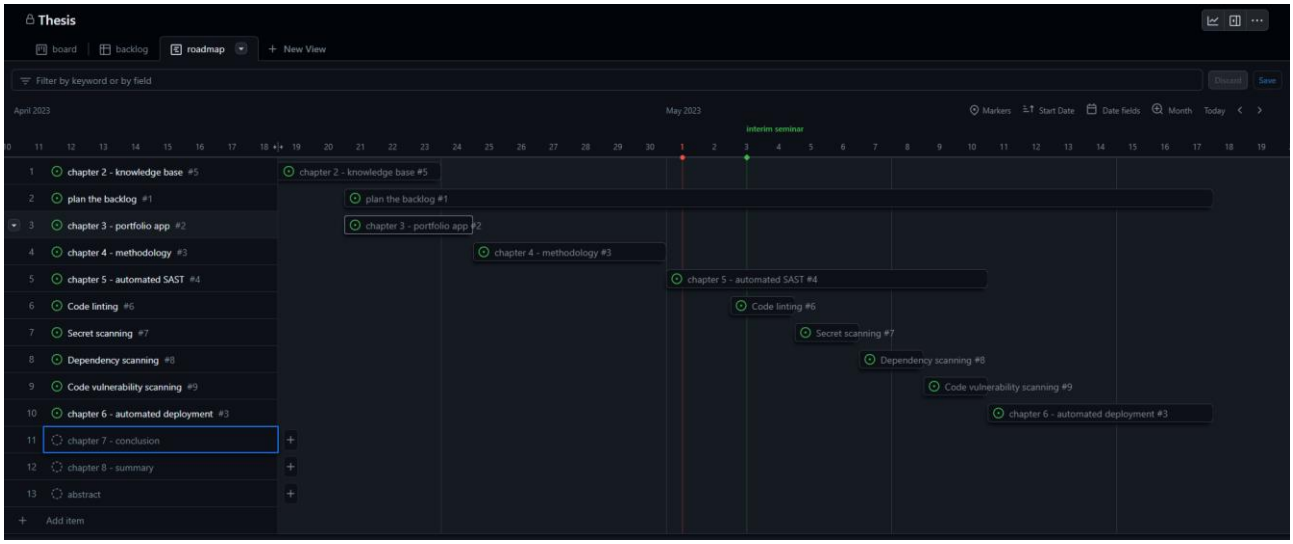


The Board view is highly customizable, and it allows us to add new columns, edit existing ones, and reorder them as needed. In addition, the view can be filtered based on criteria such as Assignee, Milestone, or other supported fields to provide a more targeted overview of project progress. For example, an "In Review" column is added in this case to provide more detailed information on the status of the task.

3.3.3 Roadmap view

Figure 3 shows that Roadmap view provides a visual representation of the project's timeline, milestones, and objectives, enabling the breakdown of the project into manageable tasks. It aids in prioritizing tasks and allocating resources to maximize productivity. It allows users to gain a clear understanding of the project's progress and timeline for completion.

Figure 3 GitHub Projects roadmap view



To utilize the roadmap view effectively, a comprehensive list of tasks required for project completion is created. The roadmap view is then used to organize these tasks into logical groups and establish realistic timelines for each group. Progress is tracked on the roadmap view, and adjustments are made accordingly.

4 Automated SAST

This chapter covers the practical aspect of implementing automated SAST by providing step-by-step instructions how each technique was set up and integrated into the Continuous Integration/Continuous Delivery (CI/CD) pipeline using GitHub Actions and Workflows. The examples presented in this chapter can be used as a guide for developers to incorporate automated SAST into their software development process and ensure that the code is thoroughly tested before being deployed to production.

Program code 1 shows the common events that were used to trigger the Workflows

Program code 1 Workflows trigger

```
on:
  pull_request:
    branches: [main, development, v0.2**]
```

It means that the workflow will be triggered when a new pull request is created on main, development or any branch that has **v0.2** prefix.

4.1 Code linting with GitHub Superlinter

GitHub SuperLinter is an open-source tool that supports several languages and performs code linting by checking the code against a set of pre-configured rules (GitHub, 2023e). This section explains how GitHub SuperLinter was set up and used in the CI/CD pipeline.

First, a **.yml** file was created within **.github/workflows** directory and named descriptively, such as **superlinter.yml**. This file consists of three primary sections: workflow trigger, GitHub Checkout action, and the GitHub Superlinter action. As outlined in the previous chapter, the common workflow trigger would be utilized within this workflow.

Program code 2 shows a code block to utilize GitHub Checkout Action into the workflow.

Program code 2 GitHub Checkout action

```
- name: Checkout Code
        uses: actions/checkout@v3
        with:
          fetch-depth: 0
```

By default, the workflow does not have access to the repository files. However, the Checkout action can be utilized to retrieve the repository and make it accessible within the workflow. Depending on the event that triggers the workflow this action will checkout the code into the branch related to the event. Since the trigger event was defined to be a pull request on main, development or any branch with **v0.2** prefix, it will checkout to the branch related to the pull request respectively as the main branch. Meaning it will checkout to the respective branch, but git will see it as it is on the main branch.

Program code 3 shows a code block to utilize GitHub Superlinter action into the workflow.

Program code 3 GitHub Superlinter action

```
- name: Lint Code Base
        uses: github/super-linter@v5
        env:
          VALIDATE_ALL_CODEBASE: true
          DEFAULT_BRANCH: main
          LOG_LEVEL: WARN
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

This code scans the main branch of the current workspace within the workflow, which serves as the related branch for the pull request. By setting the **LOG_LEVEL** variable to **WARN**, the logging system will only report errors and warnings, thereby improving log readability.

Program code 4 shows how the final code should look like.

Program code 4 superlinter.yml

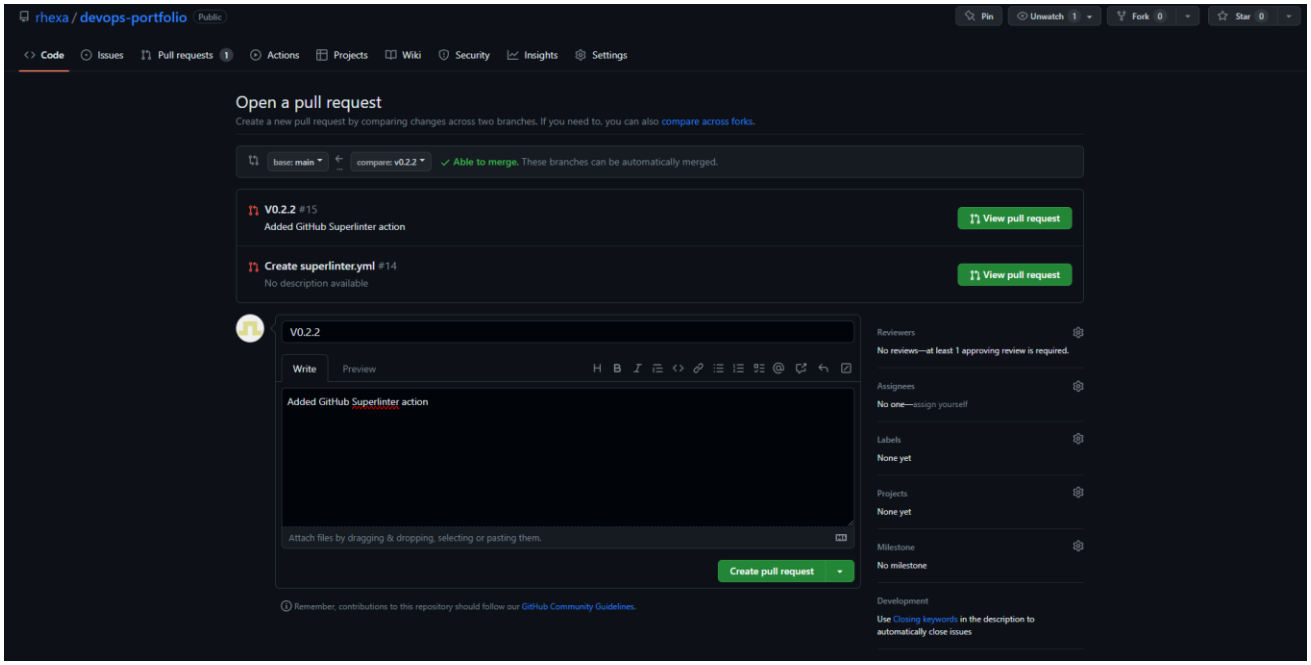
```
name: Lint Code Base
on:
  pull_request:
    branches: [main, development, v0.2**]
jobs:
  lint:
    name: Lint Code Base
    runs-on: ubuntu-latest
```

```
steps:
  - name: Checkout Code
    uses: actions/checkout@v3
    with:
      fetch-depth: 0
  - name: Lint Code Base
    uses: github/super-linter@v5
    env:
      VALIDATE_ALL_CODEBASE: true
      DEFAULT_BRANCH: main
      LOG_LEVEL: WARN
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

To confirm the successful implementation of the GitHub Superlinter action, a new pull request was created and directed towards the specified branch, which in this case was the main branch.

Figure 4 shows a page where pull request can be created.

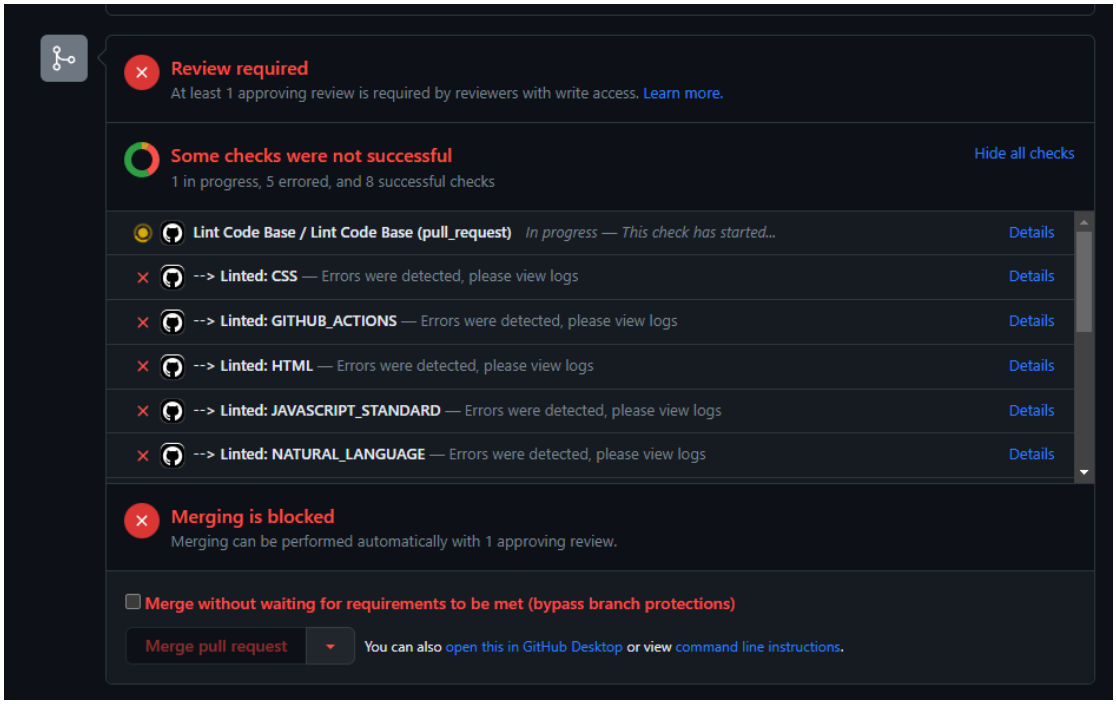
Figure 4 Creating a new pull request



To create a pull request, click the pull request tab. Select new pull request, specify the main branch as the target and **v0.2.2** branch as the source, compose a brief description, and click the create pull request button.

Figure 5 shows the status of the workflow.

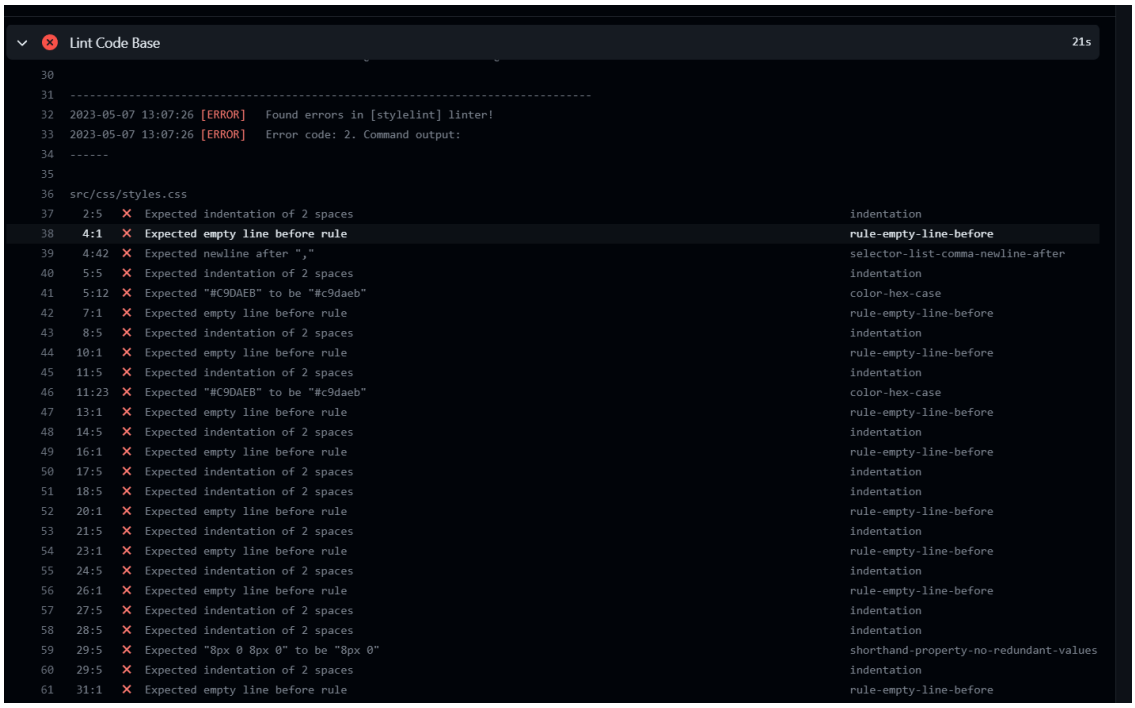
Figure 5 GitHub Superlinter action status on pull request



Upon creation of a pull request, the workflow will be triggered automatically, and its status can be viewed on the pull request page. Click the Details link to see the detailed status or error found by the GitHub Superlinter action.

Figure 6 shows the action log that contains a list of linting errors and warnings including their respective line numbers.

Figure 6 GitHub Superlinter action logs



This is a good sign that the GitHub Superlinter action is functioning as intended.

4.2 Secret scanning with GitHub Secret Scanner

Secrets, such as passwords and API keys, are sensitive information that must be kept secure. GitHub Secret Scanner is a tool that scans the code for potential secrets and alerts the developer if any are found (GitHub, 2023d). This section explains how GitHub Secret Scanner was set up and used.

To activate the secret scanner feature, it has to be enabled on the repository settings.

Figure 7 shows the main page of the repository where the repository settings can be found.

Figure 7 GitHub repository main page

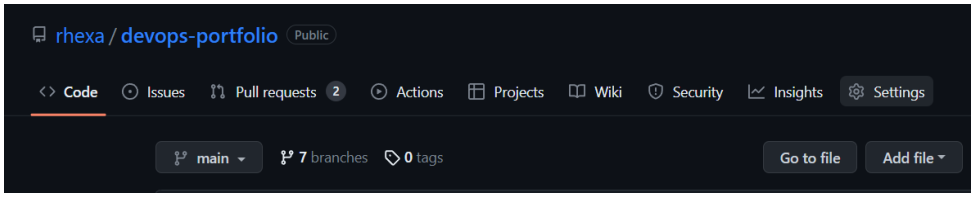
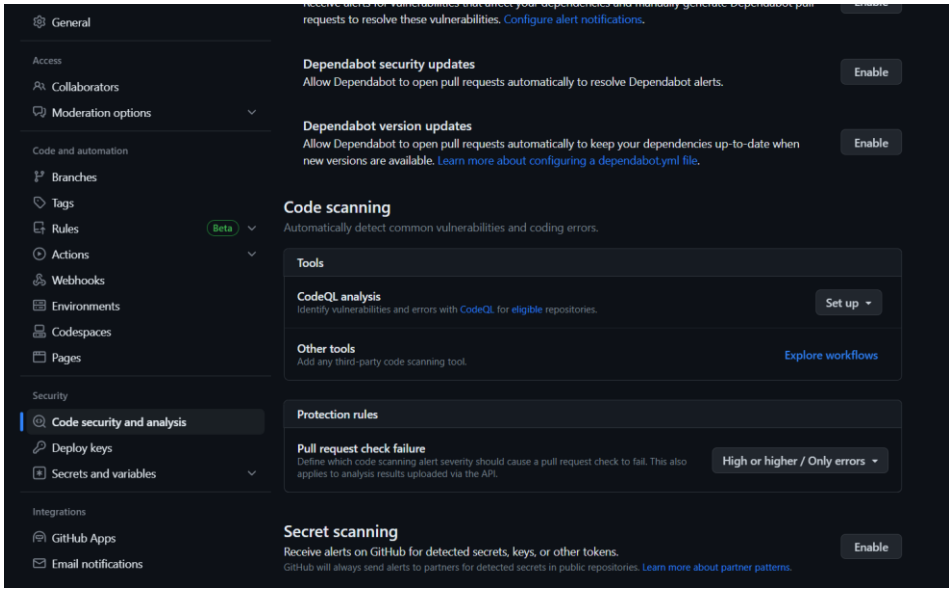


Figure 8 shows the available settings for the repository.

Figure 8 GitHub repository settings



Click Code security and analysis in Security section on the sidebar. Scroll down and find the Secret scanning feature then click the Enable button.

To verify whether the secret scanning is operational as planned, a dummy API token is needed. Program code 5 shows the dummy API token used for this research.

Program code 5 Bash dummy API token

```
HEROKU_API_KEY='dummy_abcdefgh1234567890'
```

This dummy API token was then added to a bash script as illustrated on Program code 6.

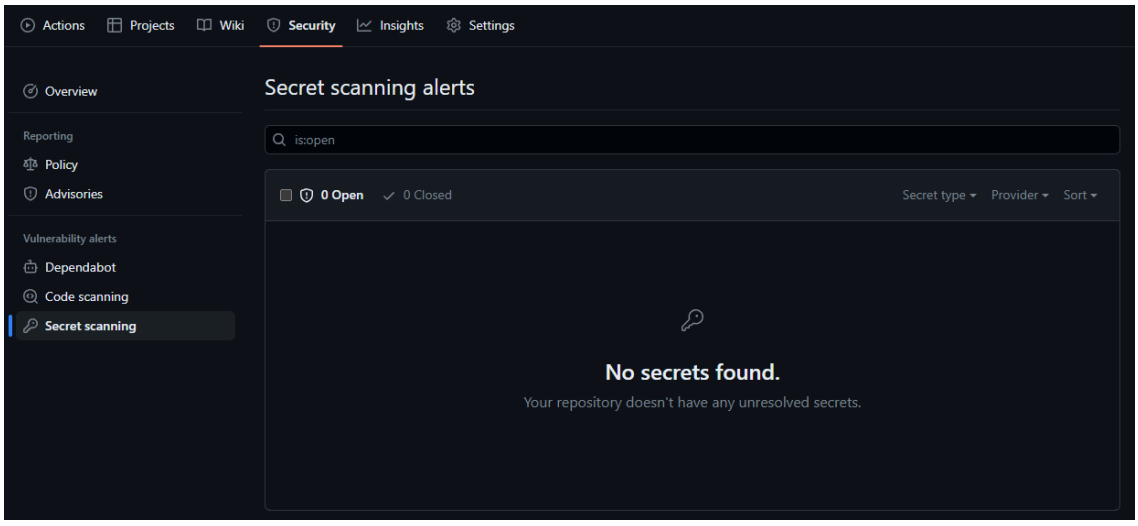
Program code 6 heroku.sh

```
#! /bin/bash

HEROKU_API_KEY='dummy_abcdefgh1234567890'

echo "machine api.heroku.com
  login $HEROKU_EMAIL
  password $HEROKU_API_KEY
machine git.heroku.com
  login $HEROKU_EMAIL
  password $HEROKU_API_KEY" > ~/.netrc
```

Once the changes had been committed and pushed to the repository, a new pull request was created to initiate the workflow.

If the secret scanning is working properly, an alert should be shown in the pull request indicating that a potential secret has been detected. Alternatively, the security page can be examined to review any secret scanning alerts that have been generated. Unfortunately, after waiting for several hours, alerts had never shown on the dashboard as shown in Figure 9. This could be due to the dummy token not being supported by the secret scanner.

Figure 9 Secret scanning alerts



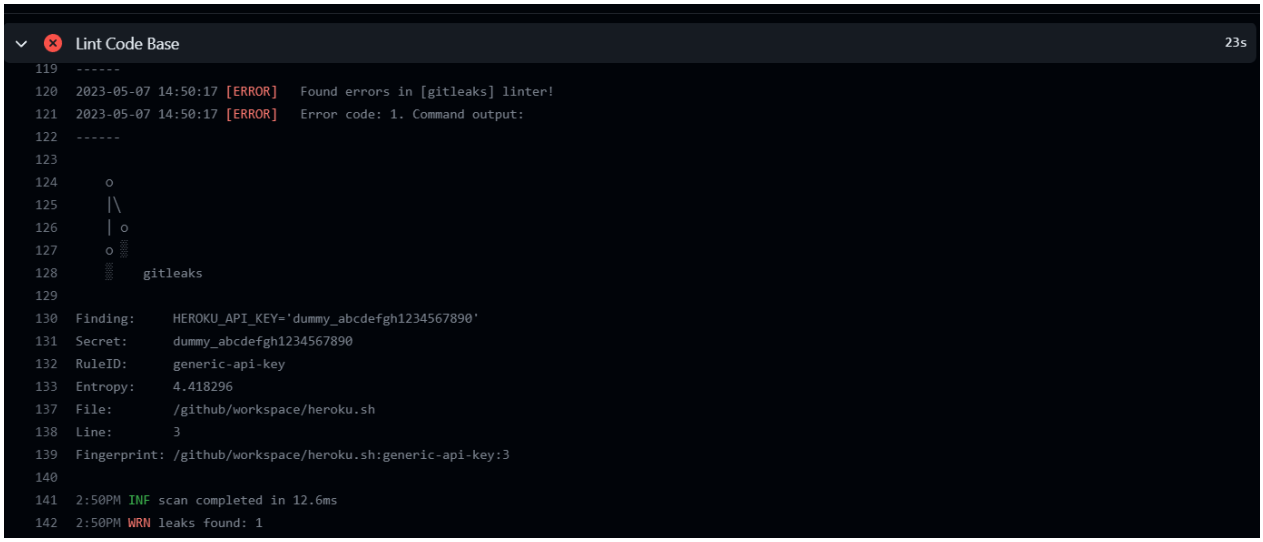
The author opted for an alternative solution by utilizing GitLeaks, which is supported by the GitHub Superlinter.

GitLeaks is a freely available, open-source secret scanner that detects various types of confidential information such as passwords, API keys, and tokens, and is compatible with several platforms,

including Windows, MacOS, Linux, and Docker (Gitleaks LLC, 2023). As the GitHub Superlinter action has been configured, this functionality is readily available and enabled by default. To assess the Gitleaks feature, the author revisited the pull request that was created when the dummy API token was committed.

As demonstrated in Figure 10, the GitHub Superlinter logs revealed that GitLeaks detected the dummy token instantly, resulting in a better user experience compared to the GitHub Secret Scanner.

Figure 10 Gitleaks status on GitHub Superlinter logs

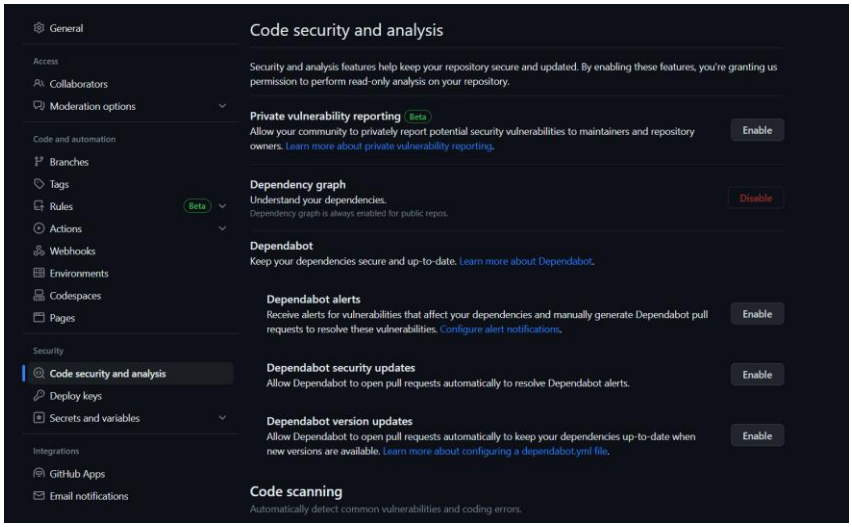


4.3 Dependency scanning with GitHub Dependabot

GitHub Dependabot is a tool that scans the application's dependencies and checks for known vulnerabilities (Mullans, 2020). This section explains how GitHub Dependabot was set up and used.

To uitilize GitHub Dependabot, it needs to be enabled on the repository settings. First, open Code security and analysis settings that can be found in the repository settings. Find Dependabot security updates setting as shown on the Figure 11 then click enable.

Figure 11 Repository settings to enable Github Dependabot



GitHub Dependabot is now enabled. To verify whether it is working as intended, a vulnerable dependency should be added to the project dependency.

Program code 7 shows a code block that contains vulnerable dependency. This code was added to the dependency section within the **package.json** file.

Program code 7 Node JS vulnerable dependency

```
"dependencies": {
  "underscore": "1.11.0"
}
```

As stated on the documentation, GitHub Dependabot will scan the default branch in the repository (GitHub, 2023f). In this case the changes made in the **package.json** were committed on the main branch.

As shown in Figure 12, a new alert had been created and showed the vulnerable dependency added previously. It is possible to see the vulnerability in detail by clicking the title of the vulnerable item.

Figure 12 GitHub Dependabot alerts

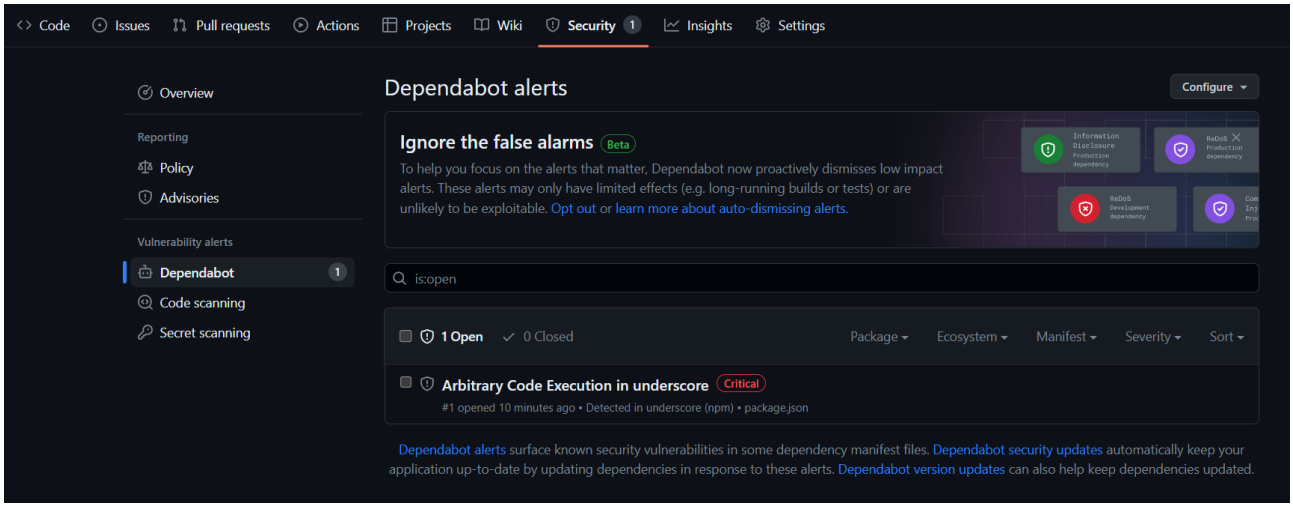
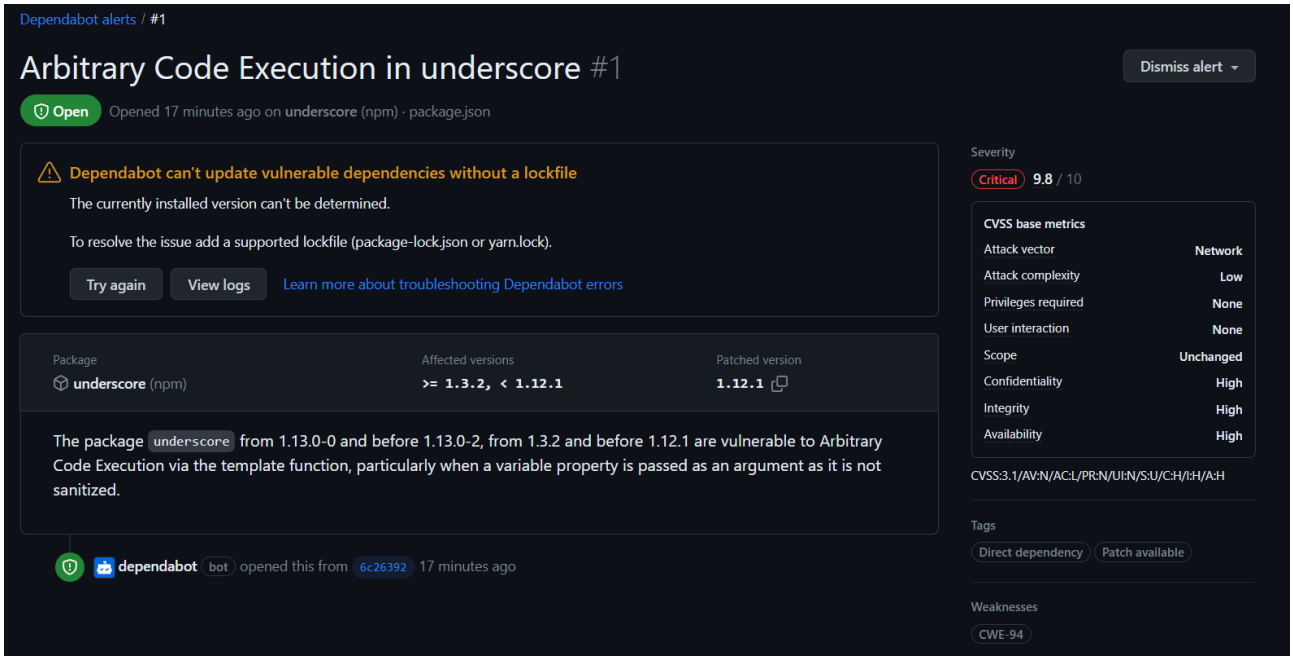


Figure 13 shows the vulnerability detailed page which includes the description of the vulnerability, affected version, severity and other details related to the vulnerable dependency.

Figure 13 GitHub Dependabot vulnerability detailed view



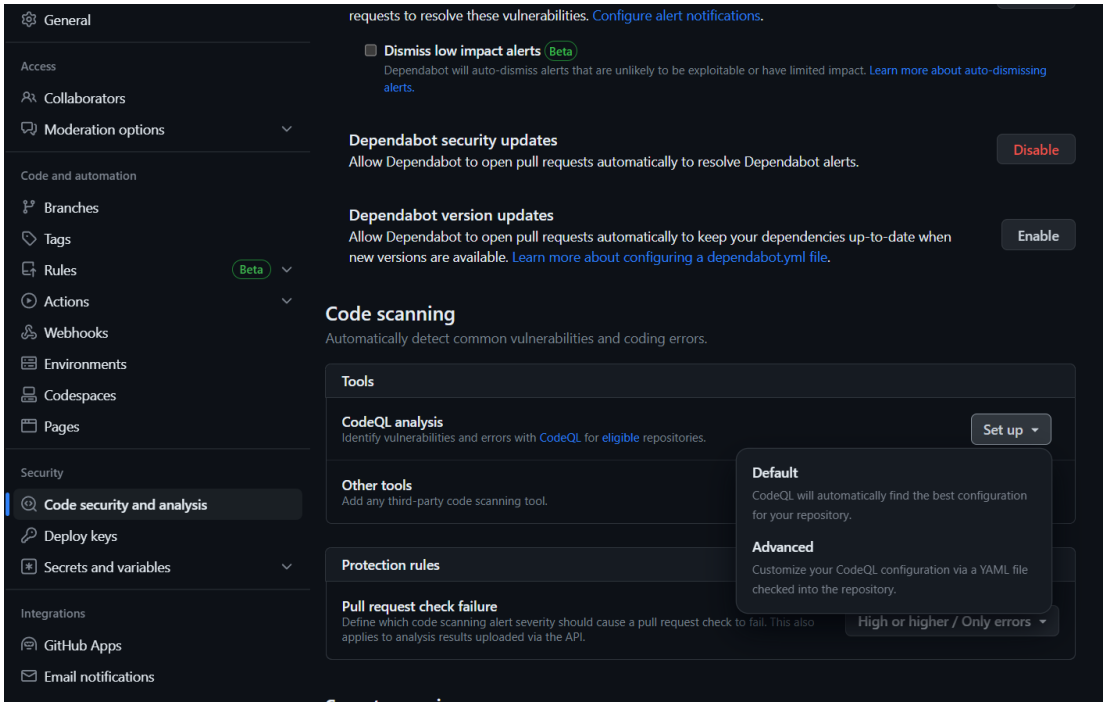
While catching vulnerabilities during a commit push to the main branch is useful, it would be even better if GitHub Dependabot could scan the repository when a pull request is created. This would allow the detection and prevention of vulnerable dependencies before they are merged into the main branch.

4.4 Code vulnerability scanning with GitHub CodeQL

CodeQL is a tool developed by GitHub that performs comprehensive code analysis to identify potential security vulnerabilities and errors (GitHub, 2023a). This section explains how GitHub CodeQL was set up and used.

Figure 14 shows the settings where GitHub CodeQL can be enabled.

Figure 14 Repository settings to enable code scanning with GitHub CodeQL



To enable GitHub CodeQL in the repository, open Code security and analysis settings that can be found in the repository settings. Find CodeQL analysis item within the Code scanning section, click Set up then select Default.

As shown on the Figure 15, CodeQL will automatically detect the existing programming language within the repository. In this case it detected JavaScript. Click the Enable CodeQL button.

Figure 15 GitHub CodeQL default configuration

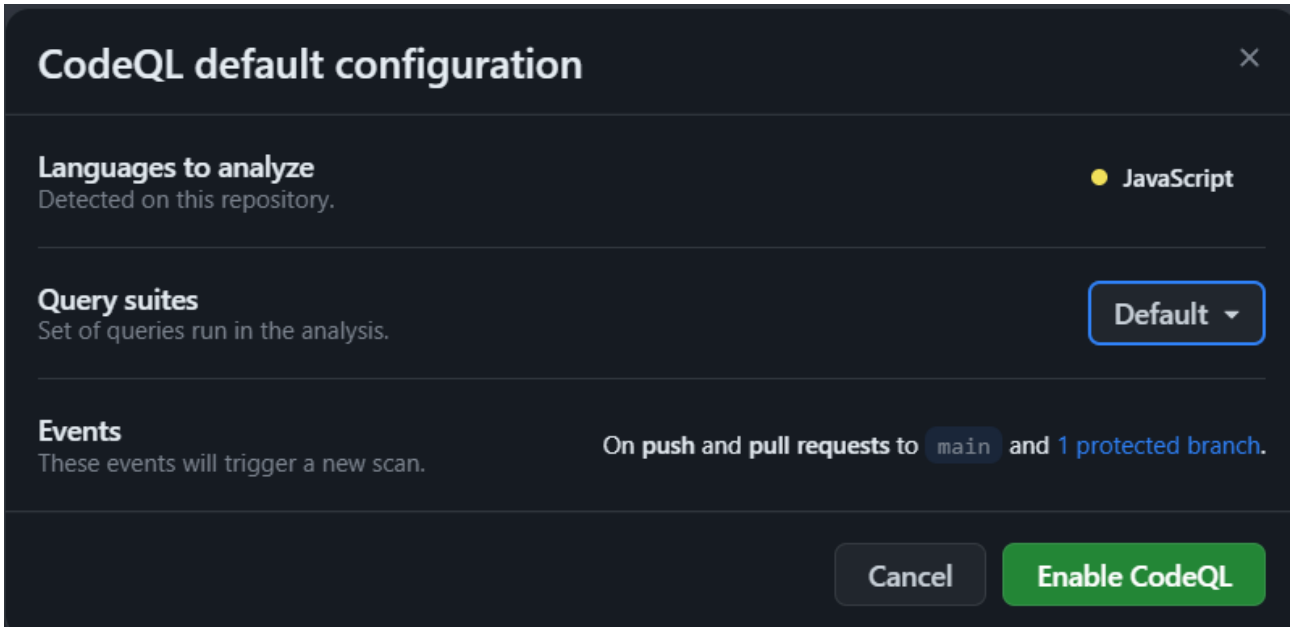


Figure 16 shows that GitHub CodeQL was configuring itself for the first time and it took some time to finish.

Figure 16 GitHub CodeQL is setting up the feature

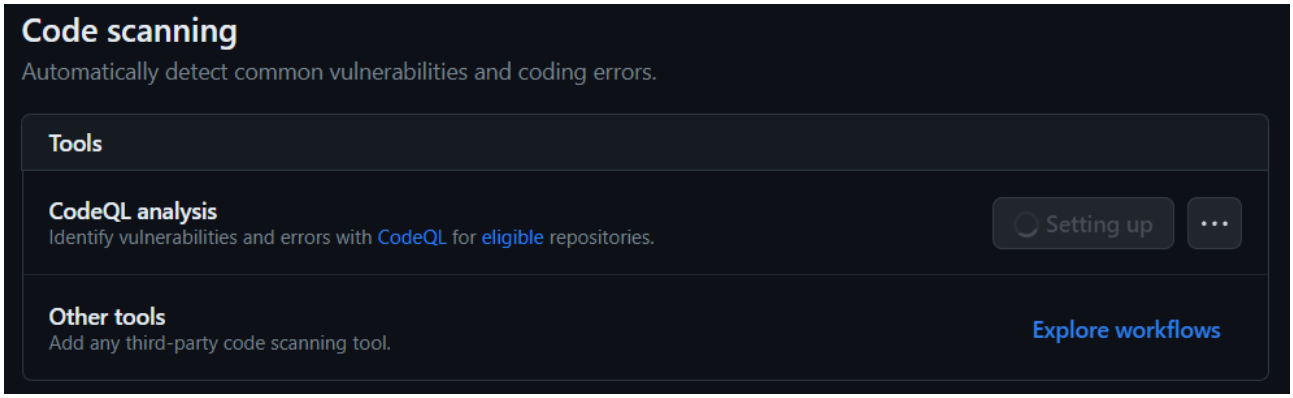
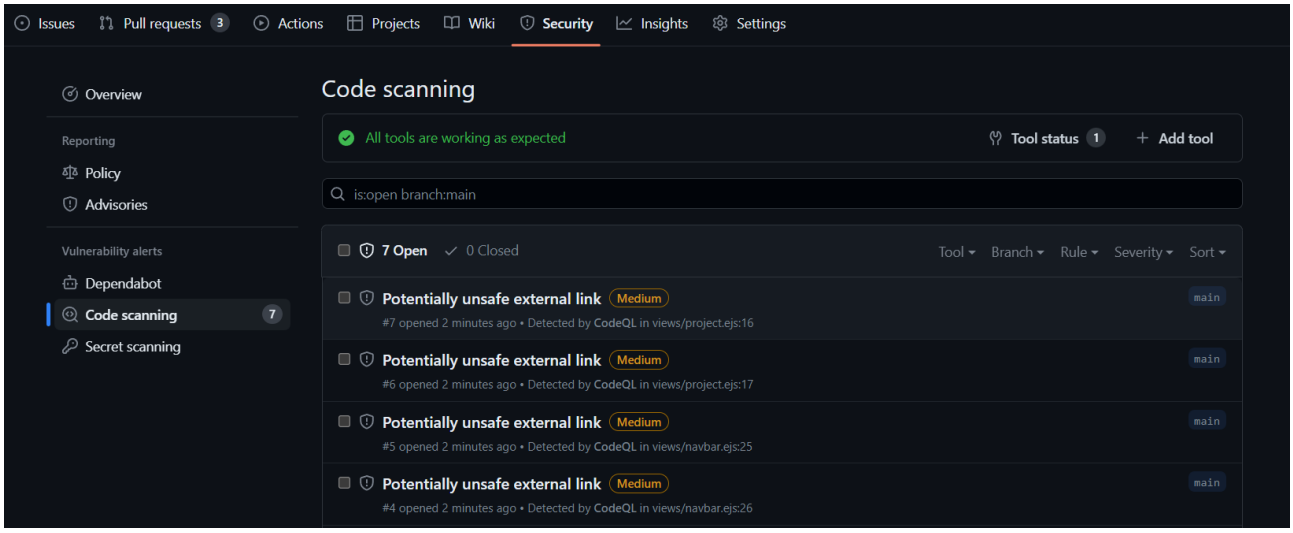


Figure 17 shows all the notifications and warnings have been listed on the code scanning dashboard.

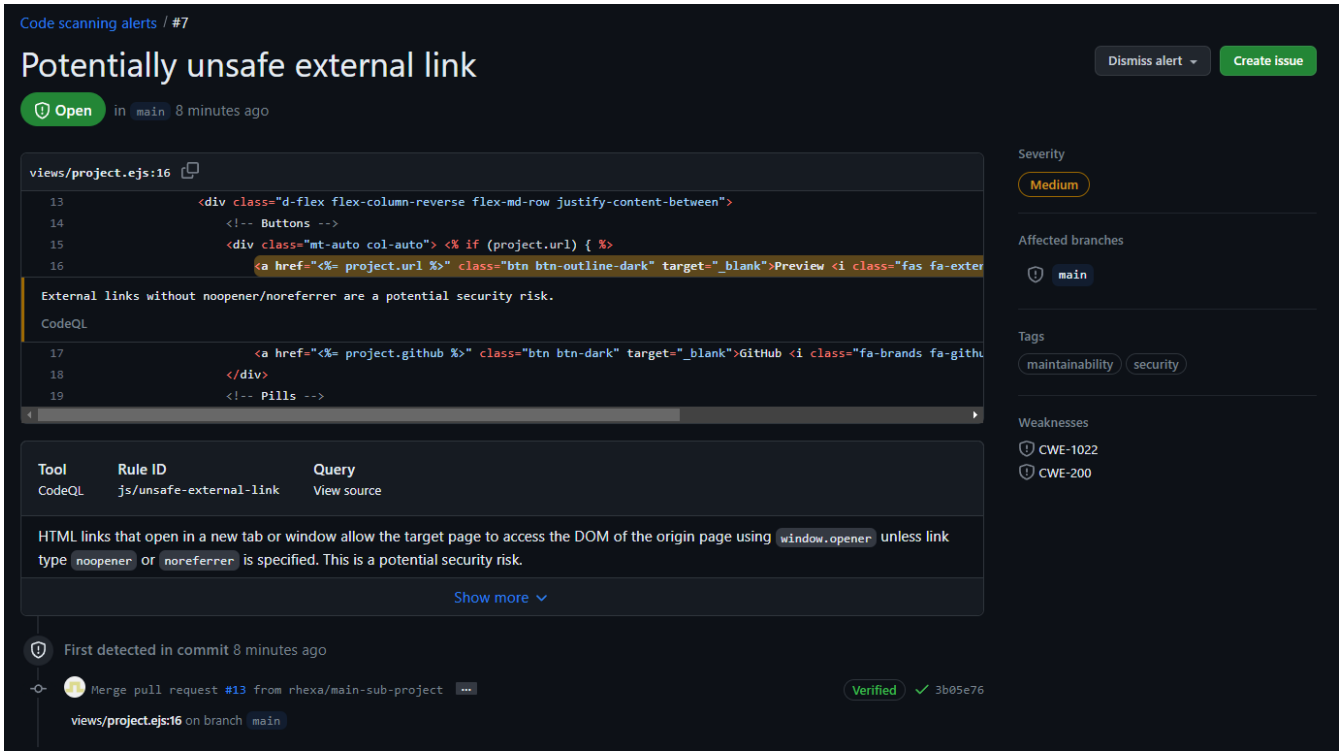
Figure 17 Code scanning dashboard



The code scanning dashboard can be accessed by clicking the Security tab on the sub menu panel then select code scanning on the left side menu. Each individual item can be clicked to see the detailed information about the vulnerability.

Figure 18 shows the CodeQL detailed page presents more information about the vulnerability such as where the vulnerability is found, what it is about, how it looks like, how to solve it, and references to external web sites that provide information about the vulnerability. Alert can be dismissed by clicking the Dismiss alert button on the top right corner of the screen. It is also possible to create an issue to track this vulnerability from the vulnerability detailed page.

Figure 18 GitHub CodeQL detailed view



5 Automated deployment

This chapter focuses on the automated deployment of the EJS web application using GitHub Actions and Workflows. It provides explanations and step-by-step instructions on how to automate the deployment process, ensuring a smooth and efficient deployment to GitHub Pages.

To begin, create two branches from the main branch, development, and github-pages respectively. Then, a **.yml** file need to be created within **.github/workflows** directory inside the github-pages branch and name it descriptively, such as **github-pages.yml**. This workflow will be utilized throughout the entire chapters and comprises three primary components: common configuration, build job, and deploy job. The subsequent sub-chapters provide detailed explanations of the build job and deploy job, respectively.

Program code 8 shows a common configuration that will be used for every job.

Program code 8 gitlab-pages.yml common configuration

```
name: Deploy GitHub Pages

on:
  push:
    branches: ["development"]
  pull_request:
    branches: ["development"]

permissions:
  contents: read
  pages: write
  id-token: write
```

The common configuration section is divided into three parts: workflow name, trigger events, and workflow token permissions. The workflow name serves to provide a distinctive identifier for the workflow. It is important to assign a descriptive name to it, allowing for easy recognition. As for the events that can trigger the workflow, they include push events on the development branch and pull request events on the development branch. The selection of the development branch ensures that all components are functioning properly before merging them with the main branch. Lastly, the workflow token permissions are specified to authorize the workflow with the necessary permissions for deploying to GitHub Pages.

5.1 Automated HTML generation from EJS

There are two deployment methods available for EJS applications. These options involve executing the EJS application using NodeJS and generating a static HTML file from the EJS files. In the subsequent chapter, the application will be deployed on GitHub Pages. As GitHub Pages exclusively supports static HTML files, the latter approach will be adopted, namely generating a static HTML file from the EJS files.

Program code 9 shows build job in detail.

Program code 9 gitlab-pages.yml build job

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Setup Node JS
        uses: actions/setup-node@v3
        with:
          node-version: 14
      - name: Build HTML
        run: npm i && npm run build
      - name: Upload artifact
        uses: actions/upload-pages-artifact@v1
        with:
          path: 'build/'
```

The build job utilizes three actions: Checkout action, Setup node action, and Upload pages artifact action. The Checkout action ensures that all files within the repository are made available within the working directory of the workflow. The Setup node action assists in the installation and configuration of Node JS, enabling the execution of npm commands within the workflow. And the Upload pages artifact action provides a solution for packaging and uploading artifact that can be deployed to GitHub Pages.

The complete workflow for the build job entails the following steps: checking out the code from the current branch, installing Node JS version 14, installing Node JS dependencies using the **npm i** command, generating HTML files from EJS using the **npm run build** command, and packaging and uploading the build folder as an artifact for the GitHub Pages deployment.

5.2 Automated deployment to GitHub Pages

Automated deployment to GitHub pages is defined withing the deploy job. Program code 10 shows the deploy job in detail.

Program code 10 github-pages.yml deploy job

```
deploy:
    needs: build
    if: ${{ github.event_name == 'push' }}
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v2
```

The deploy job only utilizes deploy-pages action to deploy the artifact uploaded on the previous chapter. **Needs** syntax on the workflow is used to specify the dependency on the build job, therefore it will wait for the build job to finish before executing the deploy job. **If** syntax is used to allow the workflow to run the job when the condition matches. In this case, it should run the job only if the workflow is triggered by push event.

5.3 Verifying successful deployment

On the previous chapter, the deployment job was defined but had not been executed. Before the deployment can be verified, it should be executed. So, first ensure that the final codes written inside the **github-pages.yml** look like the snippet shown on the Program code 11. And commit the changes made into the github-pages branch.

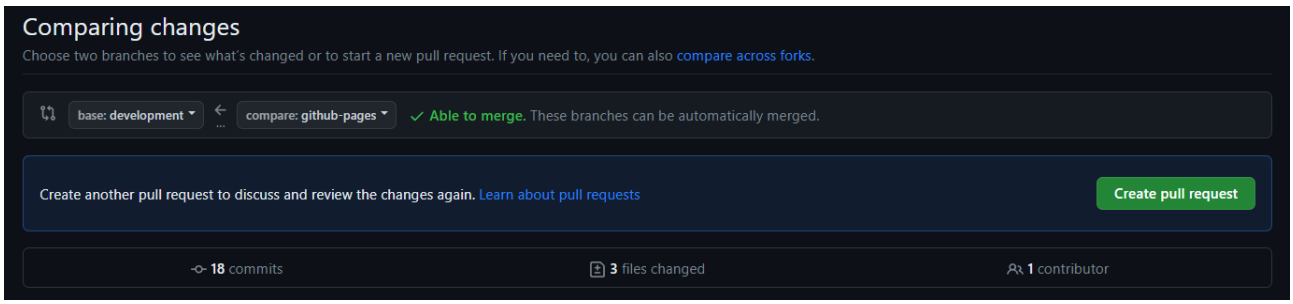
Program code 11 github-pages.yml

```
name: Deploy GitHub Pages
on:
  push:
    branches: ["development"]
  pull_request:
    branches: ["development"]
permissions:
  contents: read
  pages: write
  id-token: write
jobs:
```

```
build:
  runs-on: ubuntu-latest
  steps:
    - name: Checkout
      uses: actions/checkout@v3
    - name: Setup Node JS
      uses: actions/setup-node@v3
      with:
        node-version: 14
    - name: Build HTML
      run: npm i && npm run build
    - name: Upload artifact
      uses: actions/upload-pages-artifact@v1
      with:
        path: 'build/'
deploy:
  needs: build
  if: ${{ github.event_name == 'push' }}
  environment:
    name: github-pages
    url: ${{ steps.deployment.outputs.page_url }}
  runs-on: ubuntu-latest
  steps:
    - name: Deploy to GitHub Pages
      id: deployment
      uses: actions/deploy-pages@v2
```

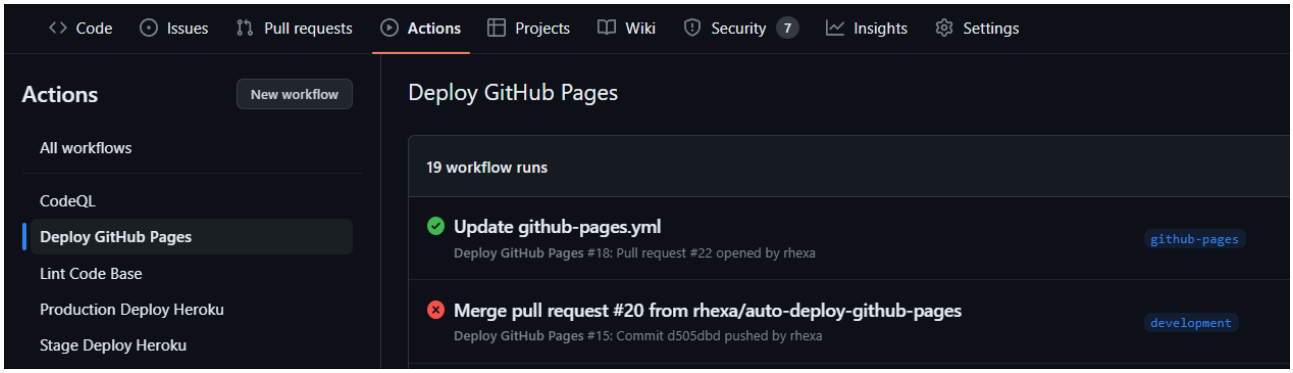
Create a pull request, set the development branch as the base branch and github-pages as the branch to compare as shown on Figure 19.

Figure 19 github pages to development pull request



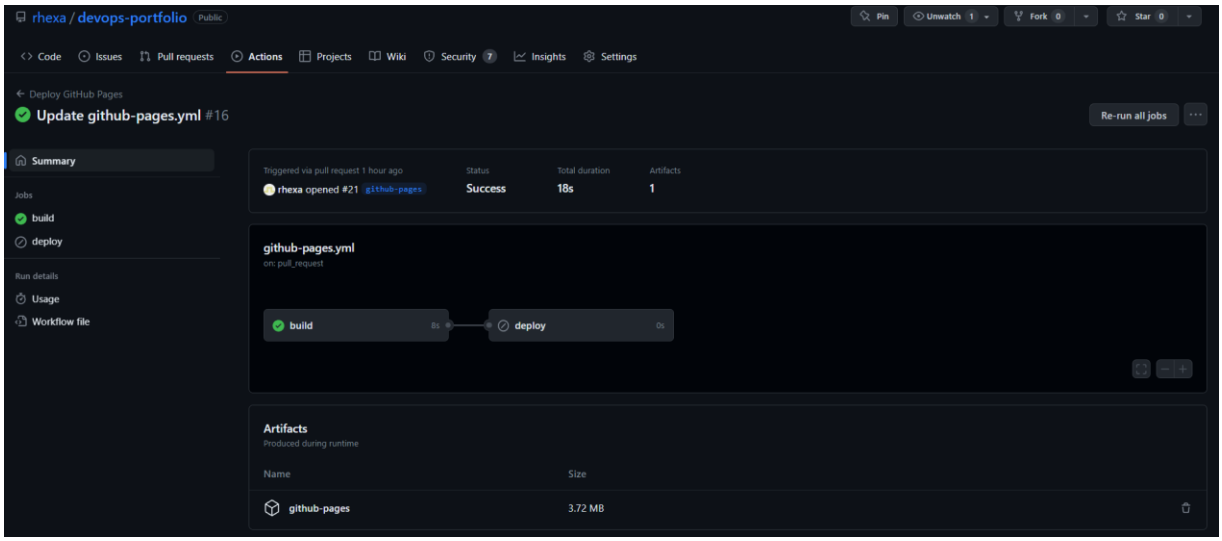
Go to the action overview page by clicking the Actions tab, select Deploy GitHub Pages workflow then select the latest workflow run which is usually located on the top of the list as shown on the Figure 20.

Figure 20 Deploy GitHub Pages workflow run



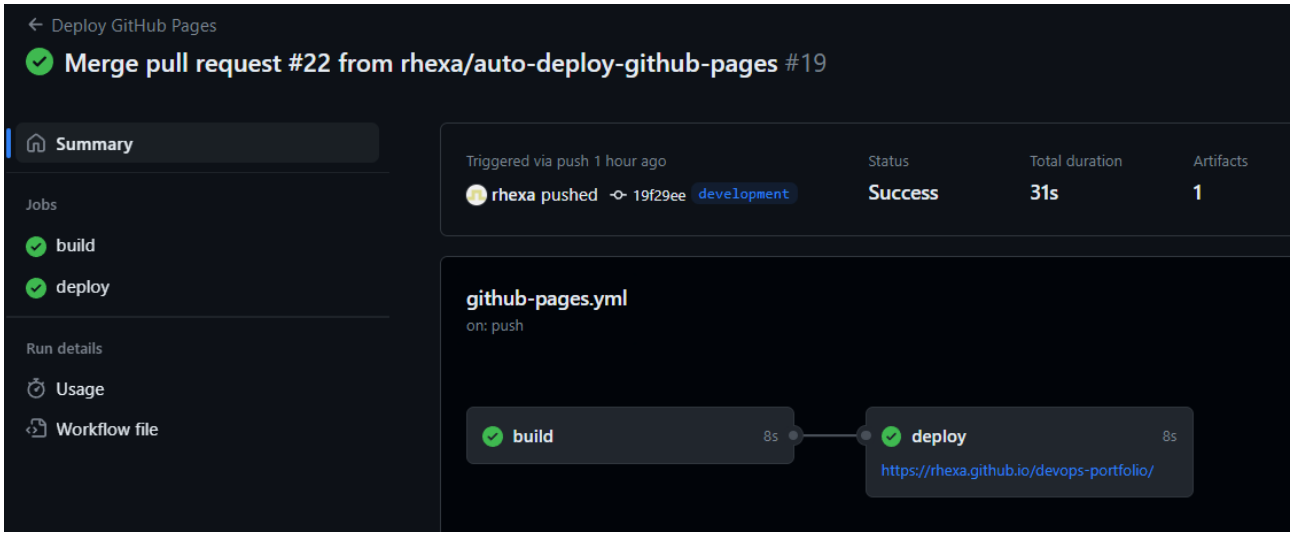
As shown on Figure 21, a successful build job will show green checklist sign and github-pages artifact within the Artifact section. Notice that the deploy job has a grey sign meaning it is skipped because it has been defined that the deploy job should only be run on push event. When creating a pull request, it will emit the pull request event, therefore the deploy job is not run.

Figure 21 pull request triggered workflow overview



Continue merging the branch and open the latest workflow run overview page. Now the deploy job shows green checklist sign with the deployed GitHub Pages URL as shown on the Figure 22.

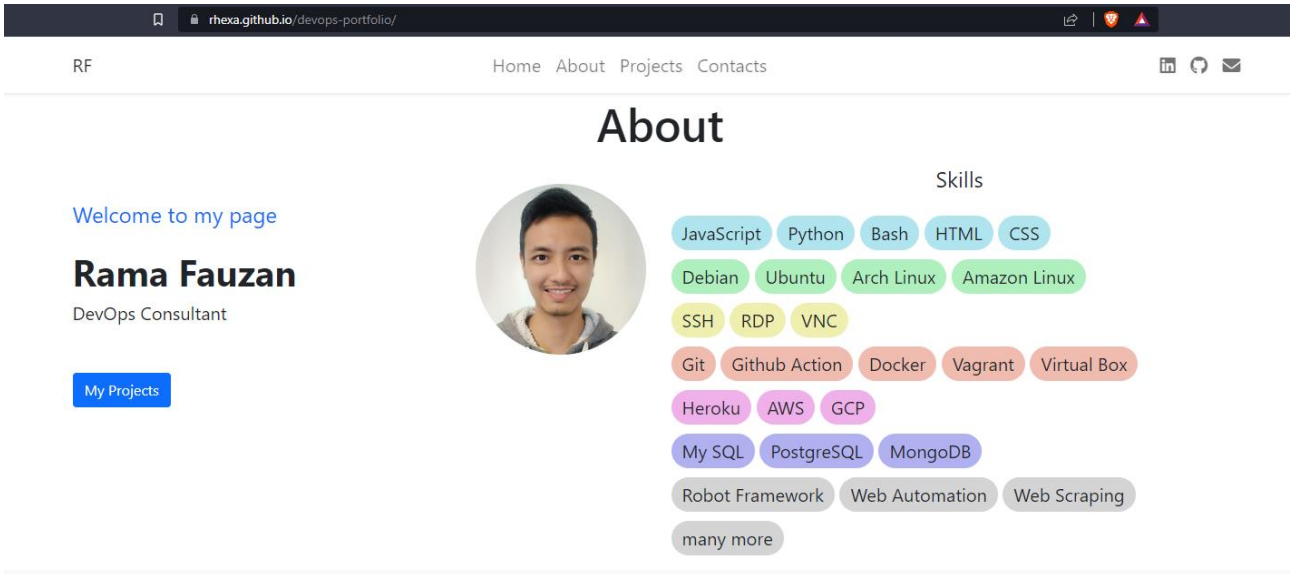
Figure 22 push triggered workflow overview



Follow the URL and verify whether the portfolio web application has been deployed successfully.

Figure 23 shows the portfolio web application is running on the URL followed on the previous step, meaning that it has been deployed successfully to the GitHub Pages.

Figure 23 portfolio web application on GitHub Pages



6 Advantages and challenges

This chapter delves into the advantages and challenges associated with the implementation of automated SAST and deployment processes using GitHub's built-in security features and GitHub Actions.

6.1 Advantages

In this section, the advantages gained from implementing automated SAST and deployment processes will be explored.

By integrating automated SAST tools into the development workflow, developers can identify and address potential vulnerabilities, code errors, and stylistic inconsistencies more efficiently. This leads to higher code quality and reduces the likelihood of introducing security vulnerabilities or bugs.

Automated vulnerability scanning, and secret detection help identify and mitigate potential security risks in the codebase. By proactively addressing security vulnerabilities, developers can protect sensitive information and ensure a more secure application.

Automation reduces the manual effort and repetitive tasks involved in the SAST and deployment processes. Developers can focus their time and energy on more critical aspects of development, such as feature implementation or performance optimization. This leads to increased productivity and faster time-to-market.

Automated deployment processes using GitHub Actions and Workflows ensure consistent and error-free deployments to GitHub Pages. With standardized workflows and configurations, developers can minimize deployment-related issues and deliver a reliable user experience.

Table 1 below summarizes the advantages that have been discussed within this chapter.

Table 1 Automated SAST and EJS deployment advantages

Advantages
Improved code quality
Enhanced security
Increased efficiency
Consistent and reliable deployments

6.2 Challenges and Limitations

While there are numerous advantages to implementing automated SAST and deployment processes, it is crucial to acknowledge the challenges and limitations that may arise. The challenges experienced during the research are explained below.

The initial setup and configuration of GitHub Actions and Workflows may require developers to familiarize themselves with new tools and technologies. This learning curve may involve understanding the syntax, configuring workflows, and troubleshooting potential issues. However, once the initial learning phase is overcome, the long-term benefits become apparent.

Integrating various SAST tools, configuring dependencies, and ensuring compatibility with the existing development stack can introduce technical complexities. Developers may encounter challenges related to tool integration, workflow customization, and maintaining a consistent development environment across different projects.

Automated SAST tools may generate false positive or false negative results, requiring manual verification and validation. Developers need to carefully review and interpret the scan results to

ensure accurate identification of vulnerabilities or issues. This process of reviewing results adds an additional layer of effort and attention to detail.

Adopting automated SAST and deployment processes may require changes to existing development workflows and organizational culture. Resistance to change or the need to align with established practices and policies can pose challenges during implementation. Ensuring proper communication, training, and collaboration among team members becomes essential for successful adoption.

Table 2 below summarizes the chalanges that have been discussed within this chapter.

Table 2 Automated SAST and EJS deployment challenges

Challenges
Learning curve
Technical complexity
False positives and negatives
Organizational adaptation

In addition to the challenges, certain limitations were identified during the implementation of automated SAST and deployment processes using GitHub Actions and Workflows.

The GitHub Dependency Scanner only scans the repository when triggered by push events to the default branch. This limitation means that vulnerabilities in dependencies may go undetected until after they are merged into the default branch.

The GitHub Secret Scanner lacks the ability to scan the repository instantly on pull request events, thereby failing to prevent secrets from being pushed to the default branch. This limitation

increases the risk of secret leakage and compromises the overall security posture. Additionally, the scanner's limited support for detecting secrets from certain platforms further restricts its effectiveness.

By addressing these challenges and limitations, developers and organizations can proactively mitigate risks, improve processes, and leverage the advantages offered by automated SAST and deployment using GitHub's built-in security features and GitHub Actions.

Table 3 below summarizes the advantages that have been discussed within this chapter.

Table 3 Automated SAST and EJS deployment limitations

Limitations
GitHub Dependency Scanner only scans the repository when triggered by push events to the default branch
GitHub Secret Scanner lacks the ability to scan the repository instantly on pull request events

7 Summary

This research study aimed to explore the utilization of GitHub's built-in security features and GitHub Actions for automating Static Application Security Testing (SAST) processes and deployment workflows for an EJS web application. The research questions addressed in this study were:

- How can GitHub's built-in security features and GitHub Actions be utilized to automate SAST processes for an EJS web application, including code linting, secret scanning, dependency scanning, and code scanning?
- How can GitHub Actions be utilized to automate the deployment of an EJS web application to GitHub Pages, including generating static HTML files from EJS?
- What are the advantages and challenges of using GitHub's built-in security features and GitHub Actions to automate SAST and deployment processes for an EJS web application?

To answer these research questions, an analysis and exploration of GitHub's security features and Actions was conducted, focusing on its utilization in automating SAST processes and EJS web application deployment. The key findings and insights derived from the study include Utilizing GitHub's Built-in Security Features and GitHub Superlinter Action for SAST Automation, Automating EJS Web Application Deployment using GitHub Actions, and Advantages and Challenges of Automation using GitHub's Security Features and Actions.

GitHub provides several built-in security features, including secret scanning, dependency scanning, and code scanning, and in addition GitHub Superlinter can be utilized to extend the built-in security features. These features can be integrated into GitHub Actions workflows, allowing for automated scanning and analysis of code for potential security vulnerabilities. By configuring appropriate workflows and incorporating these security features, developers can ensure that their EJS web applications undergo thorough security checks during the development process.

GitHub Actions can be leveraged to automate the deployment of EJS web applications to GitHub Pages. Workflows can be created to build the web application, generate static HTML files from EJS templates, and publish them to GitHub Pages. This automation streamlines the deployment

process and ensures that the latest version of the application is readily accessible on GitHub Pages.

The utilization of GitHub's security features and Actions for automating SAST and deployment processes offers several advantages. It improves development efficiency, enhances security by detecting potential vulnerabilities early in the development cycle, and simplifies the deployment process. By automating these processes, developers save time, reduce manual effort, and increase productivity. However, there are also challenges to consider. The configuration and setup of GitHub Actions workflows may require a learning curve, and maintaining and updating the workflows over time can be complex. Integration with external services or customizing the workflows for specific project requirements may pose additional challenges. It is also crucial to consider security best practices, such as securely storing secrets and managing access controls, to protect sensitive information within the workflows.

In conclusion, this thesis has demonstrated the effectiveness and benefits of utilizing GitHub's built-in security features and Actions for automating SAST processes and EJS web application deployment. By incorporating these features into workflows, developers can enhance security, improve development efficiency, and simplify deployment processes. However, it is important to be mindful of the challenges associated with workflow configuration, maintenance, integration, and security considerations. The findings of this study contribute to the growing understanding of automation practices using GitHub's tools and highlight the potential for future advancements in secure development and deployment workflows.

References

Codefresh. (2023). *Software Deployment in 2023: Checklist, Strategies & Tips*. Codefresh. https://codefresh.io/learn/software-deployment/

EJS. (2023). *EJS -- Embedded JavaScript templates*. https://ejs.co/

GitHub. (2023a). *About code scanning*. GitHub Docs. https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning

GitHub. (2023b). *About GitHub Pages*. GitHub Docs. https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages

GitHub. (2023c). *About Projects*. GitHub Docs. https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects

GitHub. (2023d). *About secret scanning*. GitHub Docs. https://ghdocs-prod.azurewebsites.net/en/code-security/secret-scanning/about-secret-scanning

GitHub. (2023e). *Super-Linter*. https://github.com/github/super-linter

GitHub. (2023f). *Troubleshooting the detection of vulnerable dependencies*. GitHub Docs. https://docs.github.com/en/code-security/dependabot/working-with-dependabot/troubleshooting-the-detection-of-vulnerable-dependencies

GitHub. (2023g). *Understanding GitHub Actions*. GitHub Docs. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Gitleaks LLC. (2023, May 7). *Gitleaks*. https://github.com/gitleaks/gitleaks

Horst, B. (2023, January 6). *What is GitHub? A Simple 2023 Beginner's Guide*. https://careerfoundry.com/en/blog/web-development/what-is-github/

Kanbanize. (2023). *What Is Kanban? Explained in 10 Minutes | Kanbanize*. Kanban Software for Agile Project Management. https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

Mullans, A. (2020, June 1). Keep all your packages up to date with Dependabot. *The GitHub Blog*. https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/

Nightfall. (2023). *The Essential Guide to Secrets Scanning | Nightfall AI*. https://www.nightfall.ai/blog/essential-guide-secrets-scanning

Ogut, C. (2021, July 4). What is Linting? How does a linter work? *Medium*. https://cogut.medium.com/what-is-linting-how-does-a-linter-work-49381f28fc60

OWASP. (2023a). *OWASP Dependency-Check | OWASP Foundation*. https://owasp.org/www-project-dependency-check/

OWASP. (2023b). *Source Code Analysis Tools | OWASP Foundation*. https://owasp.org/www-community/Source_Code_Analysis_Tools

Tech Target. (2023, March 17). *What is network vulnerability scanning? | Definition from TechTarget*. https://web.archive.org/web/20230317181925/https://www.techtarget.com/searchsecurity/definition/vulnerability-scanning

Annex 1: Material management plan

The Material Management Plan outlines the procedures for collecting, processing, storing, and disposing of research materials during the thesis process. It ensures the protection of data security and compliance with regulations. The plan also addresses the storage and backup of research materials.

During the development project, the following material management practices have been implemented:

Storage of Research Materials

GitHub Issues and Projects, GitHub Issues and Projects were used to manage and track the progress of the development project. These platforms provided a centralized location to document technical information and tasks related to the project.

Zotero for Reference Management, Zotero application was utilized for storing and managing references and bibliographic information. It allowed for efficient organization, citation, and retrieval of research materials**.**

Retention Period

Research materials stored in GitHub Issues, Projects, and Zotero will be retained for at least one year from the date of thesis approval. This retention period ensures that the research materials are available for reference or verification if needed.

Protection of Data Security

No confidential or personal information is used or collected in the thesis. The research materials solely consist of publicly available code, documentation, and references from open sources. As a result, there is no risk of storing or mishandling sensitive information, and the thesis complies with data protection and privacy regulations.

By following this Material Management Plan, the thesis project has ensured the proper management, storage, and retention of research materials. The utilization of GitHub Issues and

Projects, as well as Zotero, provided efficient organization and tracking of project-related tasks and references. Adhering to data security guidelines ensured the protection of research data throughout the thesis process.