**Pipeline extension scenario 1 default configuration**

**Step 1**

* Add the detect-secrets scanner into the pipeline.
* Include the publish task underneath the detect secrets task to publish the results to a test tab.
* Please use the default YAML in the documentation folder as a starting point.

Graphical user interface, text, application

Description automatically generated

**Step 2**

The figure above demonstrates default scanning parameters.

Source Code Location: by default it will scan the root of the directory, otherwise if you would like to point your scanner to a specific directory modify the yaml to point to the new location e.g., '$(Build.Repository.LocalPath)/desired\_path'

scan files which are not tracked by GitHub: This setting skip all of the GitHub files in the repository

Location of the secrets word-list file: by default, this will not run, but if this field gets populated then an allow list will ignore secrets in the codebase.

setFailuresAsWarnings: This setting can be configured to fail with a warning by checking the box for the value otherwise the task will fail once any findings are detected.

excludeFiles: This feature takes in a list of comma separated file paths which will be ignored when running the scanner.

**Step 3**

Run the scanner in the pipeline.

The quality gate for this scanner fails on secrets > 0 detected in the project, otherwise it is green and good.

**Pipeline extension scenario 2 allow-list configuration**

**Step 1**

Add an allow-list to a location in your repository. For this demonstration, I just added my allow-list to the root of the repository.

Graphical user interface, text, application

Description automatically generated

**Step 2**

Modify existing extension by hitting the “settings” at the top and add in a path to your new allow-list.

Graphical user interface, text, application

Description automatically generated

The task will now be modified with an allow list, just save and run the pipeline.

- task: CSEDetectSecrets@1

  inputs:

    sourceLocation: '$(Build.Repository.LocalPath)'

    usingwordListFile: true

    wordListFile: 'allow\_list.txt'

**Step 3**

Continuously add false positives to the allow-list to continue with CI credential free code!

**Pipeline extension scenario 3 .secrets.baseline configuration**

**Step 1**

Create and add .secrets. Baseline to a location in your repository. To create one, run the scanner with the default settings shown above in “Pipeline extension scenario 1 default configuration” the only thing that will need to be configured is pointing the scanner to where it should scan, if this setting is not set it will run at the repositories root by default. After run is finished, retrieve the .secrets.baseline file and commit it into your repository, or a secure file location.

**Step 2**

Once the .secrets.baseline file has been added to your repository, you must modify the yaml to use the .secrets.baseline file.

Graphical user interface, text, application, email

Description automatically generated

- task: CSEDetectSecrets@1

  inputs:

    sourceLocation: '$(Build.Repository.LocalPath)'

    usingBaselineFile: true

    baselineFile: '.secrets.baseline'

**Step 3**

Once the scanner has been configured to run with a baseline file, you can now triage and mitigate against the secrets by looking through the. secrets.baseline file. The scanner works currently by detecting new secrets added or removed from your baseline file and will alert the user on new changes to the baseline file.

It is then up to the user to re-commit the newest baseline file into the repository to continue the cycle, and to utilize the wordlist file from the scenario “Pipeline extension scenario 2 allow-list configuration.” This way you can mitigate against false positives within the wordlist.

**User Story features**

Text

Description automatically generated

**Step 1**

View the test results in the "Tests" tab from a finished run

Graphical user interface, text, application, email

Description automatically generated

This will lead you to a page that shows you what and where the issues were happening.

Graphical user interface, text, application, email

Description automatically generated

**Step 2**

Create stories on the true positives.

Navigate to "Test Plans" -> "Runs" and double click on the report you are interested in.

Graphical user interface, text, application, email

Description automatically generated

Navigate to "Test Results" highlight an item and click "create bug" to make a backlog item.

Graphical user interface, application

Description automatically generated with medium confidence

Enter in a title for the bug and notice the link at the far right contains the actual issue in the story.

Graphical user interface, text, application

Description automatically generated

Save and close and navigate to the "Boards" to see the backlog item.

Graphical user interface, application

Description automatically generated

**Extra features**

View the test results in the "Job" section at the top of the pipeline, here can you see that two artifacts have been produced from the pipeline.

Text

Description automatically generated

Application

Description automatically generated with low confidence

Download the report.xml to witness the secrets detected in the source code.

Text

Description automatically generated with medium confidence

The secrets.baseline file can be downloaded and used to audit the baseline with the yelp-detect secrets run. NOTE this functionality cannot be done within the pipeline, instead you must download the tool and run it by hand. The detect-secrets audit functionality will return a list of the non-hashed secrets.

Text

Description automatically generated