CSCE689 Project 2 - Container Safety Determination Report

Qingqing Li 424001753, Shijin Tang 523005303, Mian Qin 725006574

Time efforts

We spent approximately 50 hours in total on this project.

Project specification

Our project aims to provide a cloud service to verify the security for docker development, preventing malicious code compromising the infrastructures. The safety scan contains two parts, first, we will check the docker image uploaded by the developer in the private registry. Second, we will scan the running docker containers in the production development. Thus to make sure the security of the docker development and deployment.

Background and motivation

Docker greatly simplifies the deployment and management of application. For example, to deploy an application consisting of a set of services one pulls corresponding docker images from a registry and wires them together.

However, there are plenty of security vulnerability across the development stacks. During the development, the developers may pull images which contains malicious code or the developers themselves maybe compromised to intentionally inject malicious code to the application. Besides, during the deployment of docker containers in the production environment. The docker image may get attacked, for example due to the security vulnerability of the production environment. In this project, we propose a solution to address these problems.

Design

In this project, we propose to build a prototype software to demonstrate our approach. The software mainly contains three parts:

1. A background crawler that pull the docker images that are pushed to the private registry.
2. A docker image scanner to determine whether the image is malicious or not. To determine if the given images are malicious or not, we intend to compare the suspicious images with the Reference Data Set (RDS) collected by National Software Reference Library (NSRL). The RDS incorporates application hash values in the hashset which may be considered malicious, i.e. steganography tools and hacking scripts.
    - A local database can be used to cache those scanned files and thus to reduce the cost of scanning.
3. A background scanner to scan the running docker containers in the production environment. We intend to implement scanning scheduling, while use 3rd party tools for container scan.

The basic software we expect to implement contains the above docker image scan and docker container scan. Further, we may focus on the performance optimization for large scale system or we may consider more security vulnerabilities for docker development and deployment and implement approach to tackle them.

Implementation

This is a service associated with a docker registry that can inspect pushed docker containers and figure out whether they are safe.

1. ClamAV is used to detect virus files.
2. sdhash values are calculated for each file for caching purpose.
3. MongoDB is used to store sdhashes, allow faster examination of previously checked files.
4. A registry application is running as a docker container. The virus checking happens every time we push an image into this registry.
5. Once we have a newly pushed image, the program will download and untar it into a local directory then do virus checking on all files there.
6. If an image is detected as suspicious, the program will delete it in the registry.
7. The results can be shown in browser with the help of flask server.
8. In the client side or production environment. A background service that monitoring the running containers.
9. If there's malware found in the container, it will delete the related containers as well as the docker images and also printing the log into the console.

Prerequisite

fuzzy_hash

Installation

```
$ sudo python -m pip install fuzzyhashlib
```

MongoDB

Installation

run following command:

1. `$ sudo apt-get install -y mongodb-org`
2. `$ cd ~; mkdir data`
3. `$ echo 'mongod --bind_ip=$IP --dbpath=data --nojournal' > mongod`
4. `$ chmod a+x mongod`

Run service

Simply by `~/mongod`

Mongo shell

- Run local Mongo shell by `mongo`
- Connect a remote MongoDB: `mongo --host 34.233.78.56`

Install pymongo

Run `sudo python -m pip install pymongo`

Create db and collection

In mongo shell, create DB needed for this application.

- Create db: `use sdhash_db`
- Create collection: `db.createCollection("good_files")`

Docker

Installation

Test on a Ubuntu 16.04 virtual machine.

Reference: https://docs.docker.com/install/linux/docker-ce/ubuntu/

Also install `docker-compose` according to https://docs.docker.com/compose/install/#install-compose

Enable HTTP request

Add configuration file to the server host to connect to insecure private registry.

```
$ sudo vim /etc/docker/daemon.json
```

Add below to the configuration file:

```
{
"insecure-registries": ["REGISTRY_IP:REGISTRY_PORT"]
}
```

Restart docker daemon:

```
$ sudo service docker stop
$ dockerd &
```

ClamAV

Installation

```
$ sudo apt-get install clamav clamav-daemon
```

Run

1. Update clamAV DB: `sudo freshclam`
2. Start clamAV: `sudo service clamav-daemon start`

pyclamd

```
$ sudo pip install pyclamd
```

Instructions on run the code

Run the service

Start helper service

1. Start MongoDB: `$ ~/mongod`
2. Update clamAV DB: `$ sudo freshclam`
3. Start clamAV: `$ sudo service clamav-daemon start`

Specify IP and PORT

In `endpoint/constants.py`, specify following fields with respect to your server. Also change `notification:endpoint:url` field in `registry/config.yml` to be `http://REGISTRY_IP:WEB_PORT/check_image`.

- `REGISTRY_IP`
- `REGISTRY_PORT`
- `WEB_PORT`

Launch Registry

1. `$ cd registry/`
2. `$ docker-compose up`

Launch Endpoint

1. `$ cd endpoint`
2. `$ python endpoint.py`

On client side

Run background_scanner

```
$ cd client
$ python background_scanner.py
```

Docker pull image from private registry

```
$ docker pull REGISTRY_IP:REGISTRY_PORT/image_name
```

Docker build image

1. See this link
2. Use `COPY` to copy files in context directory to the new image.
3. See the reference link for `RUN` and `CMD` command.
4. Build with following command.

```
$ docker build -t your_container_name:version_tag context_dir_path
```

Docker push image

Enable push to insecure registry

Add configuration file to the server host to connect to insecure private registry.

```
$ sudo vim /etc/docker/daemon.json
```

Add below to the configuration file:

```
{
"insecure-registries": ["REGISTRY_IP:REGISTRY_PORT"]
}
```

Restart docker daemon:

```
$ sudo service docker stop
$ dockerd &
```

Here is a reference.

Push to registry

```
$ docker tag image_name:tag REGISTRY_IP:REGISTRY_PORT/container_name
$ docker push REGISTRY_IP:REGISTRY_PORT/image_name
```

Docker save image

```
$ docker save -o a.tar container_name:version_tag
```

Docker run container

```
$ docker run image_name:tag
```

Docker list all containers

```
$ docker ps -a
```

Docker list all images

```
$ docker image ls
```

View push results

Push results can be viewed through browser at `http://REGISTRY_IP:WEB_PORT/results`. Successful pushes are marked with OK. Failed pushes lists those suspicious files.

Example

This service is deployed along with a public Docker registry on a virtual server on Digital Ocean. Anyone can configure their client side and push&pull images to/from this registry. Our program provides a guard for this registry to identify and remove malicious images.

Registry side

The endpoint is able to recognize the name, url of the images once a new image is pushed. Afterwards, the images are pulled and saved as compressed files as shown in Fig. 1. Further, the safety check will examine the pushed images.

The Fig. 2 above shows the logs coming from the safety check. It also prints out the number of bad files and the name of the suspicious files in the image. Once a suspicious image is detected, the endpoint implements deletion which removes the suspicious images from private registry.

Fig. 3 show the printed JSON in the notification. It lists the two operations conducted on endpoint after a new image is pushed into the registry, one is pulling the image for safety check, another one following is the deletion once a malware is detected.

The results can be shown in a web page as Fig. 4.

Client side (production environment)

The client side or production environment side service provide a background scanner to check the running container regularly and kill/remove the malicious containers/images. The motivation here is when deploy the containers in the production environments, after pulling the image from our private registry, the image may be modified (build as new image in the client side) with malicious files and runs. The example here demonstrates how this is prevented by our background scanner.

First, on the production environment pull the non-malicious image **hello** and re-build with some malware file in the new image **mian:v1** as Fig. 5 demonstrates.

Then, we run two containers with the malicious image, see the images and running containers in Fig. 6.

Meanwhile, we have already started our background scanner and here is the output of our backgrounder scanner. It will capture the malicious containers and delete the malicious container as well as the malicious images as Fig. 7 shows.

Finally, we list the containers and the images in the machine and we can see that the malicious containers and images are deleted. Only the safe containers are still exist. See Fig. 8.

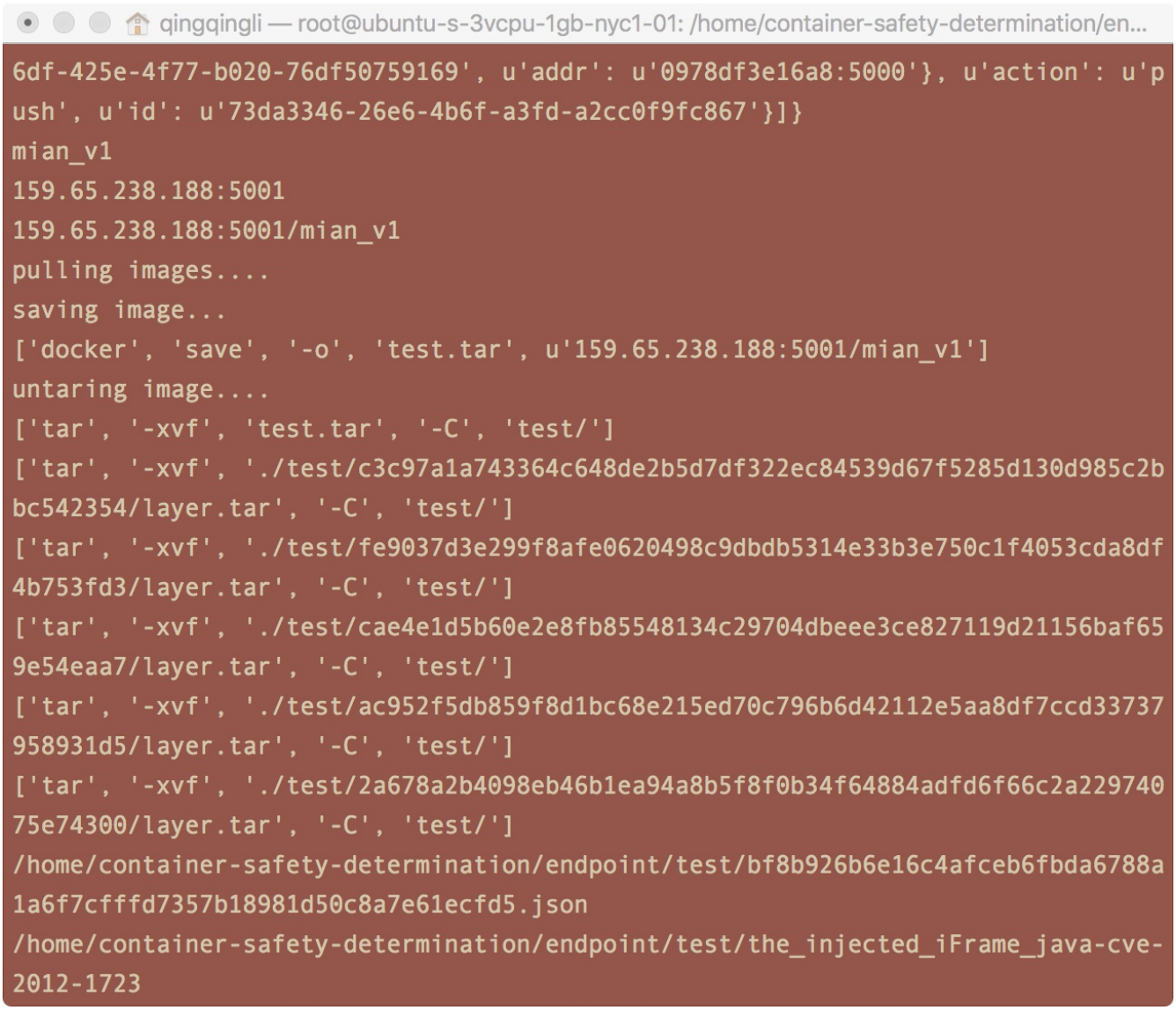

Figure 1: pulling images

Figure 2: safety check

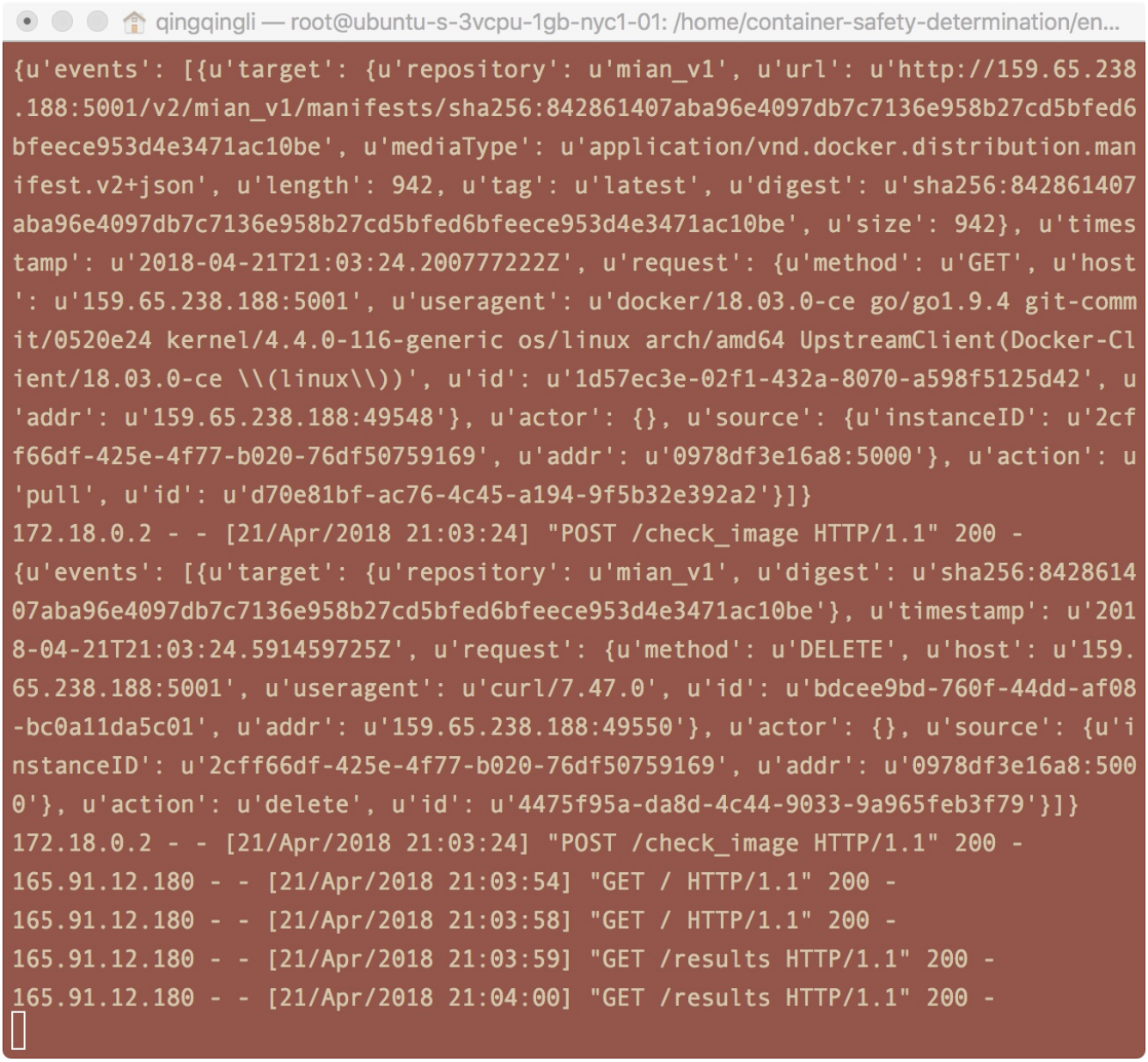

Figure 3: json examples

Registry push results:

2018-04-21 19:59:55.242814; Image: 159.65.238.188:5001/qql; OK

2018-04-21 20:03:46.318220; Image: 159.65.238.188:5001/mian_v1; OK

2018-04-21 20:03:46.513988; Image: 159.65.238.188:5001/mian_v1; OK

2018-04-21 20:03:46.742169; Image: 159.65.238.188:5001/mian_v1; OK

2018-04-21 20:03:47.236065; Image: 159.65.238.188:5001/mian_v1; OK

2018-04-21 20:03:48.043879; Image: 159.65.238.188:5001/mian_v1; Suspicious with files:

- /home/container-safety-determination/endpoint/test/linux-chapros_ E022DE72CCE8129BD5AC8A0675996318
- /home/container-safety-determination/endpoint/test/the_zeus_binary_chapros
- /home/container-safety-determination/endpoint/test/wirelesskeyview.exe

2018-04-21 21:01:43.682463; Image: 159.65.238.188:5001/mian_v1; Suspicious with files:

- /home/container-safety-determination/endpoint/test/linux-chapros_ E022DE72CCE8129BD5AC8A0675996318
- /home/container-safety-determination/endpoint/test/the_zeus_binary_chapros
- /home/container-safety-determination/endpoint/test/wirelesskeyview.exe

Figure 4: web result

```
celery@celery-VirtualBox:~/mydockerbuild$ docker build -t mian:v1 ./
Sending build context to Docker daemon  499.7kB
Step 1/4 : FROM 159.65.238.188:5001/hello
 ---> e38bc07ac18e
Step 2/4 : COPY ./a.out /
 ---> 68aa34c28c7a
Step 3/4 : COPY ./malware-examples/* /
 ---> 1abe1850a1b7
Step 4/4 : CMD [ "/hello" ]
 ---> Running in 7c056e7f5143
Removing intermediate container 7c056e7f5143
 ---> 9e1a63d557fe
Successfully built 9e1a63d557fe
Successfully tagged mian:v1
celery@celery-VirtualBox:~/mydockerbuild$ docker run mian:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Figure 5: pull and build malicious image

```
celery@celery-VirtualBox:~/mydockerbuild$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
096b839f9ace        mian:v1             "/hello"            26 seconds ago
Exited (0) 25 seconds ago                       modest_brown
c1218567e2b1        mian:v1             "/hello"            28 seconds ago
Exited (0) 27 seconds ago                       vibrant_davinci
fd94d61bb9f3        hello-world         "/hello"            About an hour ago
Exited (0) About an hour ago                    eager_franklin
celery@celery-VirtualBox:~/mydockerbuild$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED
        SIZE
mian                        v1                  9e1a63d557fe        41 seconds a
go      488kB
159.65.238.188:5001/hello   latest              e38bc07ac18e        10 days ago
        1.85kB
hello-world                 latest              e38bc07ac18e        10 days ago
        1.85kB
celery@celery-VirtualBox:~/mydockerbuild$
```

Figure 6: list container and image

```
^Ckeyboardinterrupt
celery@celery-VirtualBox:~/mydockerbuild$ python background_scanner.py
[2018-04-21 16:51:28.967195] do scanning
[2018-04-21 16:51:29.179456] no malicious comtainer/image found
[2018-04-21 16:52:29.223631] do scanning
[2018-04-21 16:52:29.423492] no malicious comtainer/image found
[2018-04-21 16:53:29.449824] do scanning
[2018-04-21 16:53:29.663044] no malicious comtainer/image found
[2018-04-21 16:54:29.667304] do scanning
[2018-04-21 16:54:30.028161] found malware /home/celery/mydockerbuild/temp/
11d7f3e7a768116a6bd085fa6949bb607cefea513db140fb65476522a59/layer.tar
[2018-04-21 16:54:30.177884] delete container: 096b839f9ace
[2018-04-21 16:54:30.346692] delete container: c1218567e2b1
[2018-04-21 16:54:30.444262] remove image: mian:v1
[2018-04-21 16:55:30.507885] do scanning
[2018-04-21 16:55:30.722671] no malicious comtainer/image found
```

Figure 7: background scanner results

```
celery@celery-VirtualBox:~/mydockerbuild$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
fd94d61bb9f3        hello-world         "/hello"            About an hour ago
Exited (0) About an hour ago                    eager_franklin
celery@celery-VirtualBox:~/mydockerbuild$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED
        SIZE
159.65.238.188:5001/hello   latest              e38bc07ac18e        10 days ago
        1.85kB
hello-world                 latest              e38bc07ac18e        10 days ago
        1.85kB
celery@celery-VirtualBox:~/mydockerbuild$
```

Figure 8: list containers and image again

Conclusion

In this project, we implement a prototype software for security purpose of the workflow of the docker container. In the registry side, it will check all the images pushed by the developer and remove the docker images with security vulnerability and notify the developer with the malicious files. In the production environment side where docker containers are actually running, it will constantly scan the running containers and delete the containers and related images with security issues. The whole solution will help get rid of the security holes in the workflow of development and deployment of applications using docker,

In this project, we learnt various basic operations on Docker, such as building images, extracting images, etc. Besides, we learn to manage and configure the private registry, including manipulating the endpoint and notifications in registry. Surprisingly, docker registry has limitation from the perspective of maximizing the waiting time for endpoint. Registry will constantly send notifications to endpoint as long as it doesn't hear anything back from endpoint in a period of time, which is a short duration. It limits our safety check on relatively small images instead of large images, such as ubuntu. Because it takes a while for endpoint to respond to registry in order to check the safety of large images.

It is also a great experience working in a group. Teamwork is indeed powerful and enable us to turn our naive idea into real product. Meanwhile, what we have learnt in class is quite useful. The idea using caching to provide more efficient services inspired us to adopt this technique in order to improve the efficiency of the virus checking. This final version of program can help both Docker registry owner and client to keep away from malicious files.

Reference

- https://okrieg.github.io/EC500/PROJECTS/2016/sastry.html
- https://www.nist.gov/software-quality-group/national-software-reference-library-nsrl
- http://roussev.net/sdhash/sdhash.html
- https://github.com/docker/distribution/issues/1874
- https://docs.docker.com/engine/reference/builder/#escape
- https://docs.docker.com/install/linux/docker-ce/ubuntu/
- https://docs.docker.com/compose/install/#install-compose