Ammar Meslmani - CBS-01

a.meslmani@innopolis.university

the repo link to check the files used in this assignment: full report

Lab 2 - Vulnerability Scanning

Task1:

- in this task, `poetry` will be used as a python virtual environment, so let's check the verion and initialize a project:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry --version
Poetry (version 2.1.1)
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry init

This command will guide you through creating your pyproject.toml config.

Package name [lab2]:
Version [0.1.0]:
Description []:
Author [Ammar Meslmani <a.meslmani@innopolis.university>, n to skip]:
License []:
Compatible Python versions [>=3.12]:

Would you like to define your main dependencies interactively? (yes/no) [yes]
        You can specify a package in the following forms:
          - A single name (requests): this will search for matches on PyPI
          - A name and a constraint (requests@^2.23.0)
          - A git url (git+https://github.com/python-poetry/poetry.git)
          - A git url with a revision         (git+https://github.com/python-poetry/poetry.git#develop)
          - A file path (../my-package/my-package.whl)
          - A directory (../my-package/)
          - A url (https://example.com/packages/my-package-0.1.0.tar.gz)

Package to add or search for (leave blank to skip):

Would you like to define your development dependencies interactively? (yes/no) [yes]
Package to add or search for (leave blank to skip):

Generated file

[project]
name = "lab2"
version = "0.1.0"
description = ""
authors = [
    {name = "Ammar Meslmani",email = "a.meslmani@innopolis.university"}
]
readme = "README.md"
requires-python = ">=3.12"
dependencies = [
]


[build-system]
requires = ["poetry-core>=2.0.0,<3.0.0"]
build-backend = "poetry.core.masonry.api"


Do you confirm generation? (yes/no) [yes]
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$
```

1.1. Bandit

- let's install `bandit`:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry add --dev bandit
Creating virtualenv lab2-Bc5w-_K2-py3.12 in /home/ammar/.cache/pypoetry/virtualenvs
Using version ^1.8.3 for bandit

Updating dependencies
Resolving dependencies... (3.0s)

Package operations: 9 installs, 0 updates, 0 removals

  - Installing mdurl (0.1.2)
  - Installing setuptools (75.8.2)
  - Installing markdown-it-py (3.0.0)
  - Installing pbr (6.1.1)
  - Installing pygments (2.19.1)
  - Installing pyyaml (6.0.2)
  - Installing rich (13.9.4)
  - Installing stevedore (5.4.1)
  - Installing bandit (1.8.3)

Writing lock file
```

- let's clone the provided `vulpy` repo:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ git clone https://github.com/fportantier/vulpy.git
Cloning into 'vulpy'...
remote: Enumerating objects: 437, done.
remote: Total 437 (delta 0), reused 0 (delta 0), pack-reused 437 (from 1)
Receiving objects: 100% (437/437), 2.90 MiB | 960.00 KiB/s, done.
Resolving deltas: 100% (223/223), done.
```

- let's run `bandit` against the cloned repo:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry run bandit -r vulpy/
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.12.7
Working... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00
Run started:2025-03-04 01:26:33.037789

Test results:
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b113_request_without_timeout.html
   Location: vulpy/bad/api_list.py:10:8
9
10          r = requests.get('http://127.0.1.1:5000/api/post/{}'.format(username))
11          if r.status_code != 200:

--------------------------------------------------
>> Issue: [B108:hardcoded_tmp_directory] Probable insecure usage of temp file/directory.
   Severity: Medium   Confidence: Medium
   CWE: CWE-377 (https://cwe.mitre.org/data/definitions/377.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b108_hardcoded_tmp_directory.html
   Location: vulpy/bad/api_post.py:6:20
5
6       api_key_file = Path('/tmp/supersecret.txt')
7

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b113_request_without_timeout.html
   Location: vulpy/bad/api_post.py:16:12
15
```

- findings:
    - high severity finding:

```
--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrar
y code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b201_flask_debug_true.html
   Location: vulpy/good/vulpy.py:53:0
52
53      app.run(debug=True, host='127.0.1.1', port=5001, extra_files='csp.txt')
54
```

- explanation:

  A Flask app is being run with `debug=True`, which enables the Werkzeug debugger. This exposes the app to potential security risks, as the debugger allows arbitrary code execution if an attacker gains access to it. Moreover it provides detailed error messages and stack traces, which can reveal sensitive information about the application.

- relevant CWE:
  - CWE-94: Improper Control of Generation of Code ('Code Injection')
- possible mitigation:
  - disable debug mode in production

- medium severity finding:

```
--------------------------------------------------
>> Issue: [B108:hardcoded_tmp_directory] Probable insecure usage of temp file/directory.
   Severity: Medium   Confidence: Medium
   CWE: CWE-377 (https://cwe.mitre.org/data/definitions/377.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b108_hardcoded_tmp_directory.html
   Location: vulpy/utils/ca-create.py:31:10
30
31      with open('/tmp/ca.key', 'wb') as out:
32          out.write(pem_private)

--------------------------------------------------
```

- explanation:

  The code uses a hardcoded temporary directory (`/tmp/ca.key`), which can lead to insecure file handling.

- relevant CWE:
  - CWE-377: Insecure Temporary File
- possible mitigation:
  - using Python's tempfile module to create secure temporary files

- low severity finding:

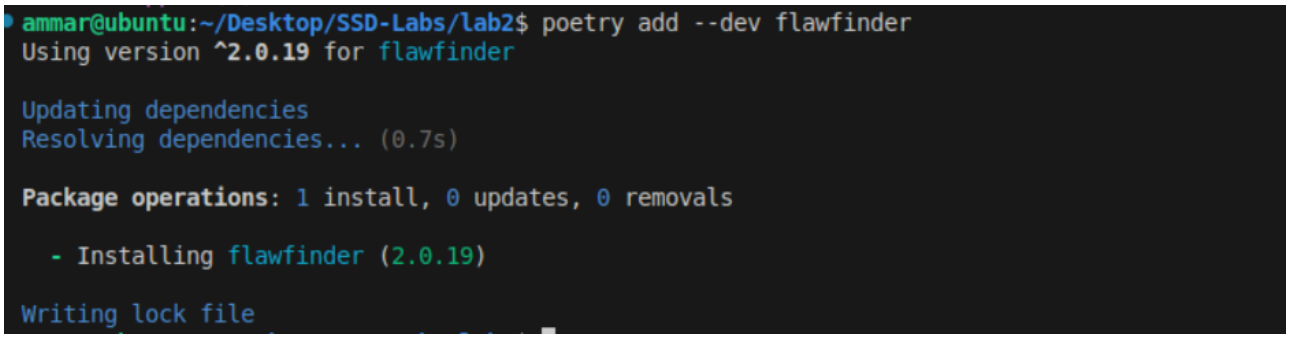
```
--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: '123aa8a93bdde342c871564a62282af857bda14b3359fde95d0c5e4b321610c1'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b105_hardcoded_password_string.html
   Location: vulpy/good/vulpy.py:17:11
16      app = Flask('vulpy')
17      app.config['SECRET_KEY'] = '123aa8a93bdde342c871564a62282af857bda14b3359fde95d0c5e4b321610c1'
18

--------------------------------------------------
```

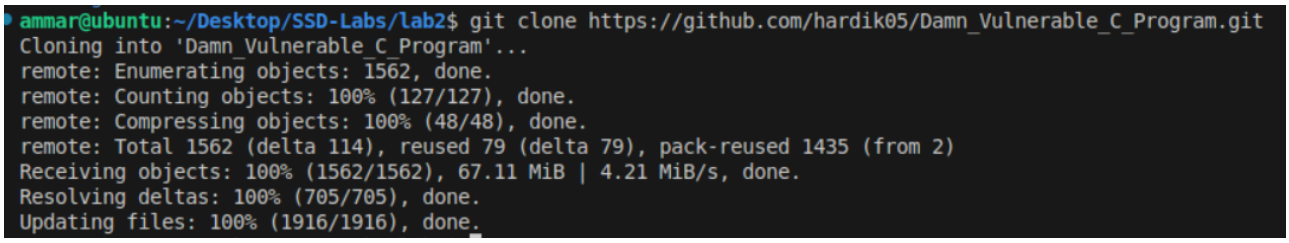
- explanation:The code contains a hardcoded password or secret key (`123aa8a93bdde342c871564a62282af857bda14b3359fde95d0c5e4b321610c1`) which could be exposed to anyone with access to the codebase. Moreover, it makes it difficult to update the secret without redeploying the whole application.

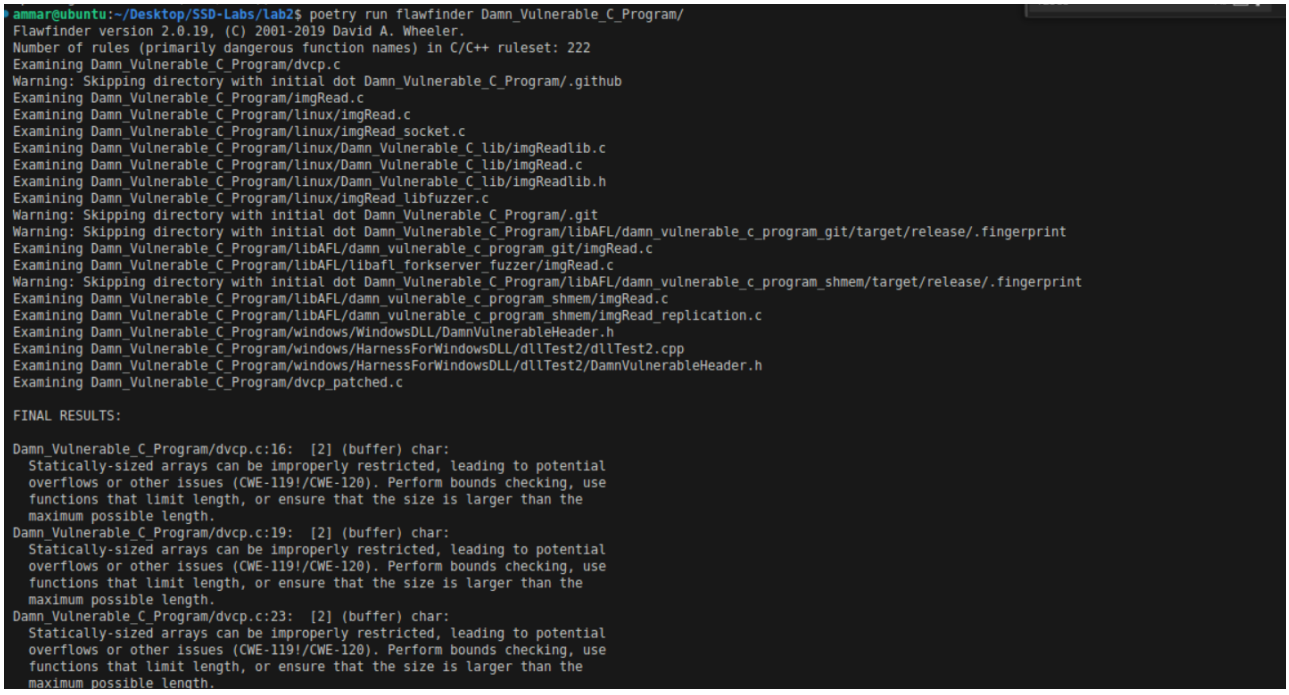
- relevant CWE:

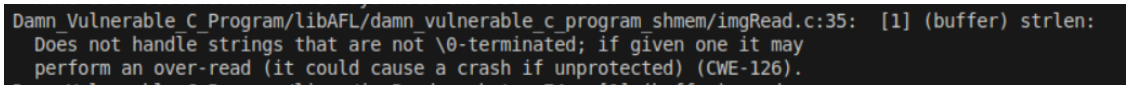
        - CWE-259: Use of Hard-Coded Password
    - possible mitigation:
        - Useing environment variables to store secrets

1.2. Flawfinder (C)

- let's install `flawfinder`:
- 

- let's clone the provided repo:
- 

- let's run `flawfinder` against the cloned repo:
- 

- findings:
    - level 1:
        - 

        - explanation: The `strlen` function is used on a string that may not be null-terminated (\0). If the string is not null-terminated, strlen will continue reading memory beyond the

intended boundary, leading to an over-read.
    - relevant CWE: CWE-126: Buffer Over-read
    - possible mitigation: Ensure that all strings passed to strlen are properly null-terminated.

- level 2:

```
Damn_Vulnerable_C_Program/linux/imgRead_socket.c:131:  [2] (buffer) memcpy:
  Does not check for buffer overflows when copying to destination (CWE-120).
  Make sure destination can always hold the source data.
```

    - explanation: The `memcpy` function is used to copy data from a source to a destination buffer without checking if the destination buffer is large enough to hold the data. This can lead to a buffer overflow.
    - relevant CWE: CWE-120: Buffer Copy without Checking Size of Input
    - possible mitigation: Always validate the size of the source and destination buffers before using `memcpy`.

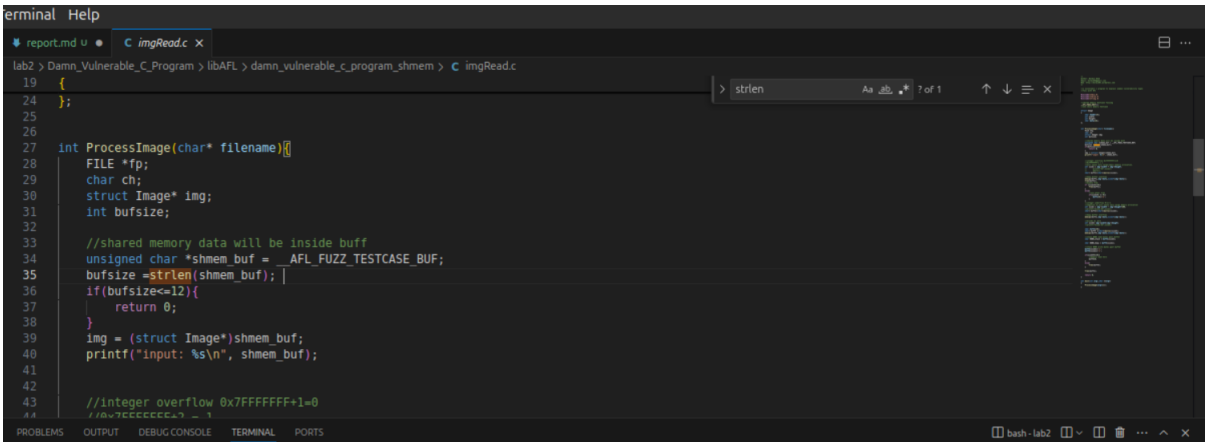
- level 3:

    - no such finding was found.

- **fasle positive**:
    - considering the following finding:

```
Damn_Vulnerable_C_Program/libAFL/damn_vulnerable_c_program_shmem/
imgRead.c:35:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given
one it may
  perform an over-read (it could cause a crash if unprotected)
(CWE-126).
```

    - let's check the file:



    - we can consider it as a false positive **only if** it's always guaranteed that the fuzzer will provide `null terminated strings` to `__AFL_FUZZ_TESTCASE_BUF`, which is a shared memory buffer used for fuzzing

1.3. njsscan

- let's install `njsscan` (after suffering with fixing versions issues and some broken packages):

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry add --dev njsscan
Using version ^0.4.3 for njsscan

Updating dependencies
Resolving dependencies... (2.3s)

Package operations: 24 installs, 0 updates, 0 removals

  - Installing opentelemetry-instrumentation (0.46b0)
  - Installing opentelemetry-util-http (0.46b0)
  - Installing billiard (4.2.1)
  - Installing click-option-group (0.5.6)
  - Installing colorama (0.4.6)
  - Installing defusedxml (0.7.1)
  - Installing exceptiongroup (1.2.2)
  - Installing glom (22.1.0)
  - Installing jsonpickle (4.0.2)
  - Installing jsonschema (4.23.0)
  - Installing opentelemetry-exporter-otlp-proto-http (1.25.0)
  - Installing opentelemetry-instrumentation-requests (0.46b0)
  - Installing packaging (24.2)
  - Installing peewee (3.17.9)
  - Installing pydantic (2.8.2)
  - Installing ruamel-yaml (0.17.40)
  - Installing tomli (2.0.2)
  - Installing wcmatch (8.5.2)
  - Installing jschema-to-python (1.2.3)
  - Installing libsast (3.1.6)
  - Installing sarif-om (1.0.4)
  - Installing semgrep (1.86.0)
  - Installing tabulate (0.9.0)
  - Installing njsscan (0.4.3)

Writing lock file
```

- let's clone the provided repo:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ git clone https://github.com/appsecco/dvna.git
Cloning into 'dvna'...
remote: Enumerating objects: 645, done.
remote: Total 645 (delta 0), reused 0 (delta 0), pack-reused 645 (from 1)
Receiving objects: 100% (645/645), 3.18 MiB | 2.17 MiB/s, done.
Resolving deltas: 100% (281/281), done.
```

- let's run `njsscan` against the cloned repo:

```
ammar@ubuntu:~/Desktop/SSD-Labs/lab2$ poetry run njsscan dvna/
- Pattern Match ████████████████████████████████████████ 148
- Semantic Grep  29

njsscan: v0.4.3 | Ajin Abraham | opensecurity.in
```

RULE ID	ejs_ect_template
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
OWASP-WEB	A1: Injection
DESCRIPTION	The EJS/ECT template has an unescaped variable. Untrusted user input passed to this variable results in Cross Site Scripting (XSS).
SEVERITY	ERROR
FILES	

File	dvna/views/app/products.ejs
Match Position	1907 - 1937
Line Number(s)	51
Match String	<%- output.products[i].code %>
File	dvna/views/app/products.ejs
Match Position	2019 - 2056
Line Number(s)	53

- findings:
  - Error:

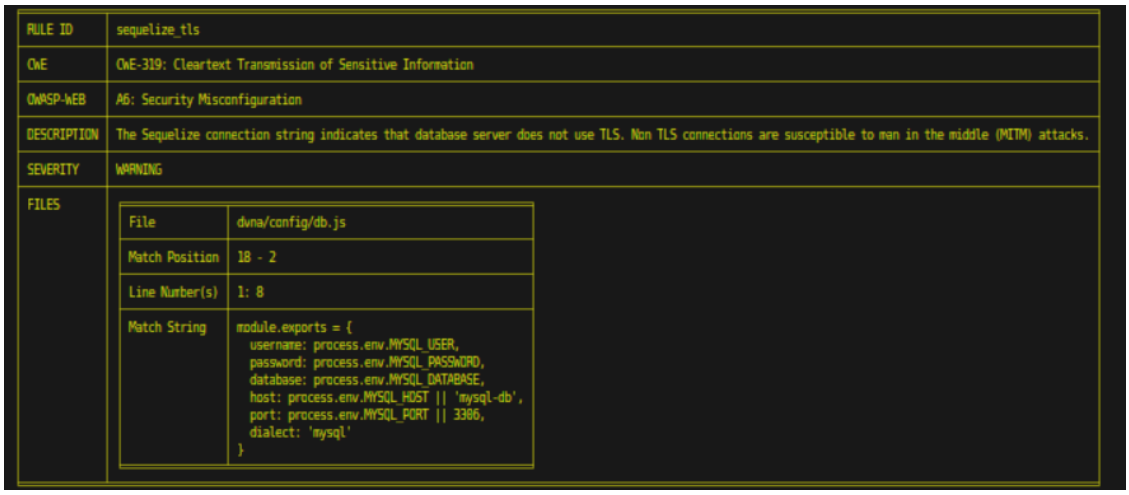
RULE ID	ejs_ect_terplate
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
OWASP-WEB	A1: Injection
DESCRIPTION	The EJS/ECT terplate has an unescaped variable. Untrusted user input passed to this variable results in Cross Site Scripting (XSS).
SEVERITY	ERROR
FILES	

File	dvna/views/app/products.ejs
Match Position	1907 - 1937
Line Number(s)	51
Match String	<%- output.products[i].code %>
File	dvna/views/app/products.ejs
Match Position	2019 - 2056
Line Number(s)	53
Match String	<%- output.products[i].description %>
File	dvna/views/app/products.ejs
Match Position	1797 - 1825
Line Number(s)	49
Match String	<%- output.products[i].id %>
File	dvna/views/app/products.ejs
Match Position	1851 - 1881
Line Number(s)	50
Match String	<%- output.products[i].name %>
File	dvna/views/app/products.ejs
Match Position	1963 - 1993
Line Number(s)	52
Match String	<%- output.products[i].tags %>
File	dvna/views/app/products.ejs
Match Position	721 - 745
Line Number(s)	20
Match String	<%- output.searchTerm %>

- explanation: The <%- %> syntax in EJS directly injects data into HTML without escaping it, which makes the application vulnerable to XSS. If an attacker can manipulate `output.products[i].name` or any other variable, they could inject malicious JavaScript into the webpage.
- relevant CWE: CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting).
- possible mitigation:

  Using `<%= %>` instead of `<%- %>` to automatically escape user input.

  - Warning:

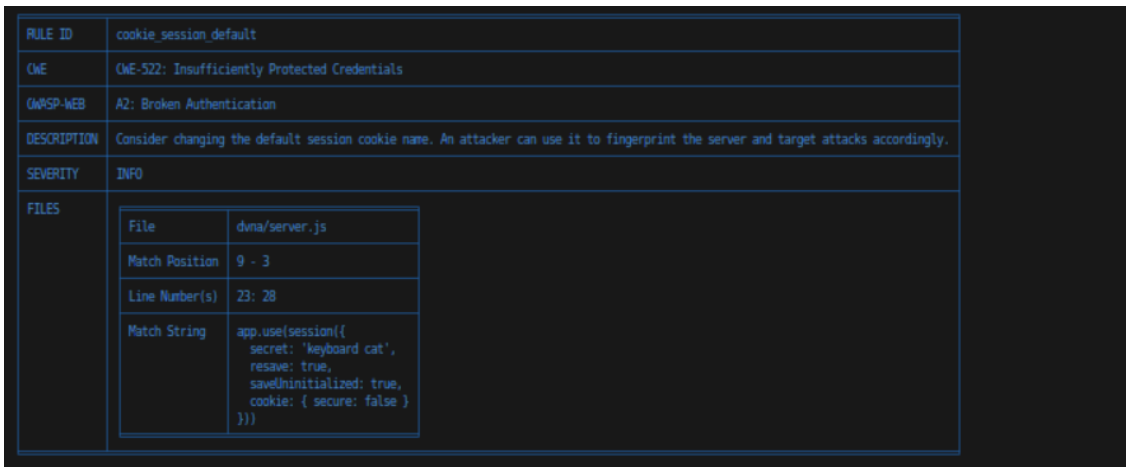


RULE ID	sequelize_tls
CWE	CWE-319: Cleartext Transmission of Sensitive Information
OWASP-WEB	A6: Security Misconfiguration
DESCRIPTION	The Sequelize connection string indicates that database server does not use TLS. Non TLS connections are susceptible to man in the middle (MITM) attacks.
SEVERITY	WARNING

FILES	File: dvna/config/db.js
	Match Position: 18 - 2
	Line Number(s): 1: 8
	Match String: module.exports = {
	username: process.env.MYSQL_USER,
	password: process.env.MYSQL_PASSWORD,
	database: process.env.MYSQL_DATABASE,
	host: process.env.MYSQL_HOST || 'mysql-db',
	port: process.env.MYSQL_PORT || 3306,
	dialect: 'mysql'
	}

- explanation: The database connection is not using TLS, making it vulnerable to Man-in-the-Middle (MITM) attacks. This means an attacker could intercept credentials and queries.
- relevant CWE: CWE-319: Cleartext Transmission of Sensitive Information.
- possible mitigation: Enabling TLS in the Sequelize configuration.

- Info:

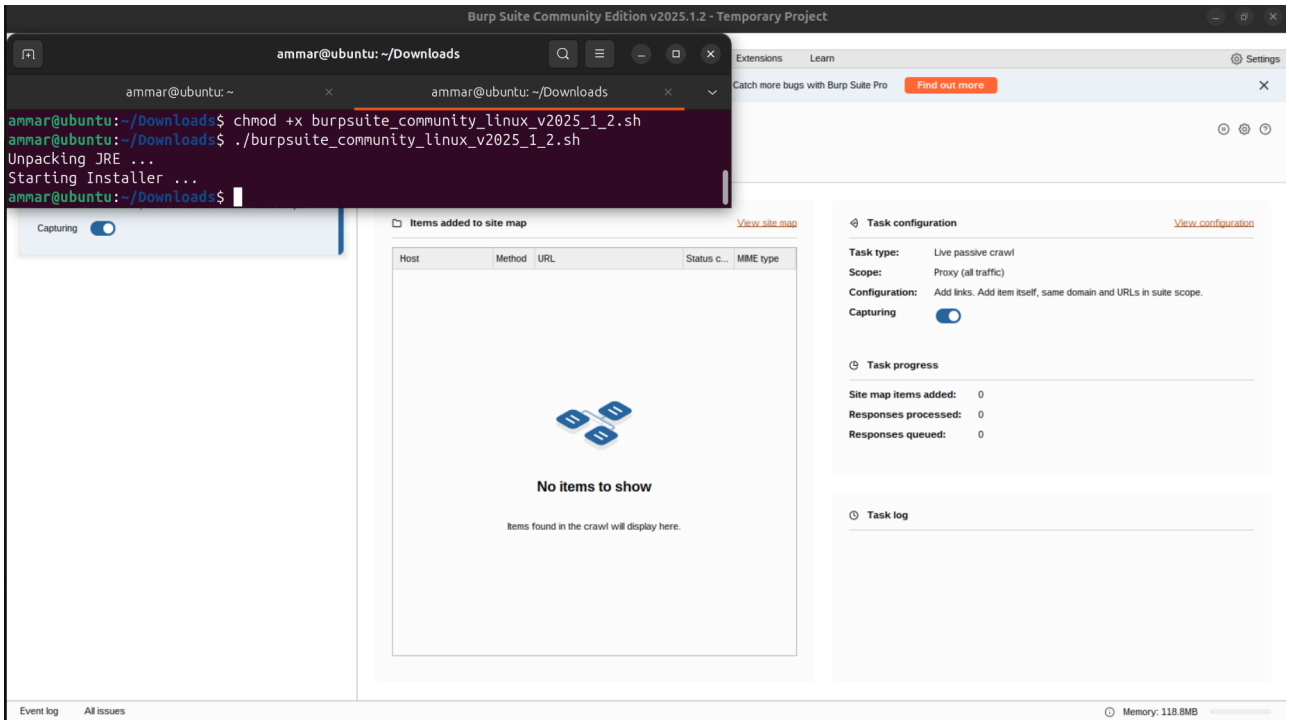


RULE ID	cookie_session_default
CWE	CWE-522: Insufficiently Protected Credentials
OWASP-WEB	A2: Broken Authentication
DESCRIPTION	Consider changing the default session cookie name. An attacker can use it to fingerprint the server and target attacks accordingly.
SEVERITY	INFO
FILES	File: dvna/server.js Match Position: 9 - 3 Line Number(s): 23: 28 Match String: app.use(session({ secret: 'keyboard cat', resave: true, saveUninitialized: true, cookie: { secure: false } }))

- explanation: The default session configuration is used, which includes secret `keyboard cat`. Attackers can fingerprint the application and potentially hijack sessions.
- relevant CWE: CWE-522: Insufficiently Protected Credentials
- possible mitigation: Changing the default session secret to a strong, random value. In addition, setting `cookie: { secure: true }` to ensure cookies are only sent over HTTPS.

Task2:

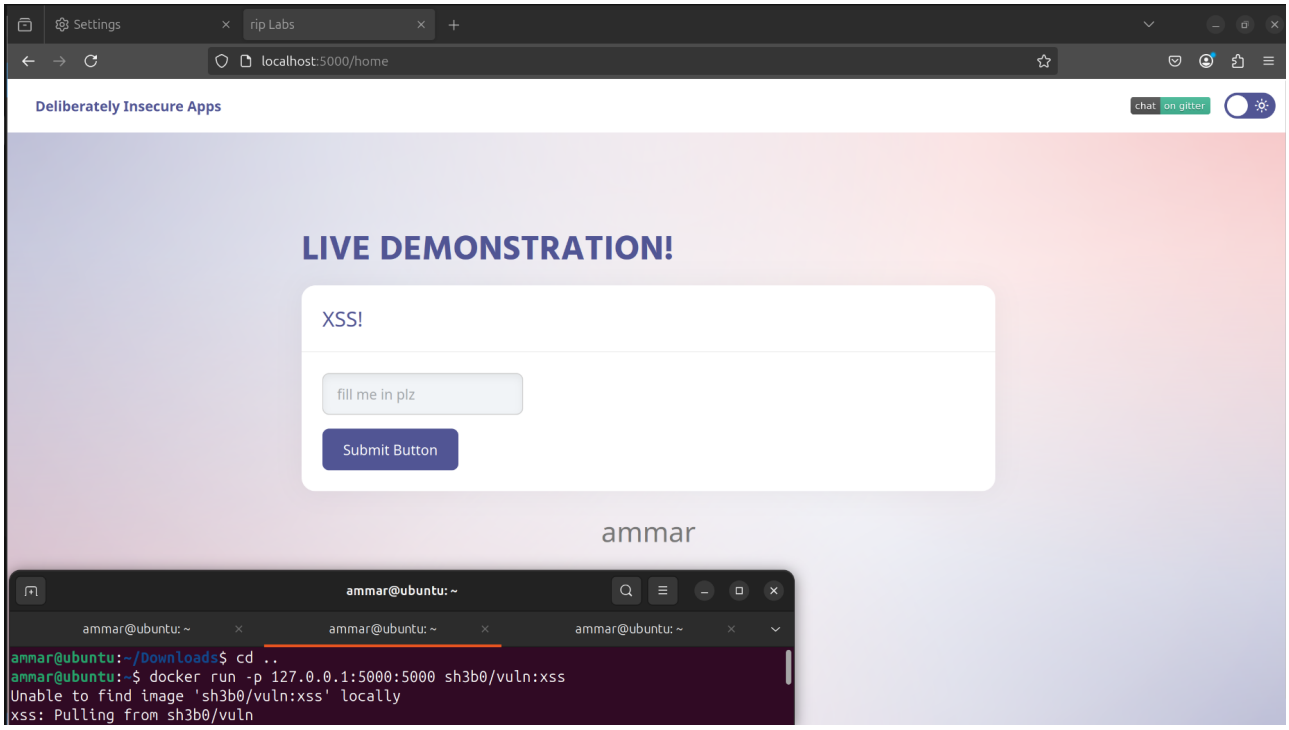
- let's install and run `burp`:



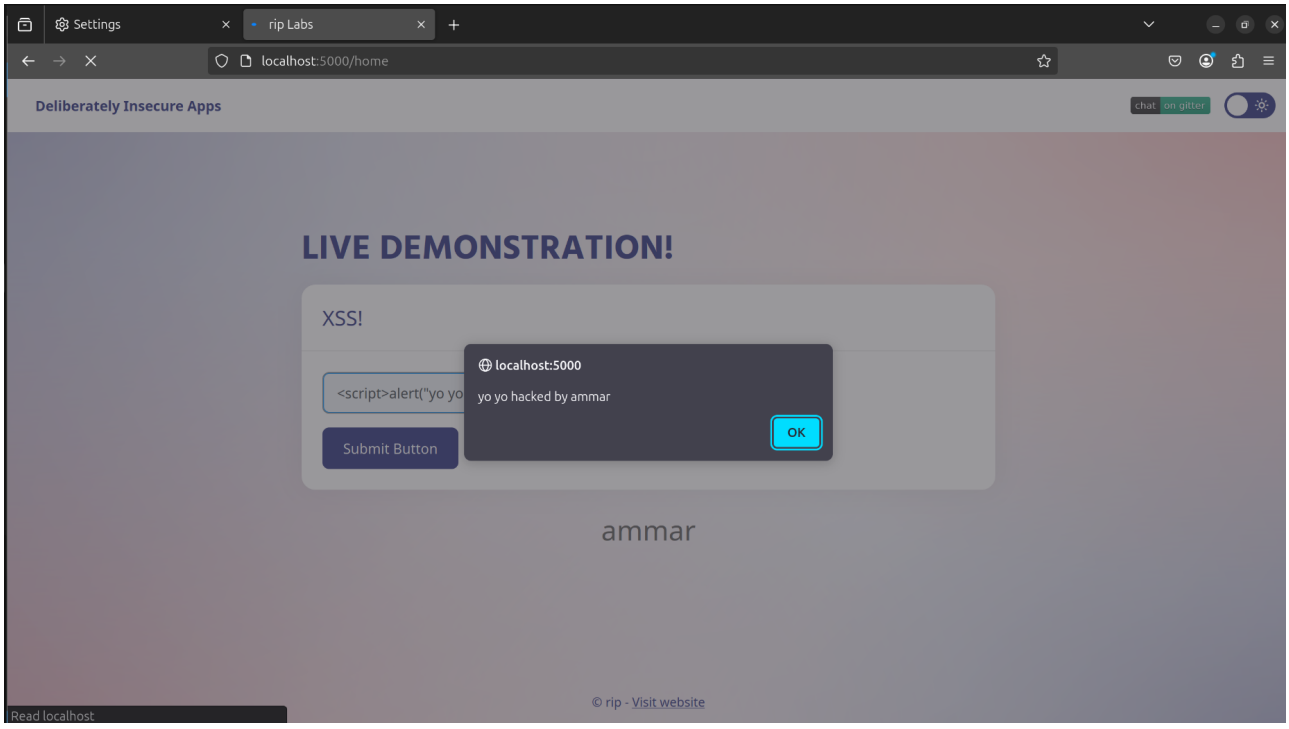
- then let's install firefox and set the proxy to `127.0.0.1:8080` to interact with `burp` (becuase i use `chromium`)
- **note**: it was **painful** to find out that this little stupid flag `network.proxy.allow_hijacking_localhost` should be toggled to allow capturing the localhost

Cross Site Scripting

- let's pull and run the docker image:



- after some tries, let's try to insert :



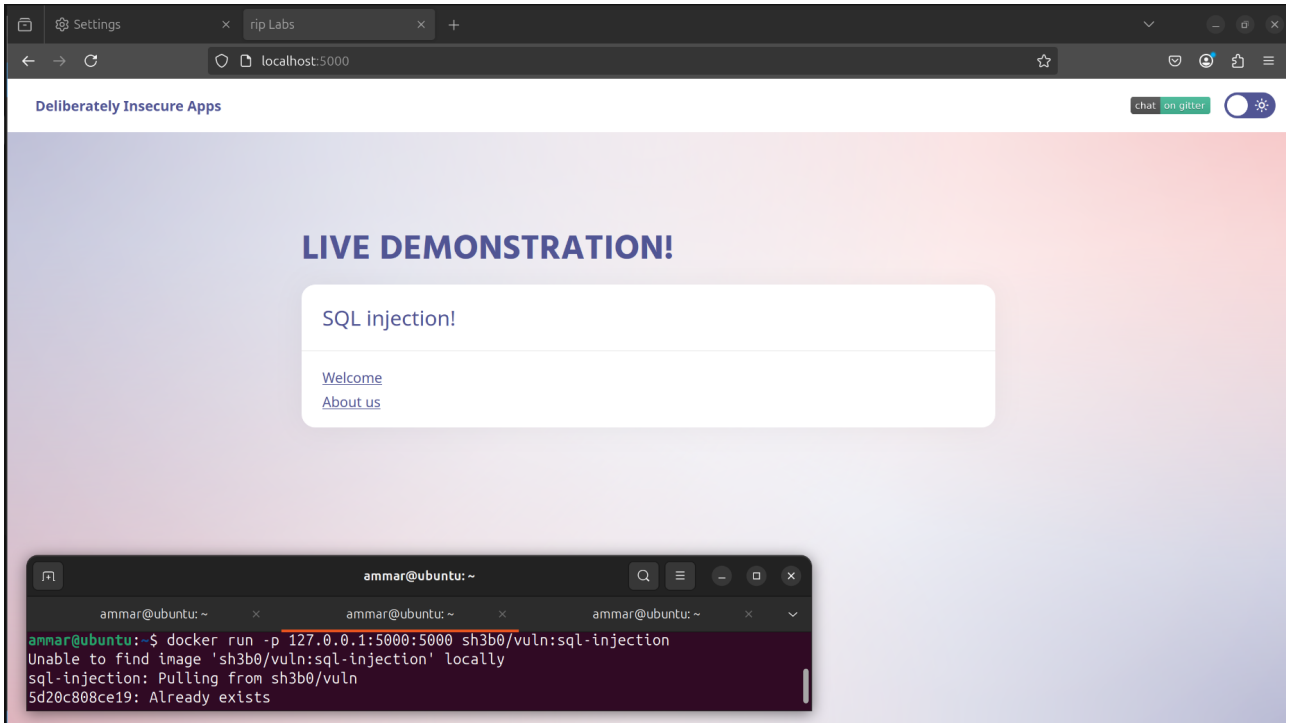
- XSS is found!
- when i find XSS, me for no reason:

- 

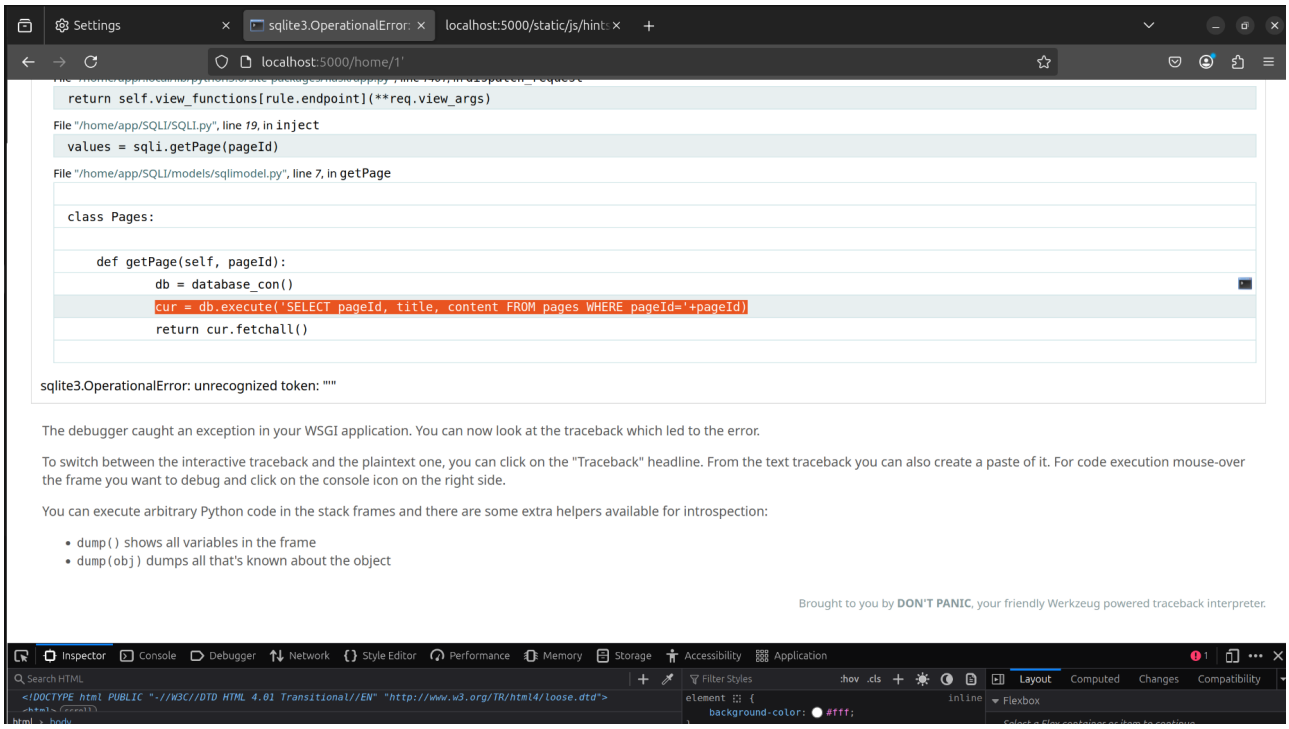
- why it's dangerous? there are a lot of reasons (like a lot!), one of them that it allows attackers to hijack cookies sessions, personal information, or login credentials
- possible mitigation:

  to prevent XSS vulnerabilities, we need to ensure that user input is properly sanitized, validated, and escaped before it is rendered in the browser

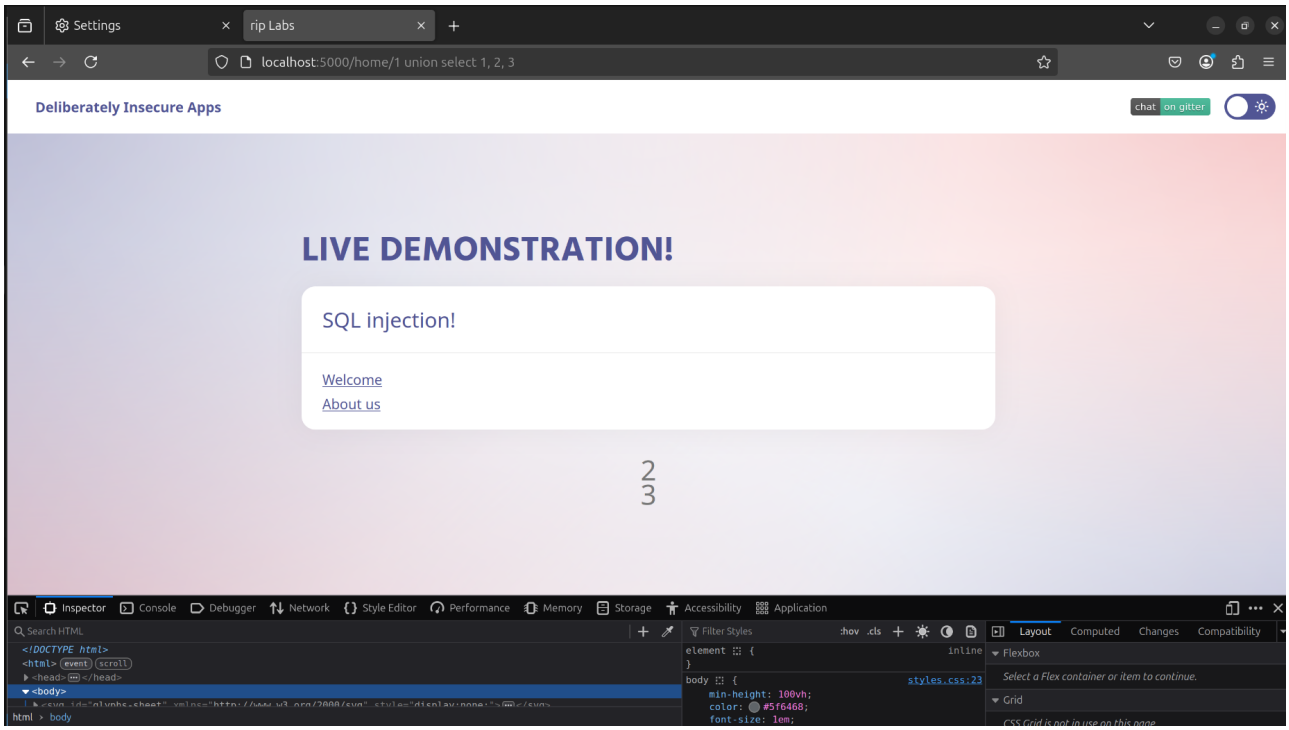
SQL Injection

- let's pull and run the docker image:
- 

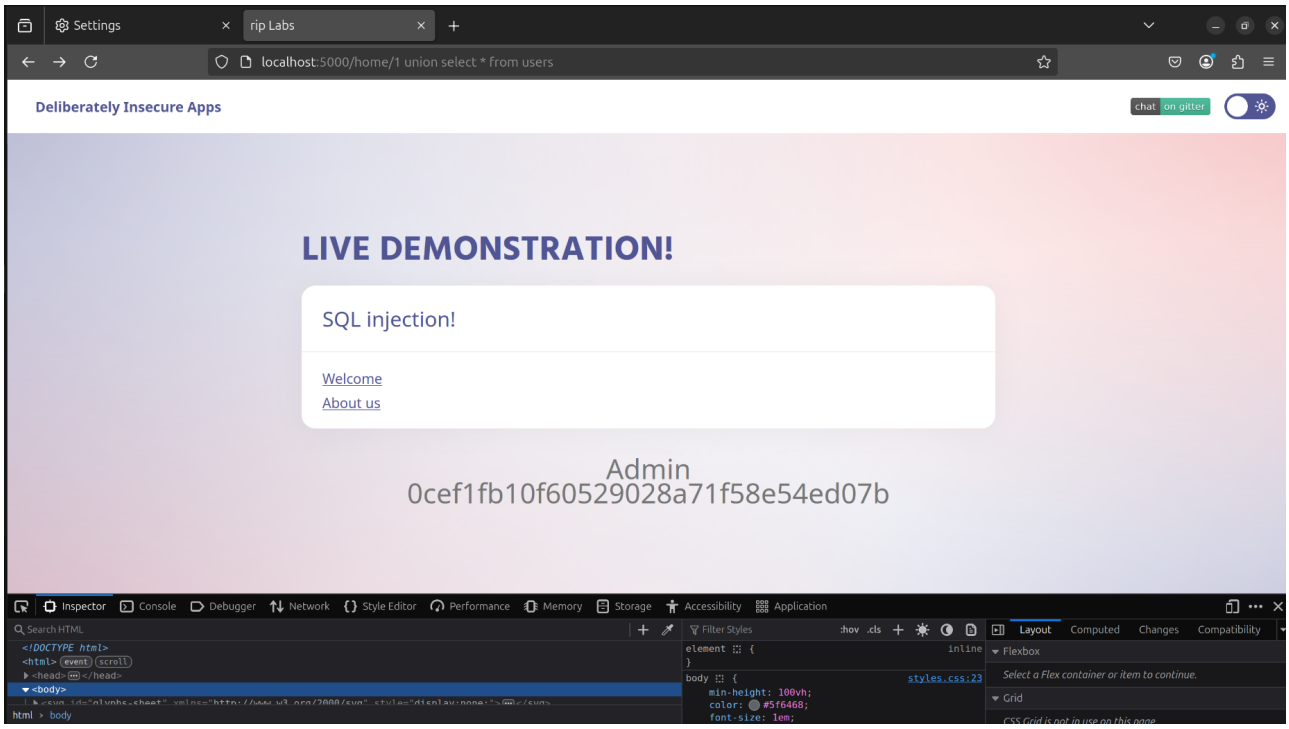
- let's try to insert a `'` in the url:



- a good start! now we know that we are dealing with 3 columns
- let's try the following:



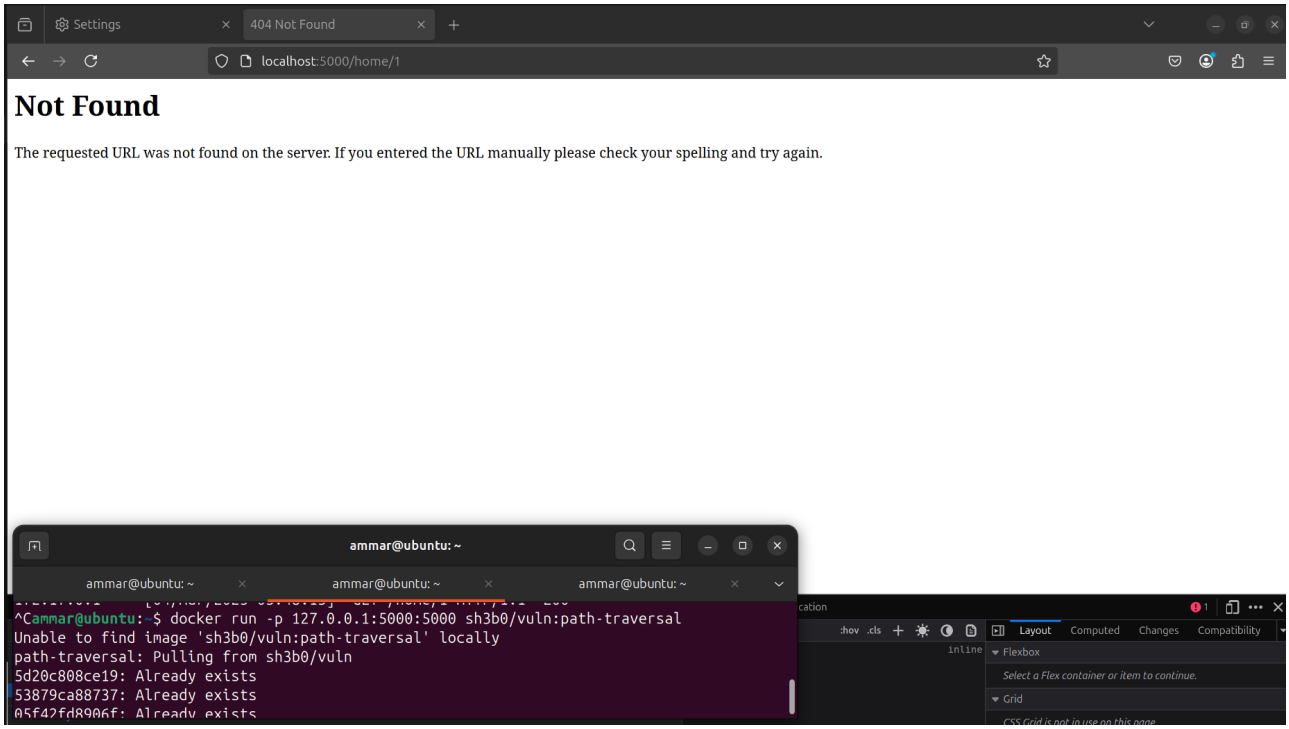
- so `title` and `content` became placeholders for data
- let's try the following by trying some guesses for the users table (user, users, ...):
- found the correct table name, `users`, now let's append `union select * from users`



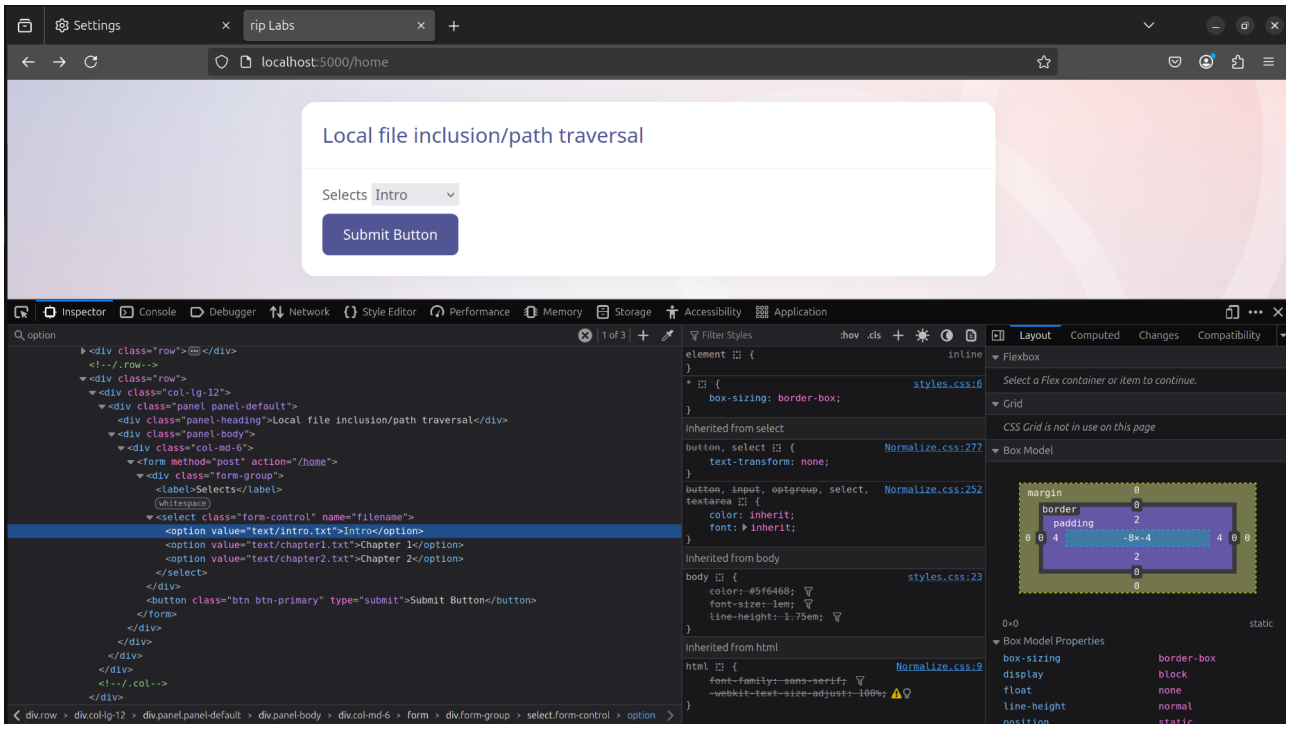
- we got the usernmae and password of the admin hehe
- why it's dangerous? for many reasons, for example attackers can extract sensitive data such as usernames, passwords, credit card numbers, and personal information from the database
- possible mitigation: using parameterized queries to separate SQL code from user input, preventing attackers from injecting malicious SQL

Path Traversal

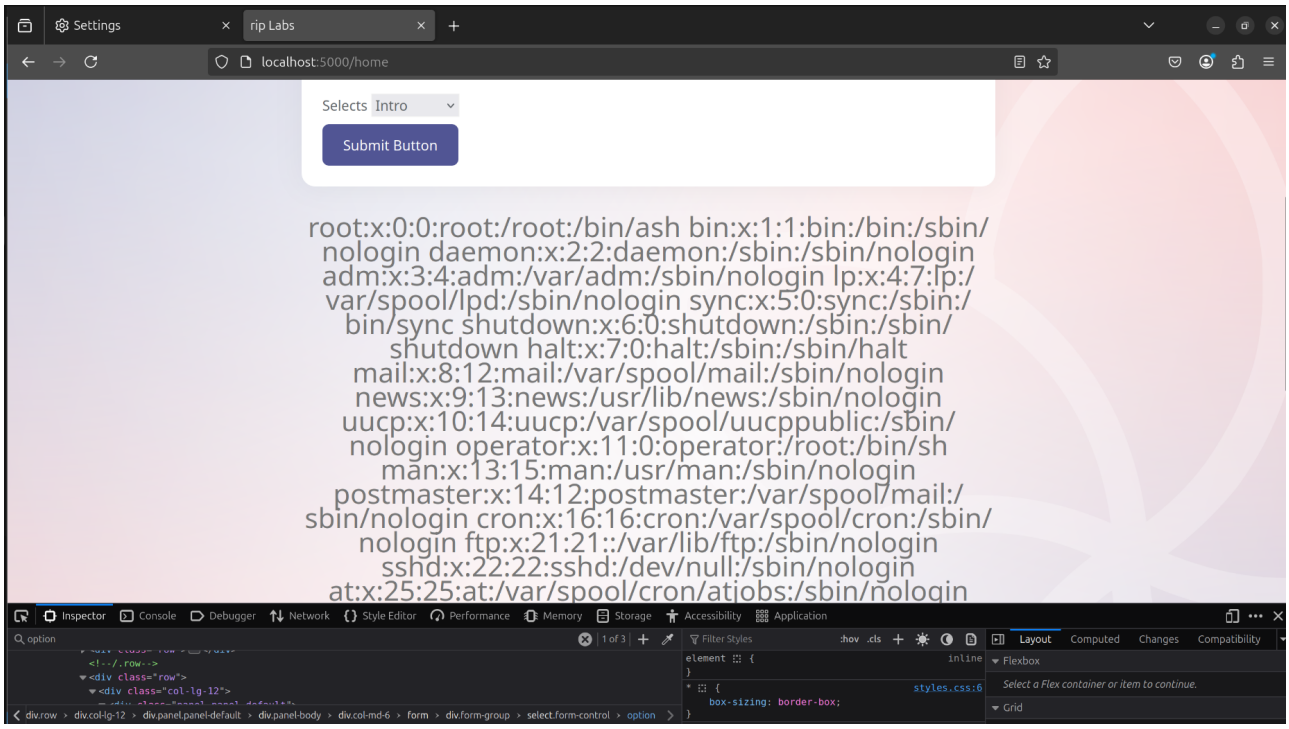
- let's pull and run the docker image:



- after we inspect the page, we can see that each option is a path for a file in the system

- 

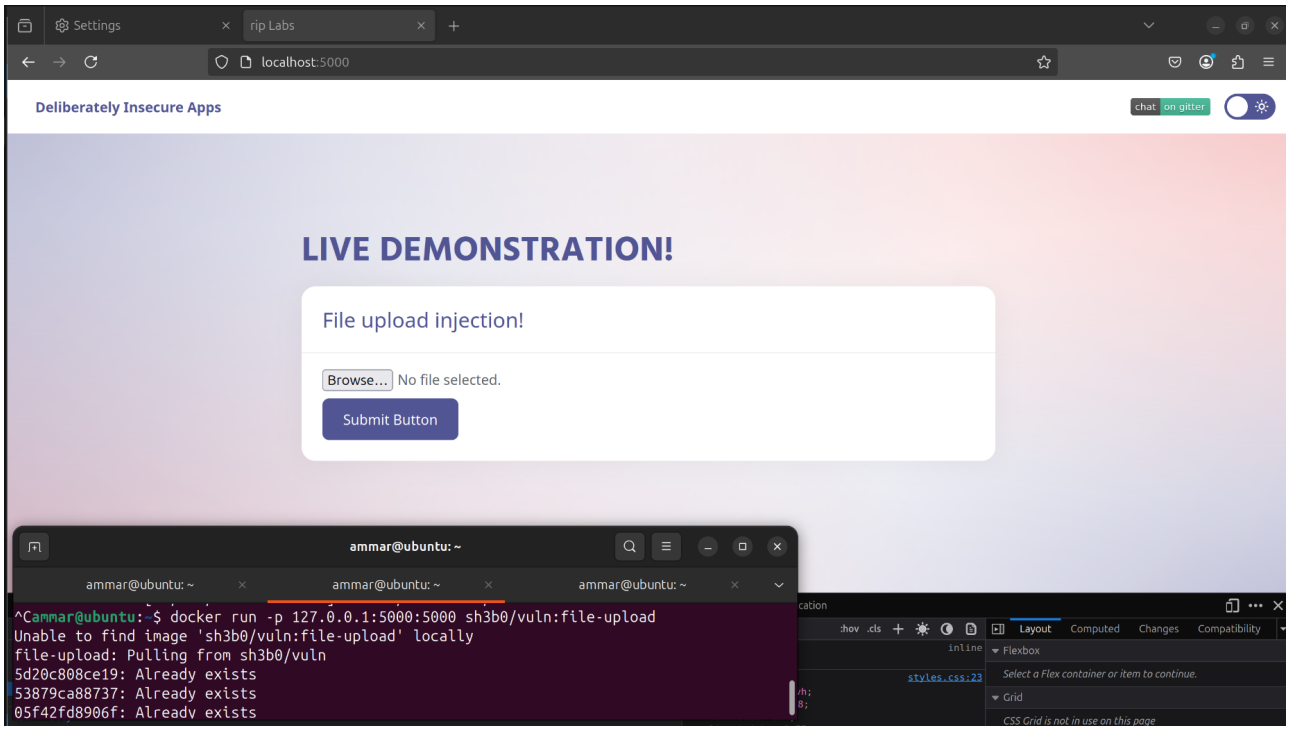
- after modifying the first option value to `../../../../../../../../../../etc/passwd` we can see:
- 



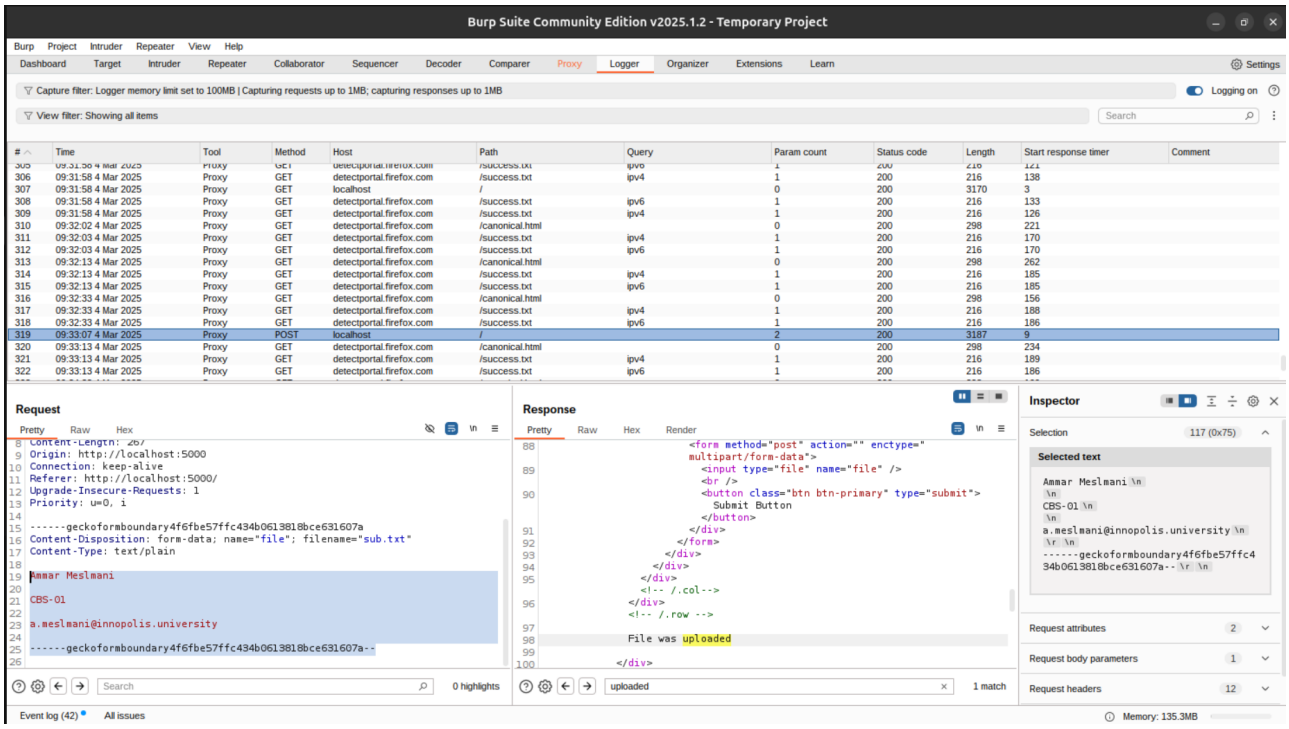
- why it's dangerous? because attackers can access sensitive files such as `/etc/passwd` (as we already did) .env, or database credentials and expose sensitive information
- possible mitigation: we need to ensure that user input is properly sanitized and validated before being used to access files

File Upload

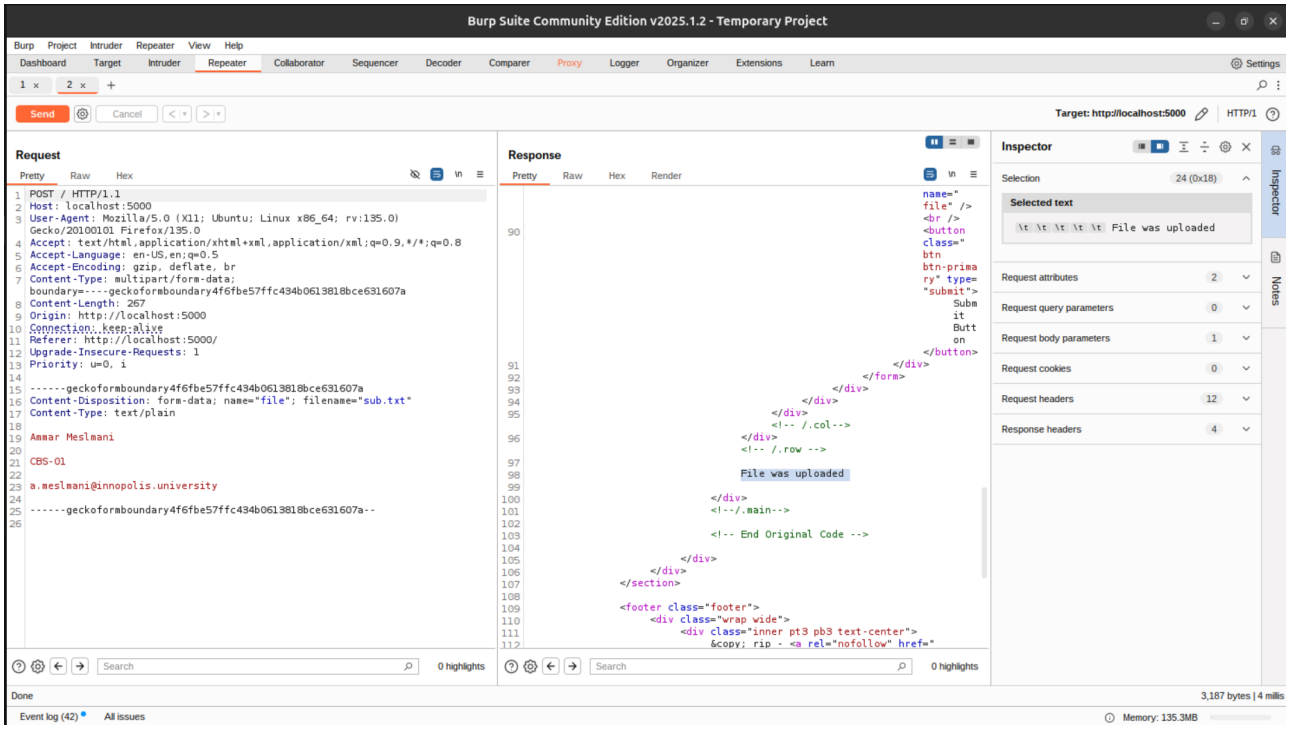
- let's pull and run the docker image:



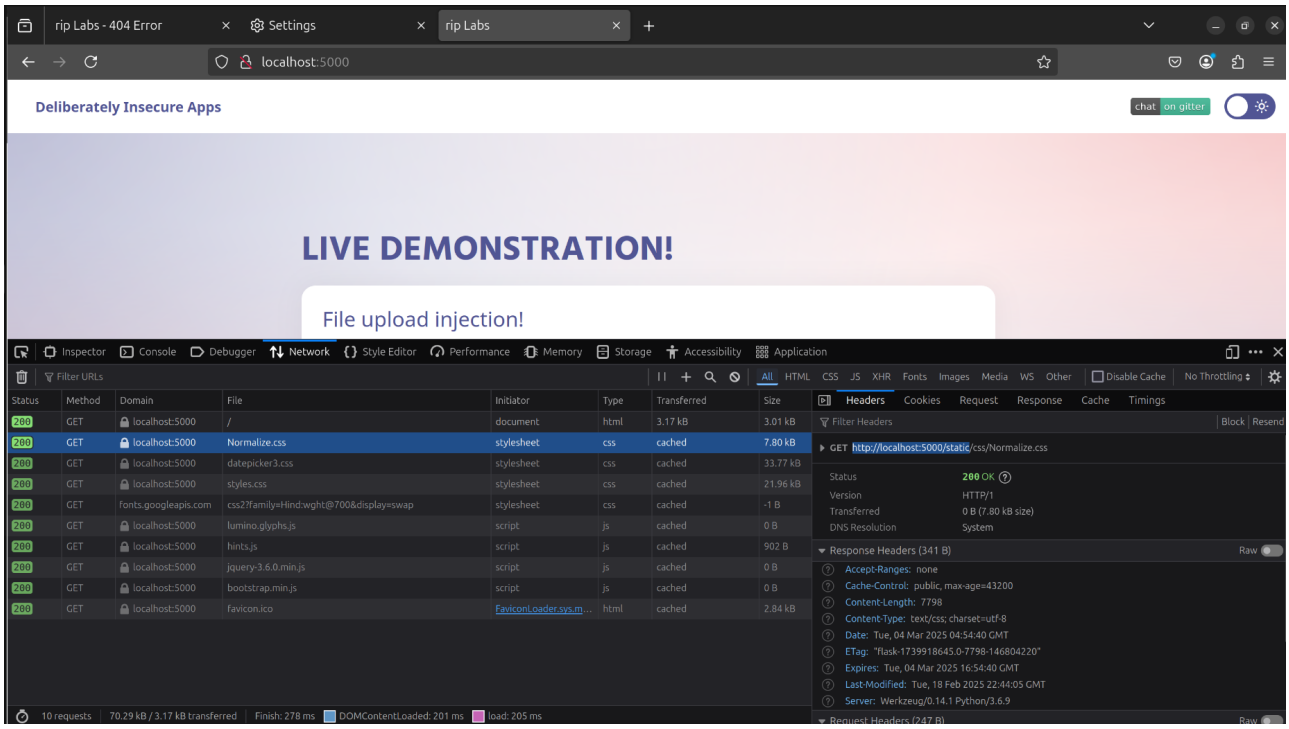
- let's try to upload a text file, and let's observe `burp`



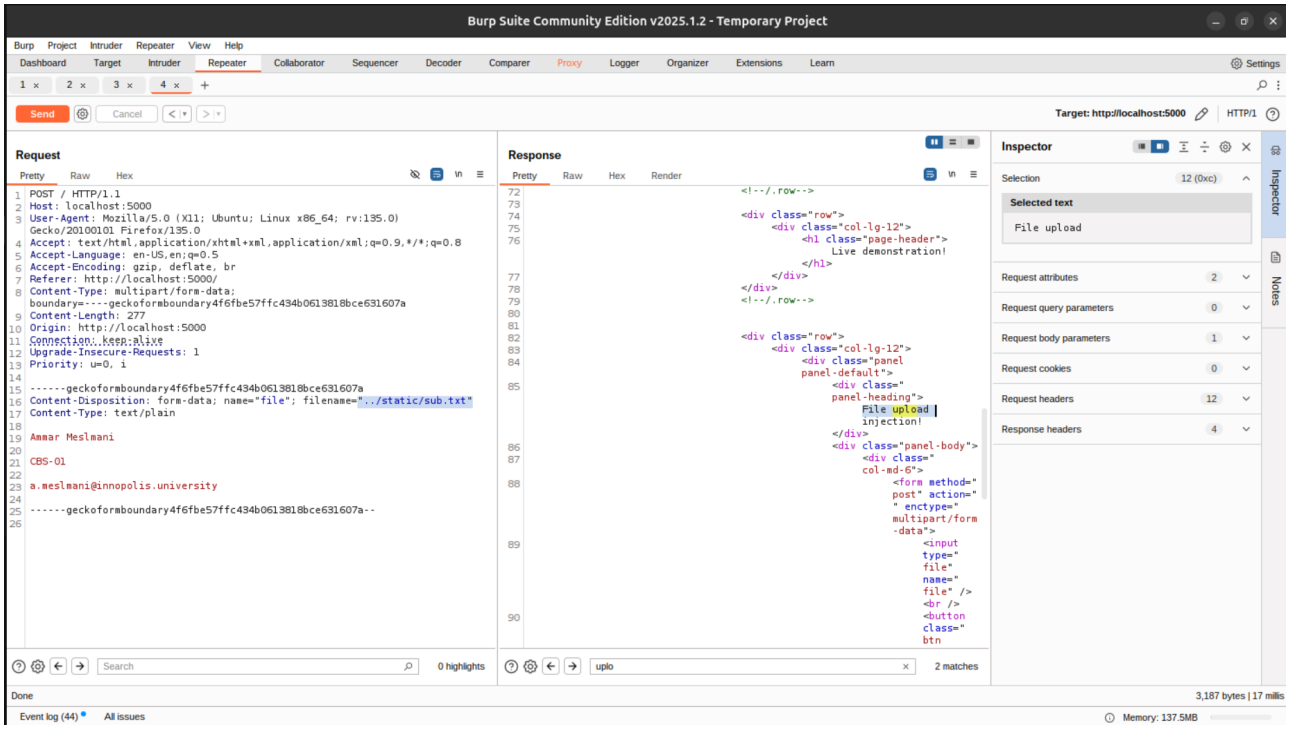
- we can see that the button is performaing a POST request, let's inspect this request



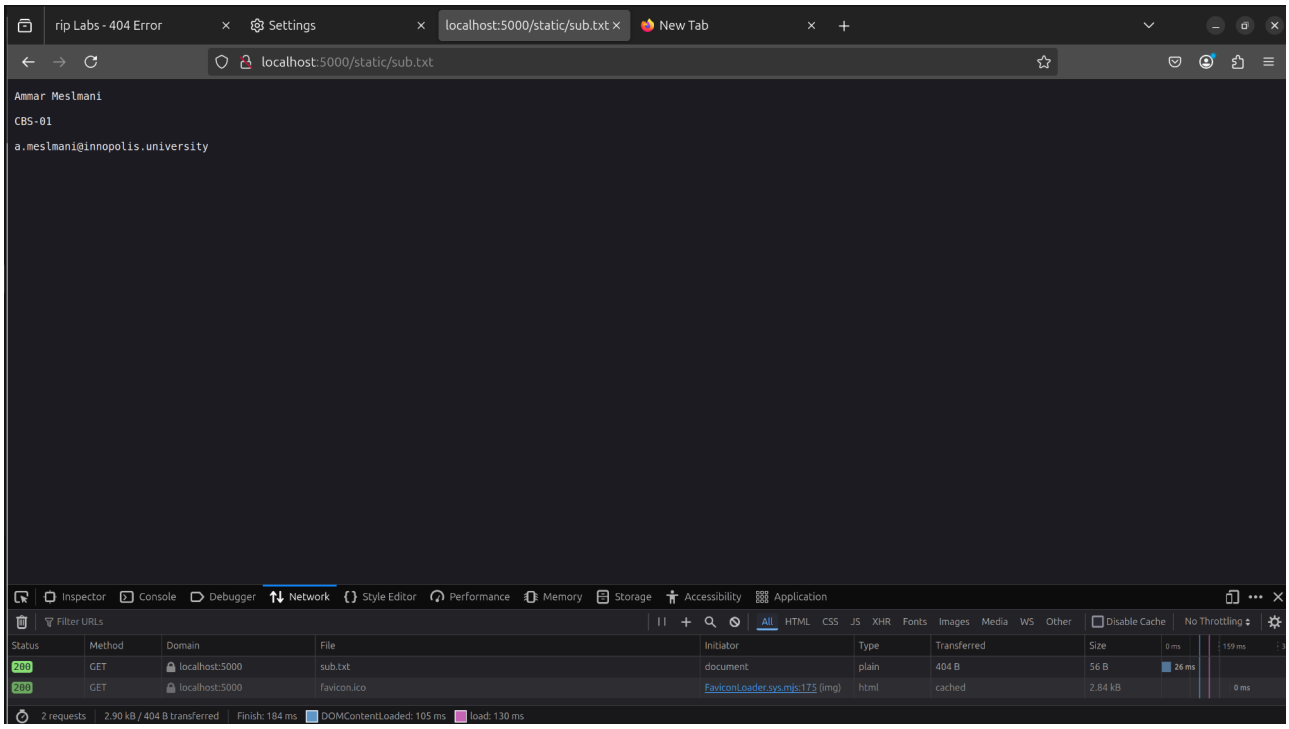
- we can see that our file got uploaded on the server, but we don't know where its locaiton
- let's inspect the headers in the network tab



- we can see that there is a folder called `/static`
- let's try to repeat our request but with changing the path of the file, and let's aim to upload it to `/static` folder



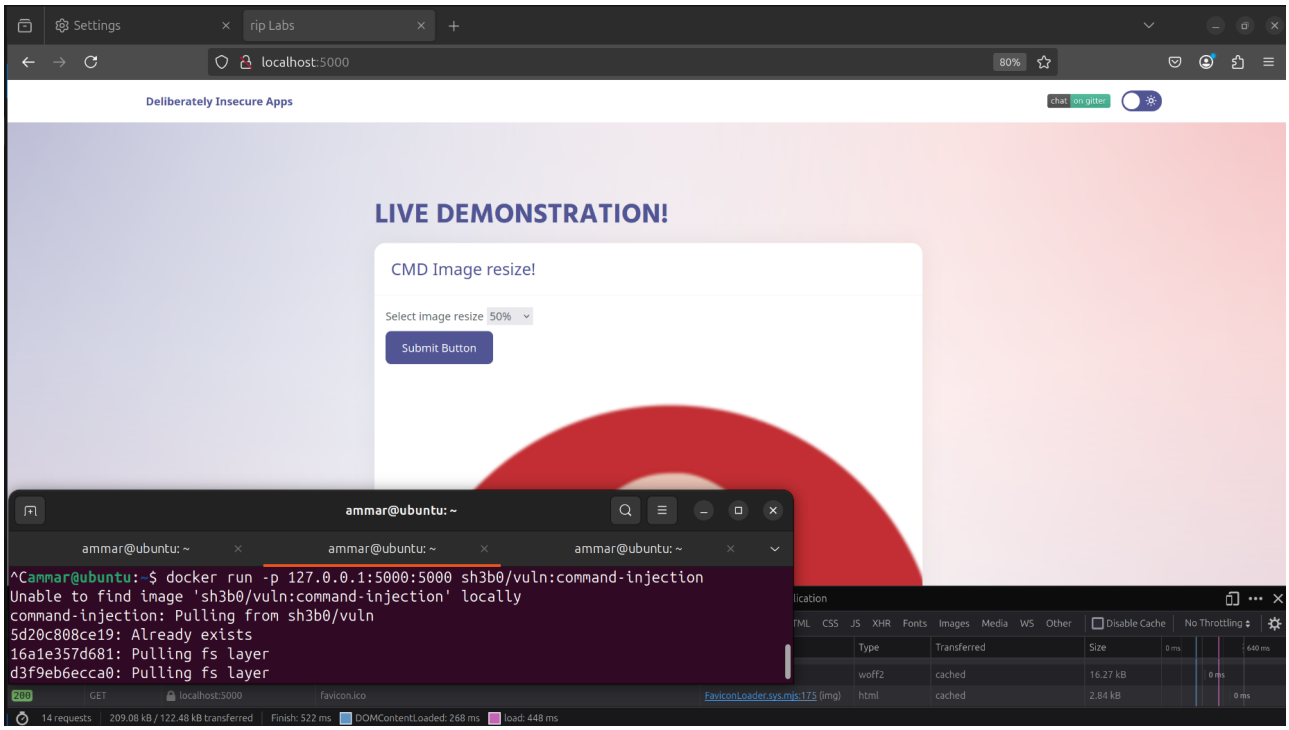
- after many trials, we succeeded!
- let's try to access the file



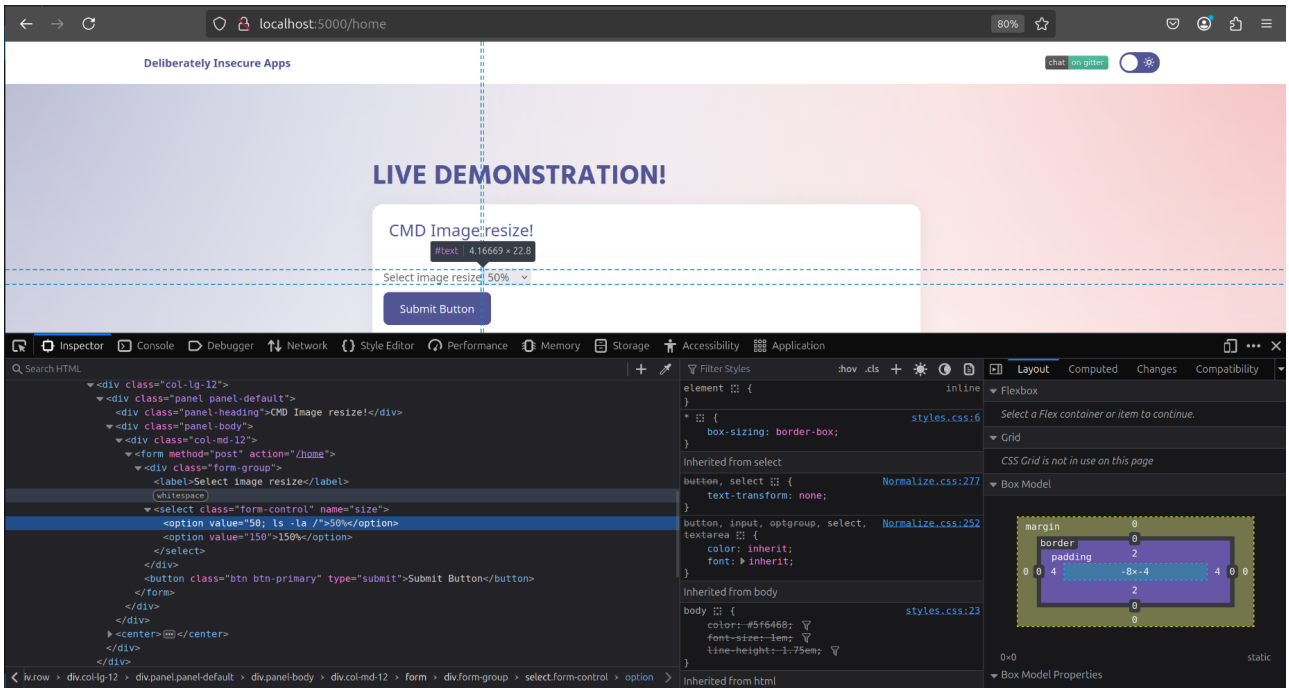
- why it's dangerous? because attackers can upload their own malicious codes and execute them on the server and cause damage or steal sensitive information, get shell access, or even perform XSS attack
- possible mitigation: the file types allowed to be uploaded should be restricted to only those that are necessary for business functionality. moreover, never accept a filename and its extension directly without having an allow list filter to avoid possible harm (for example, why a user should be able to upload a script to website of downloading and uploading images)

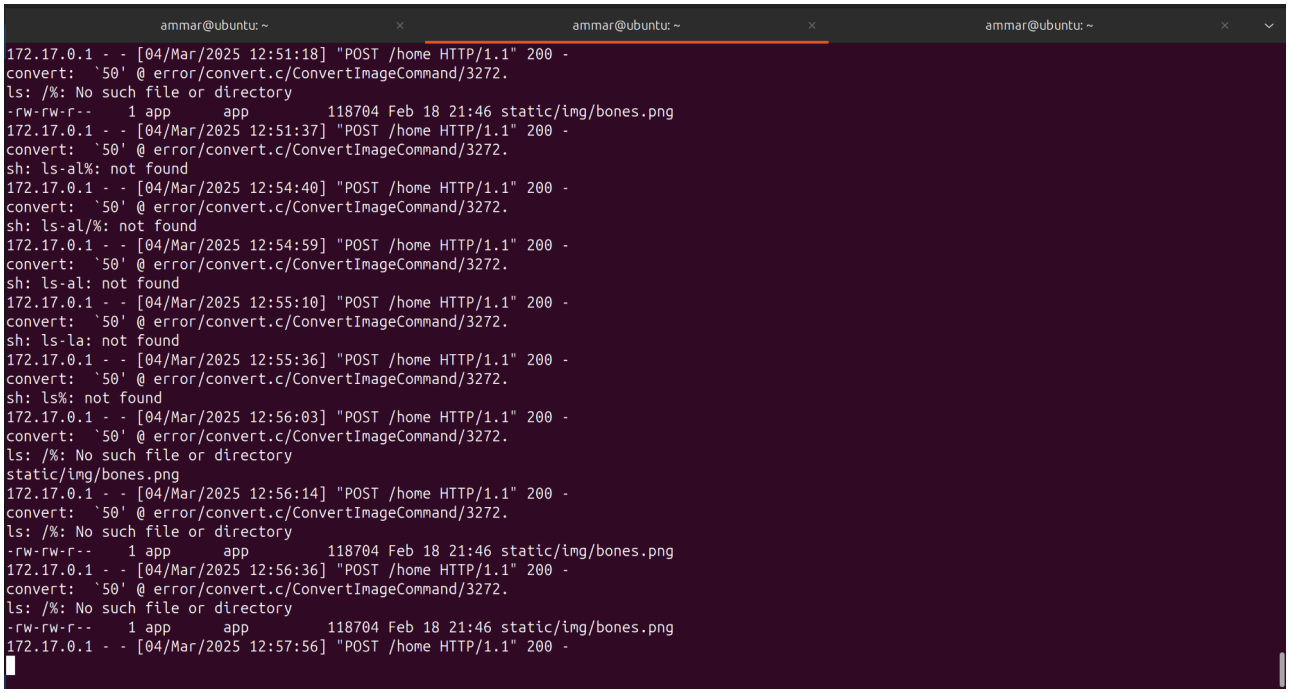
Command Injection

- let's pull and run the docker image:

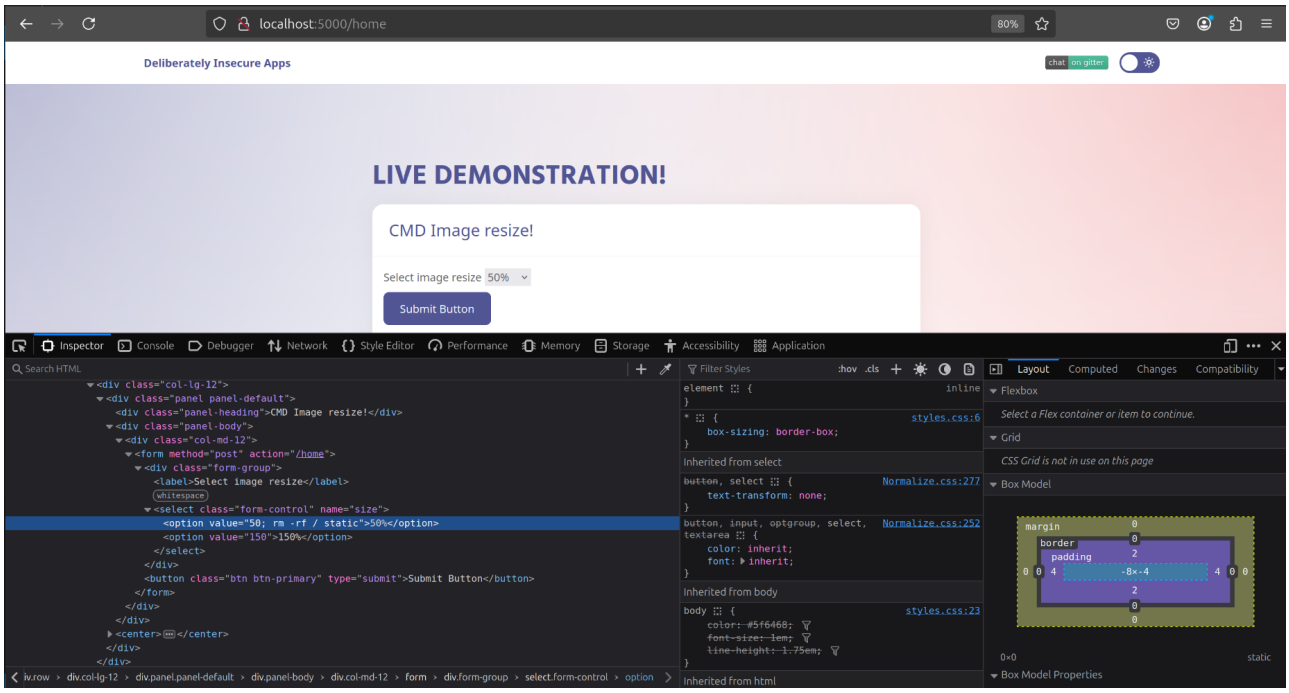


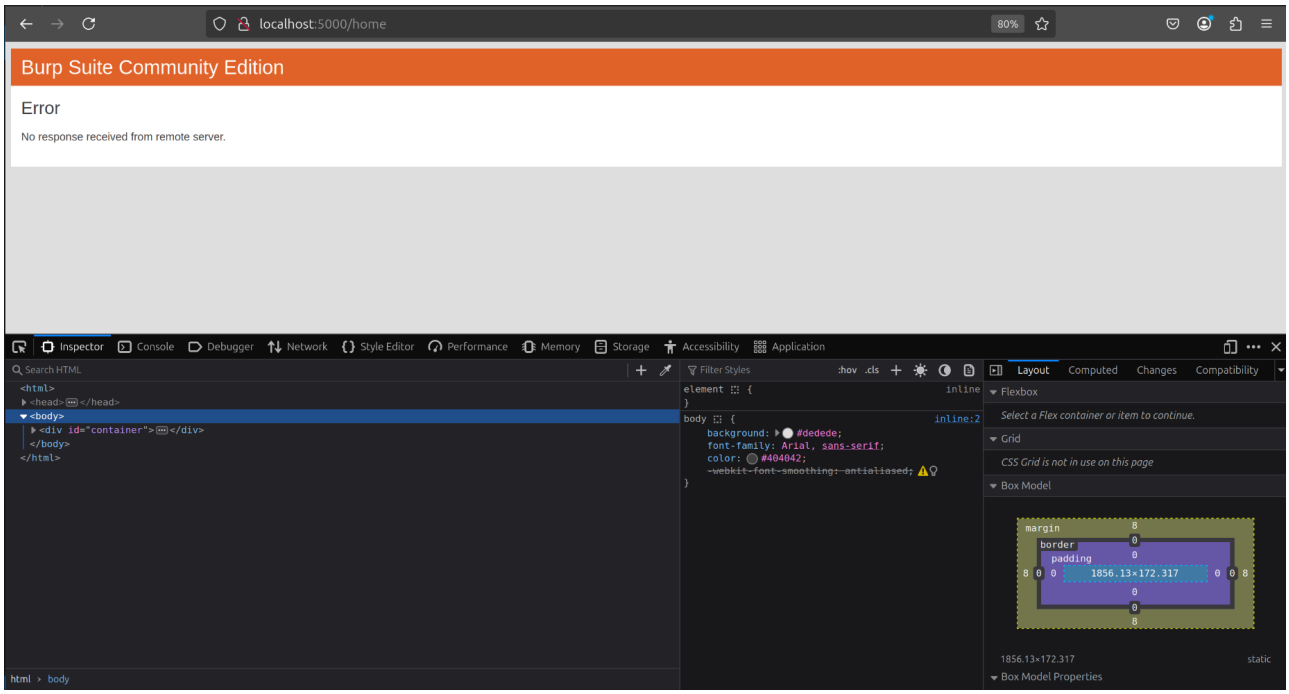
- let's try to inject the following command `50; ls -la /`:

- 

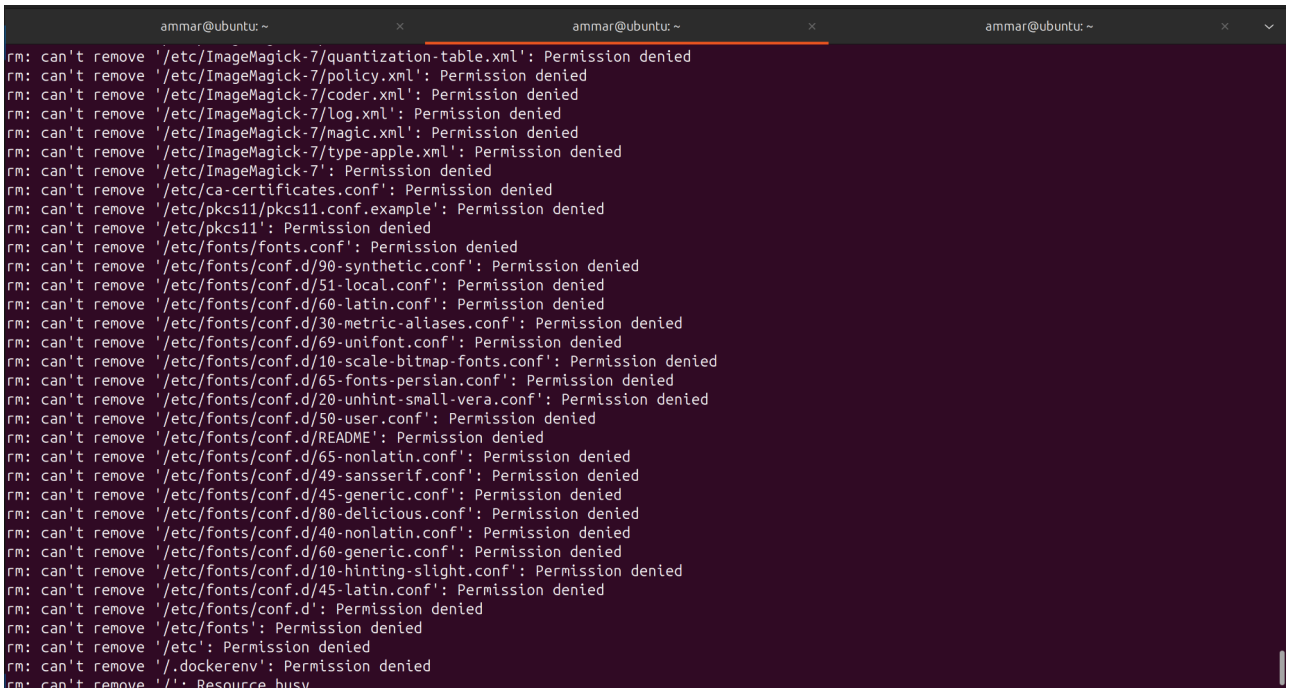
- let's check the terminal (i executed it many times to check the correct syntax):
- 

- let's try to inject a command to recursively remove all files in `/static`:

- 

- 

- let's check the terminal:



- baaang! we exposed the whole system structure and its files

- why it's dangerous? because it allows an attacker to execute arbitrary system commands on a server leading to full system compromise, data theft, and destruction of files
- possible mitigations:
  - avoiding executing system commands
  - useing parameterized Inputs
  - using secure APIs instead of shell execution