



Vulnerability Management and Remediation

Key Takeaways

All rights reserved to nnSoftware GmbH

No part of this publication may be reproduced, copied, transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of nnSoftware GmbH

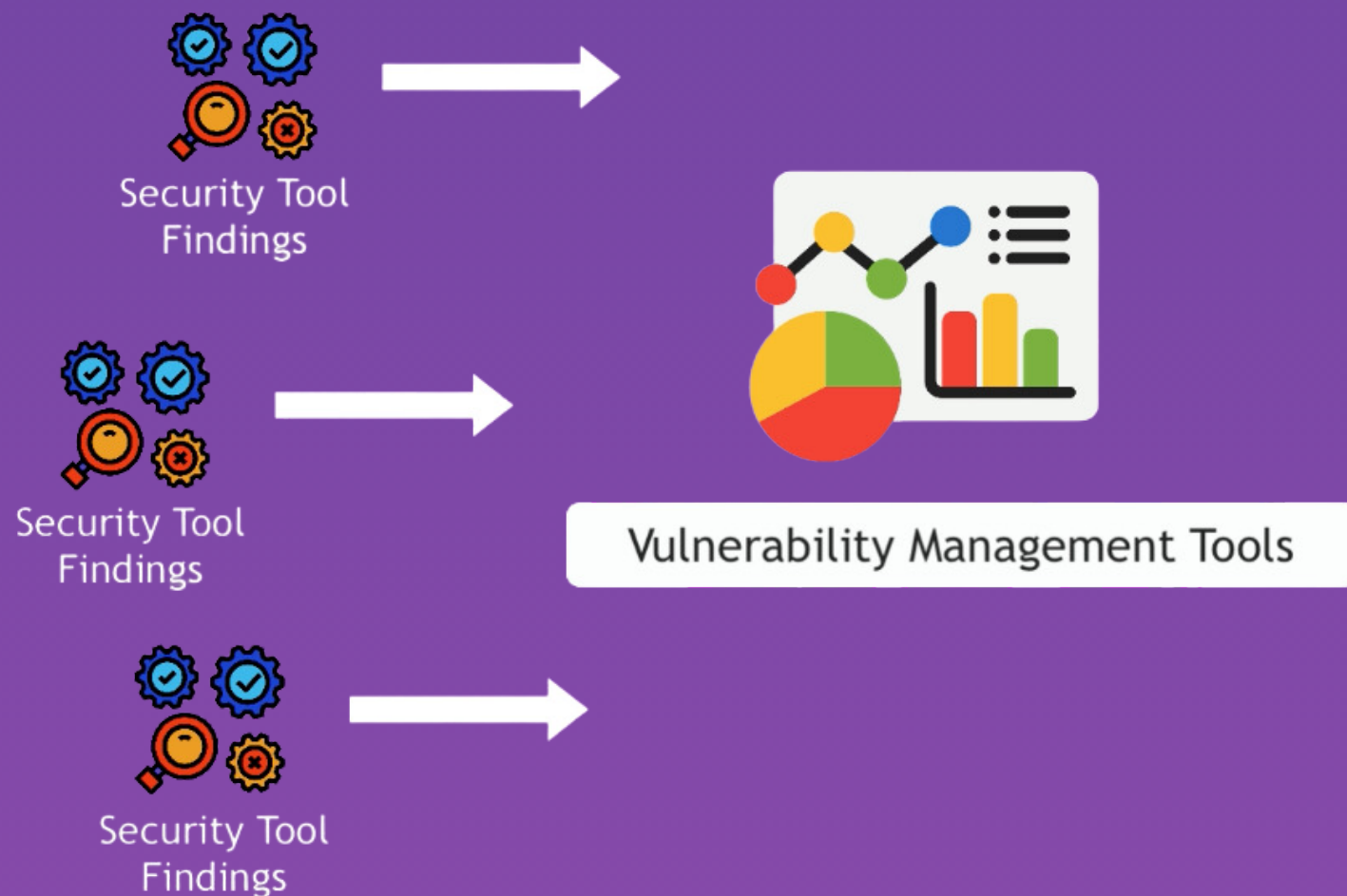
About TechWorld with Nana

TechWorld with Nana is an established name in the DevOps and Cloud industry, and it stands for the quality trainings helping 1,000s of engineers acquire the most in-demand skills in this field.



Our mission is enable individual engineers as well as companies to take advantage of the recent developments in Cloud and DevOps fields, to use technologies and concepts in order to create efficient, automated, streamlined DevSecOps processes in organisations.

Vulnerability Management Tool



Why and what they are able to do

- ✓ 1 central place - Centrally manage vulnerability findings of different tools
- ✓ UI instead of logs in CI tool
- ✓ Enriches and refines vulnerability data
- ✓ Triage vulnerabilities and push findings to other systems

How it works

DefectDojo

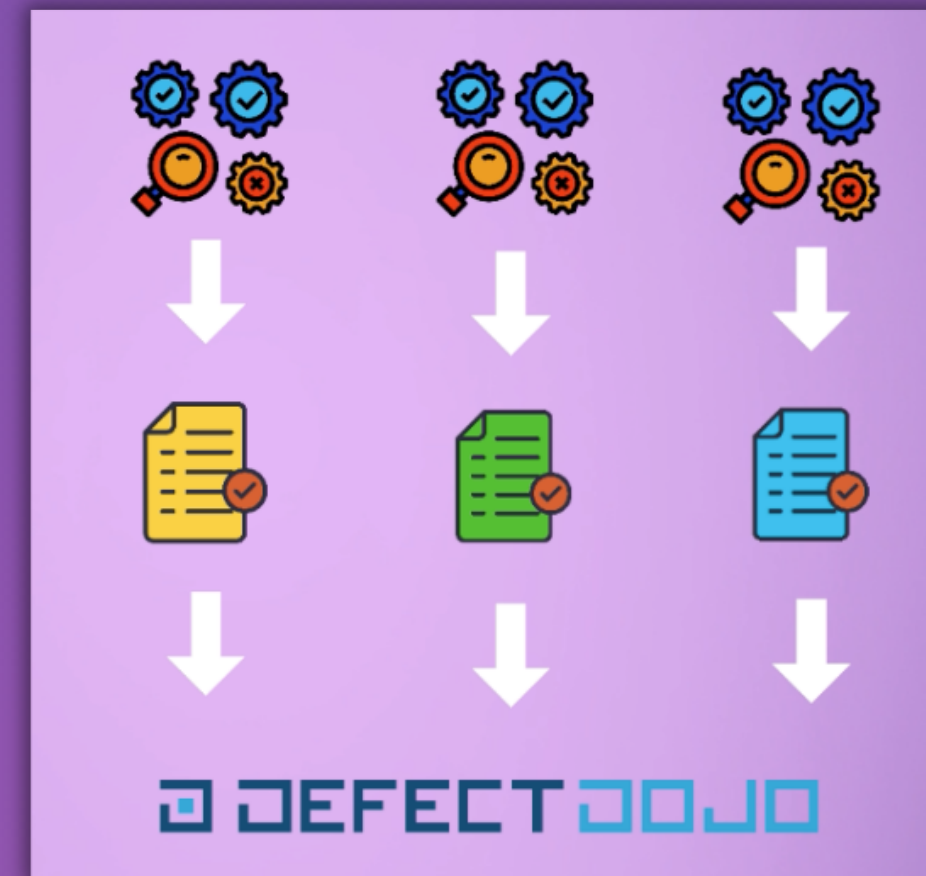
- One open source vulnerability management tool is



How it works



1. Configure each security tool to generate security findings report file
 - a. Produce files in a format that visualization tool can consume
2. Run DefectDojo
3. Feed report files to DefectDojo by importing it



Generate Scan Report Files

Producing Scan Reports in GitLab CI


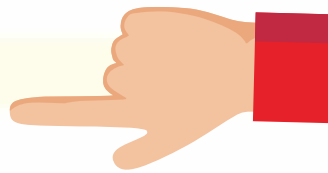


- Jobs can output files and directories
- Job artifacts can be downloaded by using the GitLab UI or the API
- We can use “job artifact” concept to export and save scan report files

Example for GitLeaks Job
















- Add parameters to tell GitLeaks tool to save output in *gitleaks.json* file
- Add “artifacts” block and indicate that we always want to produce that file no matter if job succeeds or fails



```
41 gitleaks:
42   stage: test
43   image:
44     name: zricethezav/gitleaks
45     entrypoint: [""]
46   script:
47     - gitleaks detect --verbose --source . -f json -r gitleaks.json
48   allow_failure: true
49   artifacts:
50     when: always
51     paths:
52       - gitleaks.json
53
```



Generate Scan Report Files

Report artifacts is available for download

Status	Pipeline	Triggerer	Stages
 warning 00:14:02 just now	Update .gitlab-ci.yml file #951656987 P main a128953c latest		  
 warning 00:13:50 1 hour ago	Update .gitlab-ci.yml file #951558409 P main 7847b91f		  
 warning 00:13:54 1 hour ago	Update .gitlab-ci.yml file #951529928 P main 1112c625		  

Download artifacts

gitleaks:archive
njsscan:archive
semgrep:archive

Time to import it in DefectDojo

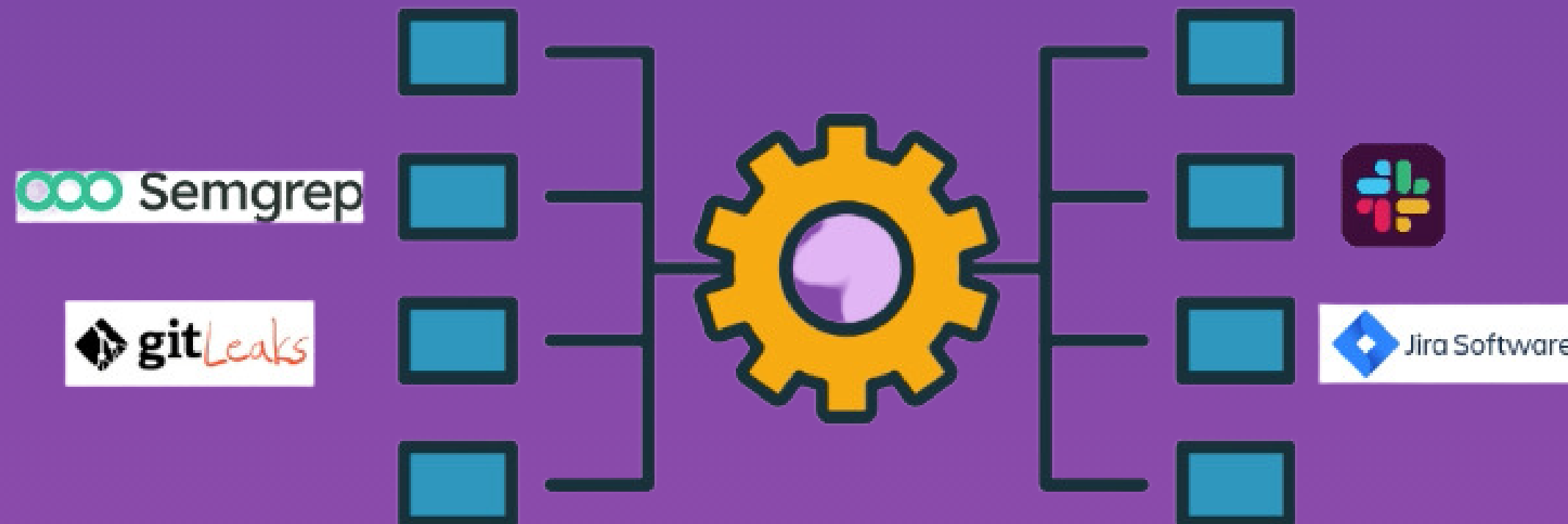


Introduction to DefectDojo

Introduction to DefectDojo

- ✓ Serves as an aggregator and provides a unified and streamlined view for security tools
- ✓ Smart features to enhance and tune the results (merge findings, remember false positives, distill duplicates)
- ✓ Bidirectional integration with Jira, Notifications, Google Sheets synchronization etc.

Supports many
different tools



Push findings
to other tools

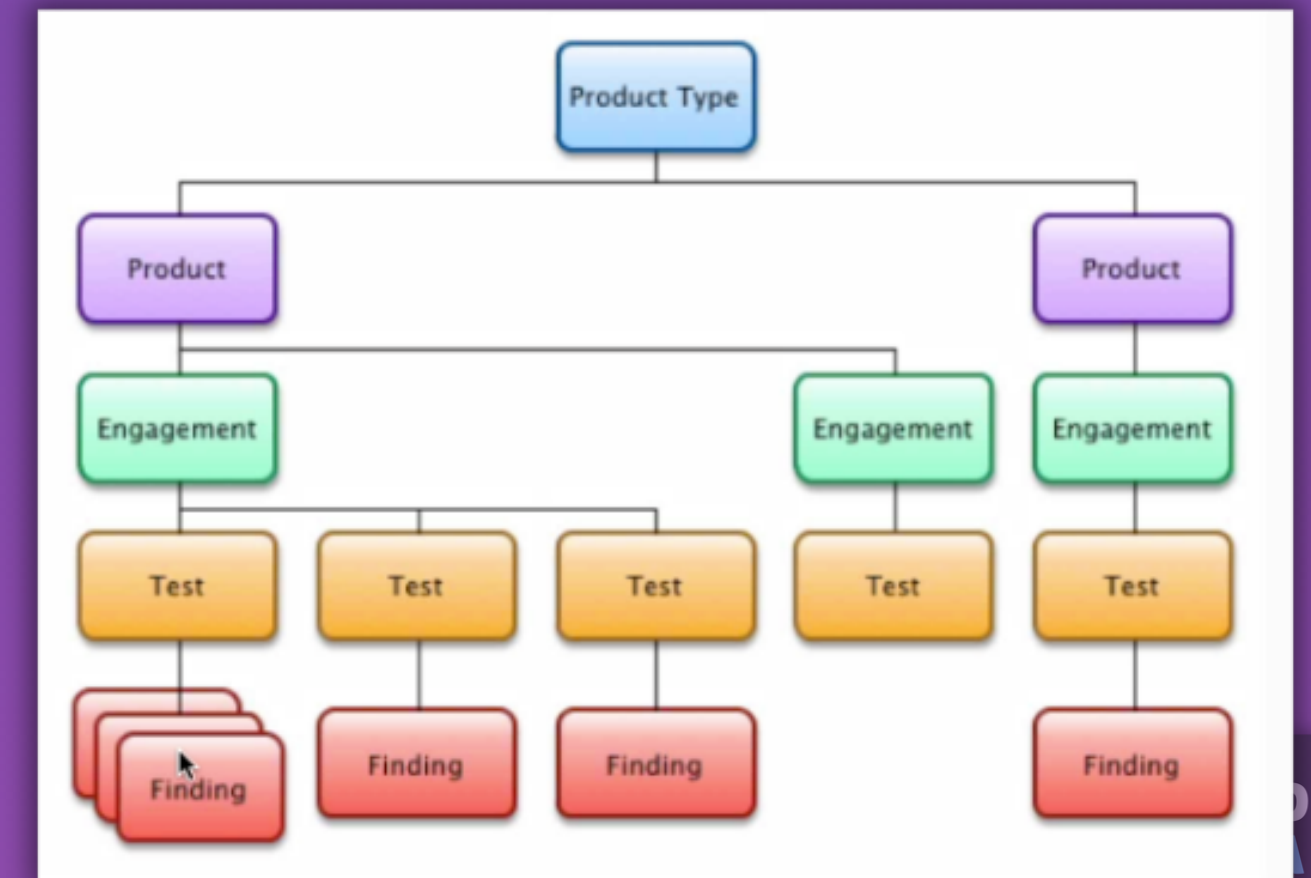
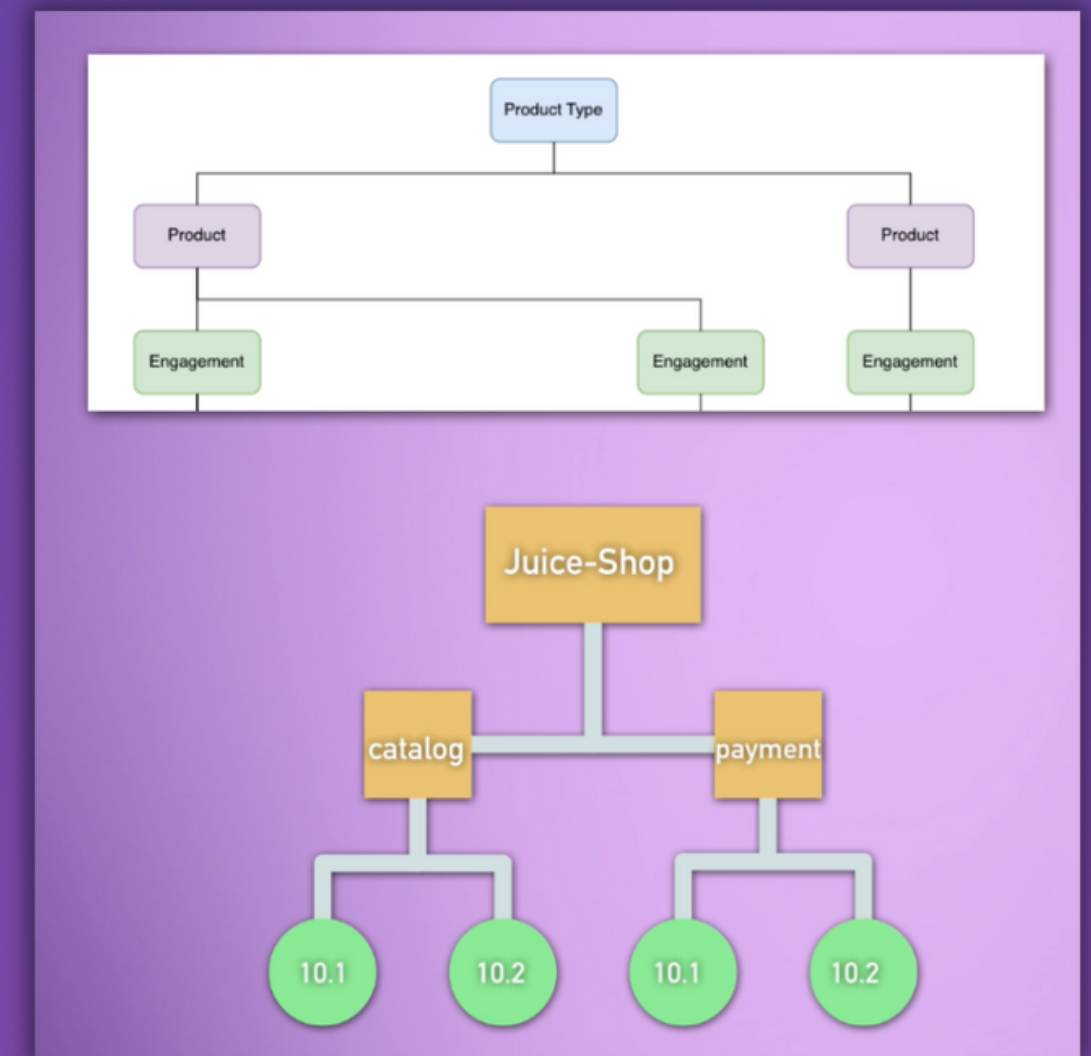
Introduction to DefectDojo

Product:Engagement model

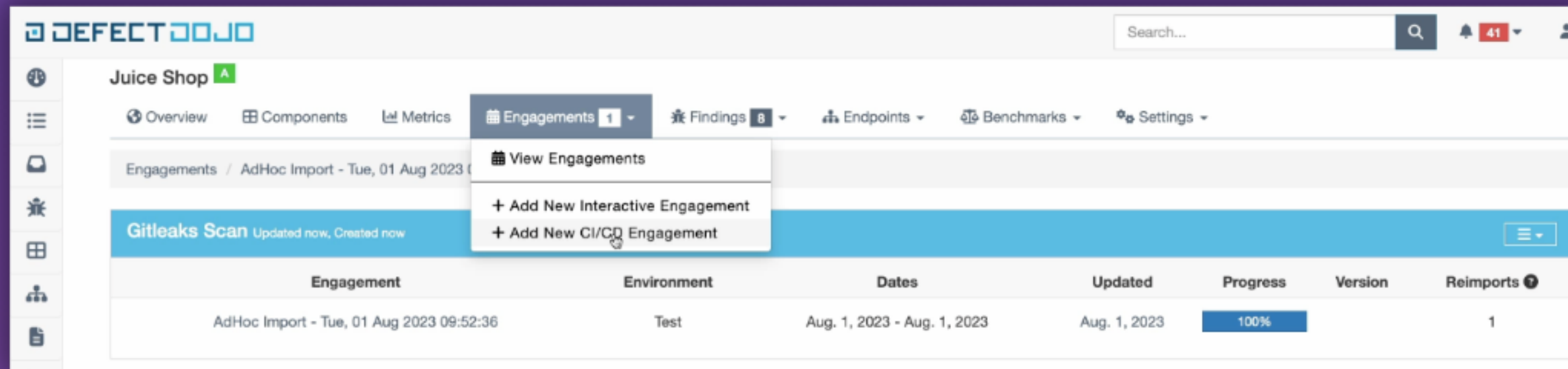
- Enables traceability among multiple projects / test cycles and allows for fine-grained reporting

DefectDojo Structure

- Working in DefectDojo starts with a Product Type
- Each Product Type can have one or more Products
- Each Product can have one or more Engagements
- Each Engagement can have one or more Tests
- Each Test can have one or more Findings



Import Findings in DefectDojo



2 Types of Engagements

Interactive Engagement:

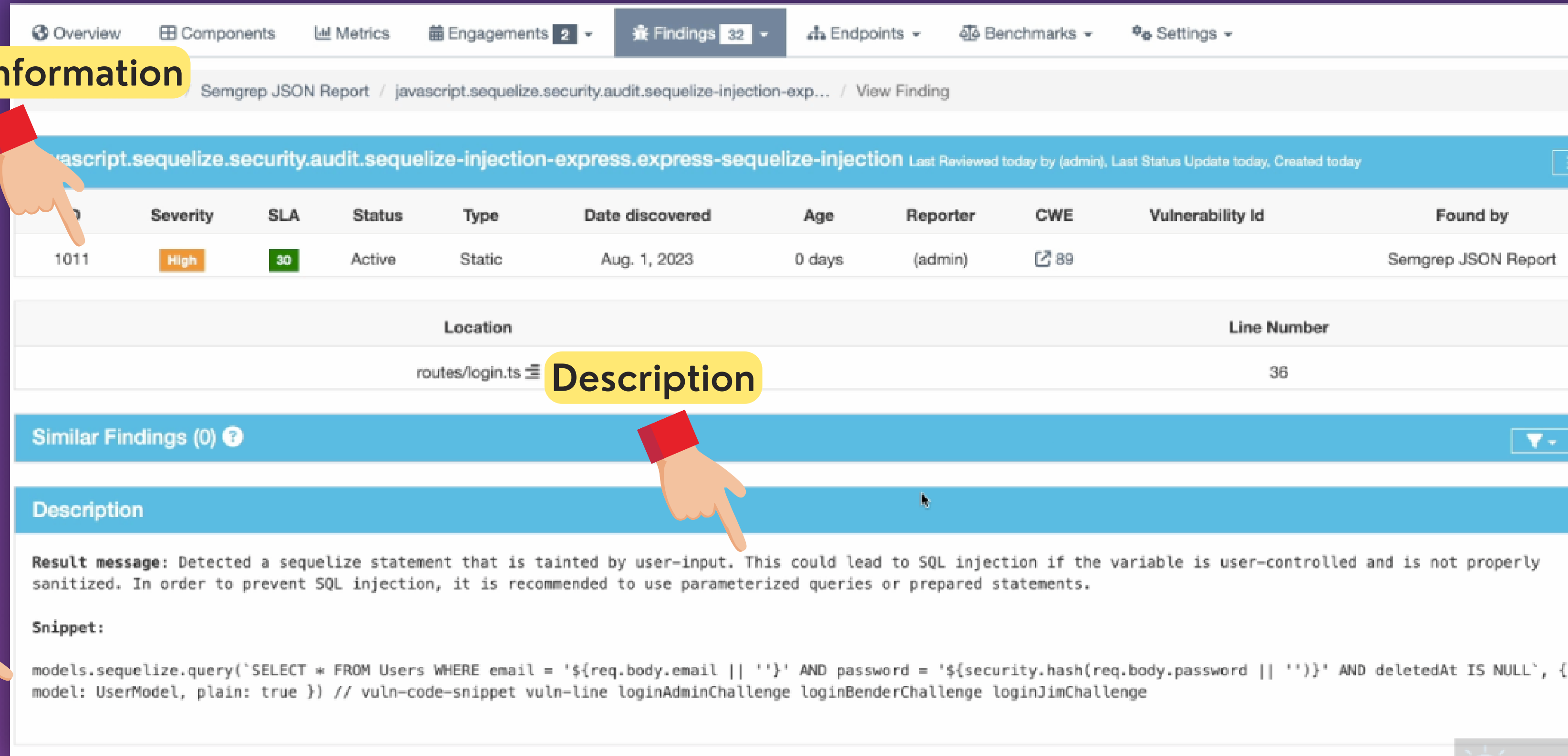
- Findings are uploaded by the engineer via UI

CI/CD Engagement:

- For automated integration with a CI/CD pipeline

Analyze Findings

Severity Information



The screenshot shows a security dashboard with a top navigation bar containing links for Overview, Components, Metrics, Engagements (2), Findings (32), Endpoints, Benchmarks, and Settings. The main content area displays a finding titled 'javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection'. A table lists finding details: ID 1011, Severity High, SLA 30, Status Active, Type Static, Date discovered Aug. 1, 2023, Age 0 days, Reporter (admin), CWE 89, Vulnerability Id, and Found by Semgrep JSON Report. Below the table, the Location is 'routes/login.ts' and the Line Number is 36. A 'Description' section contains a 'Result message' about a tainted sequelize statement and a 'Snippet' of code. A 'Similar Findings (0)' section is also present. Annotations include a hand pointing to the finding ID 1011, another pointing to the 'Description' header, and a third pointing to the code snippet.

ID	Severity	SLA	Status	Type	Date discovered	Age	Reporter	CWE	Vulnerability Id	Found by
1011	High	30	Active	Static	Aug. 1, 2023	0 days	(admin)	89		Semgrep JSON Report

Location

routes/login.ts

Line Number

36

Description

Result message: Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.

Snippet:

```
models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}' AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, {  
  model: UserModel, plain: true }) // vuln-code-snippet vuln-line loginAdminChallenge loginBenderChallenge loginJimChallenge
```

Similar Findings (0)


Exact Code
Snippet that
contains
vulnerability

Analyze Findings

CWE Information

CWE = Common Weakness Enumeration

- Community-developed list of common software and hardware weakness types
- More detailed description of the issue compared to OWASP
 - What causes it
 - How it can be fixed
- Express vulnerability using numeric information



Severity	Name	CWE	Vul Id
High	javascript.sequelize.security.audit.sequelize-injection-exp... <code></></code>	89	
High	javascript.sequelize.security.audit.sequelize-injection-exp... <code></></code>	89	

CWE Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (4.12)

Home | About | CWE List | Mapping | Top-N Lists | Community | News | Search

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Weakness ID: 89
Abstraction: Base
Structure: Simple

View customized information:

Description

The product constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

Extended Description

Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or product package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	✓	943	Improper Neutralization of Special Elements in Data Query Logic
ParentOf	✓	564	SQL Injection: Hibernate
CanFollow	✓	456	Missing Initialization of a Variable

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	✓	137	Data Neutralization Issues

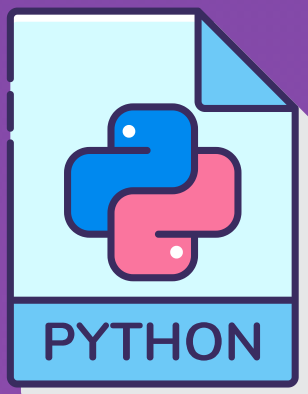


Upload Findings automatically

Automatically upload findings to DefectDojo

DevOps is all about automation

- We don't want to manually upload report files. Pipeline may run multiple times per day
- It's essential to configure integration with DefectDojo to upload the scans automatically

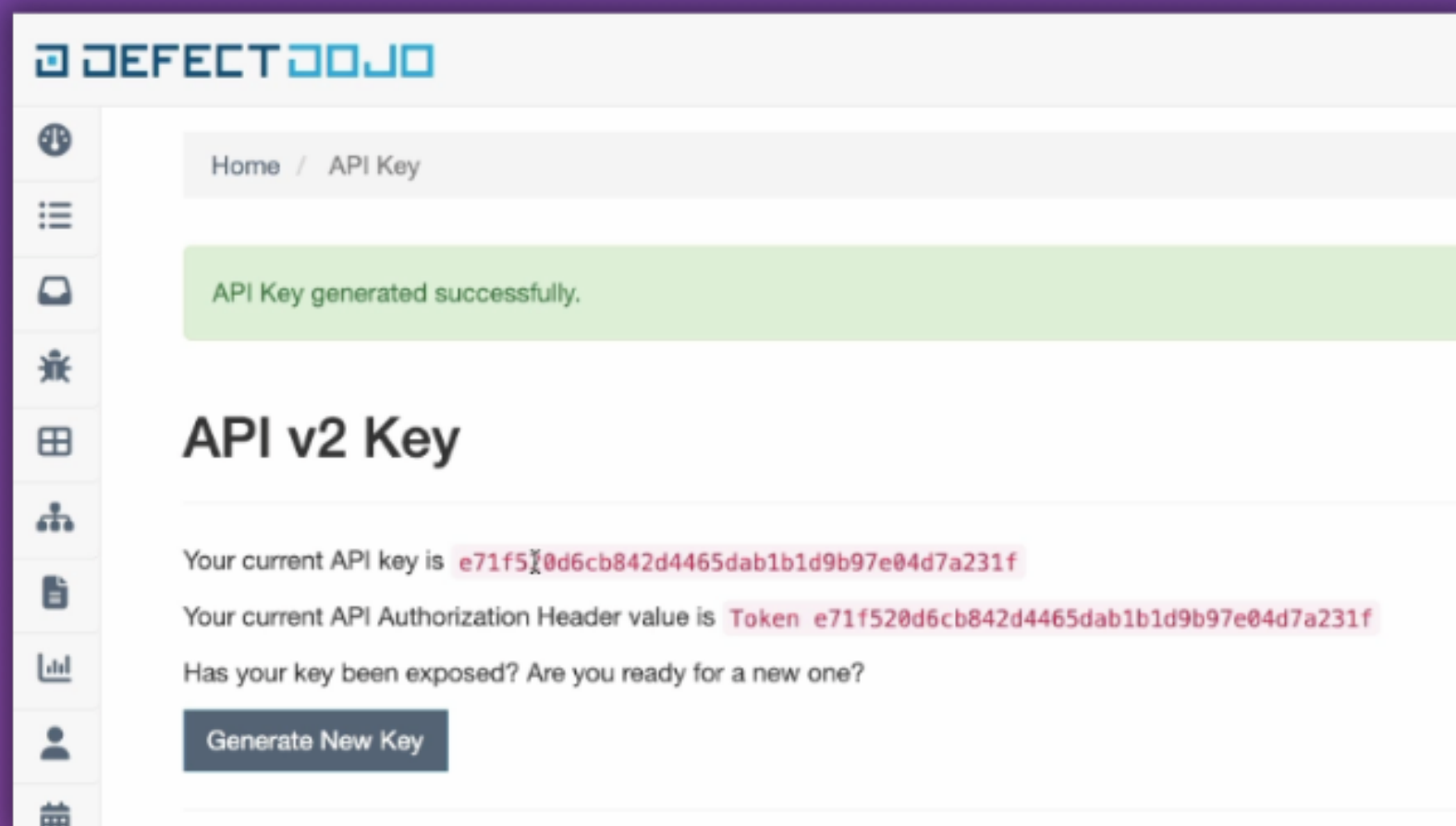


For that, we can use Python or any other programming language

Automatically upload findings to DefectDojo

1 - Authentication

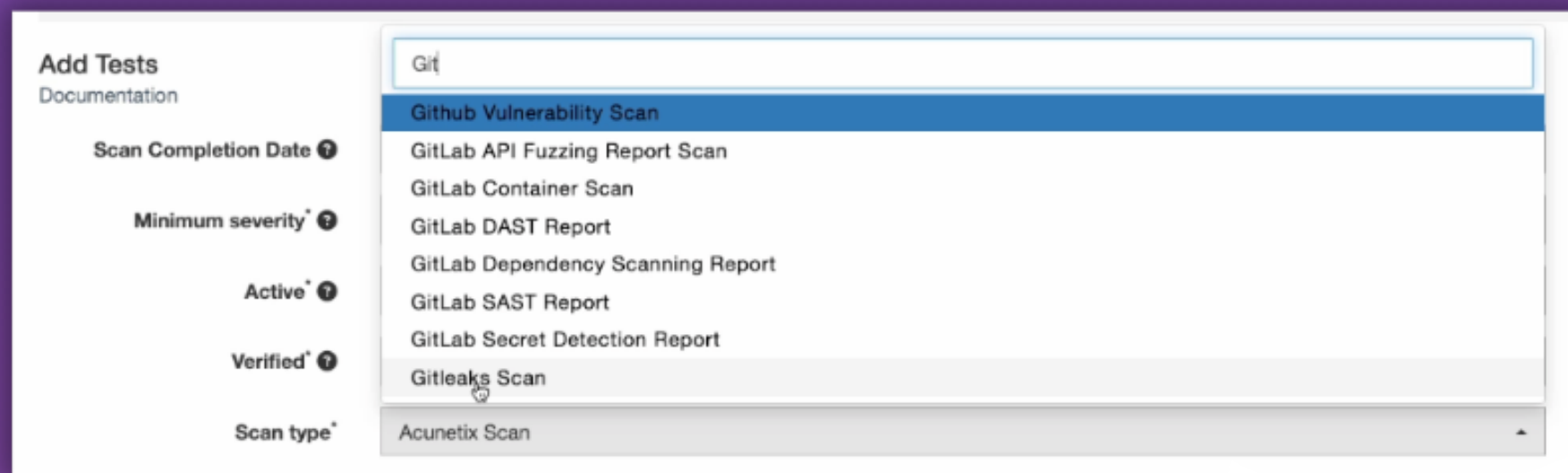
- We need an API key to authenticate to DefectDojo and access its API
- Needs to be sent with the HTTP request



Automatically upload findings to DefectDojo

2 - Write Python Script

- Use requests library to send HTTP request
- Check necessary values in DefectDojo UI or documentation



```
upload-reports.py U •
upload-reports.py
1  import requests
2
3  headers = {
4      'Authorization': 'Token e71f520d6cb842d4465dab1b1d9b97e04d7a231f'
5  }
6
7  url = 'https://demo.defectdojo.org/api/v2/import-scan/'
8
9  data = {
10     'active': True,
11     'verified': True,
12     'scan_type': 'Gitleaks Scan',
13     'minimum_severity': 'Low',
14     'engagement': 19
15 }
16
17 files = {
18     'file': open('gitleaks.json', 'rb')
19 }
20
21 requests.post(url, headers=headers, data=data, files=files)
```

Automatically upload findings to DefectDojo

3 - Add CI job to execute Python script

- Install needed libraries in before_script
- Add script block to execute the script
- Wait for previous jobs to execute ("needs") or introduce new stage that only executes after scanning jobs

```
80  upload_reports:
81      stage: test
82      image: python
83      needs: ["gitleaks", "njsscan", "semgrep"]
84      when: always
85      before_script:
86          - pip3 install requests
87      script:
88          - python3 upload-reports.py gitleaks.json
89          - python3 upload-reports.py njsscan.sarif
90          - python3 upload-reports.py semgrep.json
91
```


Wrap Up



Remediation Examples

Broken or Weak Algorithm



Found vulnerable code

```
const md5 = crypto.createHash('md5')
```



Fixed code

```
const sha256 = crypto.createHash('sha256')
```



Over time, new vulnerabilities and weaknesses can be discovered, making **previously considered secure algorithms insecure**

So continuous, automated security scans are so important



Cryptographic Failures

One of the OWASP Top 10 category



MD5

Using **broken or weak algorithm** to encrypt data

Similar Findings (1) ?

Description

Result message: MD5 is a weak hash which is known to have collision. Use a strong hashing function.

Snippet:

```
const md5 = crypto.createHash('md5')
```

Rule name: NodeMd5

SQL Injection



Found vulnerable code

```
models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE '%${criteria}%') AND deletedAt IS
```



Fixed code

```
models.sequelize.query("SELECT * FROM Products WHERE ((name LIKE :searchQuery OR description LIKE :searchQuery) AND deletedAt IS NULL) OR  
  replacements: { searchQuery: '%' + criteria + '%' },  
  type: models.Sequelize.QueryTypes.SELECT
```

- Use placeholder instead of variable
- Placeholder ensures user input is **treated as data, not executable code**
- It's a key element in preventing SQL injection