

Secure Coding and Vulnerability Detection

2nd Assignment for Design and Development of Secure Software

Work done by:

- Pedro Afonso Martins (2019216826)
- António Correia (2019216646)

Index

1. Construction of website and vulnerabilities	3
1.1. Website walkthrough	3
1.2. Good coding practices	4
1.3. Added vulnerabilities	8
1.4. Cross reference between vulnerable pages and vulnerabilities	9
2. Testing and Exploits' Development	10
2.1. Testing Analysis	10
2.1.1. Burp Professional Scanner	10
2.1.2. OWASP ZAP Scanner	14
2.2. Static Code Analysis	17
2.2.1. Bandit	17
2.2.2. Pylint	19
2.3. Tools discussion	21
2.4. Exploit development	21
2.4.1. OS Injection	21
2.4.2. Cookie manipulation	23
2.4.3. SQL Injection	23
3. Final remarks	23

1. Part 1 – Construction of website and vulnerabilities:

1.1. Website walkthrough

In this topic only the functionalities and website top view will be aborded. Remarks regarding the present vulnerabilities will be made in *topic 1.3* and *1.4*. Good practices involving these options will be depicted in *topic 1.2*. We would also like to highlight the choice made to separate the vulnerable and correct version pages, so that we can add and customize further vulnerabilities much more easily and effectively.

Registration Vulnerable	a)
Registration Correct	b)
Part 1.1 Vulnerable	c)
Part 1.1 Correct	d)
Part 1.2 Vulnerable	e)
Part 1.2 Correct	f)
Part 1.3 Vulnerable	g)
Part 1.3 Correct	h)

Figure 1 - Description of the website's initial page

- a) **Registration Vulnerable** allows the user to insert a new account into the database requesting a username and password.
- b) **Registration Correct** has the same purpose as a) but it requires the user to reintroduce the same password and in case of success redirects them into the 2FA page, where he is prompted with a secret code to insert in Google Authenticator to enable 2FA protection.
- c) **Part 1.1 Vulnerable** consists of a login page that takes the username, password to compare with the database. If it finds a match, the user is then authenticated and can access the rest of the functionalities. It also possesses a "Remember Me" field, that user cookies with a defined expiration date to keep them logged in in case they leave.
- d) **Part 1.1 Correct** has the same purpose as c) but it also requires a 2FA code that should be available in the user's Google Authenticator. If the user, password, and code are all verified, all the other functionalities become available. Only users registered via the correct form can authenticate via this page.

- e) **Part 1.2 Vulnerable (Requires Authentication!)** displays all messages within the database and allows for the authenticated user to insert new messages into the system. If he chooses to do so, the page will be automatically refreshed with the new message in display.
- f) **Part 1.2 Correct (Requires Authentication!)** has the exact workflow as e), with the created vulnerabilities removed.
- g) **Part 1.3 Vulnerable (Requires Authentication!)** consists of an enormous search query which is responsible for displaying all the books from a book table that match several parameters (title, author, category, price more and less than, search for specific keywords or entire phrases, date options and range, show and sort options). In case no parameters are given before pressing search, the output is all the books present on the website. This output box is displayed to the right of the parameters.
- h) **Part 1.3 Correct (Requires Authentication!)** follows the same principles as g), with the vulnerabilities removed and protections added.

1.2. Good coding practices

- **Usage of Two Factor Authentication**

When registering on the website via the correct form, the user will be redirected to another page to conclude the Two Factor Authentication and add a time-based code to his account. To do so he must follow the instructions depicted in **Figure 2**:

Please follow the instructions:

- Download [Google Authenticator](#) on your mobile device.
- Click on the "+" icon in the corner and select the **setup key** method.
- Provide a name of your choice and the secret token displayed below.
 - Select the time-based authentication option.
 - Introduce here the OTP generated to confirm the 2FA.

Secret Token:

Generated OTP:

Figure 2 – Two Factor Authentication Page

Once everything is set up the user can then authenticate via the correct form using the code displayed in their mobile device. If the code is valid, he's successfully logged in, otherwise he will have to input the code again until it's valid. Regarding how this process is done internally, when registering the application will generate a token and associate it with the user being created on the database. He is then redirected to another page, with the username and token passed on as parameters. The application is then responsible for checking if the passed token and user correspond to the data present in

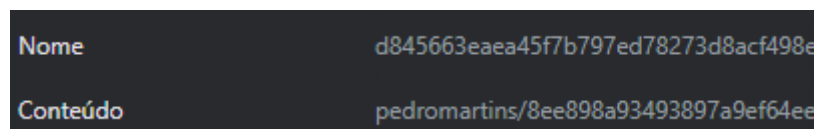
the database and, if positive, display the page in **Figure 2**. Once the user inputs the required code, he is then redirected to the main page, with the secret token stored in the database to be used later for validation.

If for some reason whilst adding the secret key the user leaves the page and does not complete the final step, upon authentication the application checks for users in the database with a generated token and no secret, so that he can be flagged, and the account deleted for not completing the requested page.

- **Password and cookie/session encryption**

When registering via the correct form, all the data is escaped to eliminate any tags/SQL content from reaching the database. A random salt is then generated using the `binascii.hexlify()` function, to then be merged with the password and hashed using the algorithm SHA256. All these data are then stored in the database encrypted, serving as an added layer of protection to the user's credentials. When logging in, the user will introduce his normal password, so the application gets its salt, merges it with the password given and hashes it to compare with the stored password. If there is a match, the user is marked as valid and only needs to complete the 2FA process to be authenticated.

As for cookies and session, when authenticating the user has an option to be remembered on the website or not. If he chooses to do so, encrypted cookies will be generated to keep the user logged in until a predefined timeout, if not, then we simply resort to use a session to keep the user authenticated while he's using the website. All these data are encrypted as well, as demonstrated in **Figure 4**, to further protect against password theft.

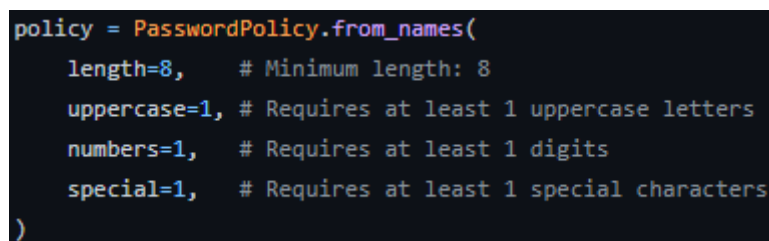


Nome	d845663eaea45f7b797ed78273d8acf498e
Conteúdo	pedromartins/8ee898a93493897a9ef64ee

Figure 3 – Cookies encryption

- **Password strength required**

When registering a new user into the system, a person is required to introduce a strong password. To achieve this, we resort to the `password_strength` library to guarantee that the input given is considered safe. In **Figure 5**, a demonstration of the structure used is available, with each criteria required marked with the number of occurrences necessary to achieve the desired level of security.



```
policy = PasswordPolicy.from_names(
    length=8,      # Minimum length: 8
    uppercase=1,  # Requires at least 1 uppercase letters
    numbers=1,    # Requires at least 1 digits
    special=1,    # Requires at least 1 special characters
)
```

Figure 4 – Password strength criteria

- **POST Usage and parameter redirection**

To safely transmit data, whenever we want to send password information, this is done via the POST method, to further protect against password hijacking. We also resort to the `url_for()` method when redirecting after using these types of information to avoid credential or sensitive data disclosure. We did not implement this in the Part 1.2 and Part 1.3 of this assignment, as our assessment concluded that the risk and impact was low enough to not warrant a change of procedures in these phases, especially given the necessary rewriting of the entire code to accommodate these changes to its' structure and functionality.

- **Parameter escape**

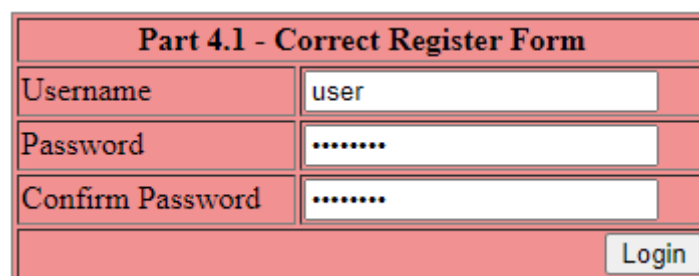
Anywhere where the user can input information, whether it is sensible or not, introduced into the database or not, this data is escaped using the `escape()` function available via the `html` library. This will remove any JavaScript tags which could be used in Cross Site Scripting attacks. Whilst important this is only the first step, due to the Jinja2 module already available via the Flask library we don't need to alter the HTML to protect against the execution of potential harmful actions, however, as will be demonstrated in the vulnerable version, appending the "`| safe`" option next to a printable variable removes this escaping feature, which limits the overall safety of the website.

- **SQL Injection protection**

Furthermore, in the correct fields we also have further protection against any unwanted actions in our database. For that we give the parameters to the SQL query directly upon execution, which limits the possibility of SQL Injection on any field where this option is used.

- **Dual password introduction and hiding**

When registering a new user, a person is required to introduce their password twice to further confirm it is well written. Since our website doesn't have a "Forgot Password" option this alternative boosts confidence that the password saved is intended. Furthermore, the password field in this page and on the login form is hidden to disallow spying of its contents. This is all depicted in **Figure 5**.



The image shows a web form titled "Part 4.1 - Correct Register Form". It contains three input fields: "Username" with the text "user", "Password" with masked characters ".....", and "Confirm Password" also with masked characters ".....". Below these fields is a "Login" button. The form is styled with a light pink background and a thin border.

Figure 5 – Correct Registration Page

- **Authentication Error Protection**

The authentication process does not provide any clues whether the username or password are incorrect, only displaying the message depicted in **Figure 6**. This avoids username disclosure protecting the user's existence within the website.

Part 1.1 - Correct Form	
Username	<input type="text" value="pedromartins"/>
Password	<input type="password" value="....."/>
2FA Code	<input type="text" value="Enter 2FA Code"/>
Remember me	<input checked="" type="checkbox"/>
<input type="button" value="Login"/>	

Authentication error!

Figure 6 – Correct Authentication Page with error message

- **Miscellaneous**

There are also some minor choices which overall contribute to the website's security, depicted as the following:

1. Usage of cookie timeout to allow for expiration date. This is a small achieved with the `COOKIE_TIMEOUT` flag, which was set to 600 seconds, protecting a user's account in case he doesn't want his account to be logged in indefinitely and allowing for a short time frame for the data storage.
2. Password strength is checked internally using a specific library as depicted in **Figure 4**, whilst in the vulnerable version we check the password with a text file of unsafe passwords to see whether it is strong or not. This also highlights how the first option is much safer and more modular, allowing for a much better optimization and parameter definition.
3. Pages that require authentication are protected from users which are not logged in in the system. This is achieved by checking whether the session or cookies contain valid information about the user in question.
4. Debug flag responsible for giving information about an error is False, further protecting the website from unwanted knowledge gain by an unauthorized third party.

1.3. Added Vulnerabilities

Throughout the vulnerable pages multiple vulnerabilities or lack of security mechanisms were purposefully introduced and are listed in the table below, ranked by their security risk:

Type	Risk	Description
1. SQL-Injection	High	All the fields are SQL-Injectable, which allows for a malicious user to execute SQL code into the database. To test it, introduce a ' in the desired field.
2 XSS Stored	High	Allows for users to inject JavaScript code into the database and, when requesting access to another page, this same code will be executed.
3. OS Injection	High	While searching for the inputted password in a file of the most common passwords to check for strength, user can insert malicious commands to be executed.
4. XSS Reflected	High	Allows for execution of JavaScript commands without storage of them. Less severe than the stored version, although still susceptible to cause big damages.
5. Password unsafe	High	When storing a password for a user in the database, there is no salt generated or hashing applied, storing the information without any encryption.
6. Lack of 2FA	High	Vulnerable accounts are not asked to add 2FA to protect their account.
7. Cookies Encryption	High	The encryption of the cookies upon authentication is not done, allowing for interception of passwords
8. GET for password submission	Medium	When submitting a form that contains sensible information like passwords, the method used is GET instead of POST.
9. Redirect with parameters	Medium	The redirect to a different page is not cleaned of previous sent parameters, which can display sensible information like passwords in the URL.
10. Password disclosure	Low	When inputting a password in the respective field, it is not hidden, which can allow for screen share or a third party to gain access to that account.
11. Lack of password reintroduction	Low	When registering a user there is no third field to reintroduce the same password, making it easier for the user to register with the wrong password.
12. Username Disclosure	Low	When trying to access an account, vulnerable fields will display a message that indicates if the user exists or not, allowing for unwanted account discovery.

Table 1 – Listed vulnerabilities added to the vulnerable portions of the website

1.4. Cross Reference between vulnerable pages and vulnerabilities

All that remains is to draw a direct association between the previous vulnerabilities and the vulnerable pages affected by them.

VULNERABILITIES	REGISTER	PART 1.1	PART 1.2	PART 1.3
V1	x	x	x	x
V2			x	
V3	x			
V4			x	x
V5	x			
V6	x	x		
V7		x		
V8	x	x		
V9	x	x	x	x
V10		x		
V11	x			
V12		x		

Table 2 – Association between vulnerable pages and vulnerabilities

2. Part 2 - Testing and Exploit development:

2.1. Testing Analysis

2.1.1. Burp Professional Scanner

We obtained the 30-day trial available for the usage of this tool via the university's email. The first scans couldn't discover many vulnerabilities, which was due to the fact that the authentication was not properly established in the scanner options. To proceed we had first to go to Proxy->Options and edit the Proxy Listener to another port (6666 in this case, as depicted in **Figure 7**. This is important so that we can access our website in port 8080 via localhost, necessary for the scan.

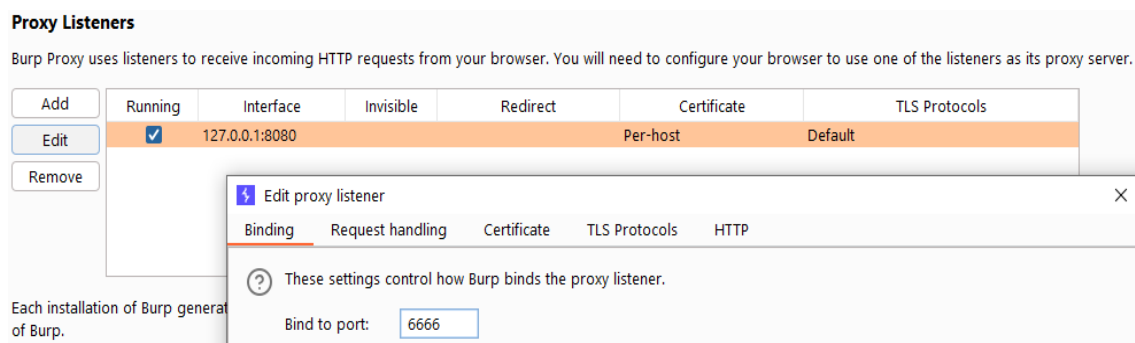


Figure 7 – Edit Proxy Listener configuration

Next we need to record our login path, going to Proxy->Open Browser and clicking on the extension to start recording, as shown in **Figure 8**. Then we just need to enter our address (localhost:8080), choose the login option and enter with a valid account. Once that's done, we simply click record on the same location as the start recording option and save the request/responses of these actions for later use.

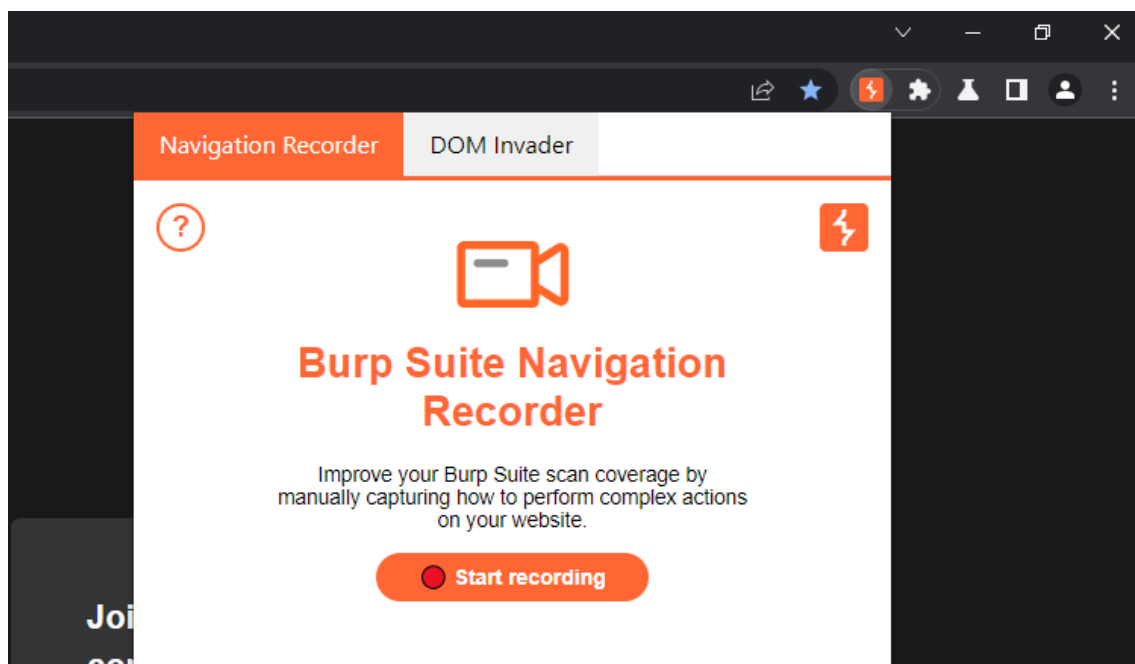


Figure 8 – Start recording login procedure

After this is concluded, we can start a new scan by going to Dashboard->New Scan. We are then presented with a lot of options. For the first page, we choose crawl and audit, enter our address, and specify our protocol (we do not use https, so we only need to scan the http version for vulnerabilities (**Figure 9**).

Scan Type

☒ Crawl and audit
☐ Crawl

URLs to Scan

Define the URLs to scan. Burp will begin crawling from these URLs, and by default will include everything beneath the specified URLs' folders.

http://localhost:8080/

Protocol settings

☐ Scan using HTTP & HTTPS ☒ Scan using my specified protocols

Figure 9 – Scan details

Then for the Scan Configuration, we will choose the Deep variant, as this will analyze in profound detail all the necessary components needed (**Figure 10**). For the first few times we used the Lightweight option to verify if all components were working and get a better understanding of the initial problems.

Scan Configuration

Scan configurations and modes are groups of settings that define how a scan is performed. Scan modes offer preset options designed to let you trade off speed and coverage. Alternatively, you can select one or more custom configurations. Burp Scanner applies any selected configurations in order, enabling you to fine-tune scanning behaviour.

☒ Use a preset scan mode ☐ Use a custom configuration

Lightweight

☐ Gain fast feedback on a site's security - for when speed is a priority. Lightweight mode will complete within 15 minutes.

Fast

☐ More thorough than a Lightweight scan, but still biased towards speed. Fast scans will generally complete within one hour.

Balanced

☐ Provides a balance between coverage and speed. You will typically see the results of a Balanced scan within a few hours.

Deep

☒ Achieve greater coverage and gain a better understanding of a site's security posture. Scanning time depends heavily on the target site's size and complexity.

Figure 10 – Scan configuration

Finally, we choose the second option in Application Login and create a New login path in the “New” button to the right, inputting all the information gathered previously. This is shown in Figure 11. Now we can start our scan by pressing the Ok option in the bottom right.

Please choose a login type

Choose whether to configure login credentials or recorded login sequences. You can configure multiple logins to scan in different user contexts.

☐ Use login credentials (username & password)
☒ Use recorded login sequences (record using Burp's Chrome extension)

Label	
UsersLogin	New ... Edit Delete Replay

Note: Recorded login sequences overrides your 'Login Functions' scan configuration.

Figure 11 – Application Login

We ran several scans in which we discovered some vulnerabilities in the correct versions which we addressed, such as redirect with some parameters in the authentication and cross site scripting in the Part 1.3, which we solved using escaping of the inputted parameters. In **Figure 12** we have the results of our final scan, with the result file attached in the deliverables of this project. Certain means the scanner is sure, firm means he’s almost positive and tentative means there is a considerable chance.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	15	5	2	22
	Medium	0	2	0	2
	Low	3	0	4	7
	Information	22	1	2	25

Figure 12 – Severity summary

1. OS command injection

- 1.1. http://localhost:8080/register_vuln.html [v_password parameter]
- 1.2. http://localhost:8080/register_vuln.html [v_password parameter]

2. SQL injection

- 2.1. http://localhost:8080/part1_vuln.html [v_username parameter]
- 2.2. http://localhost:8080/part2_vuln.html [username cookie]
- 2.3. http://localhost:8080/part2_vuln.html [v_text parameter]
- 2.4. http://localhost:8080/part3_vuln.html [v_author parameter]
- 2.5. http://localhost:8080/part3_vuln.html [v_category parameter]
- 2.6. http://localhost:8080/part3_vuln.html [v_name parameter]
- 2.7. http://localhost:8080/part3_vuln.html [v_pricemax parameter]
- 2.8. http://localhost:8080/part3_vuln.html [v_pricemin parameter]
- 2.9. http://localhost:8080/part3_vuln.html [v_search_input parameter]
- 2.10. http://localhost:8080/register_vuln.html [v_password parameter]
- 2.11. http://localhost:8080/register_vuln.html [v_username parameter]

3. Cross-site scripting (stored)

4. Cross-site scripting (reflected)

- 4.1. http://localhost:8080/part2_vuln.html [v_text parameter]
- 4.2. http://localhost:8080/part3_vuln.html [v_author parameter]
- 4.3. http://localhost:8080/part3_vuln.html [v_category parameter]
- 4.4. http://localhost:8080/part3_vuln.html [v_pricemax parameter]
- 4.5. http://localhost:8080/part3_vuln.html [v_pricemin parameter]
- 4.6. http://localhost:8080/part3_vuln.html [v_search_input parameter]

5. Cleartext submission of password

- 5.1. http://localhost:8080/part1_correct
- 5.2. http://localhost:8080/register_correct

6. Web cache poisoning

- 6.1. http://localhost:8080/part2_correct.html [c_text parameter]
- 6.2. http://localhost:8080/part2_vuln.html [v_text parameter]

7. Open redirection (DOM-based)

- 7.1. http://localhost:8080/part2_correct.html
- 7.2. http://localhost:8080/part3_correct.html
- 7.3. http://localhost:8080/part3_vuln.html
- 7.4. http://localhost:8080/register_vuln.html

8. Password field with autocomplete enabled

- 8.1. http://localhost:8080/part1_correct
- 8.2. http://localhost:8080/register_correct

9. Unencrypted communications

10. File path manipulation

- 10.1. http://localhost:8080/part3_vuln.html [v_pricemax parameter]
- 10.2. http://localhost:8080/part3_vuln.html [v_pricemin parameter]

11. Input returned in response (stored)

- 11.1. http://localhost:8080/part2_correct.html [c_text parameter]
- 11.2. http://localhost:8080/part2_correct.html [c_text parameter]
- 11.3. http://localhost:8080/part2_correct.html [c_text parameter]
- 11.4. http://localhost:8080/part2_correct.html [c_text parameter]
- 11.5. http://localhost:8080/part2_vuln.html [v_text parameter]
- 11.6. http://localhost:8080/part2_vuln.html [v_text parameter]
- 11.7. http://localhost:8080/part2_vuln.html [v_text parameter]
- 11.8. http://localhost:8080/part2_vuln.html [v_text parameter]

12. Input returned in response (reflected)

- 12.1. http://localhost:8080/part2_correct.html [c_text parameter]
- 12.2. http://localhost:8080/part2_vuln.html [v_text parameter]
- 12.3. http://localhost:8080/part3_correct.html [c_author parameter]
- 12.4. http://localhost:8080/part3_correct.html [c_category parameter]
- 12.5. http://localhost:8080/part3_correct.html [c_pricemax parameter]
- 12.6. http://localhost:8080/part3_correct.html [c_pricemin parameter]
- 12.7. http://localhost:8080/part3_correct.html [c_search_input parameter]
- 12.8. http://localhost:8080/part3_correct.html [c_sp_c parameter]
- 12.9. http://localhost:8080/part3_vuln.html [v_author parameter]
- 12.10. http://localhost:8080/part3_vuln.html [v_category parameter]
- 12.11. http://localhost:8080/part3_vuln.html [v_pricemax parameter]
- 12.12. http://localhost:8080/part3_vuln.html [v_pricemin parameter]
- 12.13. http://localhost:8080/part3_vuln.html [v_search_input parameter]
- 12.14. http://localhost:8080/part3_vuln.html [v_sp_c parameter]

13. Frameable response (potential Clickjacking)

Figure 13 – All vulnerabilities discovered using the Burp Scanner

As we can see in **Figure 13**, we have detected several high priority vulnerabilities in the vulnerable versions, such as SQL Injection, XSS and OS Injection. Regarding the point 5, it is also listed as a high risk, and it appears on the correct version. Upon deeper investigation we concluded that it was due to unencrypted communication between the client and server when sending sensible authentication data, as reported by point 9. A possible fix would be to use HTTPS via certificates to encrypt all the data, although we could use self-generated or local certificates, which defeats all their purpose entirely as any browser will not recognize them as legitimate. For the scope of this project, points 5 and 9 are marked as **false positives**.

Points 6 and 7 also contain correct pages in the vulnerability and, although acknowledged, we conclude that these reports are not of concern for the overall safety of the website. Point 8 concerns autocomplete of the password when authenticating or registering and overall does not contain a risk high enough to be considered.

Finally, points 11 and 12 concern the input returned (parameters) when redirecting to another page, which as previously explained, is only protected in register and login to deter credential exposure. The report giving part2 and part3 correct as flagged does not pose a significant threat to the website's functionality.

2.1.2. OWASP Scanner

The other tool we decided to use was OWASP ZAP, an open-source web app scanner claiming to be the world's most widely used one. Once again we had to set the scanner's proxy settings so that it would be able to work, which we did by going to Tools->Options->Network->Local Servers/Proxies:

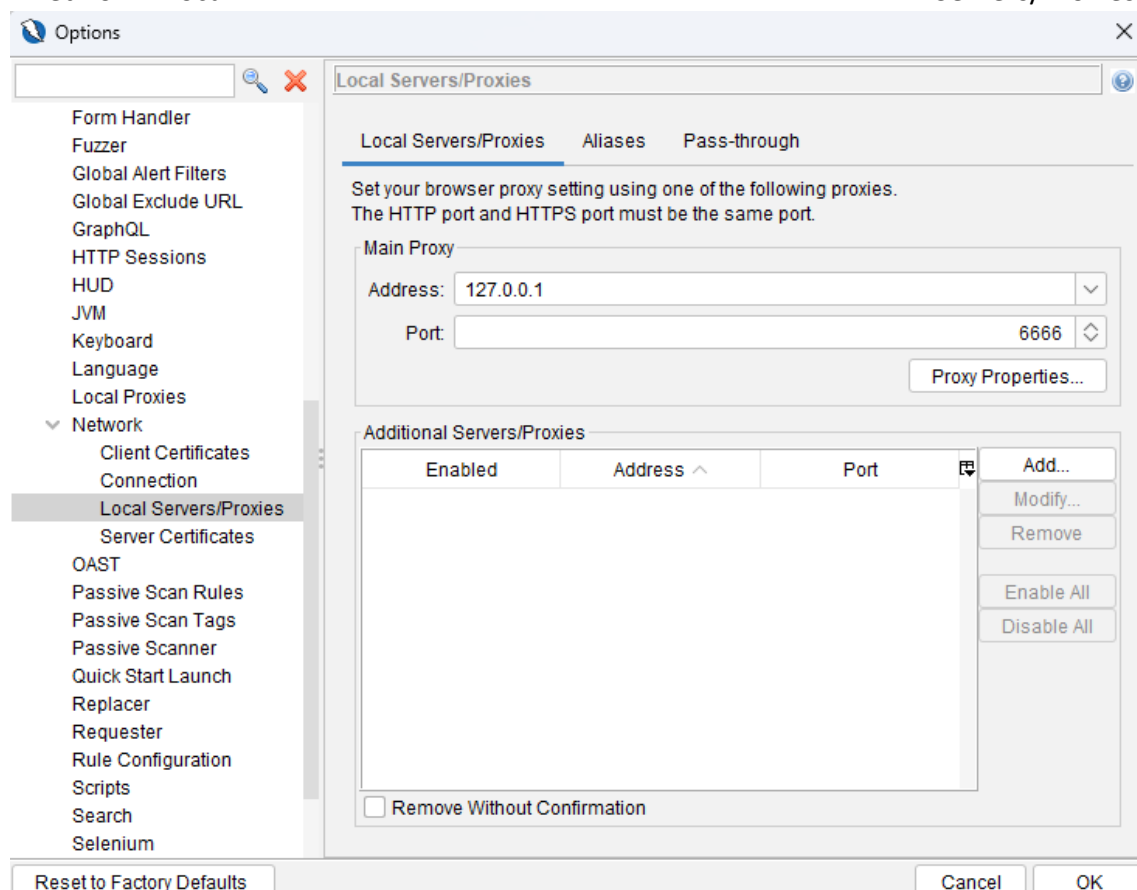


Figure 14 – Proxy configurations

After that we need to “Explore” our website, feeding the scanner with what request our website handles. To do that we go to every page and submit information on the ones that support it, leaving us with a list of the requests of the website:

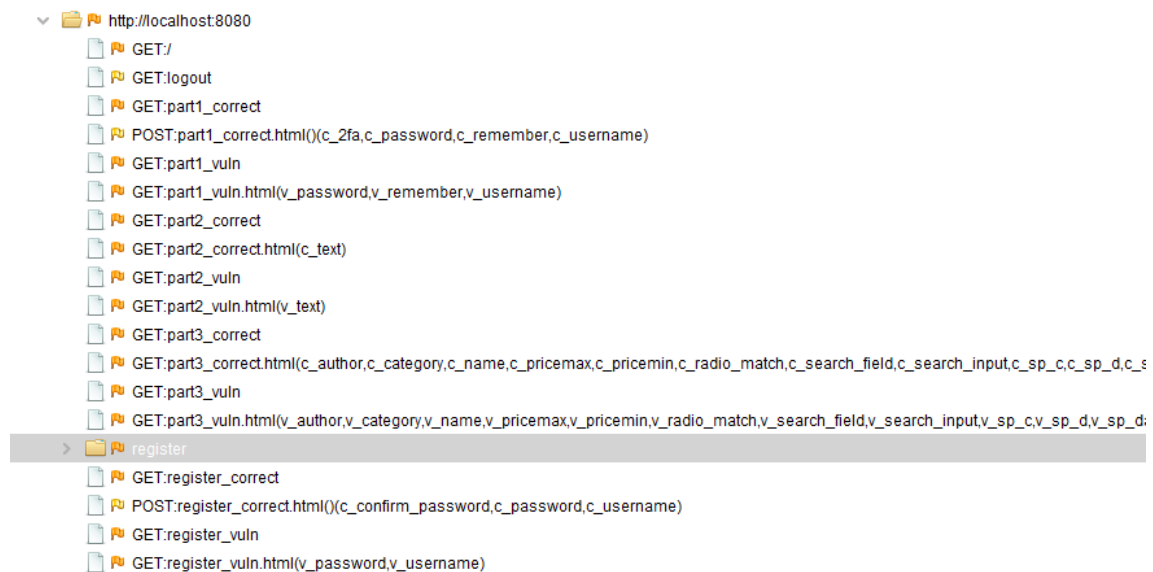


Figure 15 – Request list

Now we need to give the tool instructions on how to authenticate itself on the website to get access to the features. We will include the request that submits the login information in a new context, configuring the Authentication and Users sections like this:

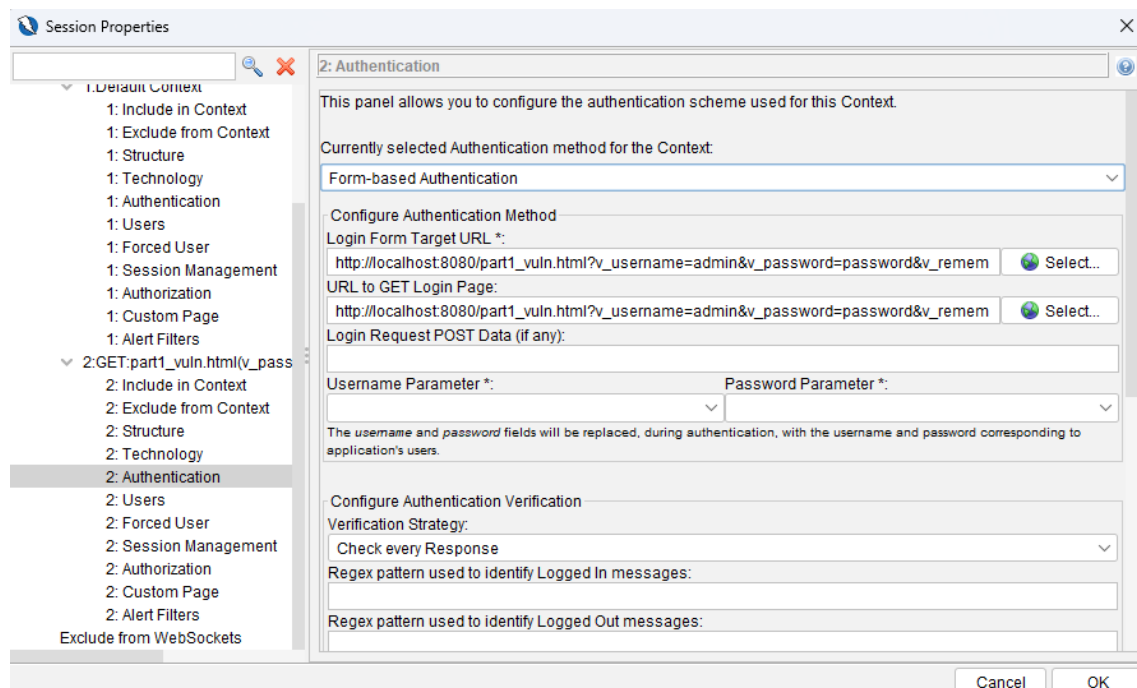


Figure 16 – Authentication configuration

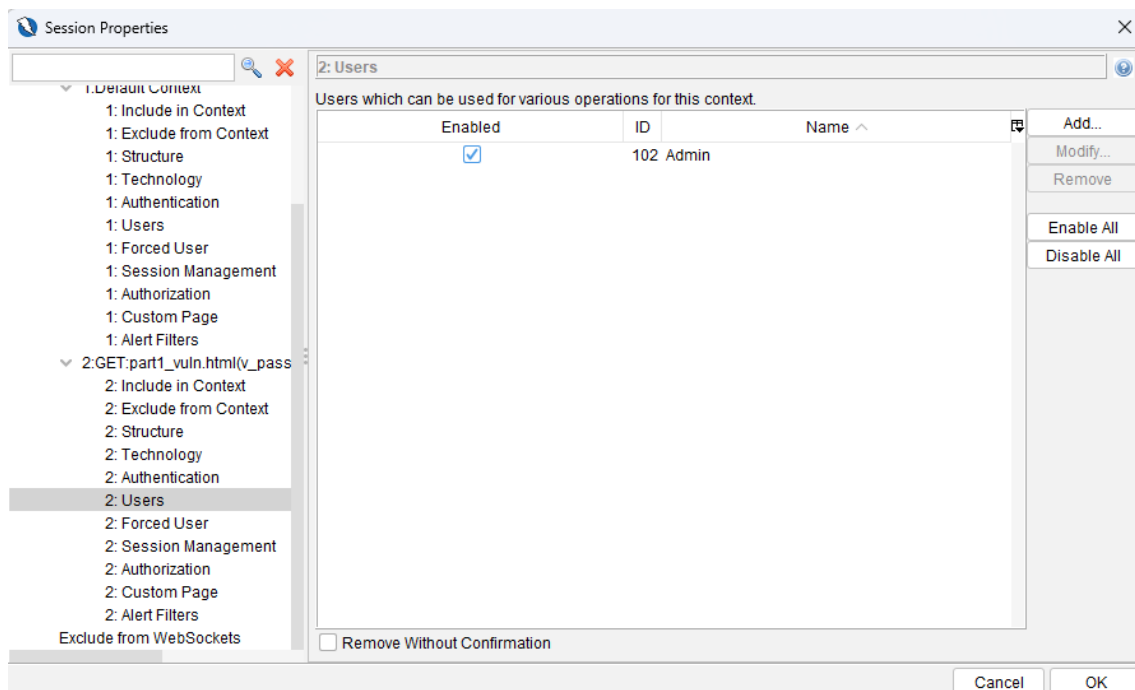


Figure 17 – Users configuration

Note that the user Admin was added in this process, giving it the credentials for one account present in the database. We also renamed the context to 'Login Vuln' for easier identification. Now we simply flag the same request as context to 'Login Vuln' (using the option 'Flag as context'), followed by including the localhost folder in the same context and flagging it. Now our configuration is completed, we can use the option 'Attack' on the localhost folder and select 'Active Scan':

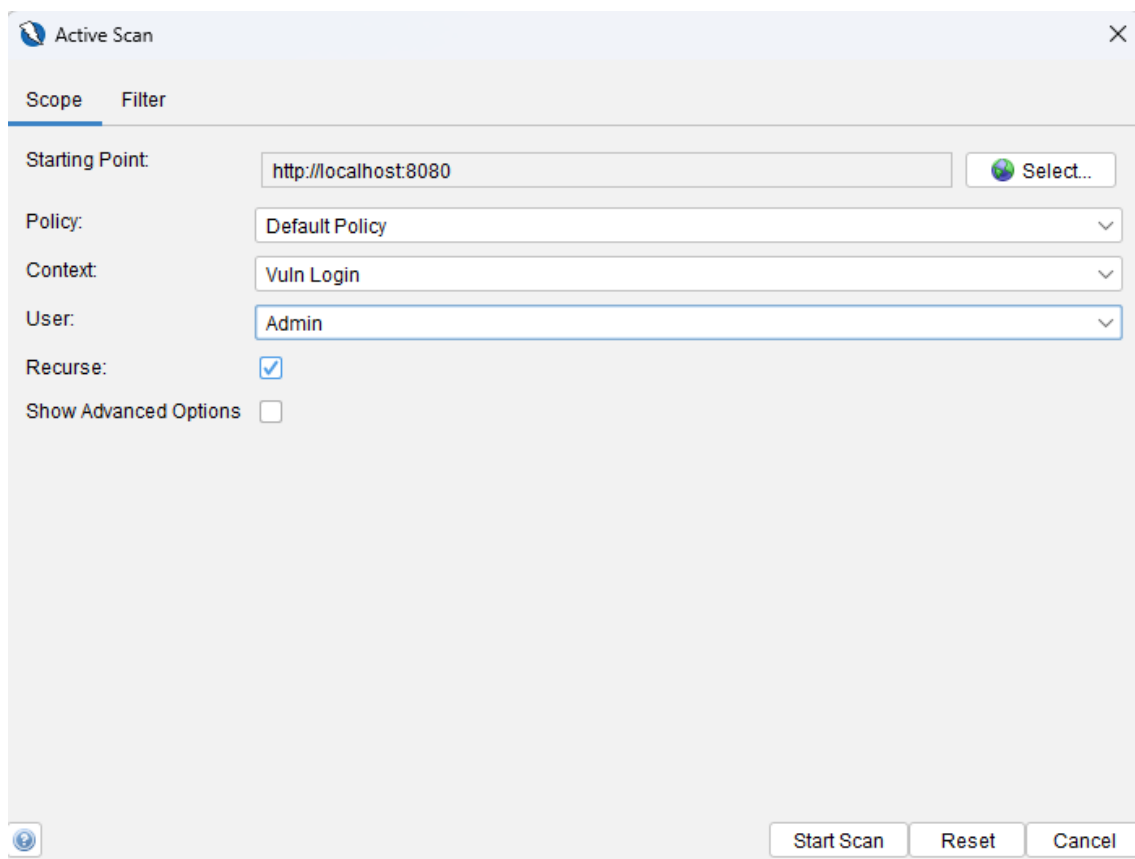


Figure 18 – Attack configuration

After the scan we can generate a report that gives us more complete information, from the complete URL used for the attack, to a description of the vulnerability and a possible solution. Here is the list of the alerts that have a high-risk level:

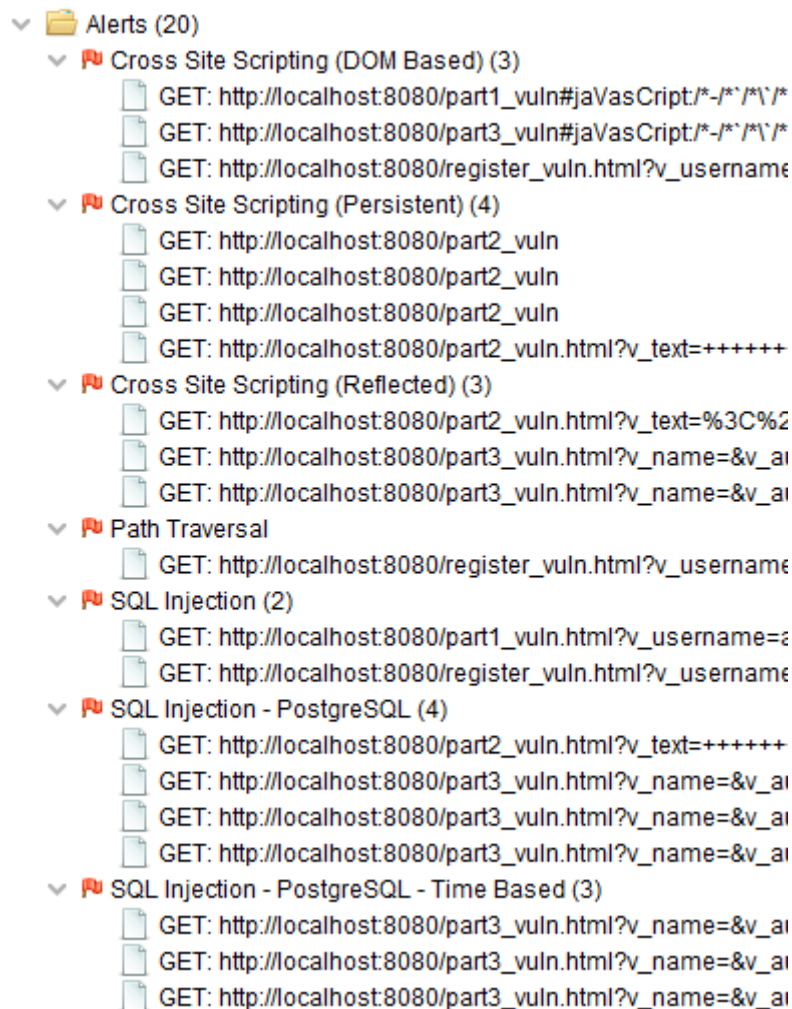


Figure 19 – All vulnerabilities discovered by OWASP ZAP

Crossing this list with the one obtained from Burp, we can see that we have less vulnerabilities found, which can result from tool limitations, especially because we can see that it found Path Traversal but not Remote OS command injection, which should be both detected the same way.

Still, we can confirm that all the high-risk vulnerabilities are still in the vulnerable parts, reassuring the safety of the non-vulnerable sections of the website.

2.2. Static Code Analysis

2.2.1. Bandit

Regarding the static code analysis, we decided to start with the Bandit library, useful to cross reference with website exploitation. For that we first installed its contents, via the “py -m pip install bandit” command. We then used cd to reach the directory of the python file containing our application and executed the command: “py -m bandit app.py”, which gave us a comprehensible output with 33 found issues:

```

Code scanned:
    Total lines of code: 601
    Total lines skipped (#nosec): 0

Run metrics:
    Total issues (by severity):
        Undefined: 0
        Low: 3
        Medium: 29
        High: 1
    Total issues (by confidence):
        Undefined: 0
        Low: 9
        Medium: 22
        High: 2
Files skipped (0):

```

Figure 20 – Summary of Bandit execution

It also gave us an extensive list with all the vulnerabilities detected. For the sake of avoiding repetition, we will only list unique ones.

```

Test results:
>> Issue: [B404:blacklist] Consider possible security implications associated with the subprocess module.
Severity: Low Confidence: High
CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
Location: app.py:11:0
More Info: https://bandit.readthedocs.io/en/1.7.4/blacklists/blacklist\_imports.html#b404-import-subprocess
10     import uuid
11     import subprocess
12
13     policy = PasswordPolicy.from_names(
-----
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'secret key'
Severity: Low Confidence: Medium
CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
Location: app.py:21:17
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b105\_hardcoded\_password\_string.html
20     app = Flask(__name__)
21     app.secret_key = "secret key"
22     COOKIE_TIMEOUT = 60 * 10
-----
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
Severity: Medium Confidence: Medium
CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
Location: app.py:65:16
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b608\_hardcoded\_sql\_expressions.html
64     cur = conn.cursor()
65     cur.execute(f"SELECT password FROM users WHERE username=%s", (username,))
66     row = cur.fetchall()
-----
>> 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)
Location: app.py:90:12
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b608\_hardcoded\_sql\_expressions.html
89     # Create SQL query (SQL INJECTABLE)
90     query = f"SELECT username FROM users WHERE username='{username}'"
91
-----
>> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True identified, security issue.
Severity: High Confidence: High
CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
Location: app.py:94:13
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b602\_subprocess\_popen\_with\_shell\_equals\_true.html
93     cmd = "grep %s pass.txt" % password
94     result = subprocess.run(cmd, shell=True, capture_output=True)
95

```

```

>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password: 'ddss-database-assignment-2'
Severity: Low Confidence: Medium
CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
Location: app.py:850:9
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b106\_hardcoded\_password\_funcarg.html
849     def get_db():
850         db = psycopg2.connect(user = "ddss-database-assignment-2",
851                               password = "ddss-database-assignment-2",
852                               host = "db",
853                               port = "5432",
854                               database = "ddss-database-assignment-2")
855         return db

-----
>> Issue: [B104:hardcoded_bind_all_interfaces] Possible binding to all interfaces.
Severity: Medium Confidence: Medium
CWE: CWE-605 (https://cwe.mitre.org/data/definitions/605.html)
Location: app.py:884:17
More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b104\_hardcoded\_bind\_all\_interfaces.html
883
884     app.run(host="0.0.0.0", threaded=True)

```

Figure 21 – Unique vulnerabilities detected via Bandit

The first two vulnerabilities concern the use of subprocess module as a potential exploitable function, which is required to make sure we can have OS Injection in the vulnerable registration field and hardcoded password for the secret application key. This last problem also occurs in the penultimate listed vulnerability, and it happens because via code reading, one could get access to credentials for the database and application. Upon further investigation we concluded that a solution for this would not be easily feasible and, given the low severity indicated by the tool, we decided to not change it.

The 3rd and 4th vulnerabilities are the most common and warn for SQL Injection. Whilst the 4th represents the incorrect way we execute queries for database interactions and should be flagged (although in our opinion its severity level should be **High** since this is a highly dangerous vulnerability with an exploitation capable of causing massive damages to the website operations), the 3rd one is how we execute the queries correctly and it has a higher confidence level than the vulnerable version. After further research we can mark the 3rd vulnerability as a **false positive**.

Finally, it also detected OS Injection in the 5th vulnerability, this time with a much higher level of confidence and severity. The reasoning provided is also correct, since it is the shell=True parameter that causes this vulnerability to occur.

2.2.2. Prospector

For the second and final static code analysis tool we decided to use prospector. The decision to choose this module stems from the usage of multiple libraries, which can give us an optimized output with a good amount of information for only one tool. Regarding installation, the process is fairly similar to the previous tool: “py -m pip install prospector” and “py -m prospector app.py”. In **Figure 22** we have present the summary of our execution

```

Check Information
=====
Started: 2022-12-25 19:17:41.446799
Finished: 2022-12-25 19:17:42.337174
Time Taken: 0.89 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: 80

```

Figure 22 – Summary of execution

```

app.py
Line: 1
pylint: unused-import / Unused import json
Line: 2
pylint: import-error / Unable to import 'flask'
pylint: unused-import / Unused g imported from flask
Line: 3
pylint: import-error / Unable to import 'password_strength'
Line: 6
pylint: import-error / Unable to import 'psycopg2'
pylint: multiple-imports / Multiple imports on one line (logging, psycopg2)
Line: 9
pylint: import-error / Unable to import 'pyotp'
Line: 48
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 65
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 68
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 69
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 93
pylint: consider-using-f-string / Formatting a regular string which could be a f-string (col 10)
Line: 94
pylint: subprocess-run-check / Using subprocess.run without explicitly set `check` is not recommended. (col 13)
Line: 107
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 121
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 138
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 141
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 142
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 176
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 180
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 186
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 197
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 201
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 211
pylint: unused-argument / Unused argument 'token' (col 32)
Line: 218
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 223
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 243
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 260
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 263
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 264
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 301
pylint: singleton-comparison / Comparison 'account[0] == None' should be 'account[0] is None' (col 15)
Line: 302
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 28)
Line: 306
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)
Line: 315
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)
Line: 332
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 15)
Line: 349
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 353
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 354

```

Figure 23 – Problems detected with Prospector (1/2)

In Figures 23 and 24 we have the full execution of our tool which demonstrates an output considerably different to Bandit, as most of the messages regard unused arguments, f-string usage and a “no-else-return” due to using else after a return message in the previous if iteration. Less commonly it also detects long lines, usage of too many variables (in part 3 due to the constitution of all the parameters needed for the search mechanism) and unused imports. In summary, although still important, this tool has a much lesser impact on the security of the website, only needed for the purpose of good coding standards regarding syntax and indentation.

```

Line: 390
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 399
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 4)
Line: 403
pylint: no-else-return / Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (col 8)
Line: 405
pycodestyle: E501 / line too long (178 > 159 characters) (col 160)
Line: 462
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 478
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 499
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 12)
Line: 525
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 541
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 556
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 560
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 16)
Line: 592
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 610
pylint: too-many-locals / Too many local variables (24/15)
mccabe: MC0001 / part3_vulnerable_form is too complex (23)
Line: 651
pylint: consider-using-in / Consider merging these comparisons with 'in' by using 'match_method in ('any', 'all')'. Use a set instead if elements are hashable. (col 15)
Line: 655
pycodestyle: E501 / line too long (175 > 159 characters) (col 160)
Line: 661
pycodestyle: E501 / line too long (213 > 159 characters) (col 160)
Line: 718
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 20)
Line: 736
pylint: too-many-locals / Too many local variables (25/15)
mccabe: MC0001 / part3_correct_form is too complex (24)
Line: 761
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 29)
Line: 764
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 29)
Line: 767
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 29)
Line: 771
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 774
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 777
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 780
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 783
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 786
pylint: consider-using-in / Consider merging these comparisons with 'in' by using 'match_method in ('any', 'all')'. Use a set instead if elements are hashable. (col 15)
Line: 790
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 34)
Line: 797
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 37)
Line: 800
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 29)
Line: 803
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 29)
Line: 808
pylint: too-many-boolean-expressions / Too many boolean expressions in if statement (6/5) (col 9)
pycodestyle: E501 / line too long (165 > 159 characters) (col 160)
Line: 811
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 814
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 33)
Line: 831
pylint: f-string-without-interpolation / Using an f-string that does not have any interpolated variables (col 11)
Line: 865
pycodestyle: E305 / expected 2 blank lines after class or function definition, found 5 (col 1)

```

Figure 24 – Problems detected with Prospector (2/2)

2.3. Tools Discussion

After reviewing these static code analysis tools, we can conclude that they are much faster and practical to execute and provide a reasonable coverage in practice standards and basic security notions, however, they **cannot** replace testing tools like OWASP or Burp Scanner, as these are much more complex and provide a full view of the security instituted and possible exploitation points for a website. All the reports for these tools are available in the directory Scanner Reports in the deliverable.

2.4. Exploit Development

2.4.1. OS Injection

The goal now is to deliver an exploit exclusive for a vulnerability (V3) introduced due to the group assignment, in this case the registration page. Firstly, with the website running the user should select the Registration Vulnerable page, since it is the only subject to this exploit, as depicted in **Figure 25**.


	Registration Vulnerable
	Registration Correct
	Part 1.1 Vulnerable
	Part 1.1 Correct
	Part 1.2 Vulnerable
	Part 1.2 Correct
	Part 1.3 Vulnerable
	Part 1.3 Correct

Figure 25 – Initial page orientation

Then he should introduce a user, pick a username and on the password field introduce the malicious command. To cause some damage we'll take advantage of the "rm" command which deletes a specific file in the website's directory.

User Registration

Part 4.1 - Vulnerable Register Form	
Username	<input type="text" value="user"/>
Password	<input type="text" value=";rm pass.txt"/>
<input type="button" value="Login"/>	

Figure 26 – Malicious command injection

In the example above, we'll try to delete a file called pass.txt which is present in the directory and is responsible to assert password security in the vulnerable system. We could theoretically delete any file we want we'd just need to replace the pass.txt parameter with any other file. Once executed the file has been deleted from the system. We can't stress enough how dangerous this vulnerability is and can lead to total website failure and, even worse, system and machine problems.

2.4.2. Cookie manipulation

The vulnerability related to cookies encryption (V7) can also be used to gain access to the website without owning an account. To do this, we can simply go to the website and open the developer console of our browser (CTRL + SHIFT + J both in Google Chrome and Microsoft Edge) and run the following command:

```
> document.cookie="username=user"
```

Figure 27 – Command to create a new cookie

This creates a new cookie with the key 'username' and the value 'user' (which doesn't really matter) since the code only check if a cookie with that username exists. This exploit also works in the correct sections of the website since we wanted to allow compatibility between the vulnerable login and the non-vulnerable parts.

2.4.3. SQL Injection

Following the same instructions in section 2.4.1. but using a different malicious command, it is also possible to execute SQL injection. We will take advantage of the non-sanitized SQL queries to delete the table 'users' rendering the site completely useless since no user will be able to login.

User Registration

Part 4.1 - Vulnerable Register Form	
Username	<input type="text" value="user"/>
Password	<input type="password" value="'); DROP TABLE users; --"/>
<input type="button" value="Login"/>	

Figure 28 – Malicious SQL injection

3. Final Remarks

In conclusion, throughout this project we were able to explore diverse concepts and vulnerabilities in practice, after learning about them in the theoretical classes by creating this website with vulnerabilities and correct coding practices security wise. We also further increased our knowledge in tools and framework scanning, essential to our master's degree.