



# The Tyranny of Zero:

Is the race to the bottom worth it in vulnerability management?



**Bryan Ross**  
Field CTO  
GitLab





We take your  
privacy and  
security very  
seriously...

Every CEO

At any company after a breach





BRYAN ROSS

# Introduction

Field CTO, GitLab

@BryanRossUK



2022 ● Executive Advisor, VMware Tanzu

2020 ● Head of Technology Products, Sky

2011 ● Senior Consultant, Sopra Group

2005 ● Co-founder, XPT Solutions



# Million-Euro Race to Detect and Respond

**€4.13M**

The average cost of a data breach

**277 days**

taken to identify and contain a data breach

Average cost saving from high levels of adoption of DevSecOps tooling

**€1.56M**

Organisations with extensive use of AI and automation are significantly faster

**108 days**

The effect of complexity on the cost of a data breach

**31.6%**



# Security and compliance are business-critical

Markets are competitive

66%

of companies are releasing 2x fast or faster

*\*GitLab DevSecOps Report*

Apps are causing breaches

8/10

of the top breaches last year stem from attacks on the application layer

*\*CrowdStrike State of Application Security Report*

Compliance costs increase

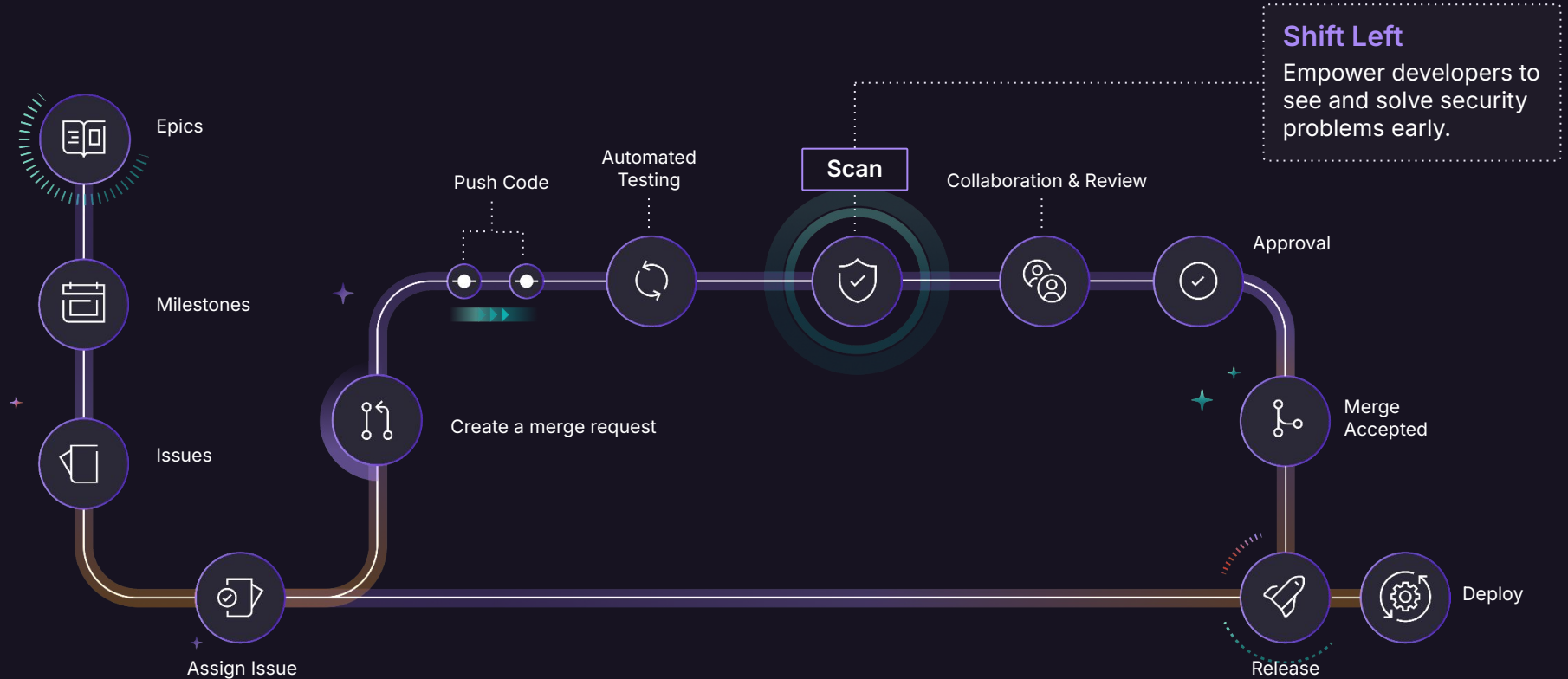
2.71x

Non-compliance costs more than early adherence

*\*Deloitte Cost of Compliance*



# Democratising Security



# Complete Coverage

Automated security scanning across all stages of the SDLC.

## SAST

Scan application source code and binaries

## DAST

Analyze web applications for runtime threats

## Dependency Scanning

Analyze external dependencies

## IaC Scanning

Scan infrastructure files for insecure practices

## Secret Detection

Check for credentials in code commits

## Container Scanning

Look for known security vulnerabilities

## API Security

Analyze APIs for runtime vulnerabilities

## Fuzz Testing

Use malformed data to measure app stability

## Bring your own Tool

Easily integrate your existing security tools

## Extract More Value

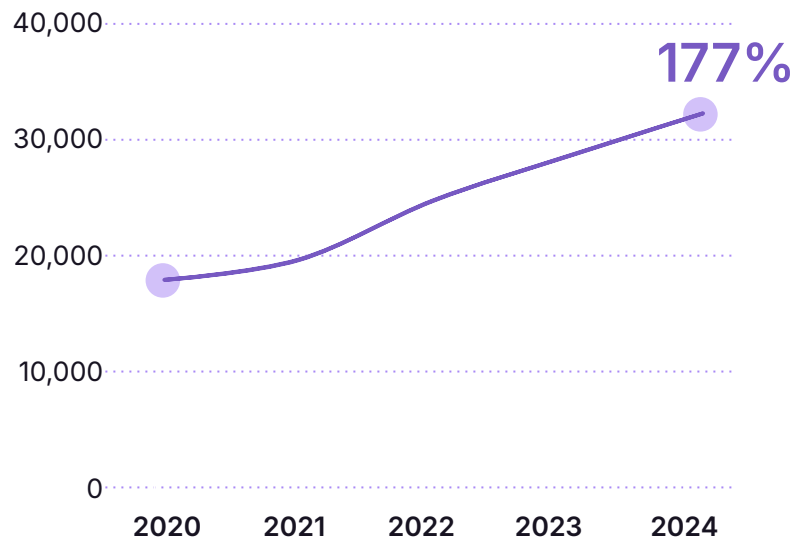
Use GitLabs policies, reporting, and guardrails



# The tyranny of zero

New vulnerabilities are found at an increasing rate that organisations cannot keep pace with.

New CVEs reported each year



Source: National Vulnerability Database, NIST





# Security

Improve Application Security by addressing vulnerabilities before they're shipped to production.



## Shift left security

Integrate security directly into the software development lifecycle, empowering developers to write secure code from the start, without compromising on velocity.



## Detection accuracy

Deliver high quality scan results to minimize the noise of false positives so security and development teams can prioritize critical vulnerabilities.



## Faster remediation

Streamline the process for finding, triaging, and fixing vulnerabilities by providing actionable insights and automating workflows, so developers can address issues swiftly with minimal friction.





*This presentation and linked pages contain information related to upcoming products, features, and functionality.*

*It is important to note that the information presented is for informational purposes only. Please do not rely on this information for purchasing or planning purposes.*

*As with all projects, the items mentioned in this presentation and linked pages are subject to change or delay. The development, release, and timing of any products, features, or functionality remain at the sole discretion of GitLab Inc.*



+



# Advanced SAST

Provides cross-function and cross-file taint analysis for popular languages:

- ✓ C#
- ✓ Go
- ✓ Java / JSP
- ✓ JavaScript / TypeScript
- ✓ Python
- ✓ Ruby



Loved by developers, trusted by app sec teams



Best in class scanning technology



More accurate and actionable security findings



Reduced alert fatigue with less false-positives



Focus on vulnerabilities that are exploitable



Detection accuracy

# Code Flow

- ✓ Cross-function, cross-file taint analysis
- ✓ Traces input data through all assignments, manipulation, and sanitization.
- ✓ Contextual remediation and fewer false positives
- ✓ Proprietary detection engine with rules informed by in-house security research

## Improper neutralization of special elements used in an OS Command ('OS Command Injection')

Details Code flow

### Steps

5 steps across 2 files

#### server/api.py

1	source	name = request.form['name']	10
2		name = request.form['name']	10
3		get_api_data_remote(name)	12

#### server/backend.py

4		def get_api_data_remote(param: str) -> None:	3
5	sink	os.system(param)	5

```
server/api.py
7
8
9
10
11
12
13
14
15
@app.route('/api/process')
def api_endpoint():
    1 name = request.form['name']
    get_api_data_local(name)
    3 get_api_data_remote(name)

def get_api_data_local(param: str) -> None:
    # This function is called with user input.
```

```
server/backend.py
1
2
3
4
5
6
7
8
import os

4 def get_api_data_remote(param: str) -> None:
    # This is the sink for a true-positive vulnerability.
    5 os.system(param)

def canned_get_api_data(param: str) -> None:
    # This code isn't used to process user input, so this isn't
```



Detection accuracy

# Static Reachability

- ✓ Reduce CVE noise by identifying dependencies actively used in code
- ✓ Improves Dependency Scanning; based on GitLab Advanced SAST
- ✓ Today: Experiment for Java & Python  
Next: UI support, additional languages

```
1 {
2   "bomFormat": "CycloneDX",
3   "specVersion": "1.4",
4   "serialNumber": "urn:uuid:929a692e-ac99-4a96-9cac-d86302d91991",
5   "version": 1,
6   "metadata": {
7     "timestamp": "2024-09-23T09:19:42Z",
8     "tools": [
9       {
10        "vendor": "GitLab",
11        "name": "Gemnasium",
12        "version": "5.7.0"
13      }
14    ],
15    "authors": [
16      {
17        "name": "GitLab",
18        "email": "support@gitlab.com"
19      }
20    ],
21    {
22      "name": "requests",
23      "version": "2.19.1",
24      "purl": "pkg:pypi/requests@2.19.1",
25      "type": "library",
26      "bom-ref": "pkg:pypi/requests@2.19.1",
27      "properties": [
28        {
29          "name": "gitlab:dependency_scanning_component:reachability",
30          "value": "in_use"
31        }
32      ]
33    }
34  }
```

Experimental





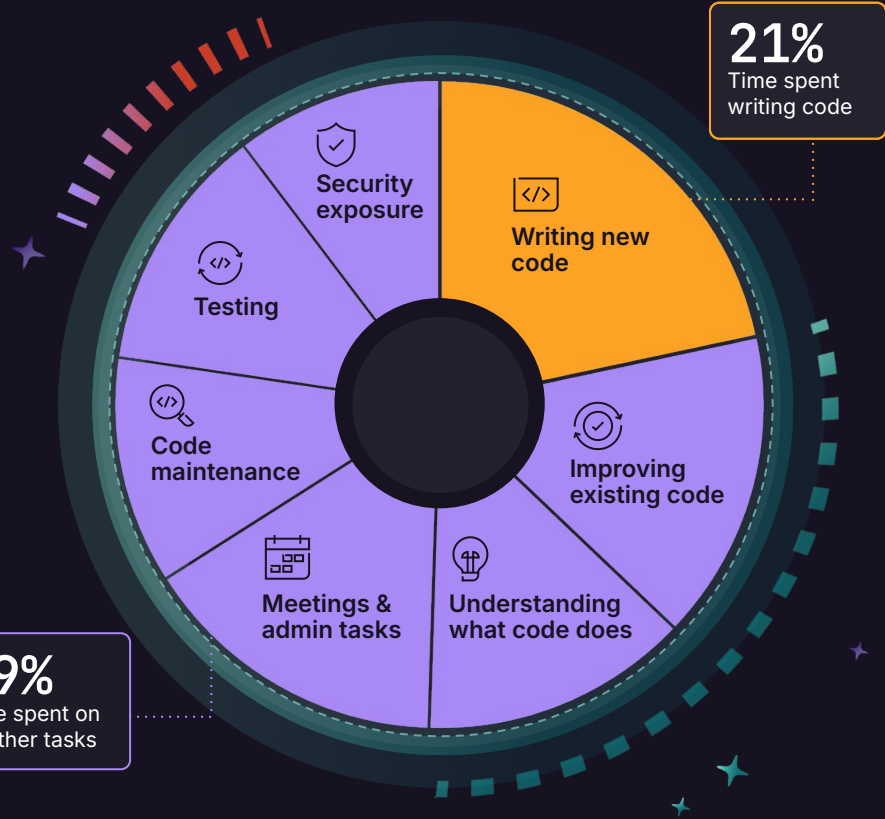
# Artificial Intelligence

# AI is central to GitLab's DevSecOps platform

**Throughout the Software Delivery Lifecycle**  
Improve DevSecOps workflow efficiency by **10x** by applying AI assisted workflows to all teams involved in delivering software value

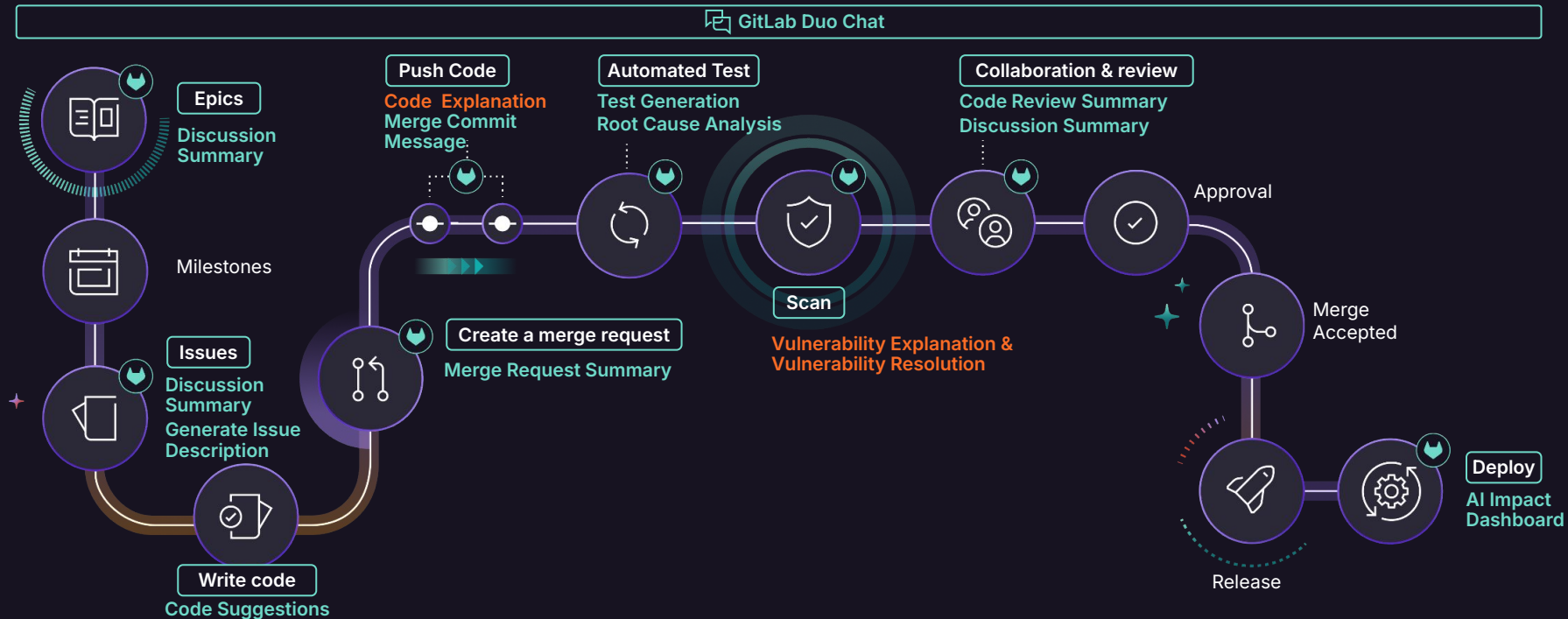
**Privacy-First, Enterprise-Grade**  
Lead with a privacy-first approach allowing enterprises and regulated organizations to adopt AI assisted workflows

**Single Application**  
Leverage the benefits of GitLab's single application to deliver more software faster, enabling executive visibility across value streams and preventing context switching





# The most comprehensive AI-powered DevSecOps platform



# Vulnerability explanation

## Continuous security improvements

- ✔ Helps developers remediate vulnerabilities more efficiently
- ✔ Helps uplevel developers to improve their skills with AI generated security training
- ✔ Enables developers to write more secure code

The screenshot displays a GitLab interface with a vulnerability report and a Duo Chat window. The report, titled 'Improper Neutralization of Special Element: Command ('SQL Injection')', is marked as 'Needs triage' and 'Critical'. It describes a SQL injection vulnerability in the file 'WebGoat/Code/SQLiteRoleProvider.cs:427-429'. The Duo Chat window provides a 'Vulnerability Explanation' and 'Exploiting the Vulnerability' section, showing a malicious payload: `string maliciousRoleName = " OR '1'='1"; cmd.CommandText = "SELECT u.Username FROM user_table WHERE (r.LoweredRoleName = $RoleName) AND (r.ApplicationId = " + uir.UserId + " ON u.UserId = uir.UserId INNER JOIN " + ROLE_TB_NAME + " ON uir.LoweredRoleName = $RoleName)";`

GitLab Duo Chat

Responses generated by AI

### Vulnerability Explanation

The code in `SQLiteRoleProvider.cs` is vulnerable to SQL injection due to the improper neutralization of user-supplied input in the SQL query. The `$RoleName` and `$ApplicationId` parameters are directly concatenated into the SQL query, which can allow an attacker to inject malicious SQL code and potentially gain unauthorized access to the database.

### Exploiting the Vulnerability

An attacker could exploit this vulnerability by crafting a malicious `$RoleName` or `$ApplicationId` parameter that contains SQL injection payload. For example, the attacker could inject the following payload into the `$RoleName` parameter:

```
string maliciousRoleName = " OR '1'='1";
cmd.CommandText = "SELECT u.Username FROM
+ " uir ON u.UserId = uir.UserId INNER JOIN
+ " WHERE (r.LoweredRoleName = $RoleName) AND (r.ApplicationId = " +
```

This payload would result in the following SQL query:

```
SELECT u.Username FROM user_table
```

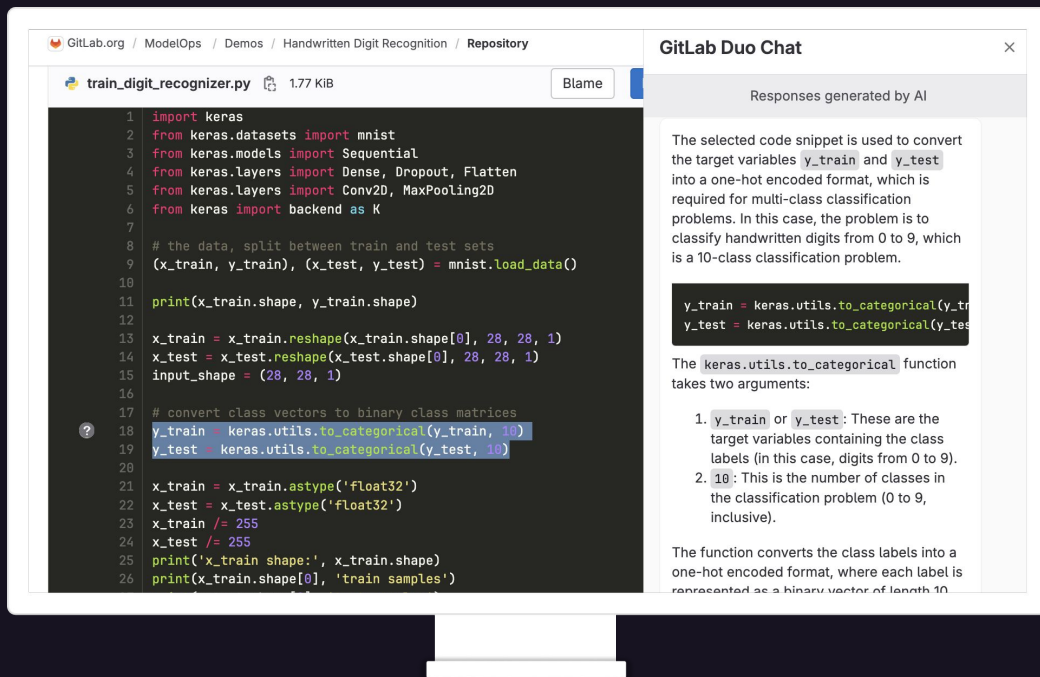
The `" OR '1'='1'` part of the payload would always evaluate to `TRUE`, effectively bypassing the role name check.



# Code Explanation

## Inline code explanations

- ✓ Enables developers to quickly understand code
- ✓ Uplevels developers through continuous learning



The screenshot displays a GitLab Duo Chat window. On the left, a code editor shows a Python script named `train_digit_recognizer.py` (1.77 KiB). The code imports Keras and processes MNIST data. A tooltip is visible over the `keras.utils.to_categorical` function call on line 18. On the right, the chat interface shows the AI's response to the tooltip query.

```
1 import keras
2 from keras.datasets import mnist
3 from keras.models import Sequential
4 from keras.layers import Dense, Dropout, Flatten
5 from keras.layers import Conv2D, MaxPooling2D
6 from keras import backend as K
7
8 # the data, split between train and test sets
9 (x_train, y_train), (x_test, y_test) = mnist.load_data()
10
11 print(x_train.shape, y_train.shape)
12
13 x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
14 x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
15 input_shape = (28, 28, 1)
16
17 # convert class vectors to binary class matrices
18 y_train = keras.utils.to_categorical(y_train, 10)
19 y_test = keras.utils.to_categorical(y_test, 10)
20
21 x_train = x_train.astype('float32')
22 x_test = x_test.astype('float32')
23 x_train /= 255
24 x_test /= 255
25 print('x_train shape:', x_train.shape)
26 print(x_train.shape[0], 'train samples')
```

**GitLab Duo Chat**

Responses generated by AI

The selected code snippet is used to convert the target variables `y_train` and `y_test` into a one-hot encoded format, which is required for multi-class classification problems. In this case, the problem is to classify handwritten digits from 0 to 9, which is a 10-class classification problem.

```
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

The `keras.utils.to_categorical` function takes two arguments:

1. `y_train` or `y_test`: These are the target variables containing the class labels (in this case, digits from 0 to 9).
2. `10`: This is the number of classes in the classification problem (0 to 9, inclusive).

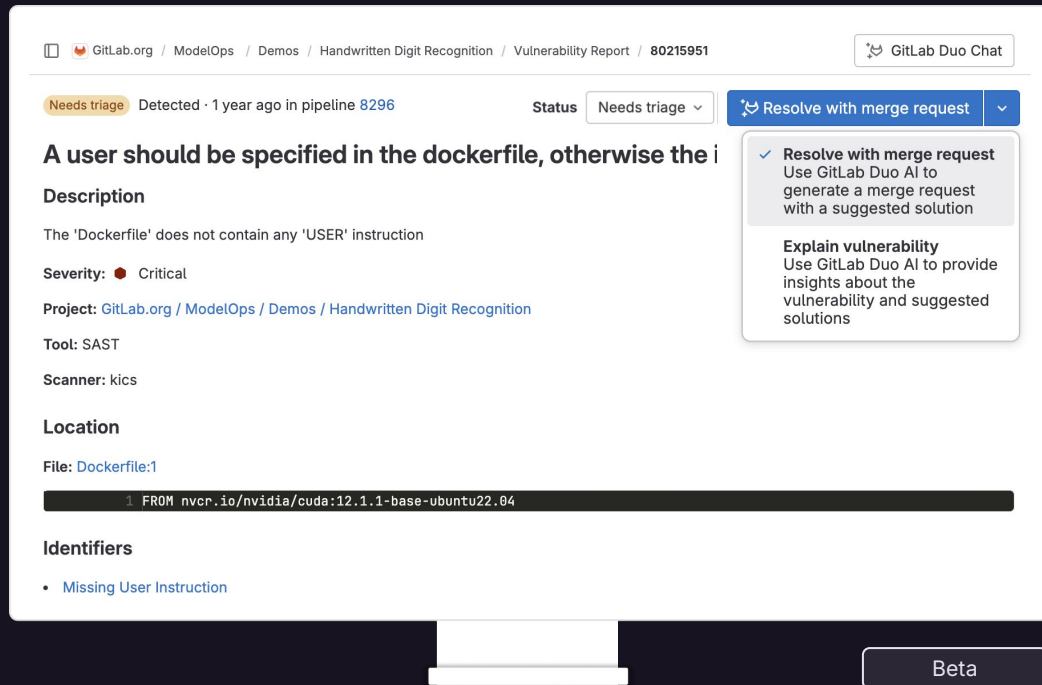
The function converts the class labels into a one-hot encoded format, where each label is represented as a binary vector of length 10.



# Vulnerability resolution

## Continuous security improvements

- ✓ Helps developers remediate vulnerabilities more efficiently
- ✓ Get a jump start on how to fix a found vulnerability with auto-generated merge requests to mitigate security risks



GitLab.org / ModelOps / Demos / Handwritten Digit Recognition / Vulnerability Report / 80215951 GitLab Duo Chat

**Needs triage** Detected · 1 year ago in pipeline 8296 Status: Needs triage Resolve with merge request

### A user should be specified in the dockerfile, otherwise the i

#### Description

The 'Dockerfile' does not contain any 'USER' instruction

**Severity:** ● Critical

**Project:** [GitLab.org / ModelOps / Demos / Handwritten Digit Recognition](#)

**Tool:** SAST

**Scanner:** kics

#### Location

**File:** [Dockerfile:1](#)

```
1 FROM nvcr.io/nvidia/cuda:12.1.1-base-ubuntu22.04
```

#### Identifiers

- [Missing User Instruction](#)

✓ **Resolve with merge request**  
Use GitLab Duo AI to generate a merge request with a suggested solution

**Explain vulnerability**  
Use GitLab Duo AI to provide insights about the vulnerability and suggested solutions

Beta



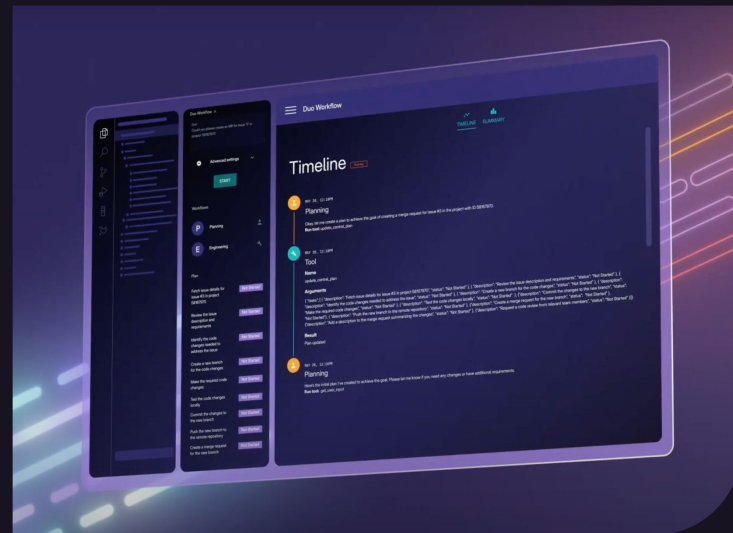
# GitLab Duo Workflow

The future of AI-driven software development

Fully autonomous AI agent that proactively support developers as they develop, deploy, and secure software. Transitioning from reactive, prompt-based AI to proactive, autonomous AI that drives software development and delivery.



GitLab Duo Workflow video



# Let's start the conversation

Having zero vulnerabilities is an unachievable standard in large / complex environments.

Prioritise by risk, not by numbers.

Build an environment that empowers everyone to collaborate and contribute towards security.

 <https://bryanross.me/>



**@BryanRossUK**

Driving transformation  
in the FTSE100 🚀

