IT 351 Cloud Security and DevSecOps

Lab 8

Fall 2021 – Professor Josh Edgar

Welcome to Lab 8. This week, we will add a fairly simple job to the beginning of our CI/CD pipelines that will scan Docker images, ensuring that they have no vulnerabilities before we begin using them.

You may recall that, in most of our GitLab CI/CD jobs, we simply specify an "image:" from the Docker Hub that uses a container maintained by a third-party, rather than building a Docker image ourselves from a Dockerfile. Normally, these images are maintained by trusted vendors, such as HashiCorp for the Terraform image, but in other cases, we've used some unofficial and unsupported images to make things work, like the "Mullnerz" image for Ansible. In either case, it's a best practice to scan these third-party images for vulnerabilities before you begin using them, ensuring that you are working with a clean environment before you introduce your code.

Before you begin:

- Your GitLab Runner instance should be started, if you previously stopped it.

What you'll achieve:

- Re-register your GitLab Runner, allowing it to expose the Docker socket to our jobs
- Add a Trivy job to your pipeline, scanning images for vulnerabilities, and reviewing/preserving the results

What you'll turn in with this lab:

- The ZIP file of JSON artifacts that you will download from the Trivy job after Task 4.

Task 1: Re-Register GitLab Runner / Expose Docker Socket

When we specify an image from the Docker Hub to use in our GitLab CI/CD jobs, our GitLab Runners create a Docker container from that image. However, that image itself does not have Docker installed on it, and the image does not even know it is running in a Docker container (strong *Matrix* vibes).

Trivy needs to use Docker to deploy a container from the image we wish to scan, which leaves us two options: Either install and start Docker in the "before_script" of our Trivy job (time-consuming and lots of code), or simply re-register our GitLab Runner with a command that allows our containers to use the Docker socket on the Runner host by "binding" the socket to each container that is deployed. This creates some security risk, since any malware that we download into a container would then have elevated privileges to our Runner, but, as usual, for the purposes of an in-class Lab, we're cutting some corners for the sake of expediency.

1. Log into GitLab in your browser. Navigate to Settings > CI/CD > Runners and remove your current Runner. This only de-registers the runner from GitLab.com – it does not stop or terminate your EC2 instance, and neither should you.
2. Start an SSH session with your GitLab Runner EC2 instance. You may need to log in to the AWS Management Console to fetch its IP address, if you don't have it notated somewhere. Don't forget to include the path to your private key file when you launch the SSH session.
3. Download the GitLab Registration Command text file from this assignment's page on Canvas. This is a long command, so it will be easiest to copy it into your terminal from a plain text editor. Before you copy and paste it over, make sure to change the registration token to reflect the one displayed on your GitLab Settings page, and update the name of the Runner to what your Runner was previously named.
4. Once you get a "Runner registered successfully" messaged, go back to the GitLab Settings page, and refresh it. Expand the Runners section again, and verify that the new Runner has been added. You may need to refresh a few more times until you see it officially come online with a green dot.

Task 2: Add Trivy Stage and Job to GitLab CI/CD File

Now, for the part that we should be fairly used to: Adding the job code to our .gitlab-ci.yml file.

5. In VS Code, open your ".gitlab-ci.yml" file.
6. At the top, add a stage to the *top* of the stages list called "Image Scanning" (with quotes).
7. Download the Trivy GitLab Job sample code from this assignment's page on Canvas. Add it into your GitLab CI/CD file above your "tf-val" job; we want this job to run before any of the others.
8. Save, commit and push your changes. Observe your pipeline's execution and review the vulnerabilities identified by Trivy.

Task 3: Scan All Images

The sample code that you downloaded from Canvas only included a script for having Trivy scan the HashiCorp Terraform image we use for our IaC jobs. We should expand this to also include scans of the other images that we use in our pipelines.

9. Duplicate the "trivy" command under the "script:" section of the Trivy job code, modifying each line so that we have Trivy scan every Docker image we use in our pipeline. Save, commit and push these changes, and observe the new report.

Task 4: Output Results as JSON Artifacts

None of our other jobs necessarily need the output from Trivy – in fact, it's probably preferable that we leave the output human-readable as it is currently, so that we could address any container image vulnerabilities before our pipeline continued to run – but this is a good exercise in learning how to preserve multiple artifacts, since Trivy isn't able to append additional scan results to a previous output file.

10. In the "trivy" command, between the word "image" and the name of the image to scan, add the following parameters:
    `-f json -o results.json`
11. Update each iteration of "results.json" to have a different filename for each image being scanned, such as `results-terraform.json`, `results-ansible.json`, etc.
12. Add an "artifacts:" section to the job code to ensure that these files are preserved. Use "`when: always`" to ensure they are preserved even if the job were to fail, if any critical vulnerabilities were identified.
13. Save, commit and push these changes.

Observe the job's execution again, this time, noticing that it does not fill the terminal preview with reporting. The reports are now being stored in the JSON artifacts, which you can download in a ZIP file as previously demonstrated. Review the contents of your JSON reports to see how they compared to the human-readable reports you saw in Task 3, and then upload the ZIP archive to Canvas for your Lab credit.

Extra Credit Opportunity: Job Failure on Critical Findings

I will offer another 1-point extra credit opportunity for those who wish to pursue it. As with Chef InSpec, Trivy's documentation is somewhat scarce, so this is another exercise in your ability to research and test your code independently – I cannot help you with achieving this extra credit.

To receive the extra credit, update your Trivy commands in the job code so that the job will fail if any critical-level findings are identified during the image scanning process. (Currently, even with 9 critical findings in the mullnerz/ansible-playbook image, the job still succeeds.)