



SECURITY ANALYSIS

Alessandro Bocci

name.surname@unipi.it

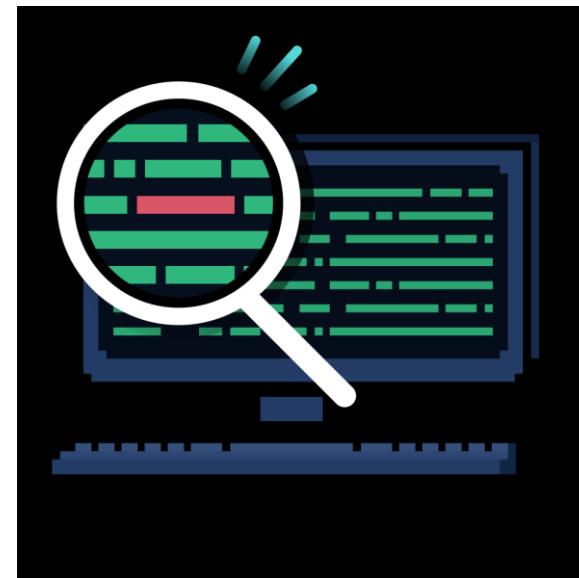
Advanced Software Engineering (Lab)

14/11/2025

What will you do?

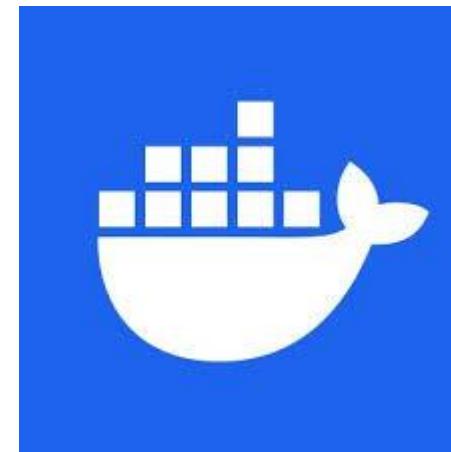
You will perform analysis on:

- 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



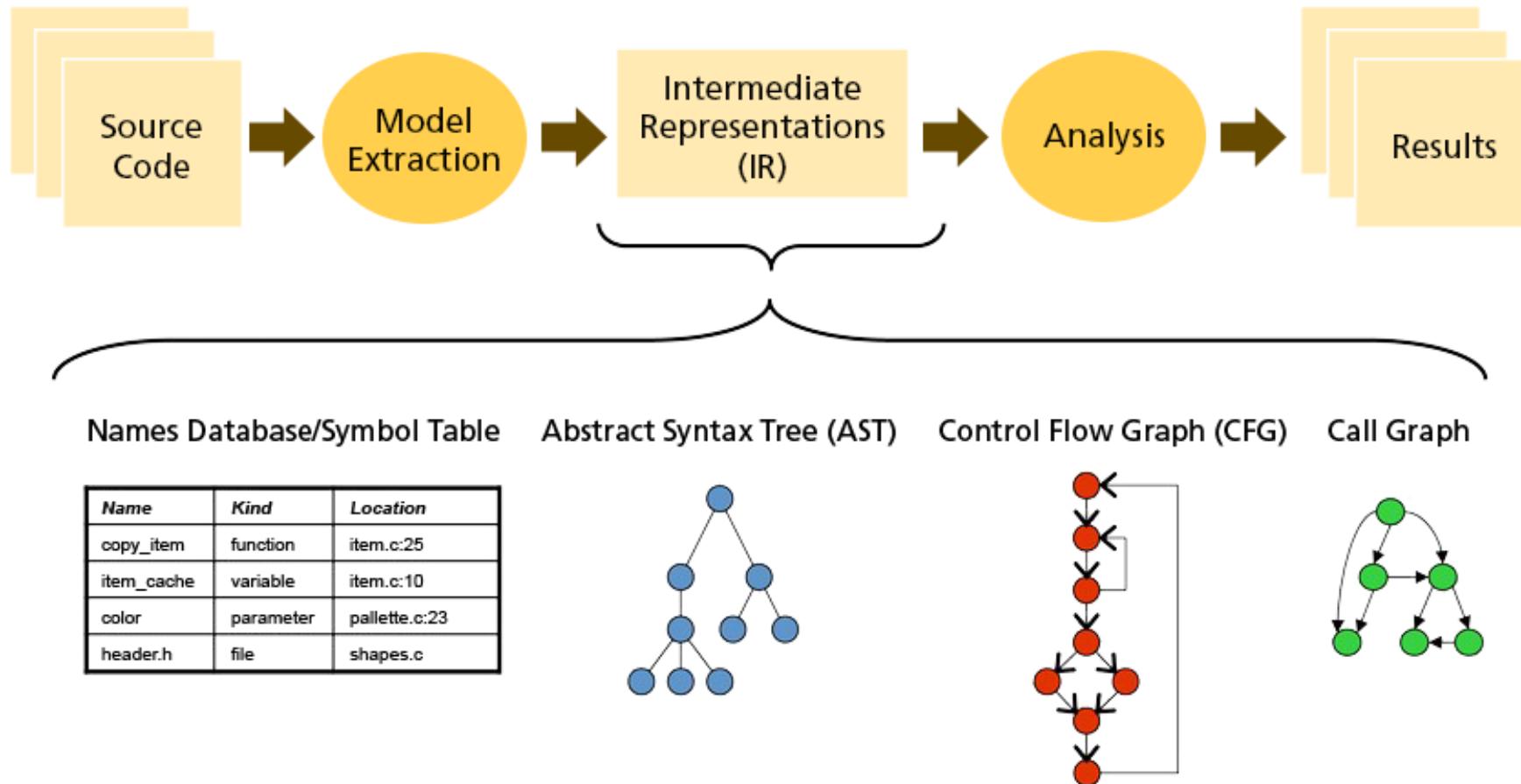
Dynamic Analysis

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.

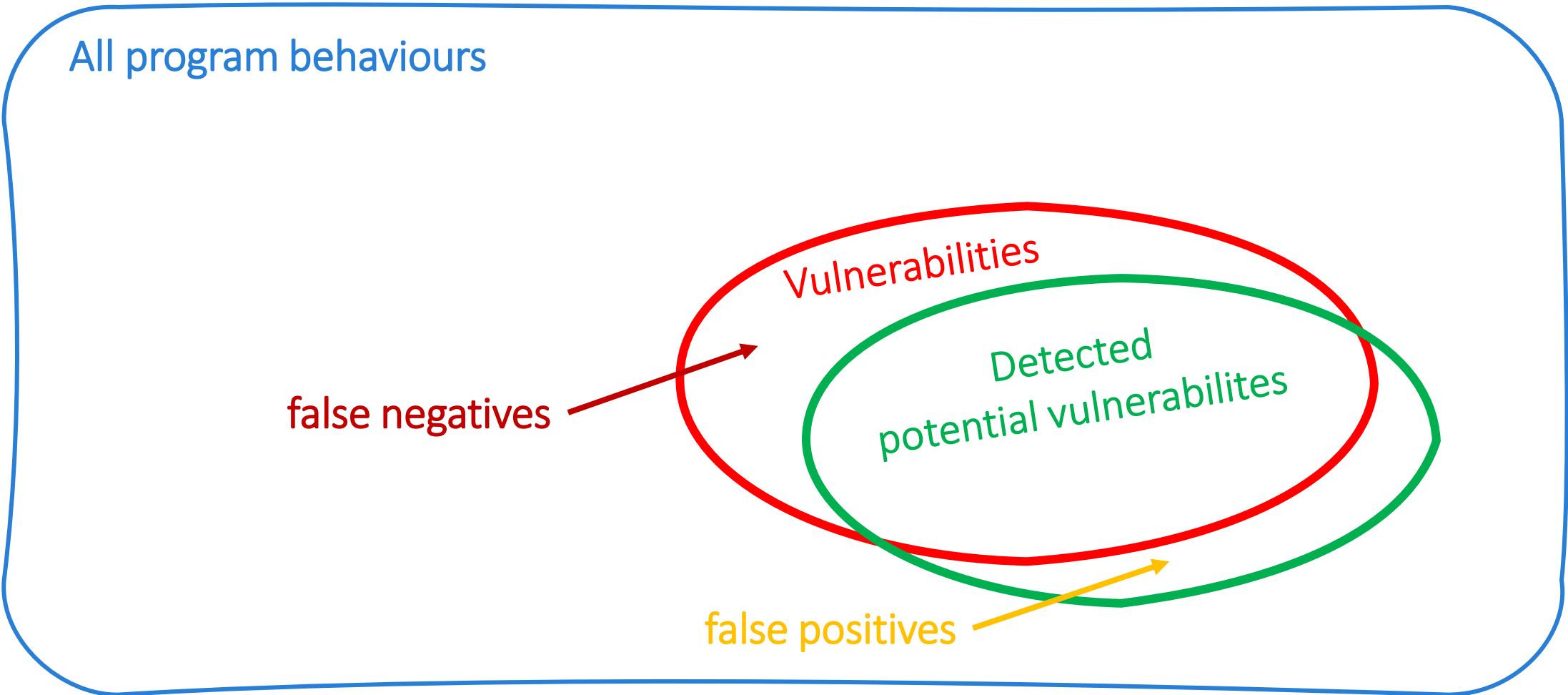
Unfortunately, there are few tools for our purposes that are free and easy to use.

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



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 scans 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:

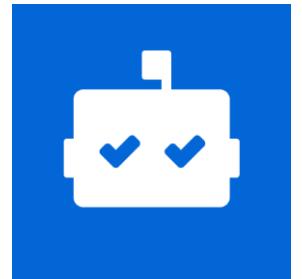
- Include the command in the build of the image.
- Build an image that execute it before building the other images.
-

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 or 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 has a blue arrow pointing to the "Docker Scout" link under the "Extensions" section. The main area displays the "Images" tab, showing one local image named "calctest" with details: Name: calctest, Tag: latest, Image ID: 6e486e381e75, Created: 1 hour ago, Size: 205.74 MB. The status bar at the bottom indicates "Engine running" and system resources: RAM 2.97 GB, CPU 0.08%, Disk: 21.41 GB used (limit 1006.85 GB). A terminal icon shows "v4.50.0".

docker desktop PERSONAL

Ask Gordon BETA

Containers

Images

Volumes

Kubernetes

Builds

Models

MCP Toolkit BETA

Docker Hub

Docker Scout

Extensions

Search

CTRL+K

?

Notification bell

Help

Settings

A

Last refresh: 17 hours ago

Showing 1 item

Name	Tag	Image ID	Created	Size	Actions
calctest	latest	6e486e381e75	1 hour ago	205.74 MB	

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 shows the Docker Desktop application window. The title bar includes the Docker Desktop logo, the text "docker.desktop PERSONAL", a search bar with placeholder "Search", and various control icons (Ctrl+K, Help, Notifications, Settings, Window controls). The left sidebar has a "Docker Scout" section highlighted, along with other options like Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, and Extensions. The main content area features a heading "Advanced image analysis with Docker Scout". It displays a message about using Docker Scout on remote repositories and links to setup instructions. Below this, it says "Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#)". A form allows selecting a "Sample image" (set to "calctest:latest") and viewing "Vulnerabilities" (labeled "Not analyzed"). A prominent blue button labeled "Analyze image" is on the right, with a large blue arrow pointing to it from the bottom right. At the bottom, there's a section titled "How to access Advanced image analysis" with a smaller screenshot of the Docker Desktop interface.

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

Analyze image

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

How to access Advanced image analysis

docker.desktop

Local Desktop Engine

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 left sidebar has a 'PERSONAL' badge and includes links for Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (which is selected and highlighted in grey), and Extensions. The main content area is titled 'Advanced image analysis with Docker Scout'. It features a 'Sample image' dropdown set to 'calctest:latest' and a 'Vulnerabilities' section with a count of 20 (highlighted in orange). A message box says 'Want to use Docker Scout on your remote repositories? Set up your integrations now'. Below this, a note says 'Understand your application's dependencies, analyze the vulnerabilities, and act quickly with suggested remediation options. [Learn more](#) and [upgrade](#)'. At the bottom, it says 'Advanced image analysis can be accessed by viewing any of the images on the [Images view](#)'. The bottom navigation bar shows 'Local Desktop Engine', 'Images' (with a 'Give feedback' link), engine status (Engine running, RAM 1.02 GB, CPU 0.08%, Disk: 21.32 GB used / limit 1006.85 GB), a terminal icon ('Terminal v4.50.0'), and a status bar with a small logo and the number 7343.

Docker scout in Docker Desktop

The screenshot shows the Docker Desktop application window. The left sidebar has a 'PERSONAL' badge and includes links for Ask Gordon (BETA), Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (which is selected and highlighted in grey), and Extensions. The main content area is titled 'Advanced image analysis with Docker Scout'. It shows a sample image dropdown set to 'calctest:latest' with a count of 20 vulnerabilities. A horizontal bar indicates the severity distribution: Critical (1), High (2), Medium (20), and Low (0). Below this, a callout box details the severity levels: Critical, High, Medium, and Low. A note says 'Advanced image analysis can be accessed by viewing any of the images on the left'. At the bottom, there's a summary of the Docker Engine status: 'Engine running', 'Local Desktop Engine', 'Images' (with a 'Give feedback' link), 'RAM 1.02 GB CPU 0.08%', 'Disk: 21.32 GB used (limit 1006.85 GB)', 'Terminal v4.50.0', and a footer with the year '1343'.

docker desktop PERSONAL

Ask Gordon BETA

Containers

Images

Volumes

Kubernetes

Builds

Models

MCP Toolkit BETA

Docker Hub

Docker Scout

Extensions

Search

ctrl+K

?

!

!

!

A

Advanced image analysis with Docker Scout

Want to use Docker Scout on your remote repositories? Set up your integrations now [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

1 2 2 20 0 [View packages and CVEs](#)

Critical

High

Medium

Low

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

How to access Advanced image analysis

docker desktop

Local Desktop Engine

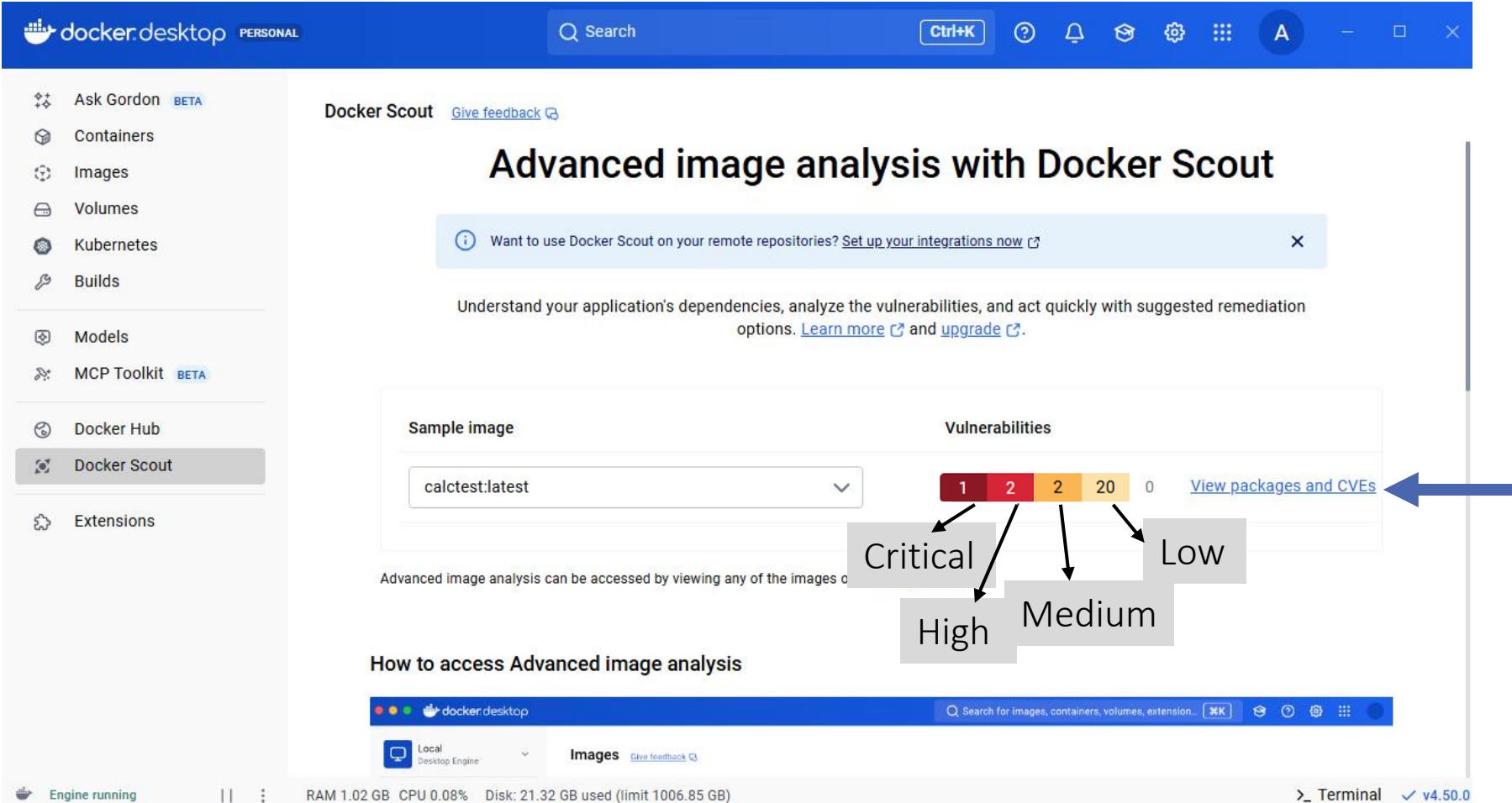
Images Give feedback

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



The screenshot shows the Docker Desktop interface with the Docker Scout tab selected in the sidebar. The main content area displays the title "Advanced image analysis with Docker Scout". A callout diagram highlights the "Vulnerabilities" section, which shows a breakdown of findings: 1 Critical, 2 High, 2 Medium, 20 Low, and 0 Unknown. An arrow points from the "View packages and CVEs" link to this callout. Below the analysis, there's a section titled "How to access Advanced image analysis" with a smaller screenshot of the Docker Desktop interface.

Sample image

Vulnerabilities

1	2	2	20	0
Critical	High	Medium	Low	

calctest:latest

View packages and CVEs

Advanced image analysis can be accessed by viewing any of the images on this page.

How to access Advanced image analysis

docker desktop

Local Desktop Engine

Images

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 shows the Docker Desktop interface with the following details:

- Left Sidebar:** Personal dashboard with links to Ask Gordon (BETA), Containers, Images (selected), Volumes, Kubernetes, Builds, Models, MCP Toolkit (BETA), Docker Hub, Docker Scout (selected), and Extensions.
- Top Bar:** Search bar, Ctrl+K, Help, Notifications, Settings, and a theme switcher.
- Image Details:** Images / calctest:latest. The image **calctest:latest** was created 59 minutes ago and has a size of 205.74 MB. It is analyzed by **docker SCOUT**.
- Layers:** Layers (16) for the image. The top layer is **python:3.12-slim** (version 20) and the bottom layer is **calctest:latest** (version 2).
- Vulnerabilities:** Vulnerabilities (22) are listed. A blue arrow points to the **Fixable** filter checkbox, which is checked. Other filters include **Present** and **Show excepted**.

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	↗	↗
- Bottom Status:** 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.

Vulnerabilities (22) Packages (144) Give feedback ↗

Package or CVE name Fixable Show excepted Reset filters

Package Vulnerabilities

Only show packages that can be upgraded to fix a 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 `microbase` 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.

