

Rezilion in GitLab CI

USER GUIDE

Requirements

- 1) Ultimate GitLab license
- 2) You are testing a container in your CI tests.

Specifications

- 1) Supported runtime environments:
 - a) Native
 - b) Java
 - c) Node
 - d) Ruby
 - e) Bash
 - f) Python
 - g) Dotnet
 - h) Perl
- 2) Supported package managers:
 - a) rpm
 - b) dpkg
 - c) Maven
 - d) NPM
 - e) Gem
 - f) pip
- 3) Rezilion only validates code that runs after its **validation_start** command, as shown in the example below.

```

test:
  stage: test
  image: rezilion/rezdemo:cve20208840ondemand

  before_script:
    - ./setup_script # Code that rezilion won't recognize
    - !reference [ .rezilion_validate_start, before_script ]
    - ./setup_script # Code that rezilion will recognize

  script:
    - ./some_test_code # Rezilion will recognize the entire script section here

  artifacts:
    paths:
      - !reference [ .rezilion_artifacts, paths ]

```

- 4) Edge Case 1:
 - a) What: Missing events (False Negative) - Mark loaded components as unloaded.
 - b) When: Loading shared libraries to short processes using **dlopen**.
 - c) Probability: Very rare.
 - d) Risk: Very low.
 - e) Technical Description: Shared libraries that are loaded using **dlopen** into processes that are being executed for less than 0.5 seconds.
- 5) Limitation:
 - a) Trivy limits rezilion running to 50 runs per hour, in order to increase the limitation to 5,000 runs per hour, a GITHUB_TOKEN environment variable needs to be set.

Setup

Initial Setup

- Please notice the use of accurate indentation when writing yaml files - A YAML file uses spaces as indentation, you can use 2 or 4 spaces for indentation, but **no tabs**.

- 1) Commit "rezilion.yml" into your git repository (received by mail).

For example:

```

git add -A rezilion.yml
git commit -m "Added rezilion yml to source control"

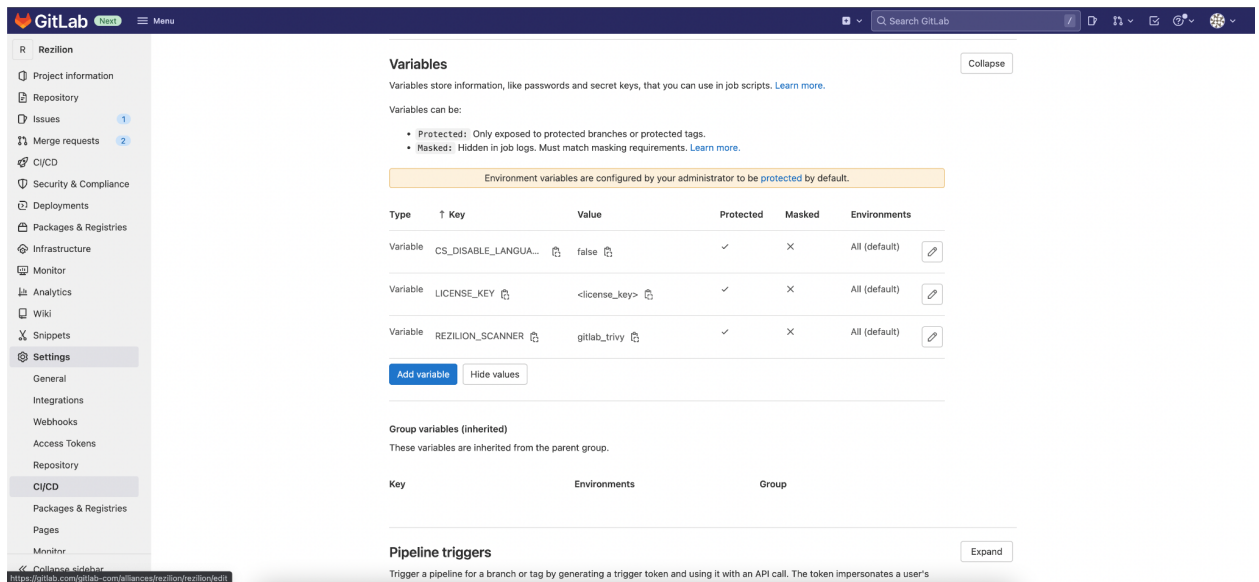
```

- 2) Add into your **include** section in ".gitlab-ci.yml" rezilion's template yml.

For example:

```
include:
  - template: Security/Container-Scanning.gitlab-ci.yml # for gitlab scanner
  - local: rezilion.yml # for rezilion
```

- 3) Declare Rezilion variables, inside GitLab UI, under your project setting go to CI/CD and add the following variables:



The screenshot shows the GitLab interface for a project named 'Rezilion'. The left sidebar contains a menu with options like Project Information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Packages & Registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, Settings, General, Integrations, Webhooks, Access Tokens, Repository, CI/CD, Packages & Registries, Pages, and Monitor. The 'CI/CD' option is selected. The main content area is titled 'Variables' and contains the following information:

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

- Protected: Only exposed to protected branches or protected tags.
- Masked: Hidden in job logs. Must match masking requirements. [Learn more.](#)

Environment variables are configured by your administrator to be [protected](#) by default.

| Type | Key | Value | Protected | Masked | Environments |
|----------|----------------------|---------------|-----------|--------|---------------|
| Variable | CS_DISABLE_LANGUA... | false | ✓ | ✗ | All (default) |
| Variable | LICENSE_KEY | <license_key> | ✓ | ✗ | All (default) |
| Variable | REZILION_SCANNER | gitlab_trivy | ✓ | ✗ | All (default) |

Buttons: [Add variable](#), [Hide values](#)

Group variables (inherited)
These variables are inherited from the parent group.

| Key | Environments | Group |
|-----|--------------|-------|
|-----|--------------|-------|

Pipeline triggers
Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's

```
REZILION_SCANNER: "<scanner_name>"
LICENSE_KEY: "<license_key>"
CS_DISABLE_LANGUAGE_VULNERABILITY_SCAN: "false"
```

Available variables

- **REZILION_SCANNER**

Available options:

- gitlab_trivy (Use GitLab's built in [Trivy scanner](#), which already have container scanning enabled for the pipeline)

- **LICENSE_KEY**

Rezilion license key

- **DOCKER_IMAGE**

An optional variable to specify the image to be scanned. If not specified, defaults to "\$CI_APPLICATION_REPOSITORY:\$CI_APPLICATION_TAG"

- **DOCKER_USER**

An optional variable to specify the username of the registry the image contains on. If not specified, defaults to "\$CI_REGISTRY_USER"

- **DOCKER_PASSWORD**

An optional variable to specify the password of the registry the image contains on. If not specified, defaults to "\$CI_REGISTRY_PASSWORD"

- **CS_DISABLE_LANGUAGE_VULNERABILITY_SCAN**

- An optional flag, that allows the gitlab trivy scanner to show vulnerabilities for non OS package managers (such as NPM, GEM, PIP)

Setup per Test

- 1) Start Rezilion's validation process: Add the following line to your "before_script" section:

```
- !reference [ .rezilion_start, before_script ]
```

For example:

```
before_script:
  - !reference [ .rezilion_start, before_script ]
```

- 2) Add the Rezilion artifact path to your test artifacts.

For example:

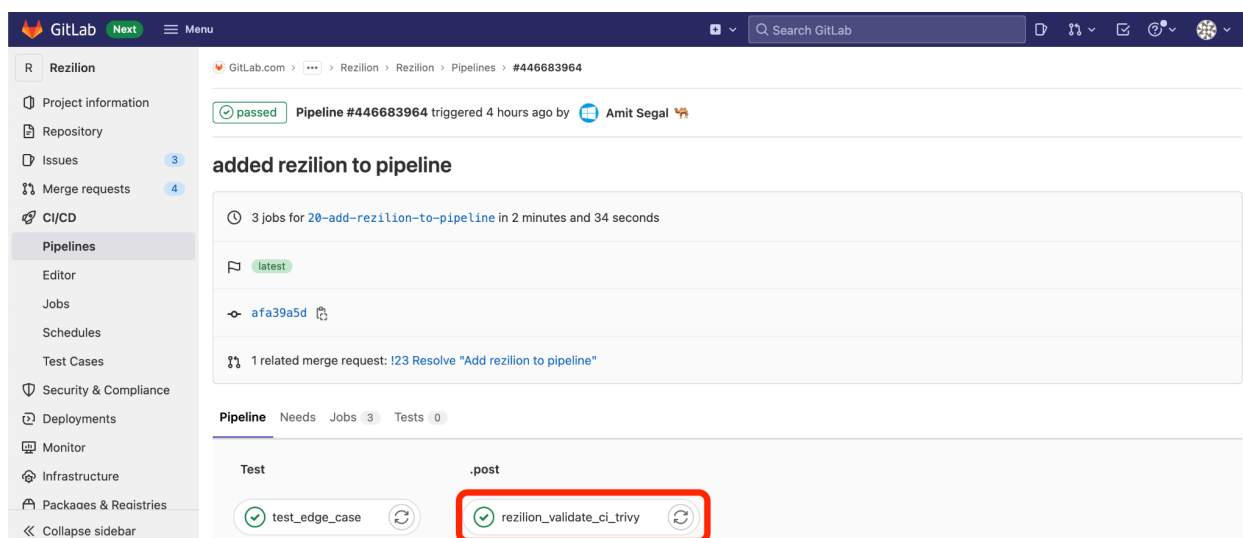
```
artifacts:
  paths:
    - !reference [ .rezilion_artifacts, paths ]
```

Run the Pipeline

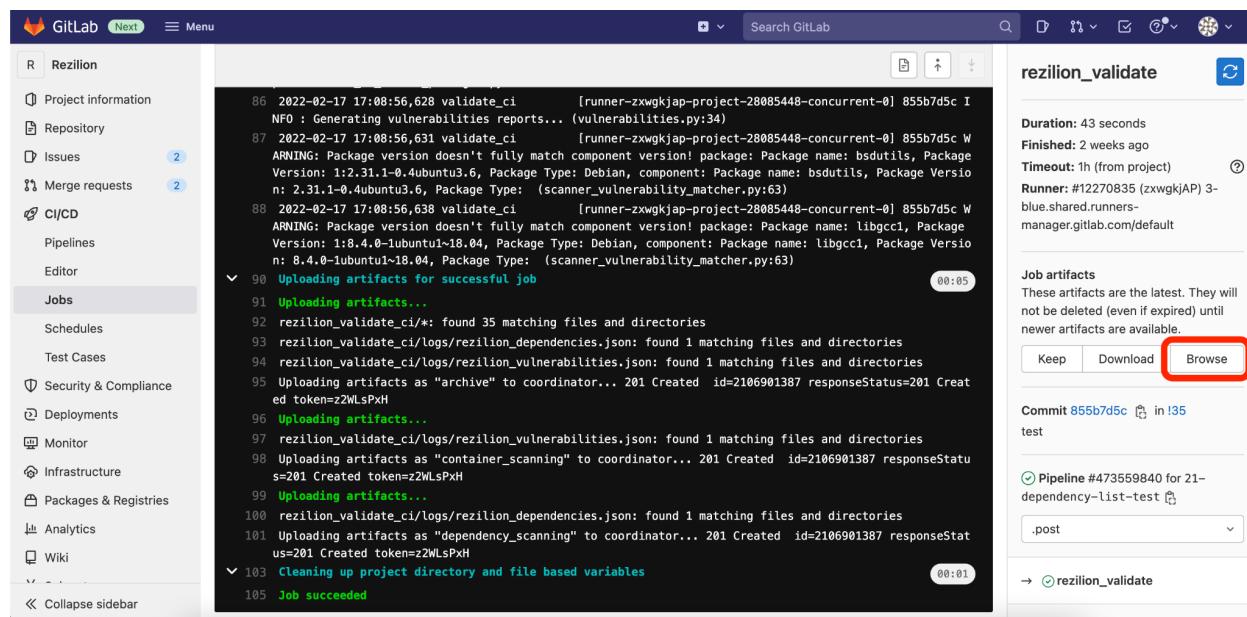
- Rezilion's validation process is executed automatically for every test Rezilion is enabled on (as configured in section B).
- A new job is added to the pipeline, which contains the results of Rezilion.
- From our testing, scanning a very large image (~14K files) should only take about 10 seconds. Not including the installation of our package which depends on the speed of your network (size - 23mb).

Explore the Result by Artifact

The results can be viewed by going into the “**rezilion_validate**” job which runs at the end of the pipeline:



And then clicking “Browse” in the artifacts section



In the folder named “rezilion_validate_ci”, you’ll be able to find these 3 html files:

The top screenshot shows the GitLab interface for a job. The left sidebar contains a menu with items like Project information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, and Snippets. The main content area shows the job status as 'passed' and the artifacts section. A table lists the artifacts:

| Name | Size |
|----------------------|------|
| rezilion_validate_ci | |

The bottom screenshot shows the same interface but with the 'Artifacts / rezilion_validate_ci' view selected. It displays a list of files within the artifact folder:

| Name | Size |
|--------------------------------|---------|
| .. | |
| logs | |
| sbom.html | 28.9 KB |
| validated_vulnerabilities.html | 699 KB |
| vulnerable_component.html | 321 KB |

SBOM (Software Bill Of Materials)

Each component Rezilion found on the scanning will be there.

Potentially loaded files happen when a nested jar/war/etc runs and finishes execution very fast (less than a second). In this scenario, the main jar/war/etc will be marked as exploitable, but the inner jars will be marked as potentially exploitable.



| <div>Search</div> | | | | | |
|--|-----------------|--------------|---|--------------------------|--------|
| Package Name | Package Version | Package Type | Loaded Files | Potentially Loaded Files | State |
| libjackson2-databind-java | 2.9.8-1~18.04 | Debian | /usr/share/java/jackson-databind.jar | | Loaded |
| com.fasterxml.jackson.core:jackson-databind | 2.9.8 | Maven | /usr/share/java/jackson-databind.jar | | Loaded |
| libjackson2-core-java | 2.9.8-3~18.04 | Debian | /usr/share/java/jackson-core.jar | | Loaded |
| com.fasterxml.jackson.core:jackson-core | 2.9.8 | Maven | /usr/share/java/jackson-core.jar | | Loaded |
| libjackson2-annotations-java | 2.9.4-1 | Debian | /usr/share/java/jackson-annotations.jar | | Loaded |
| com.fasterxml.jackson.core:jackson-annotations | 2.9.4 | Maven | /usr/share/java/jackson-annotations.jar | | Loaded |
| | | | /usr/lib/jvm/java-11-openjdk-amd64/lib/libverify.so, /usr/lib/jvm/java-11-openjdk-amd64/lib/jrt-fs.jar, /usr/lib/jvm/java-11-openjdk- | | |

Validated Vulnerabilities

Each vulnerability the scanner found, with an **Exploitable/Unexploitable** state. When clicking on a vulnerability, the affected components will be shown.

- The table can be sorted by one of those fields: **CVE ID, Severity, Description, State.**
 - By clicking on the CVE ID you will redirect to more details about it in NVD site (NATIONAL VULNERABILITY DATABASE)
- There is an option to search by a given string in the table, using the search textbox.

| CVE ID | Severity | Description | State |
|--------------------------------|----------|--|-------------|
| CVE-2020-27350 | Medium | APT had several integer overflows and underflows while parsing .deb packages, aka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/debfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 versions prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubuntu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.1; | Unloaded |
| CVE-2019-18276 | Low | An issue was discovered in disable_priv_mode in shell.c in GNU Bash through 5.0 patch 11. By default, if Bash is run with its effective UID not equal to its real UID, it will drop privileges by setting its effective UID to its real UID. However, it does so incorrectly. On Linux and other systems that support "saved UID" functionality, the saved UID is not dropped. An attacker with command execution in the shell can use "enable -f" for runtime loading of a new builtin, which can be a shared object that calls setuid() and therefore regains privileges. However, binaries running with an effective UID of 0 are unaffected. | Exploitable |
| CVE-2018-7738 | Low | In util-linux before 2.32-rc1, bash-completion/umount allows local users to gain privileges by embedding shell commands in a mountpoint name, which is mishandled during a umount command (within Bash) by a different user, as demonstrated by logging in as root and entering umount followed by a tab character for autocompletion. | Unloaded |
| CVE-2016-2781 | Low | chroot in GNU coreutils, when used with --userspec, allows local users to escape to the parent session via a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input buffer. | Exploitable |

Vulnerable Components

A list of components found by the vulnerability scanner.

Each row represents a component, with relevant evidence in case it is loaded.

In case when a nested jar/war/etc runs and finishes execution very fast (less than a second). The main jar/war/etc will be marked as exploitable, but the inner jars will be marked as potentially exploitable.

When clicking on a component, a list of its associated vulnerabilities will be displayed.

- The table can be sorted by one of those fields: **CVE ID, Severity, Description, State.**
 - By clicking on the CVE ID you will redirect to more details about it in NVD site (NATIONAL VULNERABILITY DATABASE)
- There is an option to search by a given string in the table, using the search textbox.



Vulnerable Components

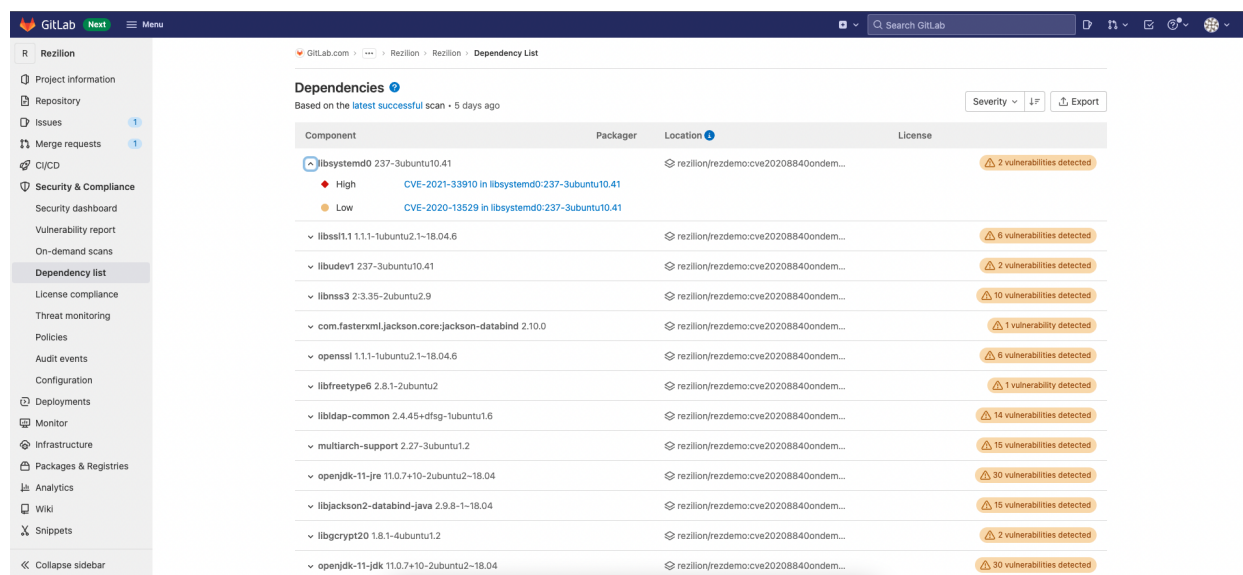
| Package Name | Package Version | Package Type | Highest Severity | State | Evidence |
|--------------|-----------------------|--------------|------------------|-------------|--|
| bash | 4.4.18-2ubuntu1.2 | Debian | Low | Exploitable | Loaded Files: /bin/bash |
| bsdutils | 1:2.31.1-0.4ubuntu3.6 | Debian | Low | Unloaded | |
| coreutils | 8.28-1ubuntu1 | Debian | Low | Exploitable | Loaded Files: /bin/sleep,/usr/bin/timeout |
| curl | 7.58.0-2ubuntu3.9 | Debian | Medium | Unloaded | |
| fdisk | 2.31.1-0.4ubuntu3.6 | Debian | Low | Unloaded | |
| gcc-8-base | 8.4.0-1ubuntu1~18.04 | Debian | Medium | Unloaded | |
| gpgv | 2.2.4-1ubuntu1.2 | Debian | Low | Unloaded | |
| krb5-locales | 1.16-2ubuntu0.1 | Debian | Medium | Unloaded | |

Explore the Results by GitLab UI

The results can also be viewed through the GitLab security dashboard:

Dependency List (SBOM)

The dependency list to review your project's dependencies and key details about those dependencies, including their known vulnerabilities. It is a collection of dependencies in your project, including existing and new findings. The dependency list only shows the results of the last successful pipeline to run on the default branch.



Vulnerability Report

The Vulnerability Report provides information about vulnerabilities from scans of the default branch.

At all levels, the Vulnerability Report contains:

- Totals of vulnerabilities per severity level.
- Filters for common vulnerability attributes.
- Details of each vulnerability, presented in tabular layout.
- The Activity column contains an icon to indicate if the vulnerability is exploitable.

