Dockerfile static analyzer

Vilho Aaltonen 2023

Introduction

Docker is a platform that makes it possible to create easily portable software. Its basic building blocks are *containers*. Those are just executable packages that contains an application's code and its dependencies. Containers are created from *Docker images*. Images are immutable templates where everything necessary is defined. This means that container is a running instance of the image that defines it. Docker image is created based on instruction from *Dockerfile*. Those instructions specifies the application code, necessary dependencies and also runtime environment of the program.

When it comes to security Docker is not a silver bullet. Although when used correctly it can provide some very secure features. Like isolating containers, signing and verifying base images and isolating networks[1]. Like in any software, the programmer is very common source of security issues. One way to create those issues is to make mistakes when writing Dockerfile. This works implements a simple static analyser that could reduce those issues before Dockerfile is build to be Docker image.

Program

Analyser itself is implemented with Python 3.9. It uses third party Dockerfile parser that is chosen based on its simplicity to read and parse Dockerfile to the easy-to-use form. Program is implement with Ubuntu 18.04. Also docker client is required to be installed to the host machine.

Program contains three modules *analyzer, docker_parser* and *utils.* Then there is *main* that calls function from analyzer to run analysis. Figure 1 shows the structure of program

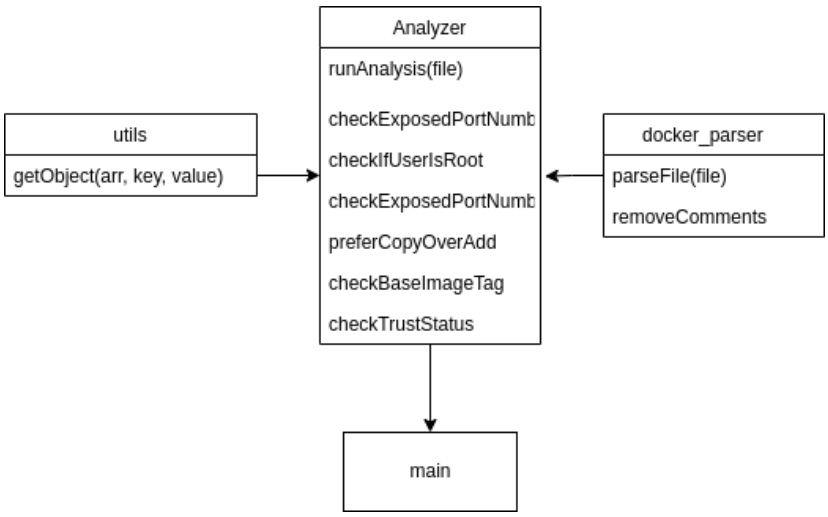

Figure 1: Diagram for program structure

1 Networks in this sense means the connection between two or more containers. This work doesn't address any issues based on networks

There are two parts where secure programming should be considered. First of all since Dockerfile is a file there might be problems when reading it. This is handled with try-except blocks when third party *dockerfile_parse* is reading the file.

Also the program uses *subprocess* library to make command */usr/bin/docker trust inspect <image>*. The image string is validated with regular expression to match format defined in https://docs.docker.com/engine/reference/commandline/tag/. This prevents that malicious commands cannot be executed within this subprocess.

Currently implemented issues

Currently the program checks following issues:

Trust status: Docker contains so called Docker Content Trust (DCT) system. That makes it possible to digitally sign the content fetched from remote Docker registery. This makes it so that one can trust the base image that is loaded. (https://docs.docker.com/engine/security/trust/)

Base image tag: This is to make sure that programmer explicitly tells what version of base image is used. Generally *latest* should be preferred.

Prefer copy over add: Both of these commands copies the content of source machine directory to the containers directory. The difference is that *add* can handle urls and decompress archives. This could possibly lead to unexpected behaviour of the running container if programmer is not careful. In a worst case scenario some malicious party could try to add archive to the directory that could cause serious security issues.

Check that user is not root: Usually docker containers executes programs with root privileges. This may lead to serious security issues if someone gets in to the container or if there is carelessly written program that tries to for example remove something that shouldn't be possible to do.

Check exposed port number: This simply checks that exposed port is not in the range of 0-1024 since those are generally reserved for operating system processes.

Running the program and tests

Program can be executed in root directory by running a command `python main.py <path-to-Dockerfile>`. If there are any issues, those are printed to the standard output and program is exited with status code 1. If everything is ok nothing gets printed and status code is 0. This was chosen so that the analyzer could be use inside deployment pipeline without any unnecessary outputs if everything is ok.

In folder *test* is simple unit tests for parser and docker_analyzer tests are implemented with pytest (https://docs.pytest.org/en/7.3.x/) . Tests can be run inside folder test with command `python -m pytest`.

Conclusion

This was a short description about static analyzer for Dockerfiles. There are few improvements that could be done in the future. For example more issues could be checked. Also issues could be categorized in some way. Other thing that could be implemented is simple way to disable some

rules. For example in some cases programmer may need to use ADD over COPY and it would be annoying if analyzer would mark it as an error all the time.

References:
https://www.docker.com/
https://www.ibm.com/topics/docker
https://docs.docker.com/engine/security/trust/
https://docs.docker.com/network/
https://github.com/containerbuildsystem/dockerfile-parse
https://docs.docker.com/engine/reference/commandline/tag/
https://docs.pytest.org/en/7.3.x/