

---

## RAPPORT TP

---

*Elèves:*

Mathis FOUCADE  
Jules PUGET  
Axelle ROUZIER

*Enseignant:*

M LOUTSCH ETIENNE

*Session 4: Sécurité dans la CI/CD*

## Première partie : Signature d'images avec COSIGN

Nous avons commencé par créer un projet public nommé "securite-cicd-lab" sur GitLab. Ce projet servira de base pour notre pipeline de CI/CD et contiendra tous les éléments nécessaires pour démontrer la sécurité dans la chaîne d'approvisionnement des conteneurs. La visibilité publique a été choisie pour faciliter l'accès au registre d'images Docker intégré à GitLab.

### Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

#### Project name

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

#### Project URL

#### Project slug

#### Project deployment target (optional)

#### Visibility Level ?

- ☐ Private  
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.
- ☐ Internal  
The project can be accessed by any logged in user except external users.
- ☒ Public  
The project can be accessed without any authentication.

#### Project Configuration

- ☒ Initialize repository with a README  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
- ☐ Enable Static Application Security Testing (SAST)  
Analyze your source code for known security vulnerabilities. [Learn more.](#)

## 1. Générer une paire de clés

```
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> .\cosign.exe generate-key-pair
Enter password for private key:
Enter password for private key again:
Private key written to cosign.key
Public key written to cosign.pub
```

Nous avons ensuite généré une paire de clés cryptographiques avec l'outil COSIGN en utilisant la commande *cosign generate-key-pair*. Cette étape est cruciale car elle crée les fichiers *cosign.key* (clé privée) et *cosign.pub* (clé publique) qui seront utilisés respectivement pour signer les images et vérifier leur authenticité. La clé privée est protégée par une passphrase, ce qui ajoute une couche de sécurité supplémentaire pour éviter les utilisations non autorisées.

Ces clés constituent la base de confiance sur laquelle repose la sécurité de notre chaîne d'approvisionnement.

## 2. Signer une image Docker et la pousser vers GitLab

```
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker build -t registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1 .
[+] Building 13.8s (6/6) FINISHED
=> [internal] load build definition from Dockerfile 0.2s
=> => transferring dockerfile: 92B 0.1s
=> [internal] load .dockerignore 0.2s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/alpine:latest 3.7s
=> [1/2] FROM docker.io/library/alpine:latest@sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3 4.9s
=> => resolve docker.io/library/alpine:latest@sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3 0.5s
=> => sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c 9.22kB / 9.22kB 0.0s
=> => sha256:1c4eef651f65e2f7daee7ee785882ac164b02b78fb74503052a26dc061c90474 1.02kB / 1.02kB 0.0s
=> => sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee5428ec8b 581B / 581B 0.0s
=> => sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB 2.0s
=> => extracting sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 2.1s
=> [2/2] RUN echo "Hello World" > /hello.txt 3.1s
=> exporting to image 0.4s
=> => exporting layers 0.4s
=> => writing image sha256:de3312f74227e8b6df206a3a294faadc0f4f0b71839897bb4afddc43489e1f6f 0.0s
=> => naming to registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1 0.0s
```

### Personal access tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

✔ Your new personal access token ×

.....

👁️ 🗑️

Make sure you save it - you won't be able to access it again.

Active personal access tokens 1 Add new token

Token name	Description	Scopes	Created	Last Used <span>?</span>	Last Used IPs <span>?</span>	Expires	Action
SC		read_registry, write_registry	Apr 28, 2025	Never	-	in 4 weeks	<span>🗑️</span> <span>🔄</span>

```
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker login registry.gitlab.com
Username: ar6472517
Password:
Login Succeeded
```

```
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker push registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1
The push refers to repository [registry.gitlab.com/ar6472517/securite-cicd-lab/image]
597ac8d099a1: Pushed
08000c18d16d: Pushed
v1: digest: sha256:f305c2b6b74fa72b0f2230b3852a6aae46cca97c9be99ea042e1e6d90705cbe8 size: 734
```

```

PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> .\cosign.exe sign --key cosign.
key registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1
Enter password for private key:
WARNING: Image reference registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1 uses a tag, not a digest, to identify
the image to sign.
    This can lead you to sign a different image than the intended one. Please use a
    digest (example.com/ubuntu@sha256:abc123...) rather than tag
    (example.com/ubuntu:latest) for the input to cosign. The ability to refer to
    images by tag will be removed in a future release.

WARNING: "registry.gitlab.com/ar6472517/securite-cicd-lab/image" appears to be a private repository, please confirm uplo
adding to the transparency log at "https://rekor.sigstore.dev"
Are you sure you would like to continue? [y/N] y

    The sigstore service, hosted by sigstore a Series of LF Projects, LLC, is provided pursuant to the Hosted Project
Tools Terms of Use, available at https://lfprojects.org/policies/hosted-project-tools-terms-of-use/.
    Note that if your submission includes personal data associated with this signed artifact, it will be part of an
immutable record.
    This may include the email address associated with the account with which you authenticate your contractual Agree
ement.
    This information will be used for signing this artifact and will be stored in public transparency logs and cannot
be removed later, and is subject to the Immutable Record notice at https://lfprojects.org/policies/hosted-project-tool
s-immutable-records/.

By typing 'y', you attest that (1) you are not submitting the personal data of any other person; and (2) you understand
and agree to the statement and the Agreement terms at the URLs listed above.
Are you sure you would like to continue? [y/N] y
tlog entry created with index: 203868257
Pushing signature to: registry.gitlab.com/ar6472517/securite-cicd-lab/image

```

Nous avons construit une image Docker à partir d'un Dockerfile simple et l'avons étiquetée avec la version v1. Après s'être connecté au registre GitLab avec notre token personnel, nous avons poussé cette image vers le registre du projet. Ensuite, nous avons utilisé COSIGN pour signer l'image avec notre clé privée, créant ainsi une signature cryptographique qui garantit l'authenticité et l'intégrité de l'image. Cette signature est stockée dans le registre GitLab aux côtés de l'image, permettant à quiconque possède la clé publique de vérifier son authenticité.

### 3. Modifier l'image et pousser une version modifiée

```

PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker run -it --rm registry.gitlab.com/ar6472517/securite-cicd-lab/image:v1 touch /tampered
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> $LAST_CONTAINER = docker ps -lq
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker commit $LAST_CONTAINER registry.gitlab.com/ar6472517/securite-cicd-lab/image:v2
sha256:caed20e6cca51b76e946d375def1692e6737044667e379207b5066e224cd8fed

```

```

PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> docker push registry.gitlab.com/ar6472517/securite-cicd-lab/image:v2
The push refers to repository [registry.gitlab.com/ar6472517/securite-cicd-lab/image]
792b4f29f4a1: Pushed
f3e0b5de2076: Pushed
f19d365f1dae: Pushed
4bd876ba25ad: Pushed
34d394b0a809: Pushed
8f0f238e0cad: Pushed
5fcfe0cf80d2: Pushed
04f5f57834ba: Pushed
70736de621ab: Pushed
9e884bd72188: Pushed
63ca1fbb43ae: Pushed
v2: digest: sha256:246578b61f9ce1212077d3f41364e699e322efae5f8e415c0708aded9e642132 size: 2624

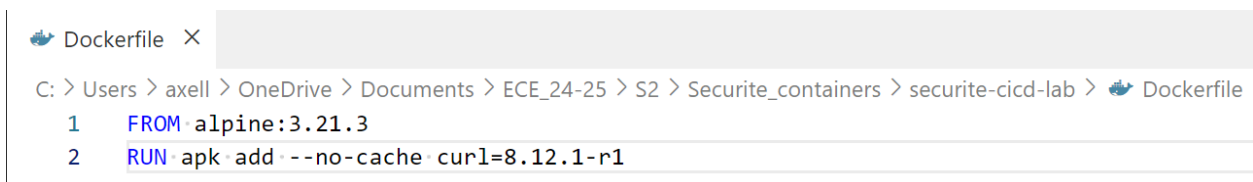
```

### 4. Vérifier la signature avant/après modification

```
PS C:\Users\axell\OneDrive\Documents\ECE_24-25\S2\Securite_containers\securite-cicd-lab> .\cosign.exe verify --key cosign
n.pub registry.gitlab.com/ar6472517/securite-cicd-lab/image:v2
Error: no signatures found
error during command execution: no signatures found
```

Pour démontrer l'efficacité de la signature, nous avons créé une version modifiée de l'image (v2) en ajoutant un fichier "tampered". Cette modification simule une tentative d'altération malveillante de l'image. Lorsque nous avons vérifié l'image originale avec la commande *cosign verify*, la vérification a réussi, confirmant que l'image v1 n'avait pas été modifiée depuis sa signature. En revanche, la vérification de l'image v2 a échoué avec l'erreur "no signatures found", prouvant ainsi que le système de signature détecte efficacement les modifications non autorisées des images. Ce mécanisme est essentiel pour se protéger contre les attaques de la chaîne d'approvisionnement où des acteurs malveillants pourraient tenter d'injecter du code malveillant dans nos conteneurs.

## 1. Créer un pipeline CI/CD avec Gitlab CI



```
.gitlab-ci.yml X
C: > Users > axell > OneDrive > Documents > ECE_24-25 > S2 > Securite_containers > securite-cicd-lab > .gitlab-ci.yml

1  stages:
2    - lint
3    - build
4    - verify
5    - scan
6
7    # Étape 1 : Scan du Dockerfile avec Hadolint
8  hadolint-scan:
9    stage: lint
10   image: hadolint/hadolint:latest-debian
11   script:
12     - hadolint --failure-threshold warning Dockerfile
13
14   # Étape 2 : Build de l'image et cosign l'image
15  build-image:
16    stage: build
17    variables:
18      COSIGN_YES: "true" # Used by Cosign to skip confirmation prompts for non-destructive operations
19      DOCKER_IMAGE_NAME: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
20      id_tokens:
21        SIGSTORE_ID_TOKEN: # Used by Cosign to get certificate from Fulcio
```

Nous avons créé un fichier .gitlab-ci.yml pour définir notre pipeline de CI/CD avec quatre étapes distinctes: lint, build, verify et scan. Cette structure permet d'introduire des contrôles de sécurité à chaque phase du processus de développement. La première version de notre Dockerfile utilisait Alpine 3.21.3 avec curl 8.12.1-r1, des versions relativement récentes et sans vulnérabilités critiques connues.


Status	Pipeline	Created by	Stages	Actions
<div><div>Passed</div><div>00:01:32</div><div>3 minutes ago</div></div>	<div>readme</div> <div>#1790749425</div> <div>main</div> <div>2521e5de</div> <div>latest</div> <div>branch</div>	<div></div>	<div><div></div><div></div><div></div><div></div></div>	<div><div></div></div>

Status	Job	Pipeline	Coverage
<div><div>Passed</div><div>00:00:22</div><div>3 minutes ago</div></div>	<div>#9855033953: trivy-scan</div> <div>main</div> <div>2521e5de</div>	<div>#1790749425 created by</div> <div>Stage: scan</div>	<div><div></div></div>
<div><div>Passed</div><div>00:00:17</div><div>4 minutes ago</div></div>	<div>#9855033952: verify_image</div> <div>main</div> <div>2521e5de</div>	<div>#1790749425 created by</div> <div>Stage: verify</div>	<div><div></div></div>
<div><div>Passed</div><div>00:00:52</div><div>4 minutes ago</div></div>	<div>#9855033949: build-image</div> <div>main</div> <div>2521e5de</div>	<div>#1790749425 created by</div> <div>Stage: build</div>	<div><div></div></div>
<div><div>Passed</div><div>00:00:16</div><div>5 minutes ago</div></div>	<div>#9855033940: hadolint-scan</div> <div>main</div> <div>2521e5de</div>	<div>#1790749425 created by</div> <div>Stage: lint</div>	<div><div></div></div>

Lors de l'exécution du premier pipeline, nous avons observé que toutes les étapes ont réussi:

- L'étape "hadolint-scan" a validé que notre Dockerfile respectait les bonnes pratiques sans émettre d'avertissements critiques.
- L'étape "build-image" a construit notre image Docker et l'a signée automatiquement grâce à COSIGN intégré dans le pipeline.
- L'étape "verify\_image" a vérifié l'authenticité de la signature, confirmant que l'image n'avait pas été modifiée entre sa construction et sa vérification.
- L'étape "trivy-scan" a analysé l'image à la recherche de vulnérabilités de sécurité connues, et n'a pas détecté de problèmes critiques ou élevés.

Cette exécution réussie démontre l'efficacité d'un pipeline CI/CD correctement sécurisé, où chaque étape vérifie un aspect différent de la sécurité, formant ainsi plusieurs couches de protection complémentaires.

 Dockerfile


C: > Users > axell > OneDrive > Documents > ECE\_24-25 > S2 > Securite\_containers > securite-cicd-lab > Dockerfile





1

FROM alpine:3.12

2

RUN apk add --no-cache curl=7.79.1-r1

Status	Pipeline	Created by	Stages	Actions
<div><div>Failed</div><div>00:01:31</div><div>3 minutes ago</div></div>	<div>Introduction d'une vulnérabilité</div> <div>#1790755775</div> <div>main 58f55708</div> <div>latest branch</div>		<div><div>✓</div><div>✓</div><div>✓</div><div>✗</div></div>	<div><div>↺</div><div>↻</div><div>⌵</div></div>

Status	Job	Pipeline	Coverage
<div><div>Failed</div><div>00:00:23</div><div>3 minutes ago</div></div>	<div>#9855076344: trivy-scan</div> <div>main 58f55708</div>	<div>#1790755775 created by </div> <div>Stage: scan</div>	
<div><div>Passed</div><div>00:00:17</div><div>3 minutes ago</div></div>	<div>#9855076341: verify_image</div> <div>main 58f55708</div>	<div>#1790755775 created by </div> <div>Stage: verify</div>	
<div><div>Passed</div><div>00:00:55</div><div>4 minutes ago</div></div>	<div>#9855076338: build-image</div> <div>main 58f55708</div>	<div>#1790755775 created by </div> <div>Stage: build</div>	
<div><div>Passed</div><div>00:00:12</div><div>5 minutes ago</div></div>	<div>#9855076335: hadolint-scan</div> <div>main 58f55708</div>	<div>#1790755775 created by </div> <div>Stage: lint</div>	

Dans la seconde partie du TP, nous avons intentionnellement introduit une vulnérabilité en modifiant notre Dockerfile pour utiliser Alpine 3.12 avec curl 7.79.1-r1, une version connue pour contenir des vulnérabilités critiques. Lors de l'exécution du pipeline avec cette modification, nous avons observé que:

- Les étapes "hadolint-scan", "build-image" et "verify\_image" ont toujours réussi, car elles ne vérifient que la syntaxe du Dockerfile et l'authenticité de la signature.
- L'étape "trivy-scan" a échoué en détectant des vulnérabilités critiques ou élevées dans la version de curl utilisée, bloquant ainsi automatiquement le pipeline.

Cet échec du pipeline démontre l'importance du scan de vulnérabilités comme dernière ligne de défense. Même si une image est correctement construite et signée, si elle contient des composants vulnérables, elle représente toujours un risque de sécurité. Cette simulation illustre parfaitement comment un pipeline CI/CD bien conçu peut empêcher le déploiement d'images potentiellement dangereuses en production, évitant ainsi des incidents de sécurité coûteux.