

## Assignment 4 – Detecting CVE's with Semgrep

---

Assigned CVE: **CVE-2024-23731**

---

### Step 1 - Analysis:

*What package/library does the CVE affect? What version was it introduced in, and which version was it patched in?*

Package: **embedchain**

Affected versions: **< 0.1.57**

Patched version: **0.1.57**

*Briefly describe the advisory's details about the vulnerability.*

Embedchain is an Open-Source (RAG) Framework for personalizing LLM responses. It makes it easy to create and deploy personalized AI apps.

The **OpenAPI** loader in **Embedchain** before 0.1.57 allows attackers to **execute arbitrary code** due to insecure usage of **yaml.load** in the **load\_data** function of **openapi.py**.

An attacker can execute arbitrary code by supplying a **crafted YAML file** that exploits the function's lack of safe loading.

*Locate and analyze the patch for this CVE. Provide the link to the patch and identify which function is affected by the patch.*

**CWE – 88** (<https://cwe.mitre.org/data/definitions/88.html>)

Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')

**CWE – 94** (<https://cwe.mitre.org/data/definitions/94.html>)

Improper Control of Generation of Code ('Code Injection')

**Patch** - mem0ai:

PR: <https://github.com/mem0ai/mem0/pull/1122>

Fixed Code:

<https://github.com/mem0ai/mem0/blob/main/embedchain/embedchain/loaders/openapi.py>

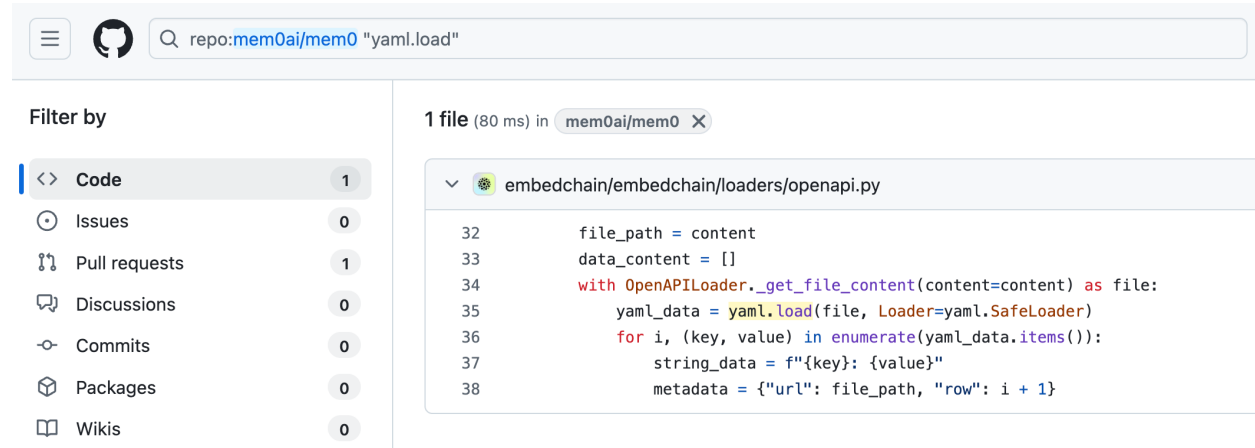
Briefly describe the patch. Note: Analyzing the entire library is not necessary; describe the patch's purpose based on its code.

**yaml\_data = yaml.load(file, Loader=yaml.Loader)**

**yaml\_data = yaml.load(file, Loader=yaml.SafeLoader)**

The patch updates the YAML loading method to use **SafeLoader** instead of **Loader** for preventing the execution of arbitrary code during deserialization.

Once you identify the affected function from the patch (Q3), trace the function to find its callers. Note: The function may not have a parent/caller. If not, describe what steps you took.



The affected function from the patch is the **load\_data** method in the **OpenAPILoader** class. As this is the only location where **yaml.load** is used with the unsafe Loader, there are no further callers to trace. The patch fixes this vulnerability by updating the **yaml.load** call to use the **yaml.SafeLoader**.

For the affected function(s)/class/component, review the library documentation and include examples of how they are meant to be used or called.

The **load\_data** function is designed to load and process OpenAPI YAML files, converting each key-value pair into a document with metadata.

Usage:

**yaml\_data = yaml.load(file, Loader=yaml.Loader) # Vulnerable**

**yaml\_data = yaml.load(file, Loader=yaml.SafeLoader) # Recommended**

SafeLoader is a safer alternative to Loader, restricting YAML constructs to simple Python objects like str, list, and dict.

Example via original source code and ChatGPT:

Example Use in `OpenAPILoader` :

```
python Copy code  
  
with OpenAPILoader._get_file_content(content=content) as file:  
    yaml_data = yaml.load(file, Loader=yaml.SafeLoader) # Safely parse YAML
```

#### Intended Usage:

- **Input:** URL or file path pointing to an OpenAPI YAML document.
- **Output:** A structured dictionary representing the YAML content, broken down into individual "documents" for further processing, with metadata.

#### Example Flow:

1. Pass a URL or file path to `load_data`.
2. YAML content is parsed securely using `SafeLoader`.
3. Each key-value pair in the YAML is extracted and stored with associated metadata (e.g., source URL, line number).
4. A unique document ID is generated using `hashlib`.

This method ensures safe parsing while maintaining the integrity and usability of OpenAPI specifications.

---

*Cross-reference other sources and summarize the information provided regarding questions 1 to 4. Note: Not all sources may provide complete information. Document whatever is available.*

CVE-2024-23731 affects the Embedchain package (versions < 0.1.57) due to insecure usage of `yaml.load` in the `load_data` function of the `OpenAPILoader` class, which could allow attackers to execute arbitrary code by providing malicious YAML files. The issue was patched in version 0.1.57 by replacing `yaml.Loader` with `yaml.SafeLoader` to prevent unsafe deserialization. Users are advised to upgrade to version 0.1.57 or later and ensure strict input validation when processing YAML files. Safe alternatives like `yaml.safe_load` should be used to mitigate such risks.

## PoC (via ChatGPT):

```
import yaml

# Insecure use of yaml.load with yaml.Loader
def load_data(file_path):
    with open(file_path, 'r') as file:
        yaml_data = yaml.load(file, Loader=yaml.Loader)
        return yaml_data

# Calling the vulnerable function
load_data('malicious.yaml')
```

Copy code

### 2. Malicious YAML File (malicious.yaml):

```
yaml

!!python/object/apply:os.system
- "echo 'Malicious code executed!'"
```

Copy code

## Explanation:

- **Malicious YAML:** The YAML file uses the `!!python/object/apply` tag, which allows for the execution of arbitrary Python code. In this case, it is calling `os.system("echo 'Malicious code executed!'")` to execute a shell command that prints a message.
- **Loader:** `yaml.Loader` allows this malicious tag to be interpreted and executed, leading to the execution of the shell command when the YAML file is loaded.



## How it works:

lib/python3.9/site-packages

embedchain

loaders

github.py

gmail.py

google\_drive.py

image.py

json.py

local\_qna\_pair.py

local\_text.py

mdx.py

mysql.py

notion.py

openapi.py

pdf\_file.py

postgres.py

rss\_feed.py

sitemap.py

slack.py

substack.py

text\_file.py

unstructured\_file.py

web\_page.py

xml.py

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

@staticmethod

def \_get\_file\_content(content):

url = urlparse(content)

if all([url.scheme, url.netloc]) and url.scheme not in ["file", "http", "https"]:

raise ValueError("Not a valid URL.")

if url.scheme in ["http", "https"]:

response = requests.get(content)

response.raise\_for\_status()

return StringIO(response.text)

elif url.scheme == "file":

path = url.path

return open(path)

else:

return open(content)

@staticmethod

def load\_data(content):

"""Load yaml file of openapi. Each pair is a document."""

data = []

file\_path = content

data\_content = []

with OpenAPIloader.\_get\_file\_content(content=content) as file:

yaml\_data = yaml.load(file, Loader=yaml.SafeLoader)

for i, (key, value) in enumerate(yaml\_data.items()):

string\_data = f'{{key}}: {{value}}'

metadata = {"url": file\_path, "row": i + 1}

data.append({"content": string\_data, "meta\_data": metadata})

data\_content.append(string\_data)

doc\_id = hashlib.sha256((content + ", ".join(data\_content)).encode()).hexdigest()

return {"doc\_id": doc\_id, "data": data}

When we “**pip install embedchain**”

We can access the openapi loader, the `OpenAPILoader` class, and the `load_data` method, which calls **yaml.load** with the previously vulnerable **yaml.Loader**.

Based on the findings, users of the embedchain library who invoke the `load_data` method from the `OpenAPILoader` class could indirectly trigger the vulnerable `yaml.load` function with `yaml.Loader`. This means the CVE (CVE-2024-23731) has a tangible impact on both:

1. **Library Developers:** They must ensure internal code is secure by updating the `load_data` method to use `yaml.SafeLoader` instead of `yaml.Loader`. (**Fixed with the patch**, commit given above.)
2. **End Users:** They are affected because the vulnerable `yaml.load` is exposed through the `OpenAPILoader` class's public API. (Just Upgrade embedchain to version **0.1.57** or higher.)

Fix?

- Upgrade embedchain to version **0.1.57** or higher.  
We can do this by running **pip install --update embedchain**.
- Input Validation: Ensure that the **YAML parser** in your OpenAPI loader **strictly validates** incoming data to avoid processing untrusted inputs.
- Safer methods: Utilize safer alternatives to **yaml.load**, such as **yaml.safe\_load**, which is designed to resist arbitrary code execution.
- <https://pyyaml.org/wiki/PyYAMLDocumentation>

## Step 2 – Writing Semgrep Rule:

pattern-either:

- pattern: |  
  yaml.load(..., Loader=yaml.Loader, ...)
- pattern: |  
  \$VAR = yaml.Loader  
  ...  
  yaml.load(..., Loader=\$VAR, ...)
- pattern: |  
  \$DICT = {...}  
  ...  
  \$VAR = \$DICT[\$KEY]  
  ...  
  yaml.load(..., Loader=\$VAR, ...)

CVE-2024-23731

Unsaved changes

Save

Share

structure **NEW**

advanced

test code

live code **NEW**

metadata

docs

```
1 rules:
2   - id: CVE-2024-23731
3     message: >
4       "Insecure use of yaml.load() detected. This can lead to arbitrary code
5       execution. Use yaml.SafeLoader instead to safely deserialize YAML
6       content."
7     languages:
8       - python
9     severity: ERROR
10    pattern-either:
11      - pattern: |
12        |   yaml.load(..., Loader=yaml.Loader, ...)
13      - pattern: |
14        |   $VAR = yaml.Loader
15        |   ...
16        |   yaml.load(..., Loader=$VAR, ...)
17      - pattern: |
18        |   $DICT = {...}
19        |   ...
20        |   $VAR = $DICT[$KEY]
21        |   ...
22        |   yaml.load(..., Loader=$VAR, ...)
```

```
1 import yaml as hritesh
2
3 # Vulnerable usage of yaml.load with loader
4 with open('example.yaml', 'r') as file:
5     data = hritesh.load(file, Loader=yaml.Loader)
6 print(data)
7
8 # Using Dict to indirectly assign
9 loader_map = {"unsafe": yaml.Loader}
10 ...
11 loader = loader_map["unsafe"]
12 yaml.load(file, Loader=loader)
13
14 # Vulnerable: using loader dynamically assigned
15 loader = yaml.Loader
16 yaml_data = yaml.load(file, Loader=loader)
```

---

### Step 3 – Testing the Rule:

```
submissions/
├── rules/
│   └── CVE-2024-23731.yaml
└── test-CVE-2024-23731/
    ├── test1.py
    ├── test2.py
    ├── test3.py
    ├── test4.py
    └── test5.py
```

Run:

**semgrep --config path/to/CVE-2024-23731.yaml path/to/test/directory**

Note: I'm encountering a false positive in *test3.py*. I'm not sure why it's detecting matches twice. If I remove the second pattern and keep the third, I end up losing a match, haha. Maybe I'll revisit this next week.

--

#### References:

- <https://github.com/advisories?query=2024-23731>
- <https://github.com/advisories/GHSA-rhhj-5436-95vf>
- <https://nvd.nist.gov/vuln/detail/CVE-2024-23731>
- <https://security.snyk.io/vuln/SNYK-PYTHON-EMBEDCHAIN-6183296>
- <https://ogma.in/cve-2024-23731-understanding-and-mitigating-vulnerabilities-in-embedchain-openapi-loader>
- <https://feedly.com/cve/CVE-2024-23731>
- <https://vuldb.com/?id.251693>
- <https://www.cve.org/CVERecord?id=CVE-2024-23731>
- <https://avd.aquasec.com/nvd/2024/cve-2024-23731>
- <https://pypi.org/project/embedchain>
- <https://docs.embedchain.ai/get-started/quickstart>

---

Thank you!