Rezilion in GitLab CI

Requirements

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

Limitations

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.
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.

**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:

```bash
  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:

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

3) Declare Rezilion variables, inside gitlab.yml, as global variables (at the outer scope):

   ```yaml
   variables:
     REZILION_SCANNER: "<scanner_name>"
     LICENSE_KEY: "<license_key>"
     DOCKER_IMAGE: "<image_name>"
     CS_DISABLE_LANGUAGE_VULNERABILITY_SCAN: "false"
   ```

**Available variables**

- **REZILION_SCANNER**
  
  Available options:
  
  - gitlab_trivy (Use GitLab's built in [Trivy scanner](https://github.com/NcHybrid/Multi-Scanner)), which already have container scanning enabled for the pipeline)

- **LICENSE_KEY**
  
  Rezilion Validate's 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 username 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:
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.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Package Version</th>
<th>Package Type</th>
<th>Loaded Files</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>libjackson2-databind-java</td>
<td>2.9.8-1~18.04</td>
<td>Debian</td>
<td>/usr/share/java/jackson-databind.jar</td>
<td>Loaded</td>
</tr>
<tr>
<td>com.fasterxml.jackson.core.jackson-</td>
<td>2.9.8</td>
<td>Maven</td>
<td>/usr/share/java/jackson-core.jar</td>
<td>Loaded</td>
</tr>
<tr>
<td>com.fasterxml.jackson.core.jackson-core</td>
<td>2.9.8</td>
<td>Maven</td>
<td>/usr/share/java/jackson-core.jar</td>
<td>Loaded</td>
</tr>
<tr>
<td>libjackson2-annotations-java</td>
<td>2.9.4-1</td>
<td>Debian</td>
<td>/usr/share/java/jackson-annotations.jar</td>
<td>Loaded</td>
</tr>
<tr>
<td>com.fasterxml.jackson.core.jackson-</td>
<td>2.9.4</td>
<td>Maven</td>
<td>/usr/share/java/jackson-annotations.jar</td>
<td>Loaded</td>
</tr>
</tbody>
</table>

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.
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 the NVD site (NATIONAL VULNERABILITY DATABASE)
- There is an option to search by a given string in the table, using the search textbox.

<table>
<thead>
<tr>
<th>CVE ID</th>
<th>Severity</th>
<th>Description</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2020-27350</td>
<td>Medium</td>
<td>APT had several integer overflows and underflows while parsing .deb packages, aka GHSL-2020-168 GHSL-2020-169, in files apt-plugin/contrib/dprintf.c, and apt-plugin/contrib/dprintf.c. This issue affects: apt 1.3.32ubuntu0 versions prior to 1.2.32ubuntu0.2, 1.6.10ubuntu0.2, 2.0.5ubuntu0 versions prior to 2.0.5ubuntu0.2, 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.1;</td>
<td>Unloaded</td>
</tr>
<tr>
<td>CVE-2019-18279</td>
<td>Low</td>
<td>An issue was discovered in disable_null_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 &quot;saved UID&quot; functionality, the saved UID is not dropped. An attacker with command execution in the shell can use &quot;enable -f&quot; 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.</td>
<td>Exploitable</td>
</tr>
<tr>
<td>CVE-2018-7738</td>
<td>Low</td>
<td>In util-linux before 2.33-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 amount followed by a tab character for autocompletion.</td>
<td>Unloaded</td>
</tr>
<tr>
<td>CVE-2016-2781</td>
<td>Low</td>
<td>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.</td>
<td>Exploitable</td>
</tr>
</tbody>
</table>
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.