



# SECURITY ANALYSIS

Alessandro Bocci

`name.surname@unipi.it`

Advanced Software Engineering (Lab)

14/11/2025

# What will you do?

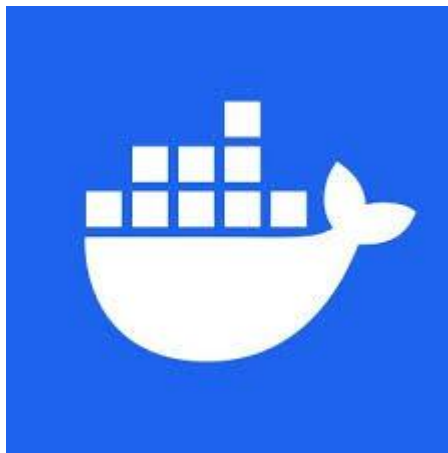
You will perform analysis on:

- The codebase.
- The codebase dependencies.
- The Docker images.



# Software Prerequisites

- Bandit (`pip install bandit`).
- pip-audit (`pip install pip-audit`).
- Docker account.
- GitHub account.
- trivy



# Secure coding practices

- Complexity of source code leads to more security vulnerabilities.
- Exponential increase of defects as number of lines of code increases.
- Functional and security testing is utterly important.
- Two types of testing and analyses:
  - Dynamic
  - Static



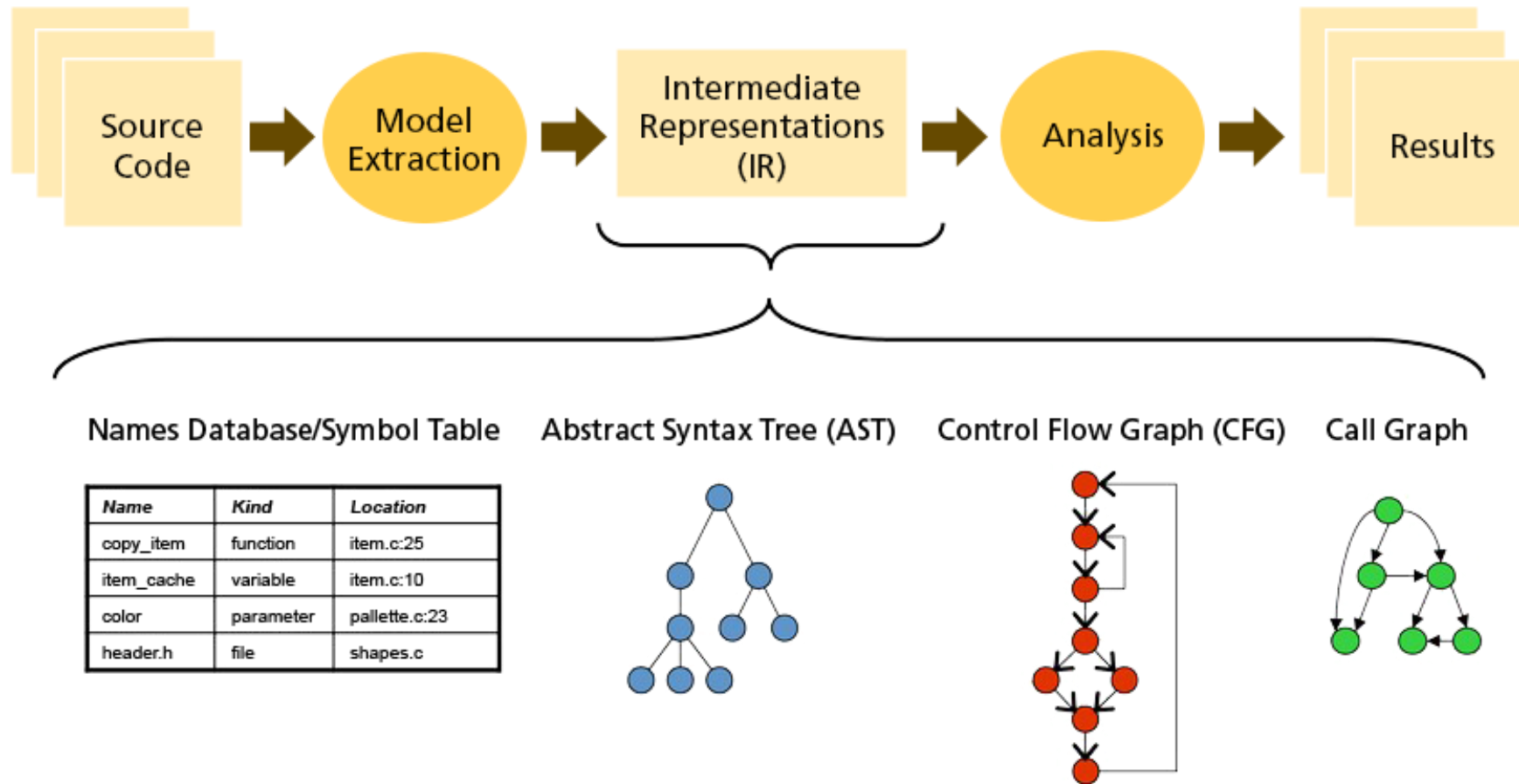
# Dynamic Analysis

Dynamic Analysis involves running applications in a controlled environment to observe their behavior and interactions in real-time.

- Identifies runtime vulnerabilities like memory leaks, data exposure, and authentication issues.
- Essential for uncovering zero-day vulnerabilities and unforeseen behaviors due to external factors.

Unfortunately, there are few tools for our purposes that are free and easy to use.

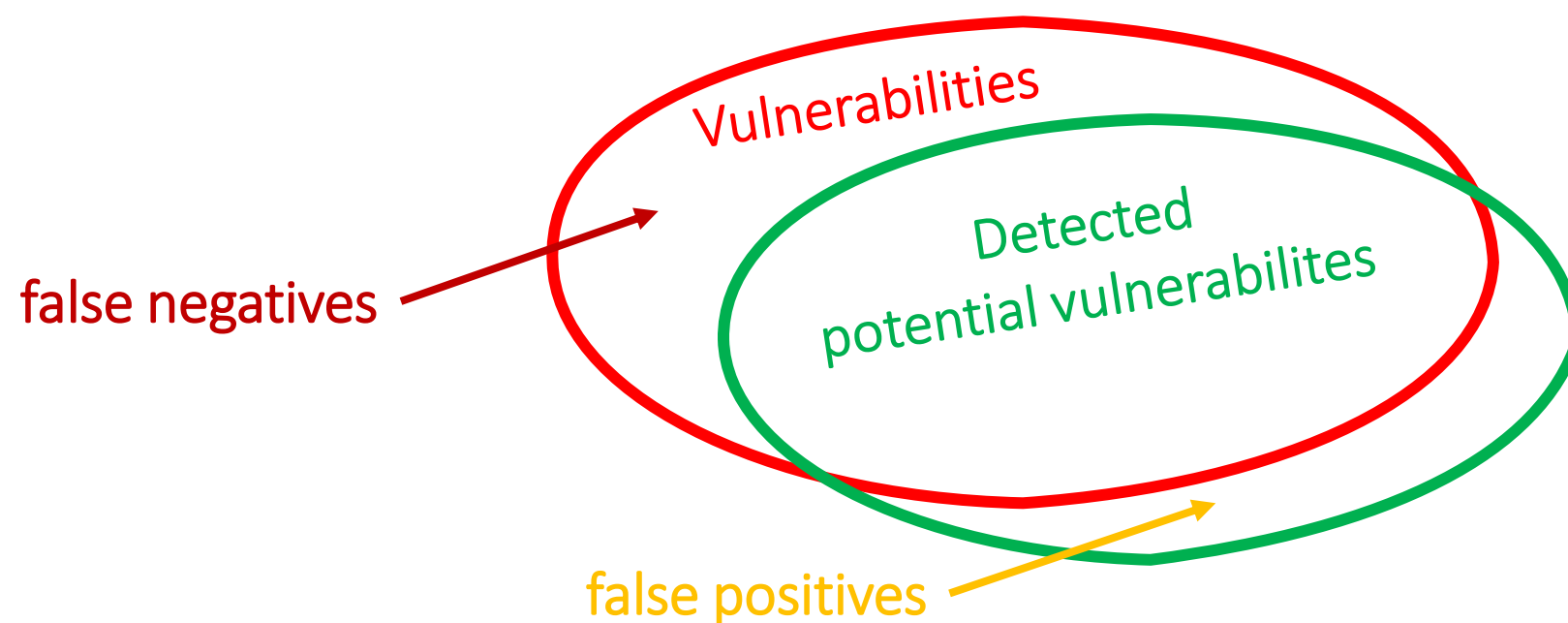
# Vulnerability Avoidance with Static Analysis



Static Analysis = Analyze the system (source code or its representation) to check some property without running it.

# A note on Static Analysis

All program behaviours



# In this lab...

We focus on static analysis:

- Source code analysis with Bandit.
- Dependency analysis with pip-audit and Dependabot.
- Docker image analysis with Docker scout and trivy.

Note: Bandit and pip-audit are for Python only.





# Source Code: Bandit

- Bandit is a static analysis tool designed to find common security issues in Python code, by exploiting known patterns (plugins).
- Bandit was originally developed within the OpenStack Security Project and later re-homed to PyCQA.
- It recognizes 70 vulnerabilities out-of-the-box.

Docs @ <https://bandit.readthedocs.io/en/latest/>



```
pip install bandit
```

```
bandit -r <path to code>
```

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False  
Severity: High    Confidence: High  
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)  
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html  
Location: ASE/lab6/test.py:2:9  
1      import hashlib  
2      result = hashlib.md5(b'ASE ASE ASE')  
3      print("The byte equivalent of hash is : ", end = "")
```

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
Severity: High    Confidence: High
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html
Location: ASE/lab6/test.py:2:9
1     import hashlib
2     result = hashlib.md5(b'ASE ASE ASE')
3     print("The byte equivalent of hash is : ", end = "")
```

Issue -> Use of MD5 (cryptographic hash function).

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
Severity: High Confidence: High
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html
Location: ASE/lab6/test.py:2:9
1     import hashlib
2     result = hashlib.md5(b'ASE ASE ASE')
3     print("The byte equivalent of hash is : ", end = "")
```

Severity of the Issue (how much the Issue is ‘dangerous’):

- Bandit classifies issues in Low, Medium and High.
- It gives a level of confidence (how much Bandit is confident about it) for every issue, also Low, Medium and High.

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
Severity: High    Confidence: High
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html
Location: ASE/lab6/test.py:2:9
1     import hashlib
2     result = hashlib.md5(b'ASE ASE ASE')
3     print("The byte equivalent of hash is : ", end = "")
```

Definition of why it is an issue.

You can use it for understand how to resolve it.

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False  
Severity: High    Confidence: High  
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)  
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html  
Location: ASE/lab6/test.py:2:9  
1     import hashlib  
2     result = hashlib.md5(b'ASE ASE ASE')  
3     print("The byte equivalent of hash is : ", end = "")
```

Bandit's info about the issue.

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
Severity: High    Confidence: High
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html
Location: ASE/lab6/test.py:2:9
1      import hashlib
2      result = hashlib.md5(b'ASE ASE ASE')
3      print("The byte equivalent of hash is : ", end = "")
```

Location of the issue. In this example:

- The path for the file having the issue is **ASE/lab6/**
- The file with the issue is **test.py**

# Example – B324: hashlib, use of weak MD5

```
>> Issue: [B324:hashlib] Use of weak MD5 hash for security. Consider usedforsecurity=False
Severity: High    Confidence: High
CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html)
More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b324\_hashlib.html
Location: ASE/lab6/test.py:2:9
1      import hashlib
2      result = hashlib.md5(b'ASE ASE ASE')
3      print("The byte equivalent of hash is : ", end = "")
```

Location of the issue. In this example:

- The path for the file having the issue is **ASE/lab6/**
- The file with the issue is **test.py**
- The line of the issue is 2 (and column 9).



# Example – B324: hashlib, use of weak MD5

How to resolve it?

Change the hash function with a secure one.

From:

```
result = hashlib.md5(b'ASE ASE ASE')
```

To:

```
result = hashlib.sha256(b'ASE ASE ASE')
```

# Example – B324: hashlib, use of weak MD5

Remember:

- Do not change the behaviour of the code!  
(in the example the hash of the string is done but with SHA256).
- Do not follow Bandit's suggestions blindly!  
(in the example the suggestion was to put `usedforsecurity=False`, ask yourself when it is correct).

# Dependencies: pip-audit

- [pip-audit](#) is a tool for scanning Python environments for packages with known vulnerabilities.
- It uses the Python Packaging Advisory Database via the [PyPI JSON API](#) as a source of vulnerability reports.
- It scan all the packages installed with pip for vulnerabilities.
- Be careful, it can break backward compatibility.
- It has an official GitHub Action.

Install

```
pip install pip-audit
```

Run

```
pip-audit
```

Fix vulnerabilities

```
pip-audit --fix
```

# pip-audit

It scans the pip environment of the system where is executed.

We install Python packages inside the Docker containers.

There are several ways to use it:

- Include the command in the build of the image.
- Build an image that execute it before building the other images.
- ....

Choose what you think is fair for your purposes.

# Dependencies: GitHub Dependabot

Dependabot is a tool integrated in GitHub to check vulnerabilities in repository dependencies.

- Go in the Settings of your repo.
- Advanced Security section.
- Enable **Dependabot security updates** and **Grouped security updates**.
- Go in the Security tab of your repo and look if Dependabot find something.
- Probably you are ok, our code use relatively recent things.
- If you want to see Dependabot at work, play with its [Demo](#).



# Docker images: Docker scout

Docker tool to scan images for vulnerabilities.  
You need a Docker account for use it easily.

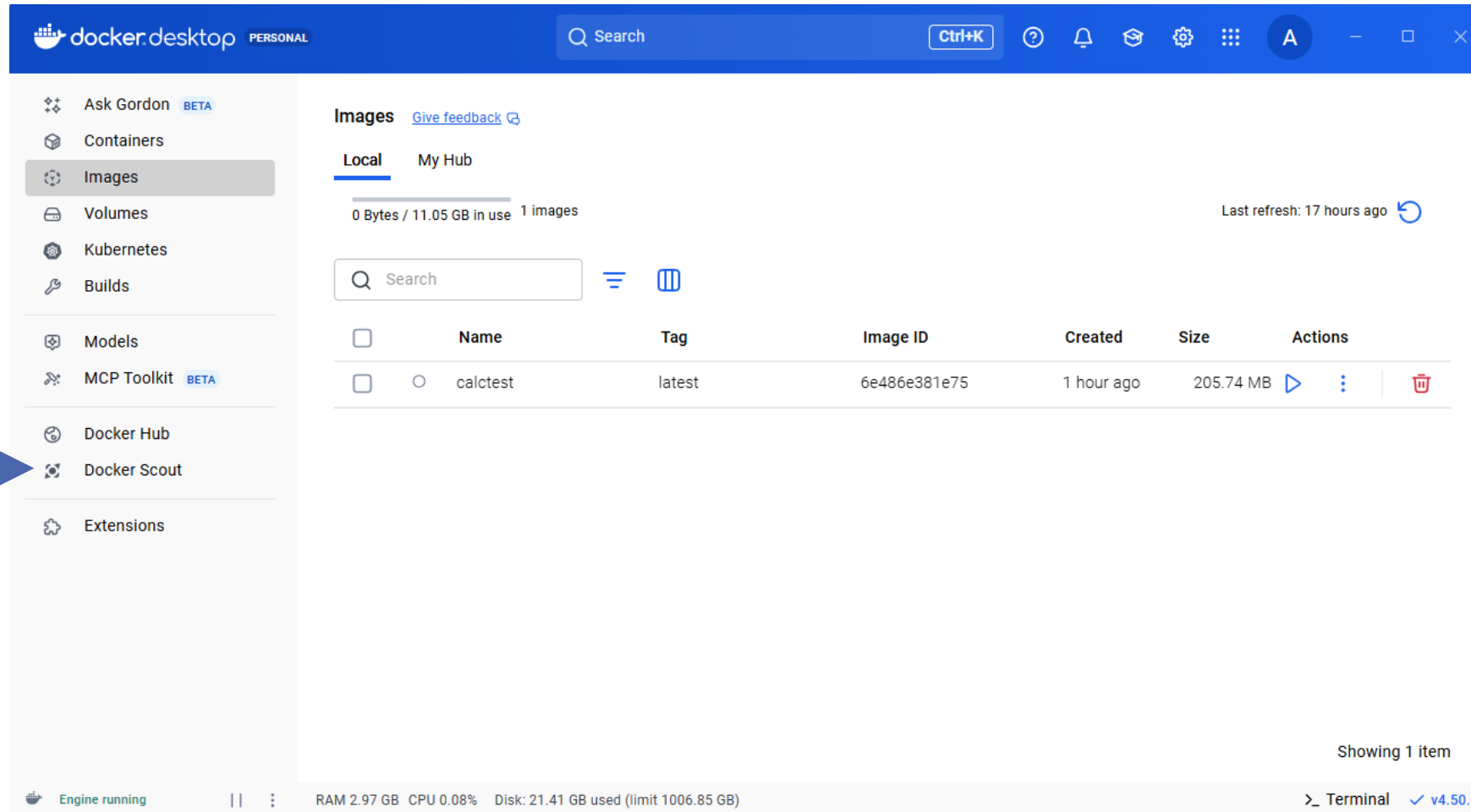
- Local images
- Remote images

Docs @ <https://docs.docker.com/scout/>

We focus on the local ones.



# Docker scout in Docker Desktop



The screenshot shows the Docker Desktop application window. The left sidebar contains a list of navigation items: Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout, and Extensions. A blue arrow points to the 'Docker Scout' item. The main panel displays the 'Images' section, with 'Local' and 'My Hub' tabs. Below the tabs, it shows '0 Bytes / 11.05 GB in use' and '1 images'. A search bar and a table of images are visible. The table has columns: Name, Tag, Image ID, Created, Size, and Actions. One image is listed: 'calctest' with tag 'latest' and Image ID '6e486e381e75'. The status bar at the bottom indicates 'Engine running', 'RAM 2.97 GB', 'CPU 0.08%', 'Disk: 21.41 GB used (limit 1006.85 GB)', and a 'Terminal' button with version 'v4.50.0'.

**Images** [Give feedback](#)

**Local** **My Hub**

0 Bytes / 11.05 GB in use 1 images Last refresh: 17 hours ago

Search

	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	calctest	latest	6e486e381e75	1 hour ago	205.74 MB	<a href="#">Play</a> <a href="#">More</a> <a href="#">Delete</a>

Showing 1 item

Engine running | RAM 2.97 GB CPU 0.08% Disk: 21.41 GB used (limit 1006.85 GB) >\_ Terminal v4.50.0



# Docker scout in Docker Desktop

The screenshot displays the Docker Desktop application window. The top blue header bar contains the Docker Desktop logo, the word "PERSONAL", a search bar, and a "Ctrl+K" shortcut button. Below the header, a left sidebar lists navigation options: Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (highlighted), and Extensions. The main content area is titled "Docker Scout" with a "Give feedback" link. Below this is a large heading "Advanced image analysis with Docker Scout". A light blue notification bar states: "Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)". A descriptive paragraph follows: "Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#)". A table with two columns, "Sample image" and "Vulnerabilities", is shown. The "Sample image" column contains a dropdown menu with "calctest:latest" selected. The "Vulnerabilities" column shows "Not analyzed" and a blue "Analyze image" button, which is pointed to by a blue arrow from the right. Below the table, a note says: "Advanced image analysis can be accessed by viewing any of the images on the [Images view](#)". A section titled "How to access Advanced image analysis" includes a smaller screenshot of the Docker Desktop "Images" view, showing a search bar and a list of images. The bottom status bar indicates "Engine running", system resources (RAM 1.77 GB, CPU 9.70%, Disk: --- GB used (limit --- GB)), and a terminal icon with the version "v4.50.0".

**Docker Scout** [Give feedback](#)

## Advanced image analysis with Docker Scout

Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)

Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#).

Sample image	Vulnerabilities
calctest:latest	Not analyzed <a href="#">Analyze image</a>

Advanced image analysis can be accessed by viewing any of the images on the [Images view](#).

### How to access Advanced image analysis

The bottom screenshot shows the Docker Desktop interface with the "Images" view selected, displaying a search bar and a list of images.

Engine running | RAM 1.77 GB CPU 9.70% Disk: --- GB used (limit --- GB) | Terminal v4.50.0



# Docker scout in Docker Desktop

The screenshot shows the Docker Desktop application window. The top bar is blue with the Docker Desktop logo, a search bar, and various icons. The left sidebar contains a list of navigation items: Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (selected), and Extensions. The main content area is titled 'Docker Scout' with a 'Give feedback' link. Below this is a heading 'Advanced image analysis with Docker Scout'. A light blue banner contains a message: 'Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)'. Below the banner, a paragraph explains the purpose: 'Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#)'. A section titled 'Sample image' shows a dropdown menu with 'calctest:latest' selected. To the right, under 'Vulnerabilities', there is a row of colored boxes representing different severity levels: 1 (red), 2 (red), 2 (orange), 20 (yellow), and 0 (green). A link 'View packages and CVEs' is next to the counts. Below this, a note states: 'Advanced image analysis can be accessed by viewing any of the images on the [Images view](#).' A section titled 'How to access Advanced image analysis' shows a smaller version of the Docker Desktop interface with the 'Images' tab selected. The bottom status bar shows 'Engine running', system resources (RAM 1.02 GB, CPU 0.08%, Disk: 21.32 GB used (limit 1006.85 GB)), and a 'Terminal' button with the version 'v4.50.0'.

**Docker Scout** [Give feedback](#)

## Advanced image analysis with Docker Scout

Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)

Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#).

Sample image	Vulnerabilities
calctest:latest	1 2 2 20 0 <a href="#">View packages and CVEs</a>

Advanced image analysis can be accessed by viewing any of the images on the [Images view](#).

### How to access Advanced image analysis

The screenshot shows a smaller version of the Docker Desktop interface with the 'Images' tab selected.

Engine running | RAM 1.02 GB CPU 0.08% Disk: 21.32 GB used (limit 1006.85 GB) | Terminal v4.50.0

# Docker scout in Docker Desktop

The screenshot displays the Docker Desktop interface with the Docker Scout extension active. The left sidebar shows navigation options: Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (selected), and Extensions. The main panel is titled 'Docker Scout' with a 'Give feedback' link. Below the title is the heading 'Advanced image analysis with Docker Scout'. A notification bar states: 'Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)'. The main content area explains: 'Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#)'. A section titled 'Sample image' shows a dropdown menu with 'calctest:latest' selected. To the right, under 'Vulnerabilities', a bar chart shows counts: 1 Critical, 2 High, 2 Medium, 20 Low, and 0 Unrated. A link '[View packages and CVEs](#)' is provided. Below the chart, text reads: 'Advanced image analysis can be accessed by viewing any of the images o'. A section titled 'How to access Advanced image analysis' shows a smaller version of the Docker Desktop interface with the 'Images' tab selected. The bottom status bar indicates 'Engine running', 'RAM 1.02 GB', 'CPU 0.08%', 'Disk: 21.32 GB used (limit 1006.85 GB)', and 'v4.50.0'.

**Docker Scout** [Give feedback](#)

## Advanced image analysis with Docker Scout

Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)

Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#).

**Sample image**

calctest:latest

**Vulnerabilities**

Severity	Count
Critical	1
High	2
Medium	2
Low	20
Unrated	0

[View packages and CVEs](#)

Advanced image analysis can be accessed by viewing any of the images o

### How to access Advanced image analysis

Local Desktop Engine

Images [Give feedback](#)

Engine running | RAM 1.02 GB CPU 0.08% Disk: 21.32 GB used (limit 1006.85 GB) | Terminal v4.50.0

# Docker scout in Docker Desktop

**Docker Scout** [Give feedback](#)

## Advanced image analysis with Docker Scout

Want to use Docker Scout on your remote repositories? [Set up your integrations now](#)

Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#).

**Sample image**

calctest:latest

**Vulnerabilities**

Critical	High	Medium	Low	Unknown
1	2	2	20	0

[View packages and CVEs](#)

Advanced image analysis can be accessed by viewing any of the images in the Docker Hub or Docker Scout interface.

### How to access Advanced image analysis

The screenshot shows a Docker Desktop window with the 'Images' tab selected. The 'Local Desktop Engine' is shown. The status bar at the bottom indicates 'Engine running', 'RAM 1.02 GB', 'CPU 0.08%', and 'Disk: 21.32 GB used (limit 1006.85 GB)'. The version is 'v4.50.0'.

# Docker scout in Docker Desktop

The screenshot shows the Docker Desktop interface with the 'Images' tab selected. The main view displays the 'calctest:latest' image, which was created 59 minutes ago and has a size of 205.74 MB. The image is analyzed by Docker Scout, showing 22 vulnerabilities and 144 packages. A blue arrow points to the 'Fixable' checkbox in the 'Vulnerabilities' section, which is currently unchecked. The 'Layers' section shows the image is built from 'python:3.12-slim' and 'calctest:latest'.

**Layers (16)**

Layer	0	1	2	3	4
python:3.12-slim	0	0	2	20	0
calctest:latest	1	2	0	0	0

**Vulnerabilities (22)** Packages (144) [Give feedback](#)

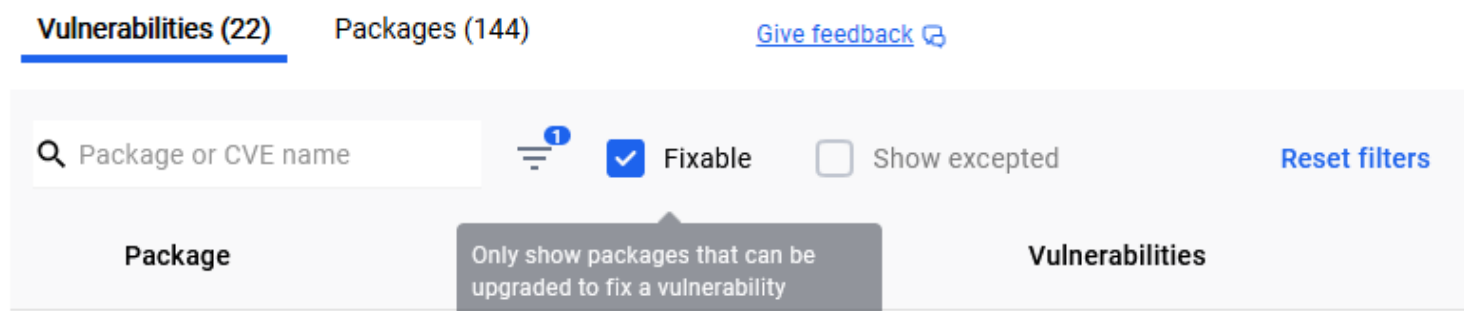
Search Package or CVE name ☐ Fixable ☐ Show excepted [Reset filters](#)

CVE ID	Severity	Fixable	Present
CVE-2025-8869	5.9 M	✓	✗
CVE-2025-45582	4.1 M		✗
CVE-2005-2541	N/A L		✗
CVE-2019-9192	N/A L		✗
CVE-2019-1010022	N/A L		✗

Engine running | RAM 2.77 GB CPU 0.50% Disk: 21.41 GB used (limit 1006.85 GB) | Terminal v4.50.0

# Docker scout in Docker Desktop

- Resolve any (fixable) **critical** and **high** vulnerability.



- Some vulnerabilities are not fixable in this moment, you can filter by fixable vulnerabilities in the details.
- Most of the time you need only to update base images (e.g. from python:3.12-slim to another one)
- Google and AI can help you 😊
- After every fix you have to build the image again.

# Docker scout in CLI

- Login in Docker Hub and insert the password when prompted:

```
docker login -u <username>
```

- Scan the image with:

```
docker scout cves <image-name>
```

- You can use filters on the vulnerabilities:

```
docker scout cves --only-severity critical,high  
docker scout cves --only-fixed
```

# Docker images: Trivy

Trivy is an all-in-one, open-source security scanner for cloud-native apps. It can scan:

- Container images.
- Local filesystems.
- Git repositories.
- Kubernetes clusters.
- ...

Docs @ <https://trivy.dev/docs/latest/>



# Trivy



Our focus is on container (Docker) images.

Trivy supports two targets for container images.

- Files inside container images.
- Container image metadata.

Looking for vulnerabilities, misconfigurations, secrets, licenses.



# Trivy



Scan the image with:

```
trivy image <image-name>
```

By default, it scans for vulnerabilities and secrets.

You can specify what scan with the option

```
--scanners
```

Adding  **vulns** , **misconfig**, or **license**. Secrets are always there.

# Trivy



Secrets are not Docker secrets.

Are rules to find if are exposed:

- AWS access key
- GCP service account
- GitHub personal access token
- GitLab personal access token
- Slack access token
- etc.

# Today's Lab



Use `microase` code or the project to familiarise with the tools.

## LAB TODO

- Run Bandit to check codebase vulnerabilities.
- Run pip-audit to check vulnerabilities in Python dependencies.
- Use GitHub's Dependabot to do another dependency check.
- Use Docker Scout and trivy to check vulnerabilities in your docker images.
- Try to resolve all the vulnerabilities.
- Check the docs of the tools to interpret reports and vulns.

# Lab take away

- ❑ Familiarise with static analysis tools for security.
- ❑ Fix vulnerabilities in codebase, dependencies and docker images.



# Project take away

- ☐ Use a static analysis tool to check for code vulnerabilities (only if your programming language(s) have one free tool for it).
- ☐ Your project should be free of dependencies vulnerabilities.
- ☐ Your docker images must be free of fixable docker image vulnerabilities with critical and high severity.

