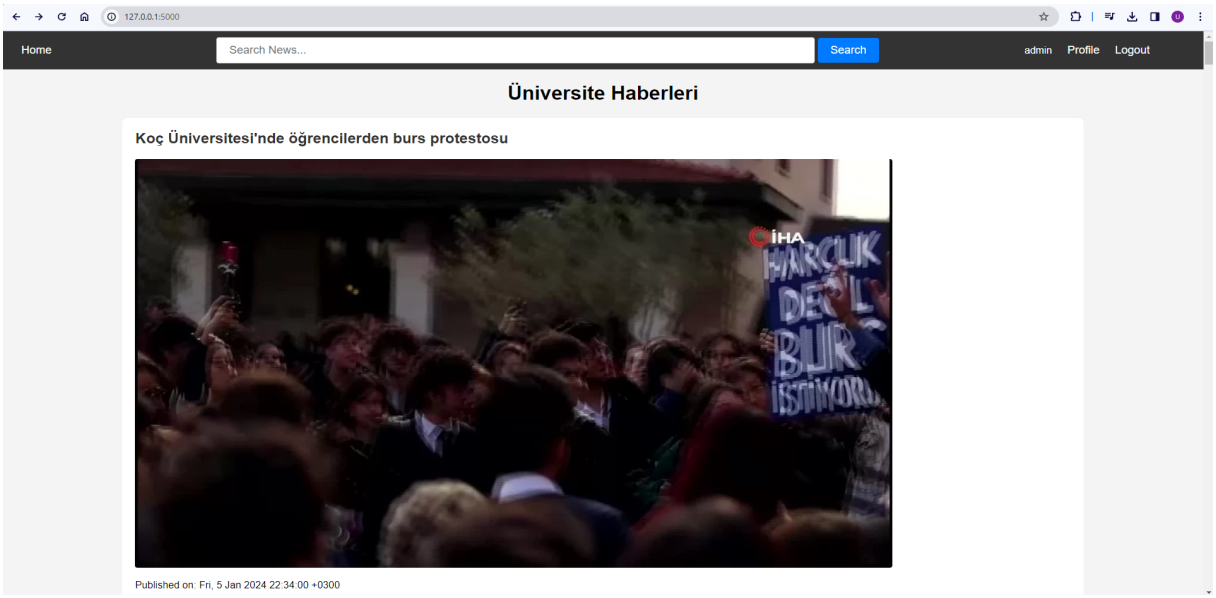
1. Website Basics

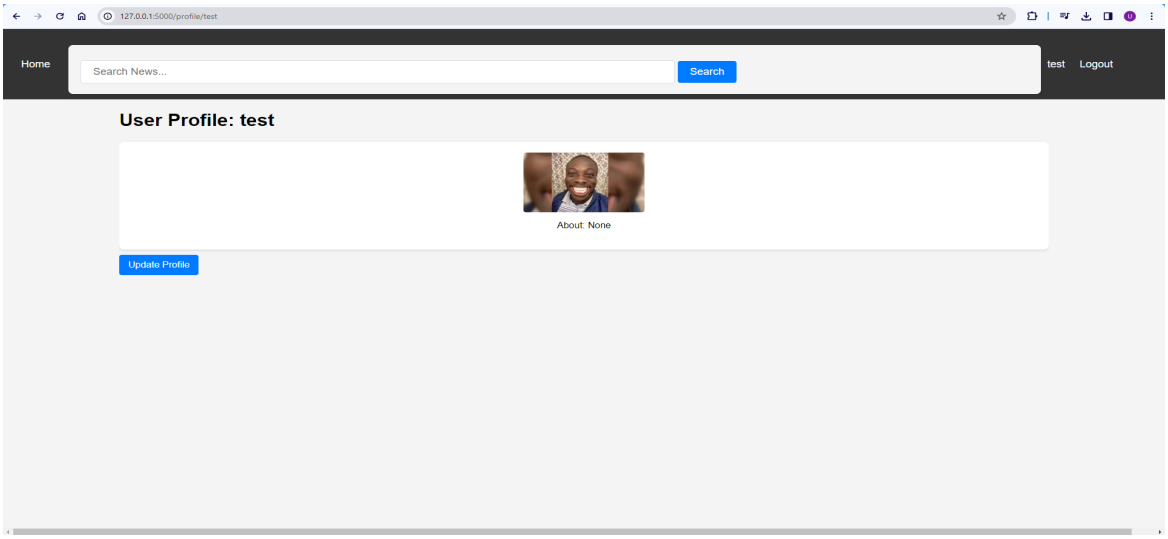
1.1 Main Page(Home Page):



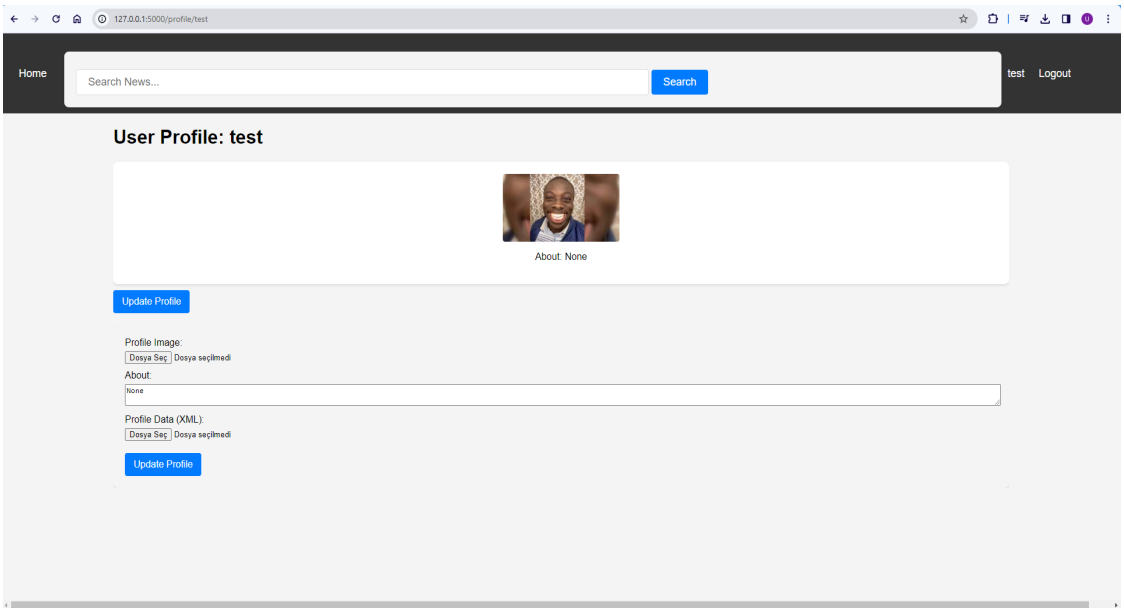
On the main page, we only show the title and picture of the news from the RSS feed. The navbar consists of 4 elements If the user logs in it can log off, go to its profile page, go to the home page, or search news/users via the search bar. If users are not logged in they can log in or register. He/She can view the comments and news but cannot comment or see other user's profiles.

1.2 Profile Page:

The profile page consists of the same navbar as the logged-in user and the profile information of the desired user. Other users can see the profile picture, username, and the about section of the user.

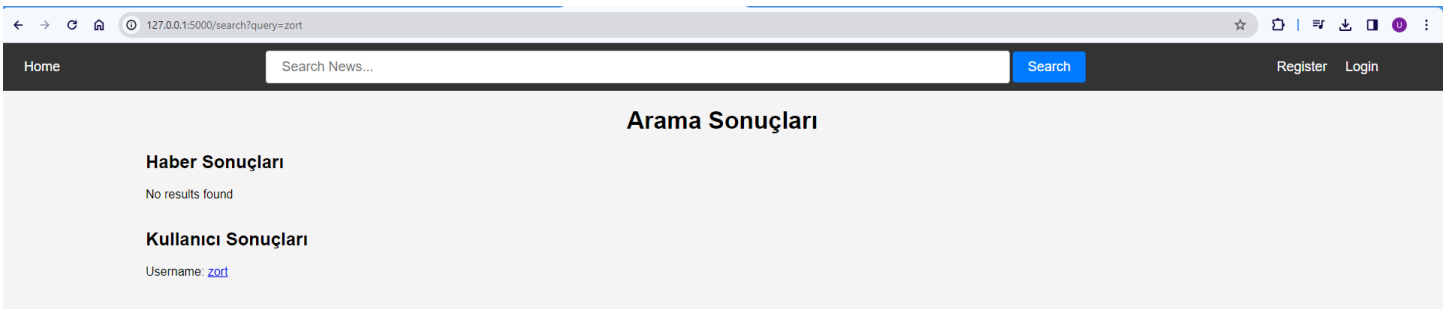


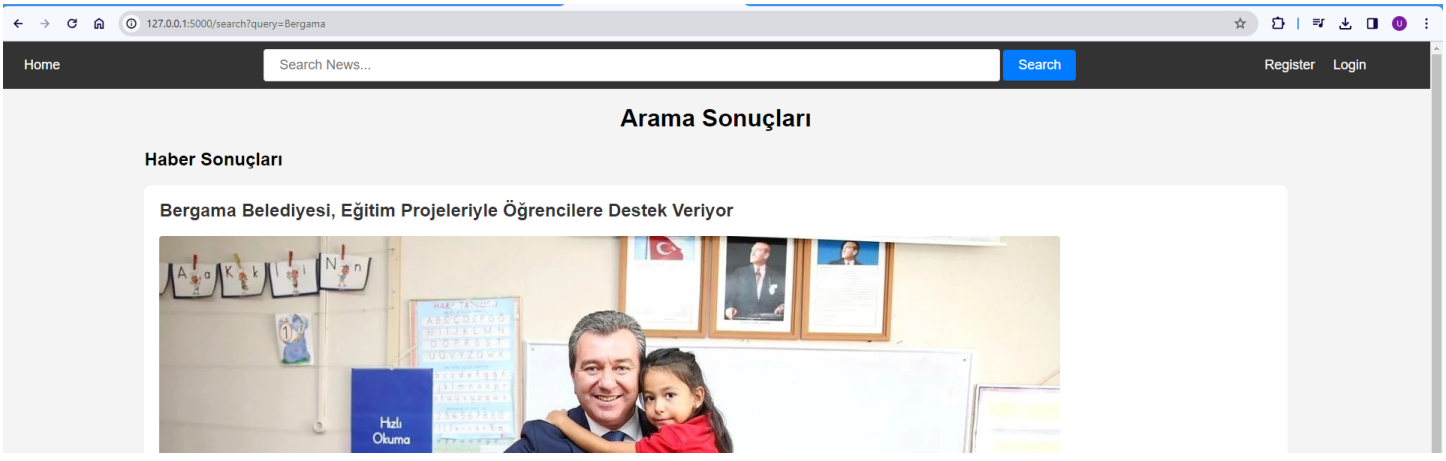
If the user goes into it’s profile He/She can update the profile picture and about section. There are two options for updating the profile user can choose a picture file and write the About section in the desired text box or submit an XML file that consists of picture and about data in it.



1.3 Search:

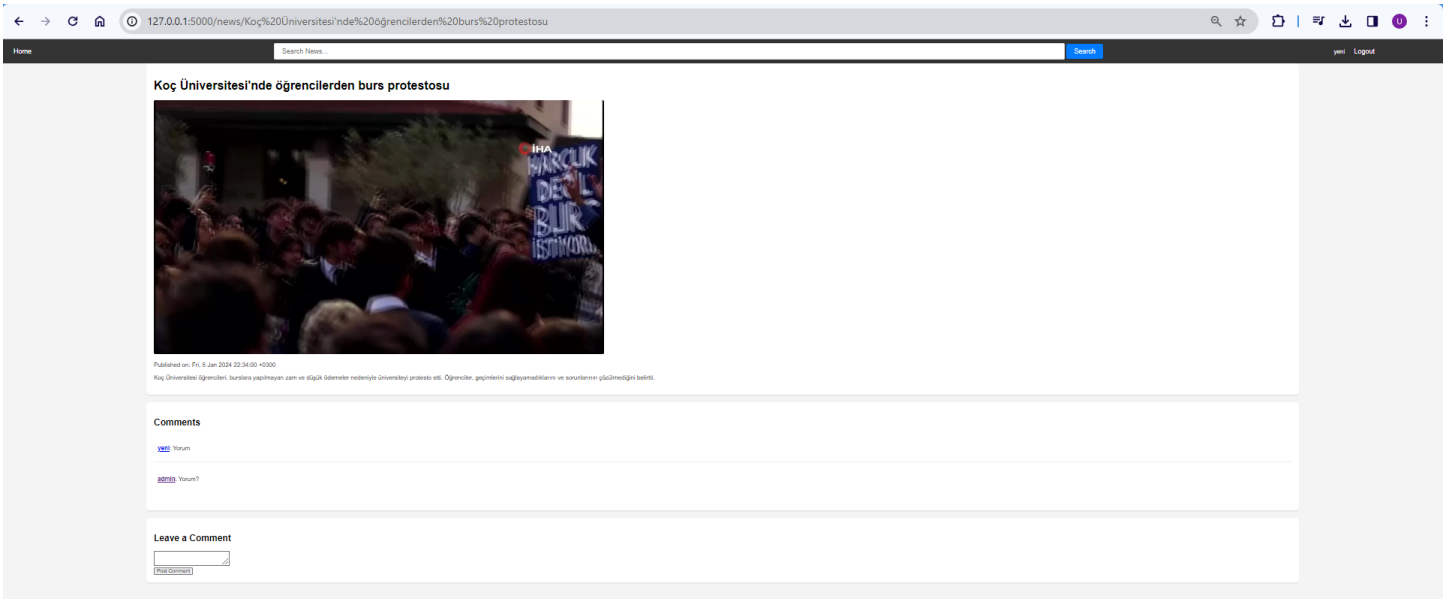
Search bar is for searching for news or users. It will show the clickable news on the main page and the user name with a redirect link to that person’s profile.





1.4 News Detail:

When users click any news on the main page they will redirected to the news detailed page. Here There He/She can view a description of that new and comments and if logged in can submit a comment.



1.5 Logging:

We logged every user activity as verbose as possible. Here is an example of one of the activity logs:

2024-01-06 09:03:35 INFO: Activity: {

'eventid': 'authentication',

'message': 'Request received',

'url': '/login',

'timestamp': '2024-01-06T06:03:35Z',

'unixtime': 1704521015.9724767,

'src_ip': '127.0.0.1',

'src_port': 35015,

'dst_port': '5000',

'request': 'POST',

'user_agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',

'accept_language': 'tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7',

'dst_ip': '127.0.0.1',

'form_data': {'username': 'admin', 'password': 'admin'}}

Timestamp (2024-01-06 09:03:35 INFO): This is the date and time when the event was logged, along with the log level indicating the importance or type of the log (INFO signifies general information).

Activity (Activity): This indicates that the log is recording an activity. In larger systems, you might see different categories like errors, warnings, etc.

Event ID (eventid: 'authentication'): This identifies the type of event. 'authentication' means the log pertains to an authentication process.

Message (message: 'Request received'): A brief description of the event. Here it indicates that an authentication request was received.

URL (url: '/login'): The specific endpoint or URL where the event occurred, indicating a login attempt.

Timestamp (timestamp: '2024-01-06T06:03:35Z'): The precise moment the event was logged in a standardized format.

Unix Time (unixtime): The time of the event in Unix time format, which is the number of seconds since January 1, 1970.

Source IP (src_ip): The IP address of the client or user making the request, often used for identifying the user's location or for security purposes.

Source Port (src_port): The port number on the client's machine used to send the request.

Destination Port (dst_port): The port number on the server that received the request.

Request Type (request): The type of HTTP request (GET, POST, PUT, DELETE, etc.). 'POST' is typical for login attempts as it contains user credentials.

User Agent (user_agent): Information about the client's browser or software making the request, useful for understanding compatibility or diagnosing issues.

Accept Language (accept_language): The preferred languages of the user's browser, which can be used for localization or understanding the user base.

Destination IP (dst_ip): The IP address of the server receiving the request.

Form Data (form_data): The actual data sent in the request, often including user input or selections. Here it includes a username and password, which are typical for a login form.

1.6 Privileges:

1.6.1: Guest: Guests cannot comment or see other users profile. They can only view the news and the comments.

1.6.2: User: Users can see other user's profiles. Can change its profile and leave comments below any news.

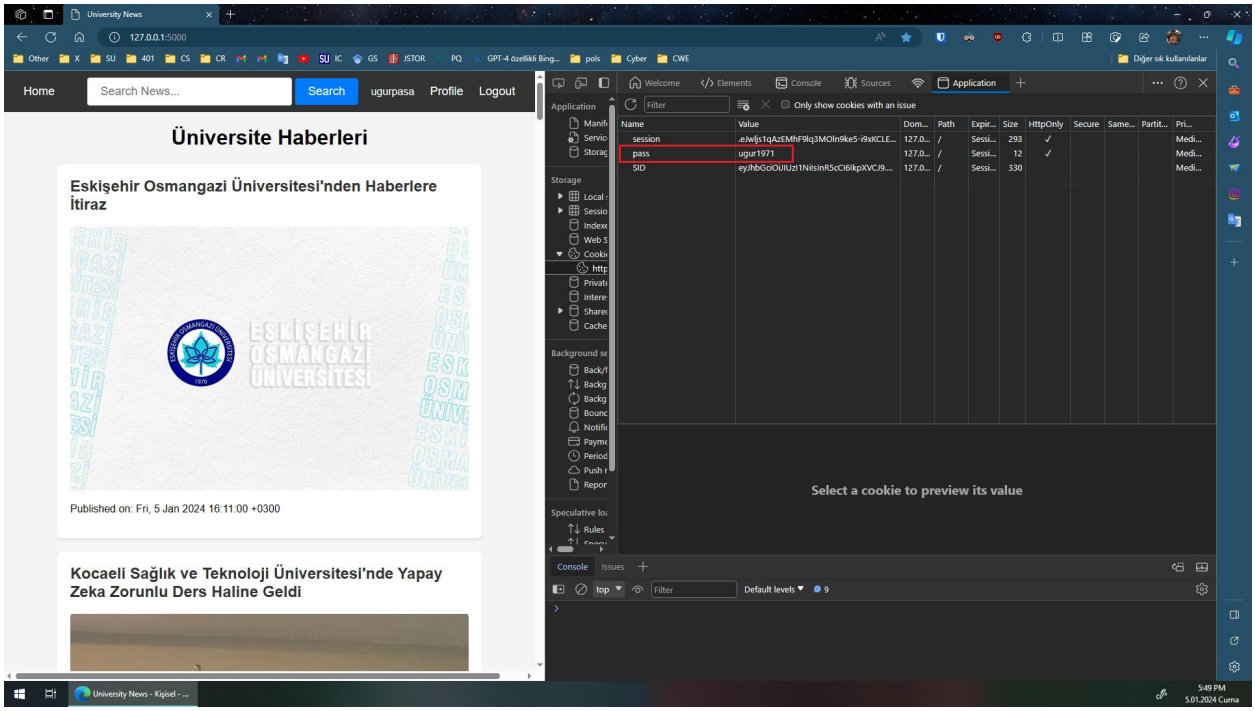
1.6.3: Admin: Admins can delete comments from users. Can delete a user permanently from the website.

2. VULNERABILITIES

2.1 CWE-315:

Our program exhibits a vulnerability by storing sensitive information, specifically user passwords, in cookies. Upon user login, the program responds with a cookie labeled 'pass,' containing the user's password in plaintext. Although the cookie is flagged as HttpOnly to prevent client-side scripts from accessing it, the inherent risk lies in storing unhashed passwords in cookies. To mitigate this, it is recommended to employ a more secure authentication mechanism, such as token-based authentication, and storing hashed passwords securely on the server.

```
514
515         response.set_cookie('pass', password, httponly=True)
516
```



2.2 XML External Entities:

Our program accepts XML input to update user profiles, including specifying profile picture URLs. We use an XML parser that is configured to allow DTDs (Document Type Definitions). This configuration can inadvertently introduce a security risk. Specifically, by allowing DTDs, attackers can define external entities within the DTD section of their XML input. These entities can be crafted to access sensitive files on the server, interact with internal systems, or exfiltrate data. For instance, an attacker might define an entity that references a file containing sensitive information (like '/etc/passwd' in Unix-based systems) and use it within the data that's processed by the application. This vulnerability, known as XXE, can lead to unauthorized data disclosure, service disruptions, or serve as a vector for further attacks if not properly mitigated.

```
def process_xml(xml_file):
    try:
        parser = etree.XMLParser(load_dtd=True, no_network=False)  # Insecure configuration
```

Example XXE payload:

<?xml version="1.0" ?>

<!DOCTYPE root [<!ENTITY read SYSTEM "file:///etc/passwd">]>

<profile>

<profile_image>&read;</profile_image>

<about>Testing XXE</about>

</profile>

Here attacker tries to get password data of the server. As result of this payload we can see this file written in user's profile_image section like this :

id	username	password_hash	is_admin	profile_image	about
3	zort	zort	f	e2b55a9b-7fcd-4369-9876-8918e5182cb6.jpeg	Zort bir yaşam tarzıdır
6	admin	admin	t	2a21385f-d511-468a-bcf8-0985d94000bc.jpeg	Welcome to Game
8	Utkualkan	Alkanutku	f	da6da958-f15e-4424-8b30-52cf0e3f2c3b.jpeg	Ponçik bir ayı
9	Süleyman5252	SızıntıYapma	f	e6589e17-25e0-45a8-8fd1-9b3959666619.jpg	Hayat kısa ama sen uzunsun
10	dila	dila	f		
11	yeni	yeni	f		XXE Test
1	test	test	f	##	Testing XXE
				# User Database	
				#	
				# Note that this file is consulted directly only when the system is running	
				# in single-user mode. At other times this information is provided by	
				# Open Directory.	
				#	
				# See the opendirectoryd(8) man page for additional information ab	

In this screen-shot we use this payload on test user.

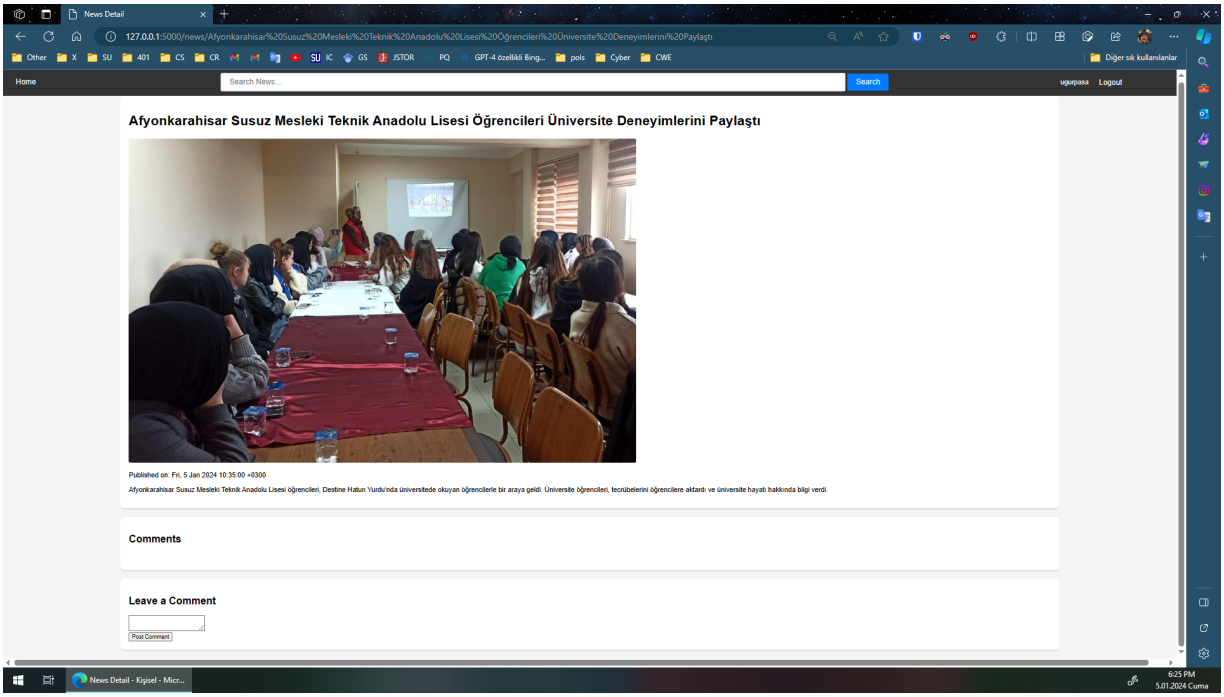
Also for logging purposes we log the entire XML payload into logs like this:

```
2024-01-06 18:24:14 INFO: Received XML content for user test: b'<?xml version="1.0" ?>\n<!DOCTYPE root [\n<!ENTITY read SYSTEM "file:///etc/passwd">\n]>\n<profile>\n    <profile_image>&read;</profile_image>\n
<about>Testing XXE</about>\n</profile>\n'
```

2.3 XSS (Cross-Site Scripting):

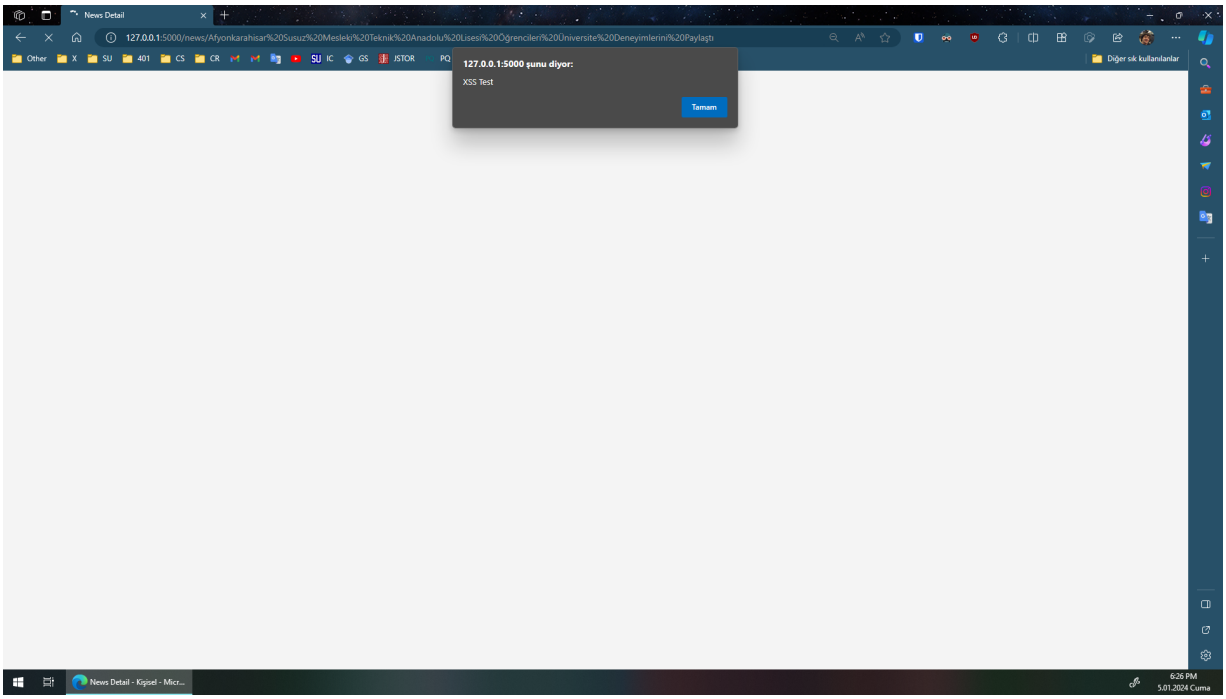
Our news website is susceptible to Cross-Site Scripting (XSS) due to improper handling of user comments. As users post comments, the program saves them directly to the database and refreshes the news page to display updated comments. The vulnerability arises from inadequate validation of entered user comment before storing into the database along with the structure of the HTML file's comment section which allows comments that are scripts to run. This is allowing users to inject malicious scripts. To address this, input validation and sanitization should be implemented, and user-generated content should be properly escaped before rendering it in the HTML. Here are some examples:

This is a news post with no comment:



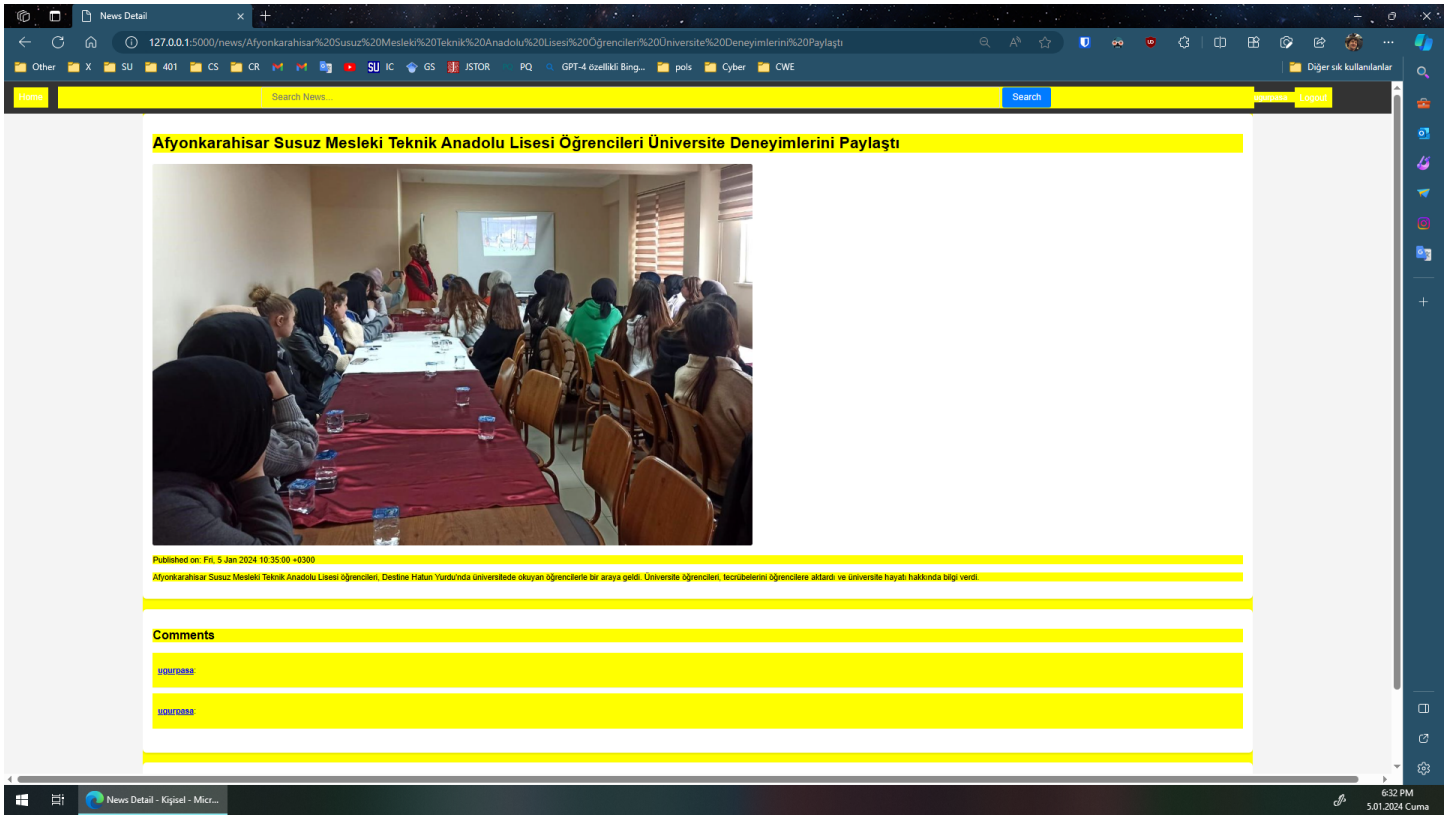
- I am entering following script as a command and post it: <script>alert('XSS Test');</script>

Whenever a user or a non-user opens this news post, the script will run:



- Another example is entering following script as a command and post it:
  <style> * { background-color: #FFFF00 } </style>

Whenever a user or a non-user opens this news post, the script will run:



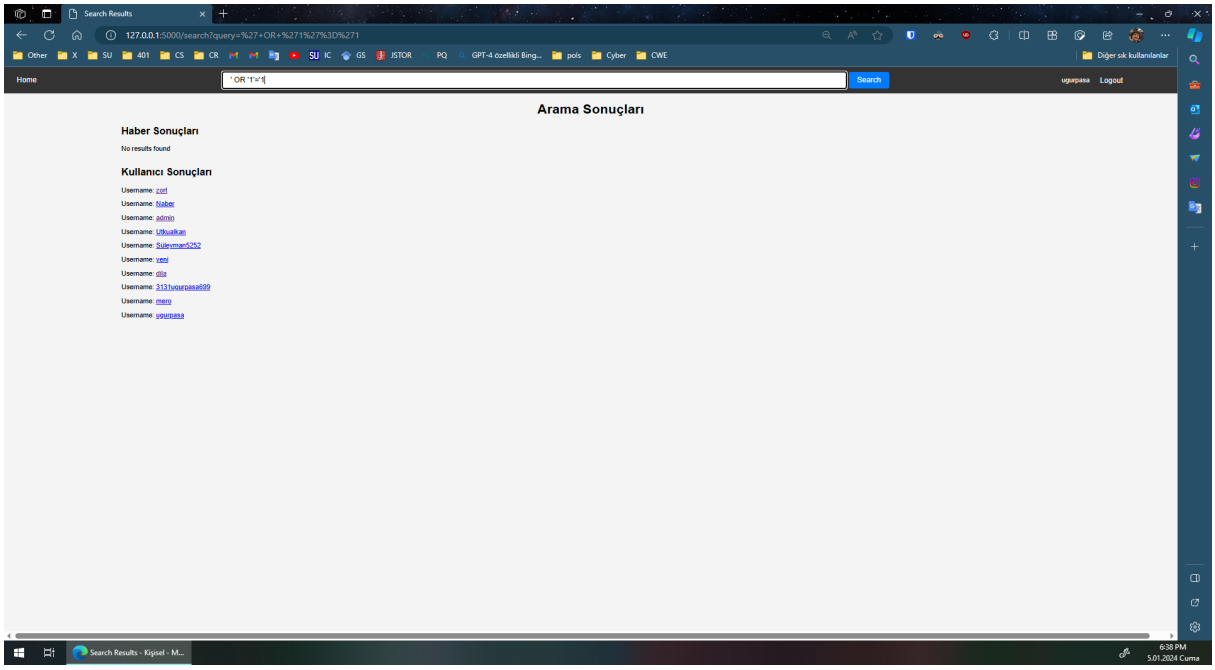
2.4 SQL Injection:

The program's search functionality poses a significant SQL Injection vulnerability. User-provided search queries are directly appended to a predefined SQL query without proper input validation or sanitization. Although the program restricts the execution of multiple SQL codes in a single query, it remains vulnerable to potential SQL injection attacks. Implementing parameterized queries or using ORM frameworks can help prevent SQL injection by ensuring that user input is treated as data, not executable code.

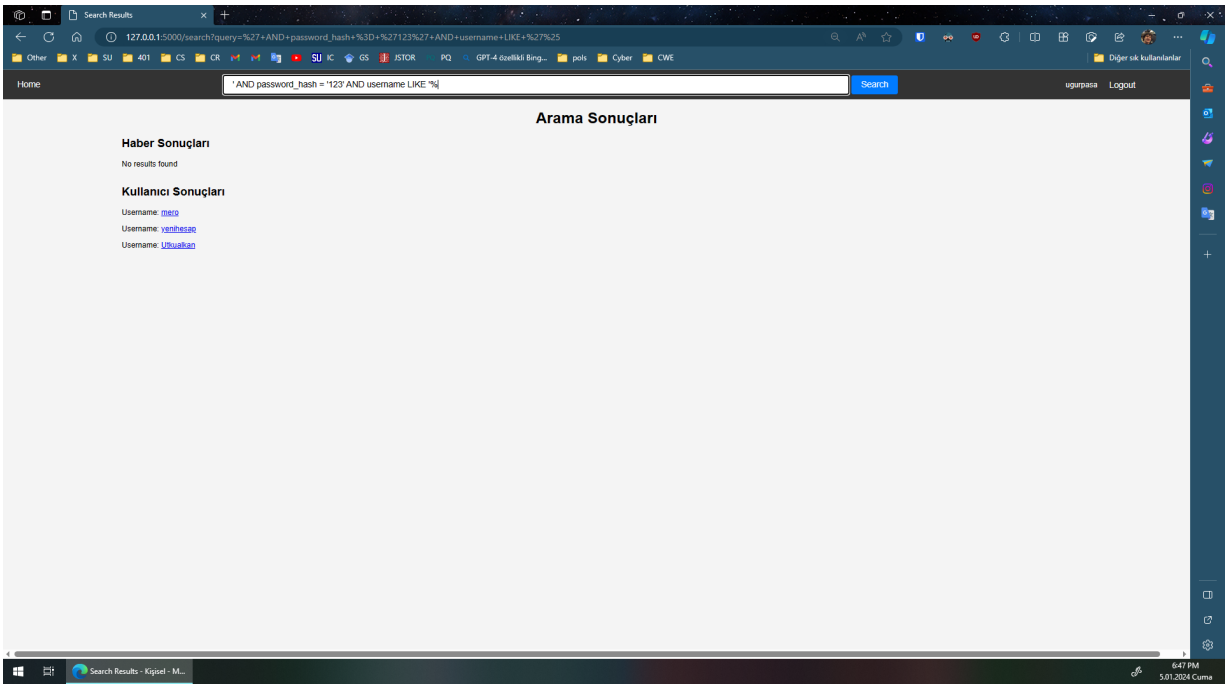
```
286
287         # Vulnerable SQL query
288         raw_sql = text(f"SELECT * FROM \"user\" WHERE username LIKE '%{query}%'")
289         print(f"q: [{raw_sql}]")
290         result = db.session.execute(raw_sql)
291         filtered_users = [dict(row._asdict()) for row in result]
```

Here are example exploits of this vulnerability:

- When I search “ ' OR '1'='1”, all users in the database are listed below. Although this situation may not seem very dangerous, listing all users is not a feature expected from a regular user.

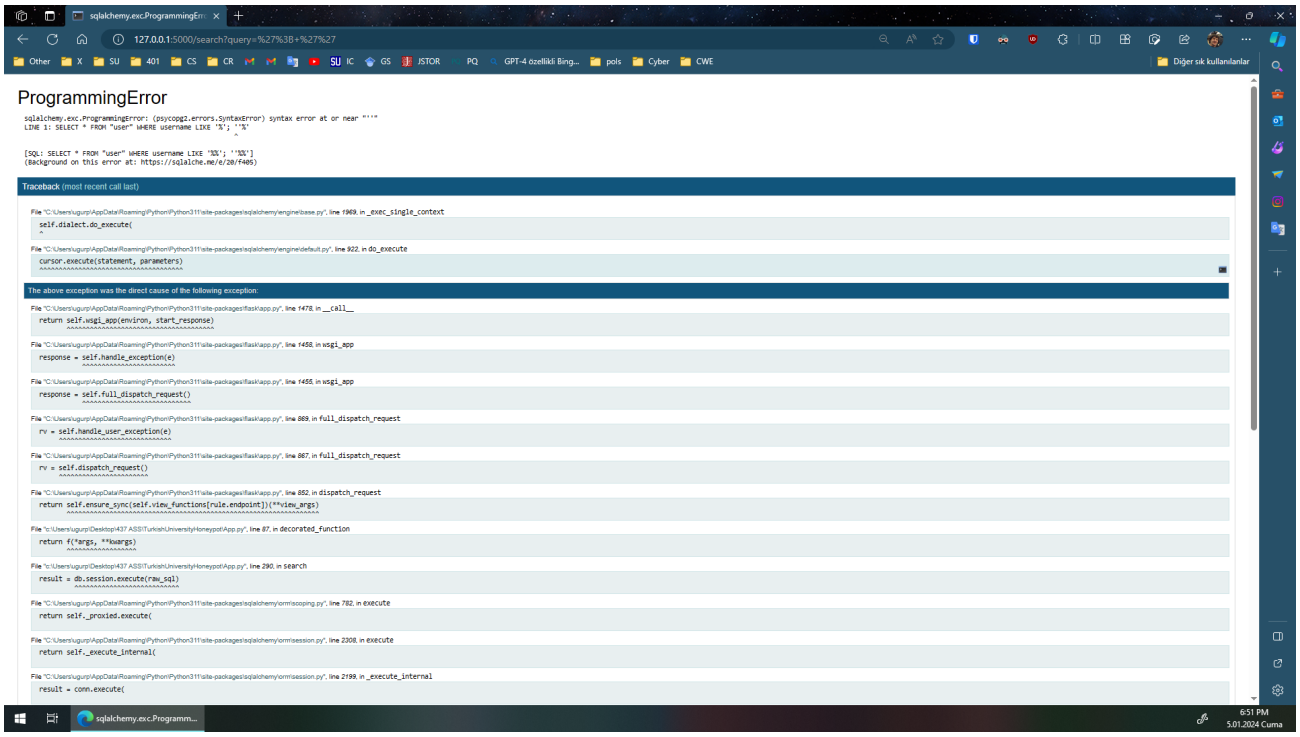


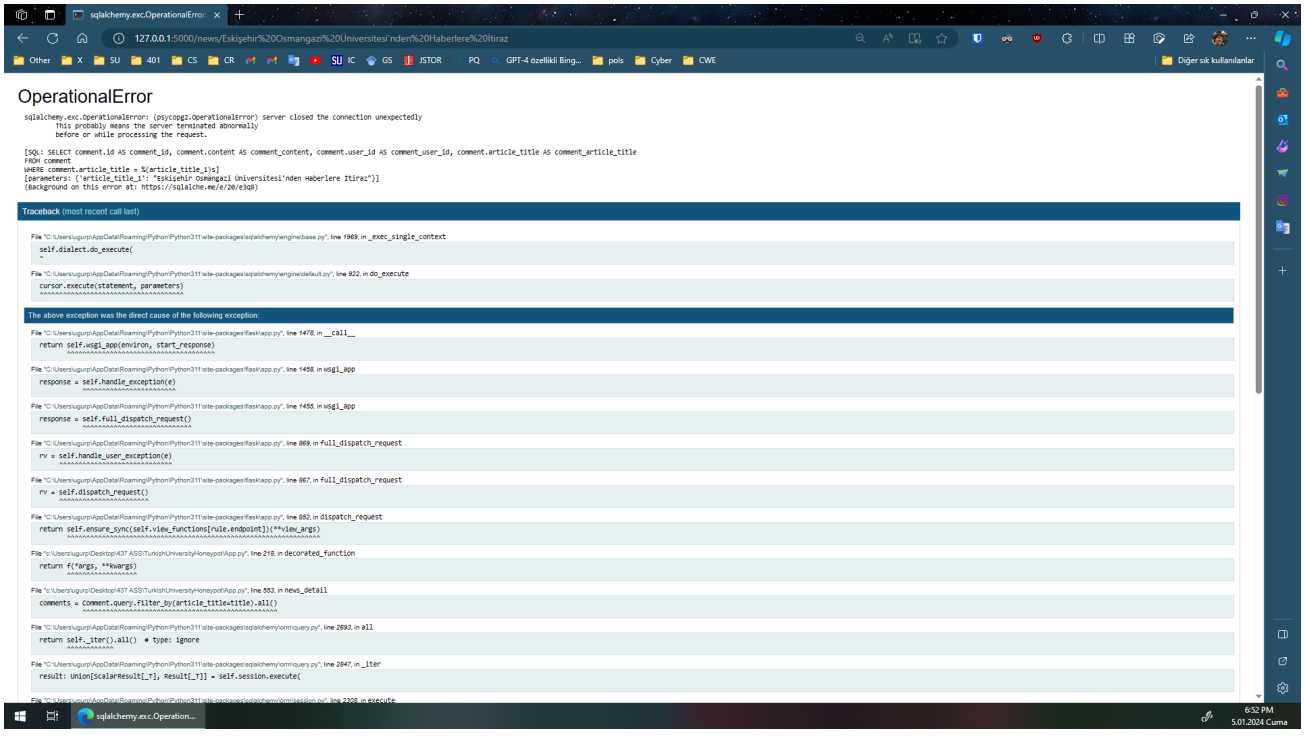
- Now, as another example exploit, I input “' AND password_hash = '123' AND username LIKE '%” into the search field. With this search, it lists all users with the password '123' for me. Since the passwords are not hashed, they can be collected through brute force along with usernames.



2.5 CWE-756: Missing Custom Error Page

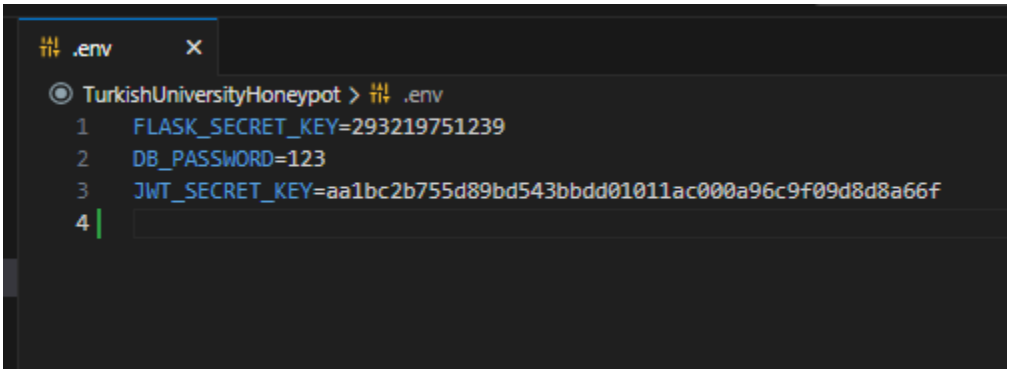
Our program lacks custom error pages, relying on Flask's default error pages. This exposes potentially sensitive information, including components of the source code or valuable database information. Implementing custom error pages for various scenarios can enhance security by providing generic error messages to users while concealing implementation details. This reduces the risk of unintentional information disclosure during error scenarios. Here are examples:





2.6 CWE-260: Password in Configuration File

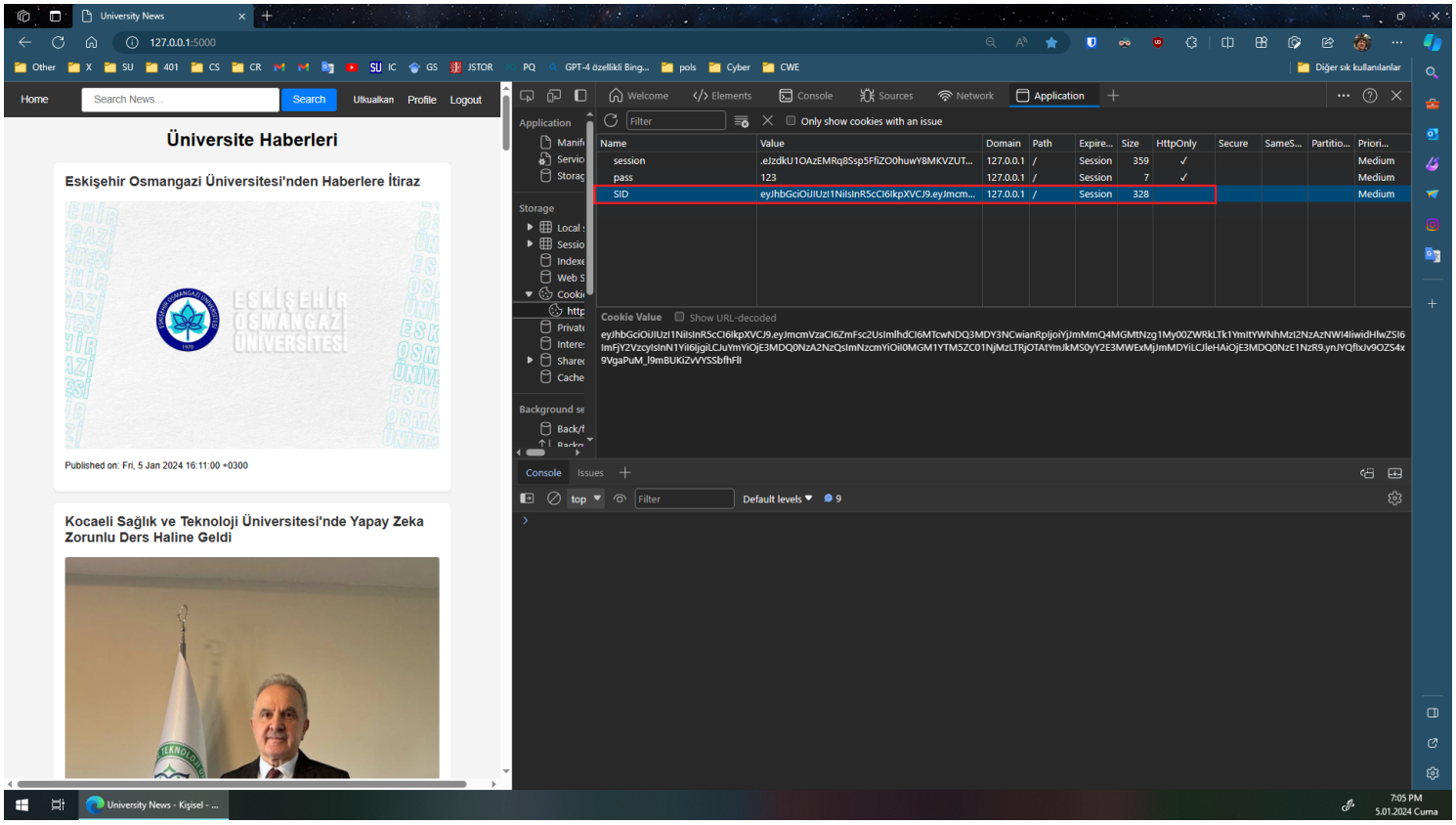
In our program, there are some keys and passwords that are used while the program is running. For example, the secret key of flask application, JWT token secret key and Postgresql database's password. These sensitive information are stored in another plain text file called '.env' and each time the application launches, they are read from that file and used in the program. This can result in compromise of the system for which the password is used. An attacker could gain access to this file and learn the stored password or worse yet, change the password to one of their choosing. Also, it is dangerous to allow developers of the application to see these important keys and passwords directly. Using a secure key management solution, encrypting sensitive configuration files, or utilizing environment variables to store sensitive information might help to mitigate this.



2.7 CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag

In our program, we have used JWT tokens for authorization. As a user is logged in, the program gives an access token to the user and this token is stored as a cookie. The problem is the HttpOnly flag is not set for the cookie.This creates a vulnerability where an attacker performing XSS could insert malicious scripts to steal the access token, as the cookie is not httponly. To enhance security, it is recommended to set the HttpOnly flag for cookies storing sensitive information, preventing client-side scripts from accessing them. This ensures that even if XSS vulnerabilities exist, the impact is limited in terms of token exposure.

```
510
511         access_token = create_jwt_token(user.id)
512
513         response.set_cookie('SID', access_token)
514
```



3. Database Configuration

We write a detailed guide for how to configure a database for this project in our git repository.

Github repository link: https://github.com/utkualkan4112/TurkishUniversityHoneypot

4. Static Code Analysis Tools

Bandit

```
C:\TurkishUniversityHoneypot-main>bandit -r C:\TurkishUniversityHoneypot-main
[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 3.11.7
Run started:2024-01-06 20:09:39.610750

Test results:
>> Issue: [B410:blacklist] Using etree to parse untrusted XML data is known to be vulnerable to XML attacks. Replace etree with the equivalent defusedxml package.
   Severity: Low   Confidence: High
   CWE: CWE-20 (https://cwe.mitre.org/data/definitions/20.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_imports.html#b410-import-lxml
   Location: C:\TurkishUniversityHoneypot-main\App.py:11:0
10      from sqlalchemy import text
11      from lxml import etree
12      import uuid

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Low
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: C:\TurkishUniversityHoneypot-main\App.py:288:23
287             # Vulnerable SQL query
288             raw_sql = text(f"SELECT * FROM \"user\" WHERE username LIKE '%{query}%'")
289             print(f"q: [{raw_sql}]")

--------------------------------------------------
>> Issue: [B320:blacklist] Using lxml.etree.parse to parse untrusted XML data is known to be vulnerable to XML attacks. Replace lxml.etree.parse with its defusedxml equivalent function.
   Severity: Medium   Confidence: High
   CWE: CWE-20 (https://cwe.mitre.org/data/definitions/20.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_calls.html#b313-b320-xml-bad-etree
   Location: C:\TurkishUniversityHoneypot-main\App.py:300:15
299             parser = etree.XMLParser(load_dtd=True, no_network=False)  # Insecure configuration
300             tree = etree.parse(xml_file, parser=parser)
301             root = tree.getroot()

--------------------------------------------------

Code scanned:
        Total lines of code: 417
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 2
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 1
                Medium: 0
                High: 2
Files skipped (0):
```

PYT (Python Taint)

```
C:\TurkishUniversityHoneypot-main>python -m pyt C:\TurkishUniversityHoneypot-main\App.py
11 vulnerabilities found:
Vulnerability 1:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 276, source "request.args.get(":
        ~call_1 = ret_flask.request.args.get('query', '')
Reassigned in:
       File: C:\TurkishUniversityHoneypot-main\App.py
        > Line 276: query = ~call_1
       File: C:\TurkishUniversityHoneypot-main\App.py
        > Line 288: ~call_3 = ret_sqlalchemy.text(f'SELECT * FROM "user" WHERE username LIKE '%{query}%'')
       File: C:\TurkishUniversityHoneypot-main\App.py
        > Line 288: raw_sql = ~call_3
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 290, sink "execute(":
       ~call_5 = ret_db.session.execute(raw_sql)
This vulnerability is unknown due to:  Label: ~call_3 = ret_sqlalchemy.text(f'SELECT * FROM "user" WHERE username LIKE '%{query}%'')

Vulnerability 2:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 276, source "request.args.get(":
        ~call_1 = ret_flask.request.args.get('query', '')
Reassigned in:
       File: C:\TurkishUniversityHoneypot-main\App.py
        > Line 276: query = ~call_1
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 295, sink "render_template(":
       ~call_7 = ret_flask.render_template('search_results.html', news=filtered_news, users=filtered_users, query=query)

Vulnerability 3:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 382, source "Framework function URL parameter":
        username
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 441, sink "url_for(":
       ~call_26 = ret_flask.url_for('profile', username=username)

Vulnerability 4:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 382, source "Framework function URL parameter":
        username
Reassigned in:
       File: C:\TurkishUniversityHoneypot-main\App.py
        > Line 441: ~call_26 = ret_flask.url_for('profile', username=username)
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 441, sink "redirect(":
       ~call_25 = ret_flask.redirect(~call_26)
This vulnerability is unknown due to:  Label: ~call_26 = ret_flask.url_for('profile', username=username)
```

```
Vulnerability 5:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 382, source "Framework function URL parameter":
         username
Reassigned in:
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 386: ~call_1 = ret_User.query.filter_by(username=username)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 386: __chain_tmp_1 = ~call_1
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 386: ~call_2 = ret___chain_tmp_1.first_or_404()
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 386: user = ~call_2
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 443, sink "render_template(":
       ~call_27 = ret_flask.render_template('profile.html', user=user)
This vulnerability is unknown due to:  Label: ~call_1 = ret_User.query.filter_by(username=username)

Vulnerability 6:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 463, source "Framework function URL parameter":
         comment_id
Reassigned in:
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 467: ~call_1 = ret_Comment.query.get_or_404(comment_id)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 467: comment = ~call_1
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 474, sink "url_for(":
       ~call_7 = ret_flask.url_for('news_detail', title=comment.article_title)
This vulnerability is unknown due to:  Label: ~call_1 = ret_Comment.query.get_or_404(comment_id)

Vulnerability 7:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 463, source "Framework function URL parameter":
         comment_id
Reassigned in:
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 467: ~call_1 = ret_Comment.query.get_or_404(comment_id)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 467: comment = ~call_1
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 474: ~call_7 = ret_flask.url_for('news_detail', title=comment.article_title)
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 474, sink "redirect(":
       ~call_6 = ret_flask.redirect(~call_7)
This vulnerability is unknown due to:  Label: ~call_1 = ret_Comment.query.get_or_404(comment_id)

Vulnerability 8:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 502, source "form[":
         password = request.form['password']
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 510, sink "set_cookie(":
       ~call_7 = ret_response.set_cookie('pass', password, httponly=True)
```

```
Vulnerability 9:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 501, source "form[":
         username = request.form['username']
Reassigned in:
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 504: ~call_1 = ret_User.query.filter_by(username=username)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 504: __chain_tmp_1 = ~call_1
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 504: ~call_2 = ret___chain_tmp_1.first()
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 504: user = ~call_2
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 47: save_8_user = user
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 512: temp_8_user_id = user.id
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 47: user_id = temp_8_user_id
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 48: ~call_10 = ret_str(user_id)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 48: ~call_9 = ret_flask_jwt_extended.create_access_token(identity=~call_10)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 48: access_token = ~call_9
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 49: ret_create_jwt_token = access_token
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 512: ~call_8 = ret_create_jwt_token
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 512: access_token = ~call_8
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 514, sink "set_cookie(":
       ~call_11 = ret_response.set_cookie('SID', access_token)
This vulnerability is unknown due to:  Label: ~call_1 = ret_User.query.filter_by(username=username)

Vulnerability 10:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 532, source "form[":
         article_title = request.form['article_title']
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 538, sink "redirect(":
       ~call_4 = ret_flask.redirect('/news/' + article_title)

Vulnerability 11:
File: C:\TurkishUniversityHoneypot-main\App.py
 > User input at line 547, source "Framework function URL parameter":
         title
Reassigned in:
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 551: ~call_2 = ret_next((item for item in feed.entries), None)
        File: C:\TurkishUniversityHoneypot-main\App.py
         > Line 551: selected_news = ~call_2
File: C:\TurkishUniversityHoneypot-main\App.py
 > reaches line 553, sink "render_template(":
       ~call_5 = ret_flask.render_template('news_detail.html', news_item=selected_news, comments=comments)
This vulnerability is unknown due to:  Label: ~call_2 = ret_next((item for item in feed.entries), None)
```

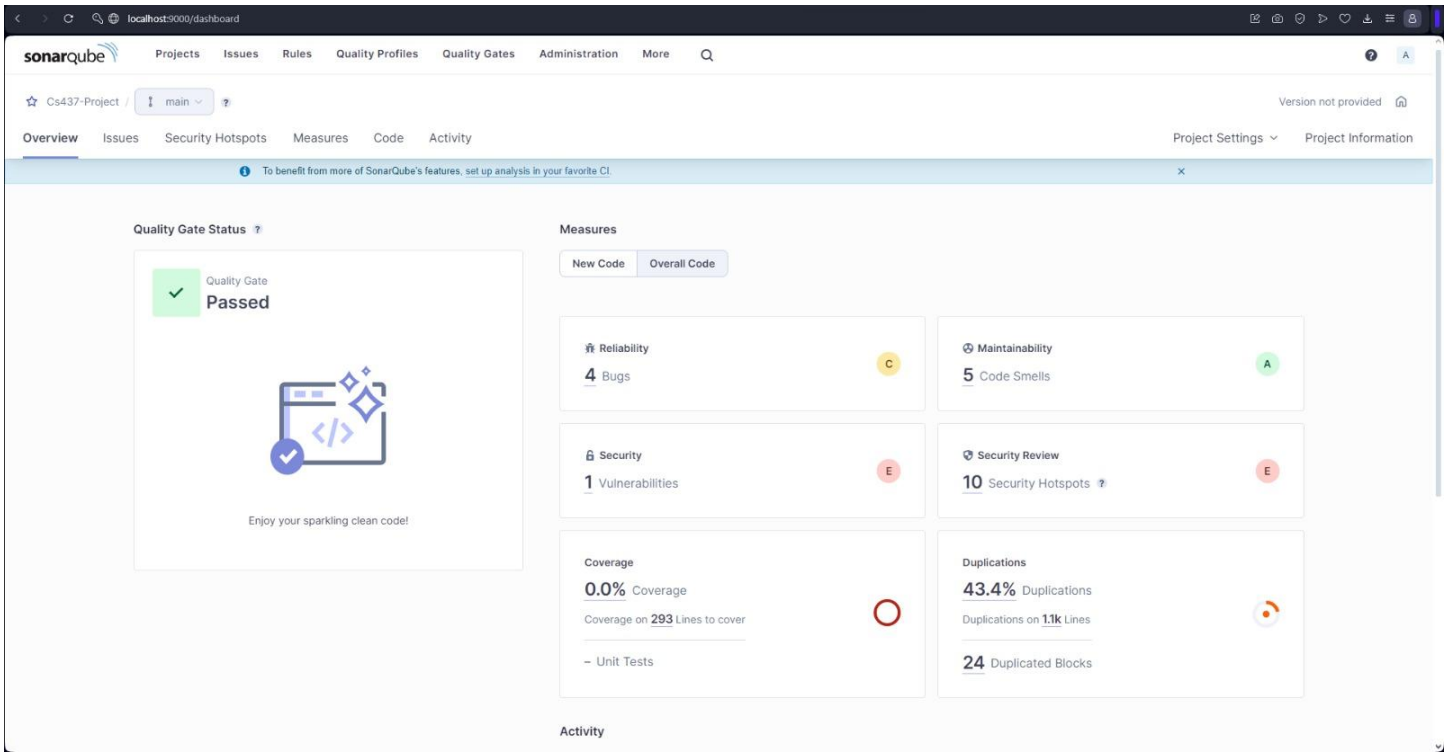
Rough-Auditing-Tool-for-Security (RATS)

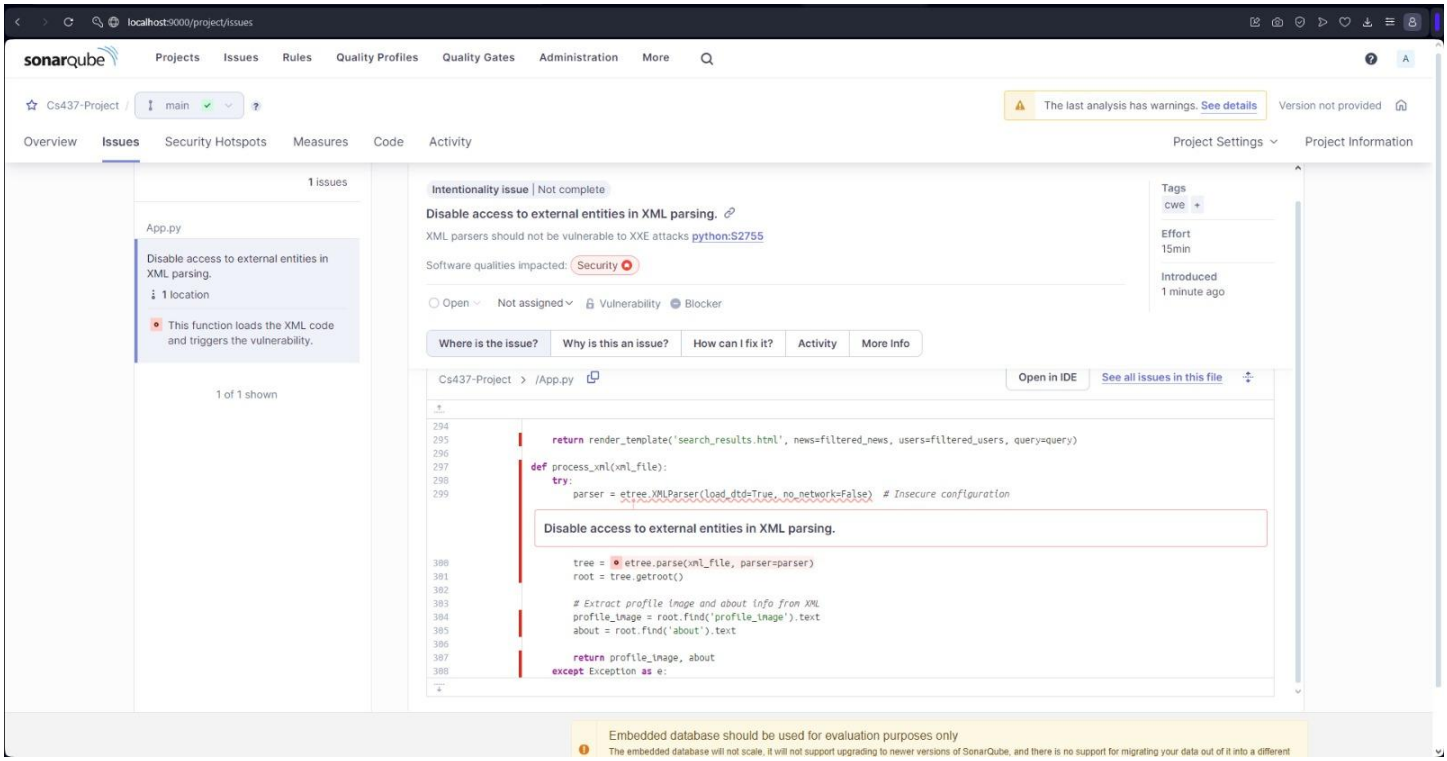
```
root@DESKTOP-QN8T1SL:/home/TurkishUniversityHoneypot# rats
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Total lines analyzed: 0
Total time 0.000004 seconds
0 lines per second
```

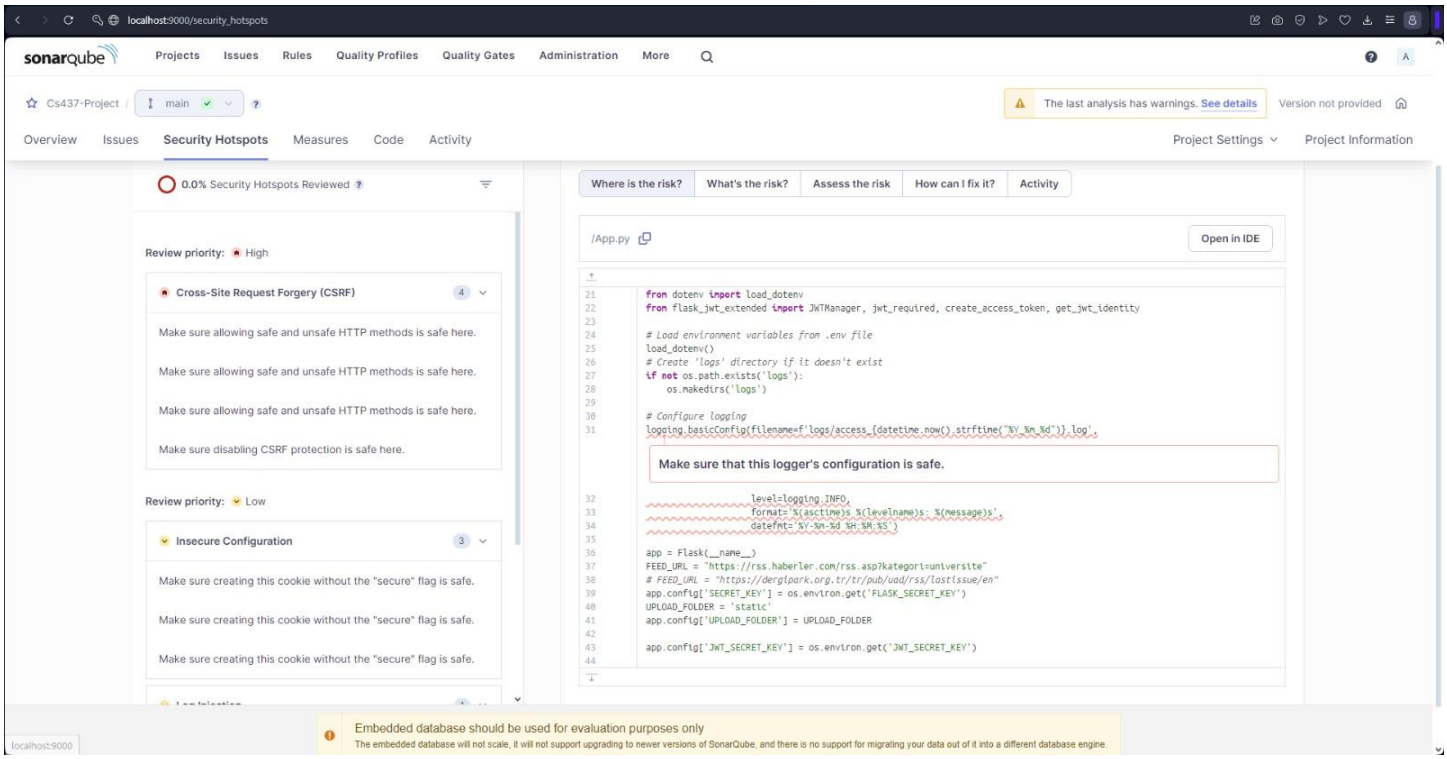
Prospector

```
C:\TurkishUniversityHoneypot-main>prospector
Messages
========

App.py
  Line: 6
    pylint: import-error / Unable to import 'feedparser'
  Line: 19
    pylint: reimported / Reimport 'Flask' (imported line 9)
    pylint: reimported / Reimport 'render_template' (imported line 5)
    pylint: unused-import / Unused abort imported from flask
    pyflakes: F811 / redefinition of unused 'Flask' from line 9 (col 1)
  Line: 20
    pylint: reimported / Reimport 'Flask' (imported line 9)
    pyflakes: F811 / redefinition of unused 'Flask' from line 19 (col 1)
  Line: 22
    pylint: unused-import / Unused jwt_required imported from flask_jwt_extended
  Line: 55
    pycodestyle: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)
  Line: 85
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 118
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 150
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 183
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 216
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 263
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 8)
  Line: 291
    pylint: not-an-iterable / Non-iterable value result is used in an iterating context (col 57)
  Line: 299
    pylint: c-extension-no-member / Module 'lxml.etree' has no 'XMLParser' member, but source is unavailable. Consider adding this module to extension-pkg-allow-list if you want to perform analysis based on run-time introspection of livi
ng objects. (col 17)
  Line: 300
    pylint: c-extension-no-member / Module 'lxml.etree' has no 'parse' member, but source is unavailable. Consider adding this module to extension-pkg-allow-list if you want to perform analysis based on run-time introspection of living o
bjects. (col 15)
  Line: 327
    pycodestyle: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)
  Line: 360
    pycodestyle: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)
  Line: 395
    pylint: logging-fstring-interpolation / Use lazy % formatting in logging functions (col 12)
  Line: 486
    pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)
  Line: 505
    pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)


Check Information
=================
        Started: 2024-01-06 22:44:30.224556
       Finished: 2024-01-06 22:44:53.364619
     Time Taken: 23.14 seconds
      Formatter: grouped
       Profiles: default, no_doc_warnings, no_test_warnings, strictness_medium, strictness_high, strictness_veryhigh, no_member_warnings
     Strictness: None
  Libraries Used: flask
      Tools Run: dodgy, mccabe, profile-validator, pycodestyle, pyflakes, pylint
  Messages Found: 23
```

SonarQube


localhost:9000/dashboard
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
Cs437-Project
main
Version not provided
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
To benefit from more of SonarQube's features, set up analysis in your favorite CI.
Quality Gate Status
Quality Gate
Passed
Enjoy your sparkling clean code!
Measures
New Code
Overall Code
Reliability
4 Bugs
C
Maintainability
5 Code Smells
A
Security
1 Vulnerabilities
E
Security Review
10 Security Hotspots
E
Coverage
0.0% Coverage
Coverage on 293 Lines to cover
– Unit Tests
Duplications
43.4% Duplications
Duplications on 1.1k Lines
24 Duplicated Blocks
Activity


localhost:9000/project/issues
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
Cs437-Project
main
The last analysis has warnings. See details
Version not provided
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
1 issues
App.py
Disable access to external entities in XML parsing.
1 location
This function loads the XML code and triggers the vulnerability.
1 of 1 shown
Intentionality issue | Not complete
Disable access to external entities in XML parsing.
XML parsers should not be vulnerable to XXE attacks python:S2755
Software qualities impacted: Security
Open
Not assigned
Vulnerability
Blocker
Tags
cwe
Effort
15min
Introduced
1 minute ago
Where is the issue?
Why is this an issue?
How can I fix it?
Activity
More Info
Cs437-Project > /App.py
Open in IDE
See all issues in this file
return render_template('search_results.html', news=filtered_news, users=filtered_users, query=query)
def process_xml(xml_file):
try:
parser = etree.XMLParser(load_dtd=True, no_network=False) # Insecure configuration
Disable access to external entities in XML parsing.
tree = etree.parse(xml_file, parser=parser)
root = tree.getroot()
# Extract profile image and about info from XML
profile_image = root.find('profile_image').text
about = root.find('about').text
return profile_image, about
except Exception as e:
Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different



Conclusion:

As a group, we chose Bandit for the best static code analysis tool. It works with both the latest versions of Python and identifies where vulnerabilities in the code are located and specifies the nature of the issues. However, we can not claim it to be completely flawless since it did not detect all vulnerabilities in our code.

I ran it on Windows

I had quite a struggle during the installation of Python-Taint (PYT) because I realized too late that it only works with Python 3.6 and Python 3.7 versions. Since I had Python 3.11 installed, I had to uninstall it and install Python 3.7. PYT identified 11 vulnerabilities and specified the line in the code where the issue occurred. However, it doesn't explain the nature of the problem as explicitly as Bandit does. Furthermore, the fact that it doesn't support the newer versions of Python adds to the challenges.

I ran it on Windows.

Rough-Auditing-Tool-for-Security (RATS) is indeed an old tool. During the installation, RATS was challenging as it has outdated documentation that is not clearly explained. Despite being confident in the correct installation, it did not provide satisfactory results, unlike Bandit. Although it read the code, I couldn't even understand if it found vulnerabilities or not because its reporting is very poor.

I ran it on Ubuntu.

Prospector's installation was easy. The documentation is well-prepared, and it can be used with the latest versions of Python. However, the results were almost unsatisfactory. It found some vulnerabilities in our code, but it also identified some irregular code lines that were unnecessary for our project. We can't claim it to be the best, especially compared to Bandit, as it found fewer vulnerabilities.

I ran it on Windows.

SonarQube was the additional static code analysis tool I chose. Based on my research, it is one of the most widely used tools, likely due to its up-to-date nature and easy installation. While it may not be as effective as Bandit in finding vulnerabilities, it stands out for its analysis of code in terms of Reliability, Maintainability, Security, and Security Review, as well as its appealing user interface. With three different versions available, I used the free version. From what I found online, the paid versions offer more advanced functionality and a more detailed analysis, although we did not make a purchase.

I ran it on Windows.

Shared Responsibilities Table

Görkem Filizöz	1. Installed and tested these four static code analysis tools on our code. 2. Conducted research to identify an additional static code analysis tool and found SonarQube. 3. Installed and tested SonarQube after the research. 4. Created reports for the results obtained from these tools.
Uğur Öztunç	1. Implemented and connected the database. 2. Implemented JWT mechanism 3. Implemented and tested the CWE-756 vulnerability. 4. Implemented and tested the CWE-260 vulnerability. 5. Implemented and tested the CWE-1004 vulnerability.
Utku Alkan	1. Developed the foundational architecture and user interface of the website. 2. Implemented logging mechanisms to monitor and understand attacker behavior. 3. Implemented and tested of the XXE vulnerability. 4. Designed and validated the SQL Injection vulnerability. 5. Implemented and verified the implementation of CWE-315 security misconfiguration. 6. Implemented and systematically tested the XSS (Cross-Site Scripting) vulnerability.

Video link

https://drive.google.com/drive/folders/1GpNrApbX8OWaSRnzr8I3-f0ySXriUahI?usp=drive_link