

# GitLab Security Scanning

**Subhashis Suara**

**Thematic CERN School of Computing on Security 2025**

Git Service - IT-PW-WA CERN

# Introduction

- Why?





# Introduction

- **Why?**
  - ... why not! 🤖
- **CERN is under constant attack!**
  - ... the need for a strong security in place!
- **GitLab 🐱 is the heart of our Dev process**
  - +17k users 👤👤
  - +170k projects 📦📁
  - +20k pipelines/month ⚙️🔄
- **A Global Security Policy for everyone!**







# Global Security Policy

The  **Global Security Policy** includes:

-  **Secret Detection:** Identifies hardcoded sensitive information, such as **passwords** or **API keys**.
-  **Static Application Security Testing (SAST):** Analyzes source code for known **security vulnerabilities**.



A **Scan Execution Policy** streamlines security scans by defining standardized rules across multiple projects:

-  **Define what scans to run:** (e.g., **Secret Detection** , **Dependency Scanning** ).
-  **Specify when these scans are triggered:** (e.g., for every **commit** or **merge** to default branches).

 Gradually injecting **Global Security Policy** in the organisation starting from the IT Dept.

# Injection of Global Scan Execution Policy

## Automated Approach

-  Scan GitLab Groups at CERN
-  Apply (or not apply if existing) the Global Security Policy
-  Notify Group Owners

# Vulnerability Report

- **Provides information about vulnerabilities and enables collaboration**
- **Contains a cumulative result of all your security scans**
  - ... making it easy for developers to manage vulnerabilities

# Vulnerability Report

## Vulnerability report

+ Submit vulnerability

↑ Export

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. [Learn more.](#)

Development vulnerabilities **1**

Operational vulnerabilities **0**

Container registry vulnerabilities **0**

Security reports last updated 1 month ago [#8461684](#)





<b>Critical</b>	<b>High</b>	<b>Medium</b>	<b>Low</b>	<b>Info</b>	<b>Unknown</b>
1	0	0	0	0	0

Group by: None

Status || Needs triage, Confirmed X Activity || Still detected X

<input type="checkbox"/> Detected	Status	Severity ↓	Description	Identifier	Tool	Activity
<input type="checkbox"/>	2024-11-06	<b>Critical</b>	Improper control of generation of code ('Code Injection') <a href="#">cal.php:16</a>	A1:2017 - Injection + 4 more	SAST	

# Vulnerability Report

- Provides information about vulnerabilities and enables collaboration
- Contains a cumulative result of all your security scans
  - ... making it easy for developers to manage vulnerabilities
- What to do with findings?
  - Check the description of the vulnerabilities to get further information
    - See whether they are remediated or they are false positives
  - Change the status (from **Needs Triage** to any of **Dismiss**, **Confirm**, **Resolve**)
  - If appropriate, create an issue to track this further ...
  - ... **and fix it!** 
- As a curiosity, the GitLab<sup>Duo</sup> AI-suite can:
  - Explain the vulnerabilities to provide insights about it 
  - Resolve the vulnerabilities, by generating a merge request with a suggested solution! 
    - ... in two clicks 

**Thank you!**

**... extra information!**



# Center of Internet Security (CIS) Benchmark

- **How to use the GitLab CIS Benchmark**

- Install the *gitlabcis* tool with:

```
pip install gitlabcis
```

- Or run a container with:

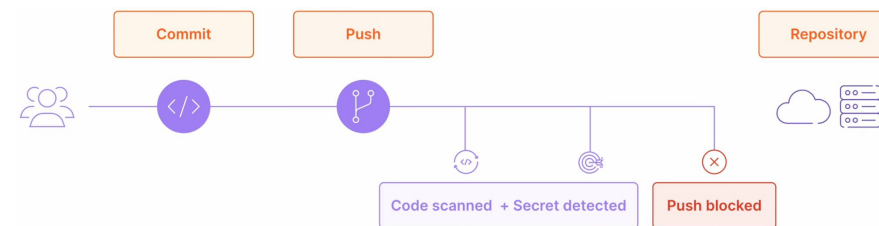
```
docker run -it --rm \
-v "$(pwd)":/gitlabcis \
-v "$(pwd)":/gitlabcis \
-w /gitlabcis \
python:3.10 sh -c "
  pip install requests gitlabcis &&
  sed -i 's/%Y-%m-%dT%H:%M:%S\\.\.\%fZ/%Y-%m-%dT%H:%M:%S\\.\.\%fz/g'
/usr/local/lib/python*/site-packages/gitlabcis/benchmarks/source_code_1/repository_management_1_2.py && bash
"

gitlabcis https://gitlab.cern.ch/<group>/<project> --token <PAT read_api scope>
```

- **ref: [GitLab CIS Benchmark](#)**

# GitLab Secret Push Protection

- Available in our GitLab instance (*as Beta*)!
  - Start using [Secret push protection](#)!
- What are the main differences?
  - (Pipeline) Secret Detection (from the scan execution policies explained before)
    - Scans committed files and identify leaked secrets (i.e., after `git push` and via triggering a pipeline).
  - Secret Push Protection
    - Secret detection when pushing files (i.e., blocks `git push`).
    - Developers will be prevented from pushing code containing secrets.



- Best of both worlds

- Combining both features maximizes secret detection throughout the development process.

# References

- [Secure your application](#)
- [Scan execution policies](#)
- [Security policy project](#)
- [Detected secrets](#)
- [Vulnerability report](#)
- [GitLab CIS Benchmark](#)