Assignment 2 - Secure Coding and Vulnerability Detection

João André Gomes Marques
Department of Informatic Engineering, University of Coimbra
uc2017225818@student.uc.pt

Leonardo Oliveira Pereira
Department of Informatics Engineering, University of Coimbra
uc2020239125@student.uc.pt

1 Introduction

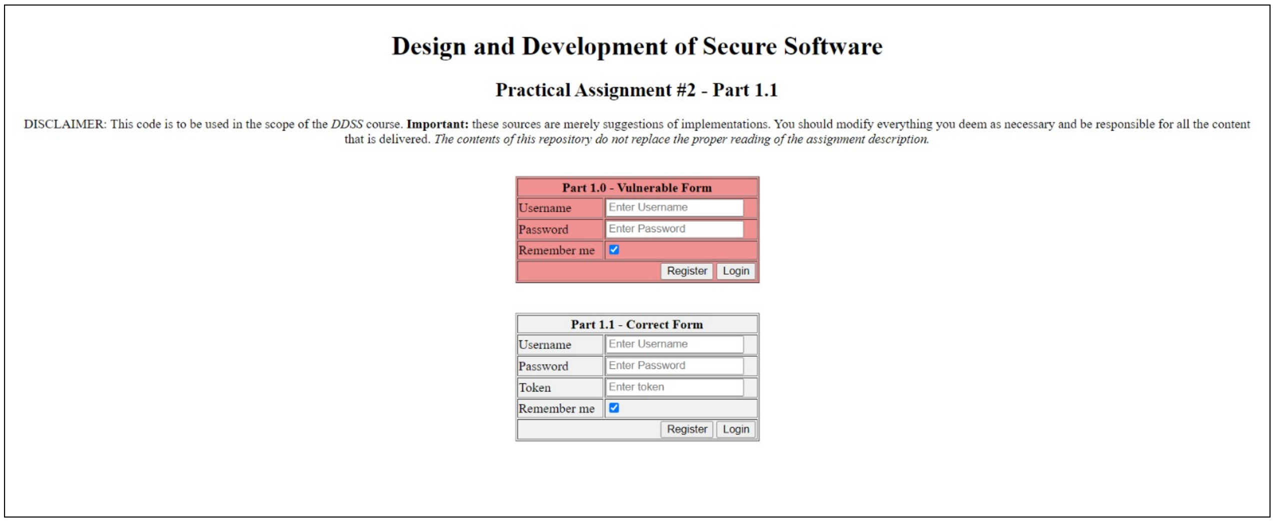
This report provides an analysis of a Python Flask web application project where each functionality was implemented twice: once with intentional vulnerabilities and the other correctly without vulnerabilities. The purpose was to demonstrate an understanding of common security flaws and how to mitigate them.

2 Methodology

The analysis was conducted by examining the purpose and implementation of each function in both its vulnerable and corrected forms, focusing on understanding the nature of vulnerabilities and the strategies used to correct them.

2.1 Part 1.1

The first functionality consists of a typical authentication form, based on a username and a password. The picture below shows the intended interface. The form at the top is vulnerable, while the one at the bottom is secure. For each function, a warning message is shown to let the users know that they are inserting wrong credentials.


Design and Development of Secure Software

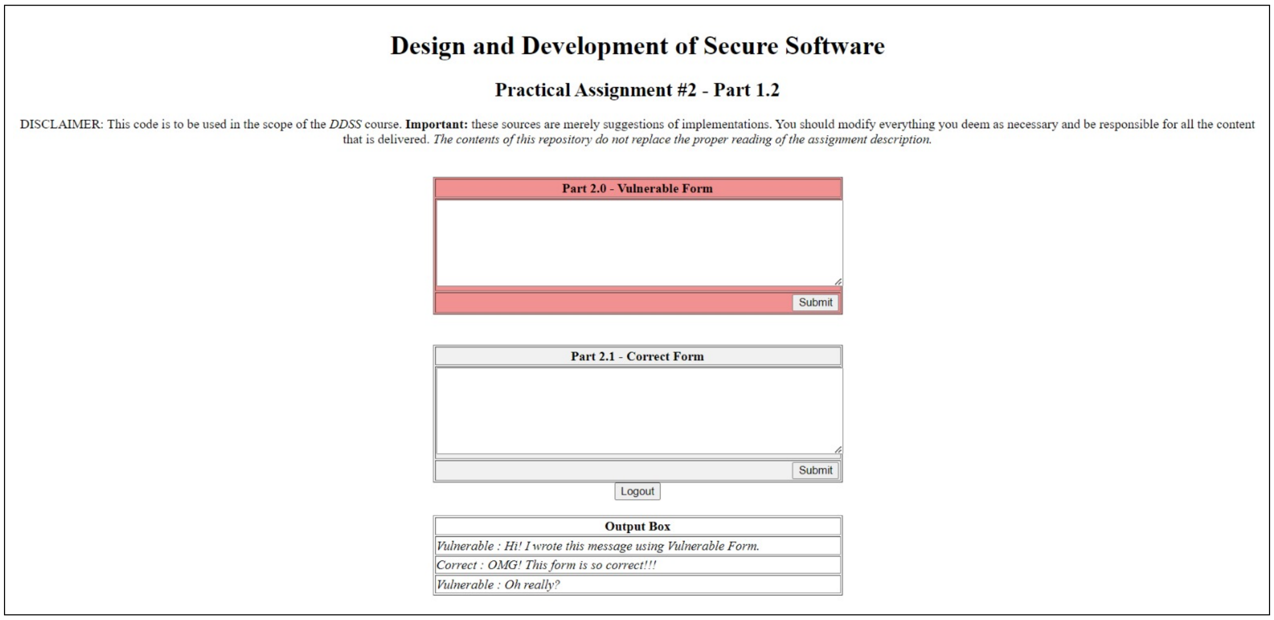
Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

Figure 1: Part 1.1

2.2 Part 1.2

The second functionality is a simple form where the user may introduce some text that is sent to the server to be stored in the database and then displayed back at the bottom of the page (as shown in the next figure). As before, the form at the top should be vulnerable, while the one at the bottom should be secure. The functionality is available only after a successful authentication.


Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

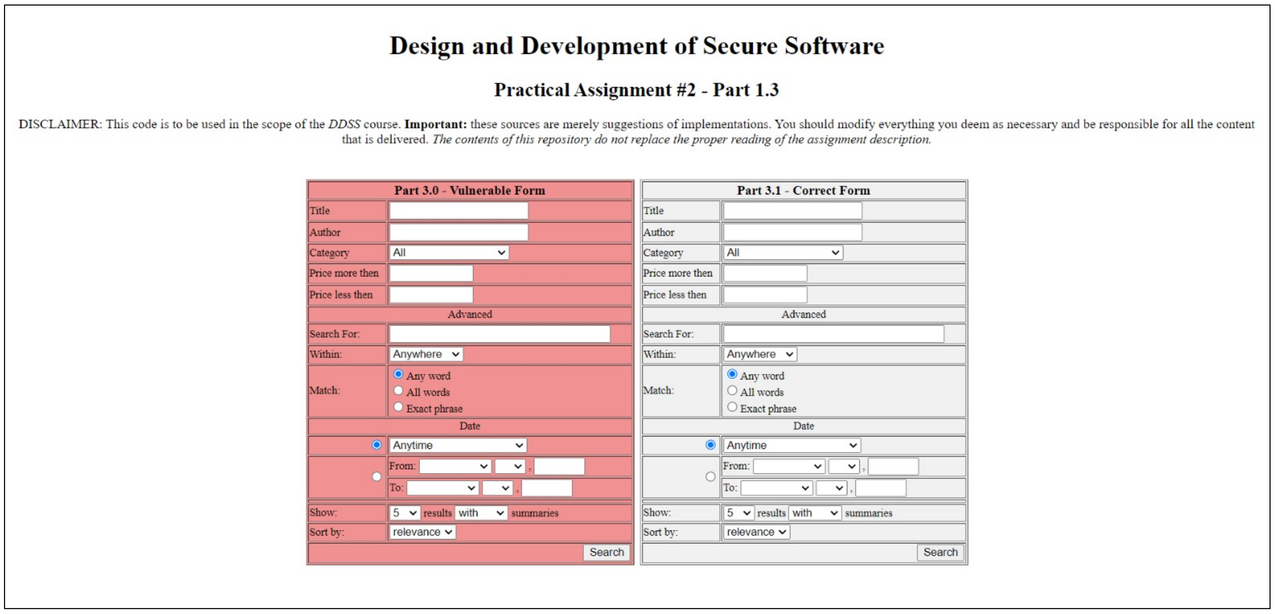
DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

Vulnerable : Hi! I wrote this message using Vulnerable Form.
Correct : OMG! This form is so correct!!!
Vulnerable : Oh really?

Figure 2: Part 1.2

2.3 Part 1.3

Finally, the last functionality consists of an advanced search form, where the user may fill different fields to define the search that will be done. The search should consider the information provided by the user and give correct answers.



Design and Development of Secure Software

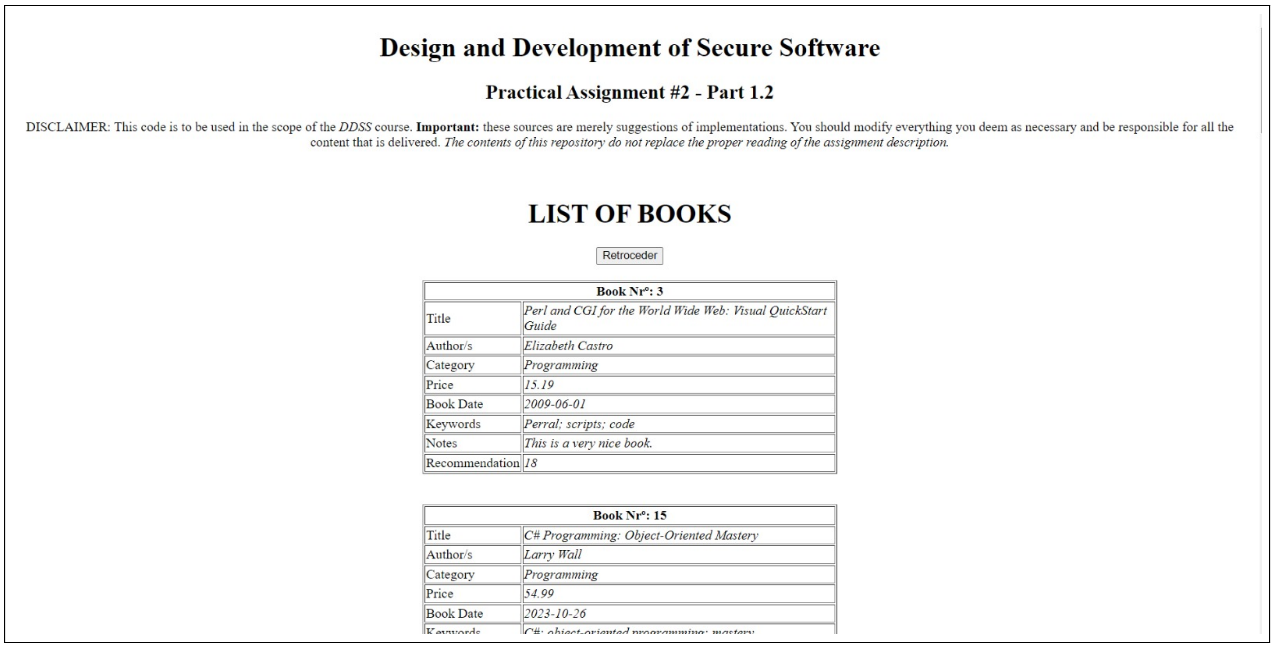
Practical Assignment #2 - Part 1.3

DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

Part 3.0 - Vulnerable Form		Part 3.1 - Correct Form	
Title		Title	
Author		Author	
Category	All	Category	All
Price more then		Price more then	
Price less then		Price less then	
Advanced		Advanced	
Search For:		Search For:	
Within:	Anywhere	Within:	Anywhere
Match:	[✓ Any word] All words Exact phrase	Match:	[✓ Any word] All words Exact phrase
Date		Date	
●	Anytime	●	Anytime
○	From: , To: ,	○	From: , To: ,
Show:	5 results with summaries	Show:	5 results with summaries
Sort by:	relevance	Sort by:	relevance
	Search		Search

Figure 3: Part 1.3

In practice, the output is the list of books that satisfy the search conditions (as shown in the next figure).



Design and Development of Secure Software

Practical Assignment #2 - Part 1.2

DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

LIST OF BOOKS

Retroceder

Book Nrº: 3	
Title	*Perl and CGI for the World Wide Web: Visual QuickStart Guide*
Author/s	*Elizabeth Castro*
Category	*Programming*
Price	*15.19*
Book Date	*2009-06-01*
Keywords	*Perral; scripts; code*
Notes	*This is a very nice book.*
Recommendation	*18*

Book Nrº: 15	
Title	*C# Programming: Object-Oriented Mastery*
Author/s	*Larry Wall*
Category	*Programming*
Price	*54.99*
Book Date	*2023-10-26*
Keywords	*C#; object-oriented programming; mastery*

Figure 4: Results from Part3

3 Analysis of Vulnerable Functions

In this section, each intentionally vulnerable function is discussed, outlining where and why it is vulnerable. The vulnerabilities include common web application security issues such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

3.1 Register User Function

In the implementation of the user registration function, a new table was created in the database to facilitate the creation of user accounts. This process exhibits several security weaknesses:

- Lack of Two-Factor Authentication (2FA): The registration procedure does not include the consideration of 2FA, thereby compromising the robustness of user verification processes.
- Unencrypted Password Storage: User passwords are stored directly in the database table without any form of encryption. This practice exposes user credentials to potential security breaches, undermining the privacy and integrity of user data.
- Vulnerability to SQL Injection: The method of inserting user data into the database is susceptible to SQL injection attacks. This vulnerability can lead to unauthorized access, data leakage, and other malicious activities, as it allows attackers to manipulate database queries through the input data.

These vulnerabilities signify a severe threat to the system's security and necessitate attention to reinforce the user registration process and safeguard user information.

3.2 Login Function

The initial implementation of the 'login' function was susceptible to Cross-Site Scripting (XSS). It did not properly sanitize user inputs, allowing attackers to inject malicious scripts that could be executed in other users' browsers. This vulnerability was particularly concerning as it could compromise user data and the integrity of the application.

Another security vulnerability arises from the system's response to incorrect user data input. Specifically, when users provide incorrect information, the system displays a message indicating which field contains the error. This behavior introduces a security risk as it potentially enables attackers to employ brute-force techniques to ascertain the existence of a valid user account, attempting to guess the correct password. The system outputs a success message and the entered data, which includes the username and password (which is not encrypted) if the user's authentication is successful.

3.3 Insert Message Function

After successful login in the web application, regardless of whether authentication was done using a vulnerable or correct method, a session is created to store the user's name. The user is then directed to part 1.2 of the project, where they can enter a message. This part displays a table listing messages in the database along with the authors of those messages. The vulnerability present in this section is SQL injection, which allows unauthorized access or manipulation of database content by injecting malicious SQL code through user input fields.

3.4 Search Books Function

The book search function encompasses a series of operations vulnerable to SQL injections. These operations generate SQL statements used to query books in the database, based on user-specified conditions. The vulnerability arises from the direct concatenation of user inputs into SQL queries. This practice permits an attacker to alter the query, potentially gaining unauthorized access to or modifying database information. Such a vulnerability significantly compromises the security and integrity of the database, underscoring the need for stringent input validation and parameterized queries in database operations, as per IEEE software engineering standards.

3.5 Monitoring the data

We implemented a specific function for monitoring the data stored within the database. This function, when invoked, allows any user to view all the data entries stored in the database. This feature, while useful for oversight purposes, introduces a significant security vulnerability. The primary concern stems from the absence of protective measures to regulate data visibility. As a result, this unrestricted access means that any individual, regardless of their authorization level, has the potential to access and view all types of information stored in the database. This lack of data protection poses serious risks in terms of data privacy and integrity, making the system susceptible to potential data breaches and unauthorized data exploitation. Addressing this vulnerability is crucial to ensuring the security and confidentiality of the stored information.

4 Analysis of Corrected Functions

This section discusses the equivalent functions that were correctly implemented without vulnerabilities. It explains the methods and coding practices used to ensure security.

4.1 Register User Function

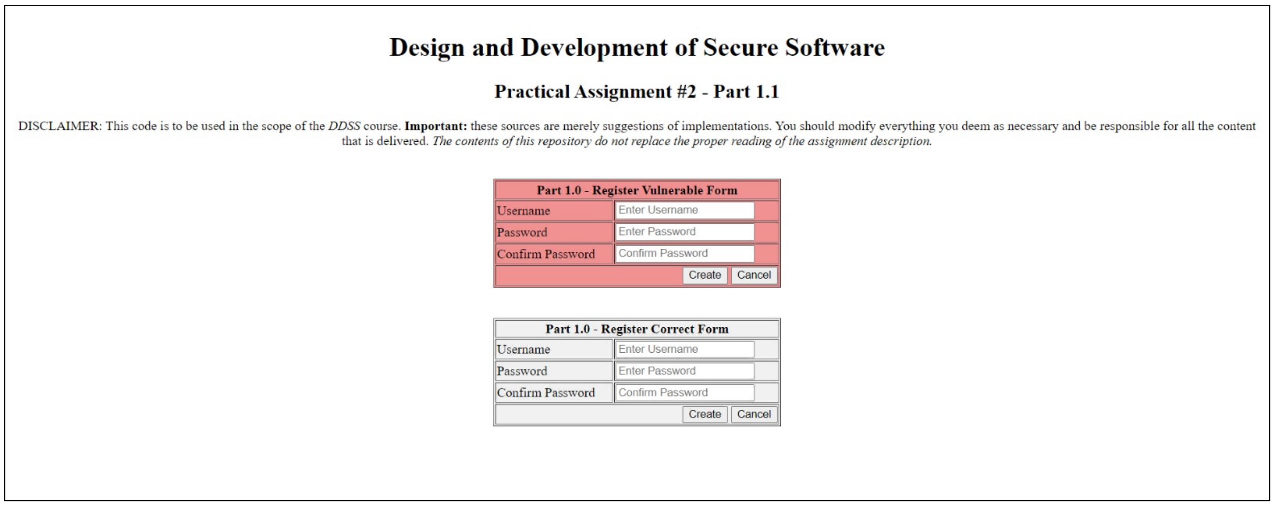
The process of registering a new user involves several critical steps to ensure security and functionality. Key steps in this process include:

- Data Entry with Encrypted Password: Users are required to input their details, including a password. The password is encrypted to ensure data security and privacy. The encryption mechanism prevents unauthorized access and enhances data integrity.

- Automatically Generated Authentication Token (2FA): Upon successful registration, an authentication token is automatically generated for each user. This token plays a crucial role in the second-factor authentication process, adding an extra layer of security. It ensures that the user's identity is verified through an additional method beyond the standard password login.

- Distinct User Table Structure: The database schema for storing user information in this secure setup is distinctly different from the schema used in the vulnerable version of the system. This distinction is critical to prevent common vulnerabilities and enhance data management.

- Protection Against SQL Injection: The function is robustly designed to resist SQL injection attacks. This is achieved through the use of prepared statements and parameterized queries, which effectively sanitize user input and prevent the execution of malicious SQL code. This safeguard is vital for maintaining the integrity and security of the database.



Design and Development of Secure Software

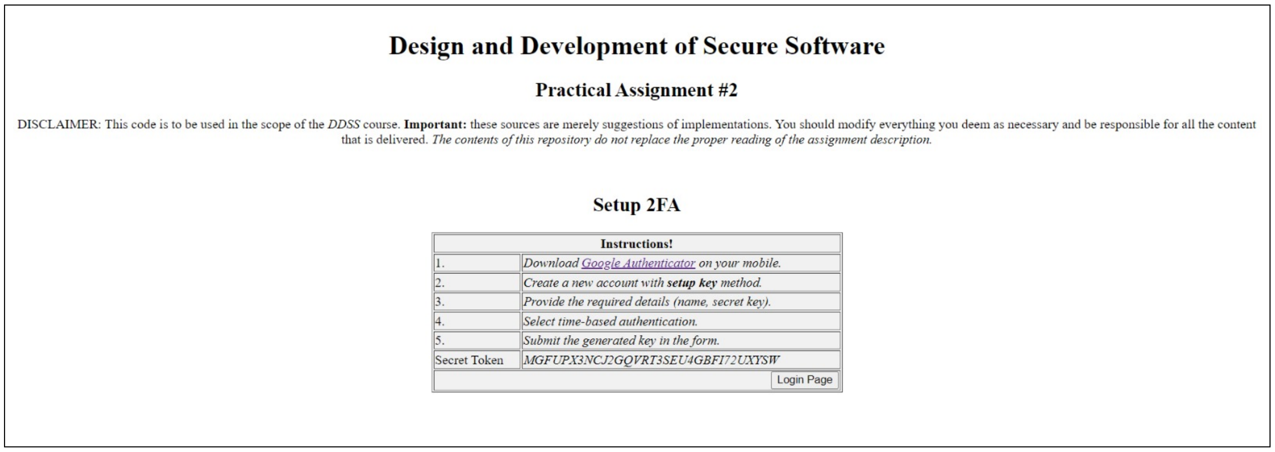
Practical Assignment #2 - Part 1.1

DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

Part 1.0 - Register Vulnerable Form	
Username	Enter Username
Password	Enter Password
Confirm Password	Confirm Password
	Create Cancel

Part 1.0 - Register Correct Form	
Username	Enter Username
Password	Enter Password
Confirm Password	Confirm Password
	Create Cancel

Figure 5: Register User Form


Design and Development of Secure Software

Practical Assignment #2

DISCLAIMER: This code is to be used in the scope of the *DDSS* course. **Important:** these sources are merely suggestions of implementations. You should modify everything you deem as necessary and be responsible for all the content that is delivered. *The contents of this repository do not replace the proper reading of the assignment description.*

Setup 2FA

Instructions!	
1.	*Download Google Authenticator on your mobile.*
2.	*Create a new account with **setup key** method.*
3.	*Provide the required details (name, secret key).*
4.	*Select time-based authentication.*
5.	*Submit the generated key in the form.*
Secret Token	*MGFUPX3NCJ2GQVRT3SEU4GBFI72UXYSW*

Figure 6: Setup 2FA

These measures collectively ensure a high level of security in the user registration process, safeguarding user data and maintaining system integrity.

4.2 Login Function

The corrected 'login' function implemented input sanitization to mitigate XSS risks. All user inputs were sanitized to remove or neutralize potentially malicious content before being processed or displayed. This approach significantly enhanced the security of the application by preventing the execution of malicious scripts. In contrast to systems with security vulnerabilities that provide explicit error messages upon incorrect data entry, this approach employs a more secure method. When a user enters data incorrectly, the system issues a general notification that the credentials are incorrect, deliberately avoiding any indication of whether the username or the password or the 2fa token is the issue. This method is particularly effective in reducing the efficacy of brute force attacks, where attackers attempt various combinations to gain unauthorized access to a user's account. By not indicating which part of the credentials is incorrect, the system increases the difficulty for attackers trying to identify the correct information, thereby enhancing the overall security of the user's account.

The system prints a success message and the entered data—password, username, and 2FA token—if the user's authentication is successful. The last two are encrypted.

4.3 Insert Message Function

Analogous to the previously discussed vulnerable component, this function shares a similar structure with one significant distinction: it incorporates appropriate measures to prevent SQL injection attacks. This proactive approach ensures robust data handling and enhances the security of user data transactions.

4.4 Search Books Function

We outlined a sequence of functions purposed to construct SQL statements for querying books within the database, contingent on criteria delineated by users. Notably, this system is resilient to SQL injection threats. This resilience is achieved by treating user inputs exclusively as parameters, rather than integrating them directly into the SQL query. As a result, the query's structural integrity is maintained regardless of user input variations. This parameterized approach acts as an effective safeguard, protecting the database against unauthorized access and manipulative activities.

5 Testing and Static Analysis Tools

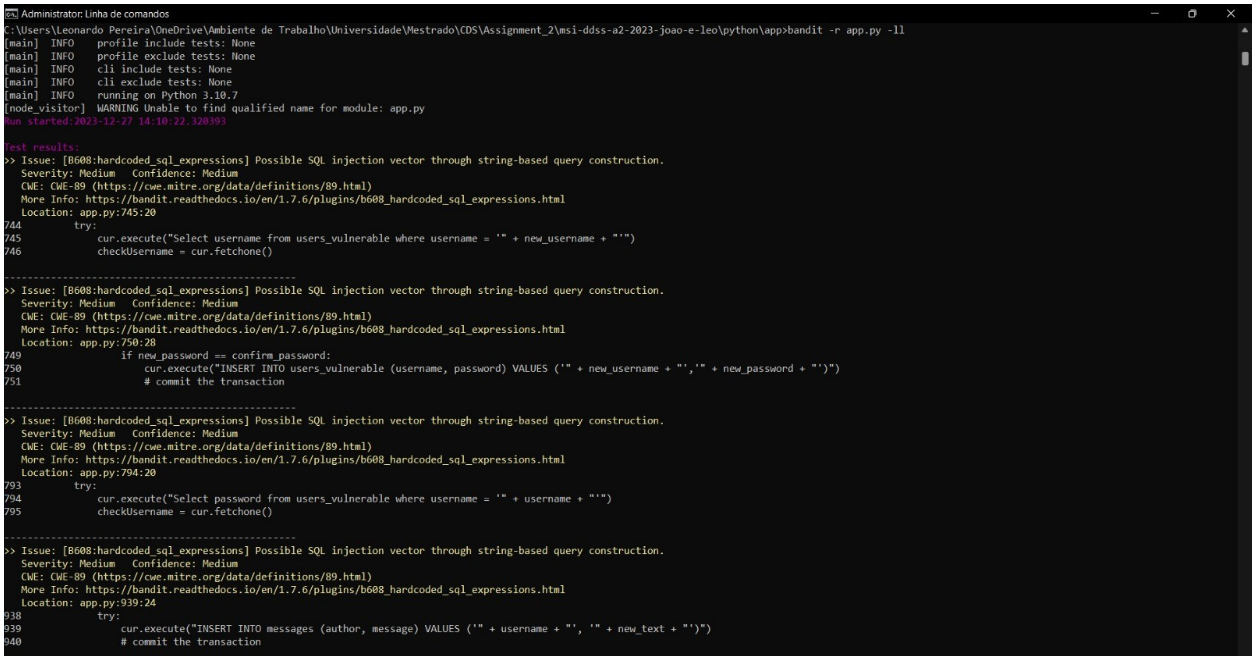
This section discusses the tools used for testing and static analysis of the code, highlighting their effectiveness and adequacy in identifying and mitigating vulnerabilities.

5.1 Bandit

Bandit, a tool for static analysis, was used to identify common security issues in Python code. It was particularly effective in detecting vulnerabilities like hard-coded passwords, SQL Injection risks, and insecure usage of subprocess calls. Its adequacy for this project lies in its ability to scan Python code and identify security issues at an early stage of development.

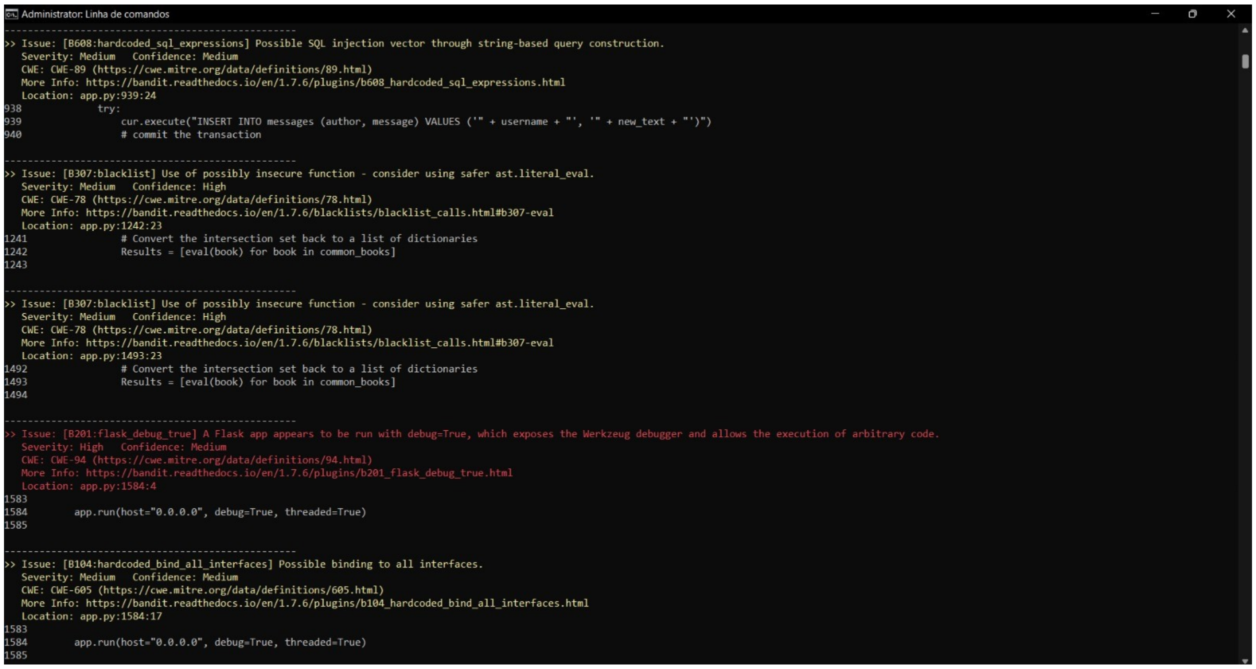
After executing Bandit on our application, numerous security issues were flagged. These issues were anticipated as we deliberately introduced vulnerabilities within the code to demonstrate the contrast between a vulnerable implementation and its secure counterpart. They findings are explained in detail as follows:

- SQL Injection Vulnerabilities (B608): Bandit has detected several instances of SQL queries that are constructed by directly concatenating strings with user inputs. This practice is inherently insecure as it opens up the possibility for SQL injection attacks. Attackers can exploit these vulnerabilities by crafting malicious input that could alter the structure of the SQL query, leading to unauthorized access or manipulation of the database. To mitigate these risks, it is recommended to use parameterized queries or Object-Relational Mapping (ORM) systems which safely handle user input.
- Unsafe Use of eval (B307): The analysis tool has raised a concern over the use of theeval function. When eval is used with input that comes from an untrusted source, it can lead to code injection vulnerabilities. This is because eval will execute the input as code, which could be malicious. The secure alternative to eval is to use ast.literal_eval, which safely evaluates an expression node or a string containing a Python literal or container display.
- Running Flask App in Debug Mode (B201): Bandit points out that the Flask application is configured to run in debug mode (debug=True). While useful during development, this setting is hazardous in a production environment. It can inadvertently expose sensitive information and debugging capabilities to attackers. Therefore, it is crucial to ensure that the debug mode is turned off in the production environment to prevent potential security breaches.
- Binding to All Interfaces (B104): The application is set to bind to all network interfaces, which might unintentionally expose the application to the internet or other networks. This configuration poses a security threat, especially if the application is intended to be confined within a private network or lacks proper security controls. A more secure approach is to bind the application only to the necessary interfaces or to implement robust firewall rules and authentication mechanisms to protect against unauthorized access.



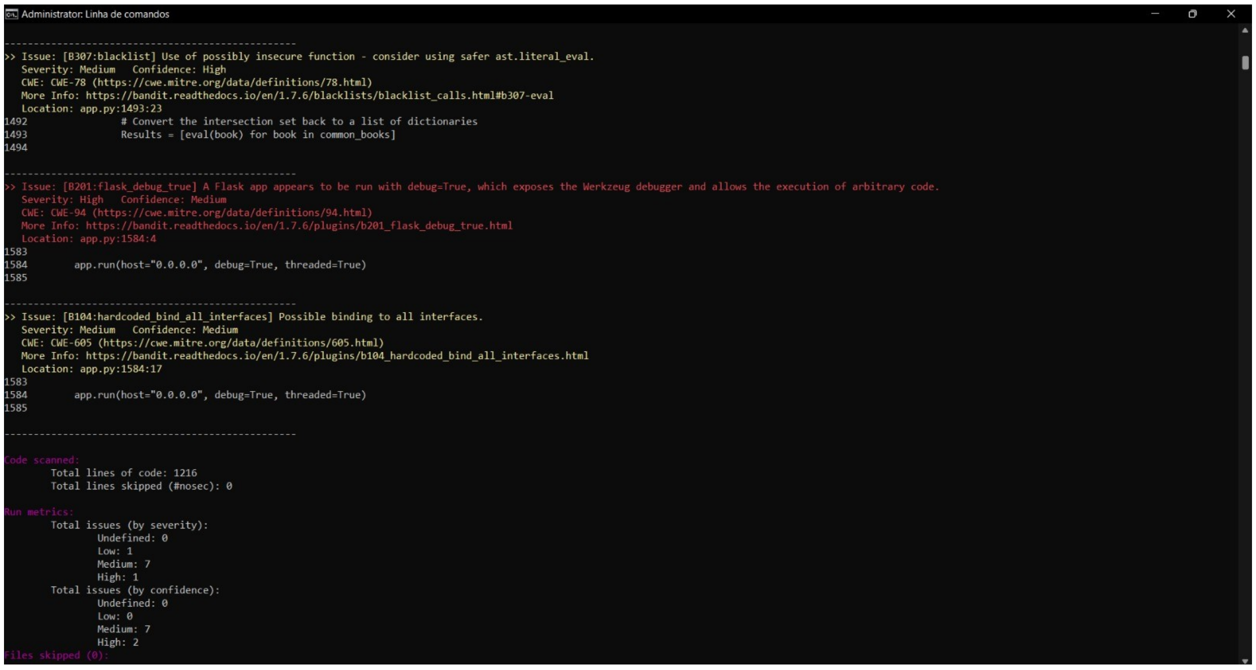
```
Administrator: Linha de comandos
C:\Users\Leonardo Pereira\OneDrive\Ambiente de Trabalho\Universidade\Mestrado\CDS\Assignment_2\msi-ddss-a2-2023-joao-e-leo\python\app>bandit -r app.py -ll
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.10.7
[node_visitor]  WARNING Unable to find qualified name for module: app.py
Run started:2023-12-27 14:10:22.320393

Test results:
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: app.py:745:20
744         try:
745             cur.execute("Select username from users_vulnerable where username = '" + new_username + "'")
746             checkUsername = cur.fetchone()

--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: app.py:750:28
749                 if new_password == confirm_password:
750                     cur.execute("INSERT INTO users_vulnerable (username, password) VALUES ('" + new_username + "','" + new_password + "')")
751                     # commit the transaction

--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: app.py:794:20
793         try:
794             cur.execute("Select password from users_vulnerable where username = '" + username + "'")
795             checkUsername = cur.fetchone()

--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: app.py:939:24
938             try:
939                 cur.execute("INSERT INTO messages (author, message) VALUES ('" + username + "', '" + new_text + "')")
940                 # commit the transaction
```

Figure 7: Results from Bandit



```
Administrator: Linha de comandos
--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b608_hardcoded_sql_expressions.html
   Location: app.py:939:24
938             try:
939                 cur.execute("INSERT INTO messages (author, message) VALUES ('" + username + "', '" + new_text + "')")
940                 # commit the transaction

--------------------------------------------------
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using safer ast.literal_eval.
   Severity: Medium   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_calls.html#b307-eval
   Location: app.py:1242:23
1241                # Convert the intersection set back to a list of dictionaries
1242                Results = [eval(book) for book in common_books]
1243

--------------------------------------------------
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using safer ast.literal_eval.
   Severity: Medium   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_calls.html#b307-eval
   Location: app.py:1493:23
1492                # Convert the intersection set back to a list of dictionaries
1493                Results = [eval(book) for book in common_books]
1494

--------------------------------------------------
>> 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 arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b201_flask_debug_true.html
   Location: app.py:1584:4
1583
1584        app.run(host="0.0.0.0", debug=True, threaded=True)
1585

--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b104_hardcoded_bind_all_interfaces.html
   Location: app.py:1584:17
1583
1584        app.run(host="0.0.0.0", debug=True, threaded=True)
1585
```

Figure 8: Results from Bandit



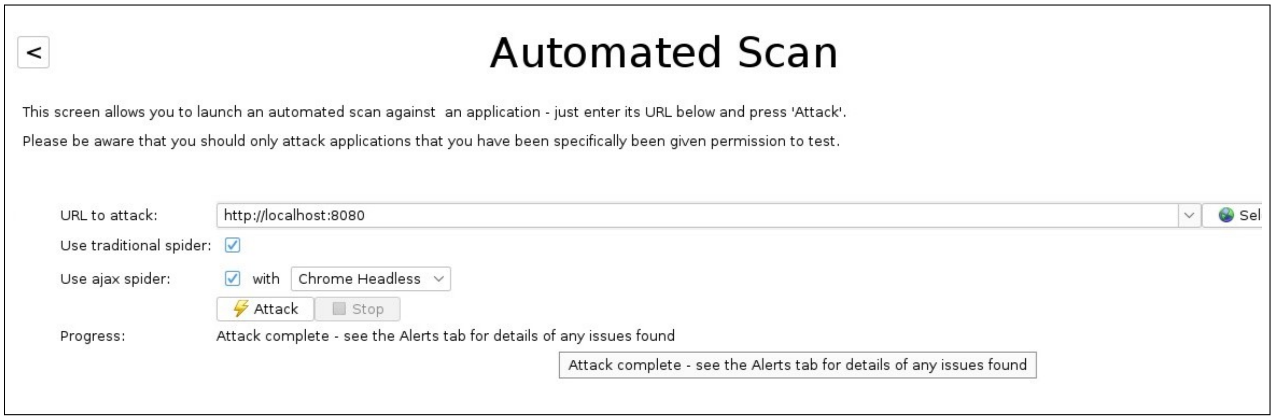
```
--------------------------------------------------
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using safer ast.literal_eval.
   Severity: Medium   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/blacklists/blacklist_calls.html#b307-eval
   Location: app.py:1493:23
1492                # Convert the intersection set back to a list of dictionaries
1493                Results = [eval(book) for book in common_books]
1494

--------------------------------------------------
>> 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 arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b201_flask_debug_true.html
   Location: app.py:1584:4
1583
1584        app.run(host="0.0.0.0", debug=True, threaded=True)
1585

--------------------------------------------------
>> 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)
   More Info: https://bandit.readthedocs.io/en/1.7.6/plugins/b104_hardcoded_bind_all_interfaces.html
   Location: app.py:1584:17
1583
1584        app.run(host="0.0.0.0", debug=True, threaded=True)
1585

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

Code scanned:
        Total lines of code: 1216
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 7
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 7
                High: 2
Files skipped (0):
```

Figure 9: Results from Bandit

After reviewing the vulnerabilities, the analysis of the system reveals critical oversights. It failed to detect a crucial flaw in the login process. Upon receiving incorrect data, the system notifies the user which field is incorrectly filled. This feature can unintentionally assist attackers in orchestrating brute force attacks by narrowing down their target fields. Secondly, a significant oversight was identified in the data monitoring function. This feature remains unprotected, granting unrestricted access to sensitive database information. Lastly, the analysis overlooked several SQL Injection vulnerabilities in key segments of the system, particularly in sections 1.2 and 1.3, posing a severe security risk. This overview highlights the necessity for a more thorough security assessment in these areas.

5.2 OWASP ZAP

The dynamic analysis focusing of the web application security was conducted using OWASP ZAP (Zed Attack Proxy). This tool, functioning as a man-in-the-middle proxy, identifies security vulnerabilities in web applications by allowing the analysis and manipulation of traffic between a browser and a web server. The selection of this tool was based on its comprehensive scanning capabilities, which are essential for thoroughly assessing web application security. It facilitated the identification and assessment of potential weaknesses that could be exploited by attackers.



Automated Scan

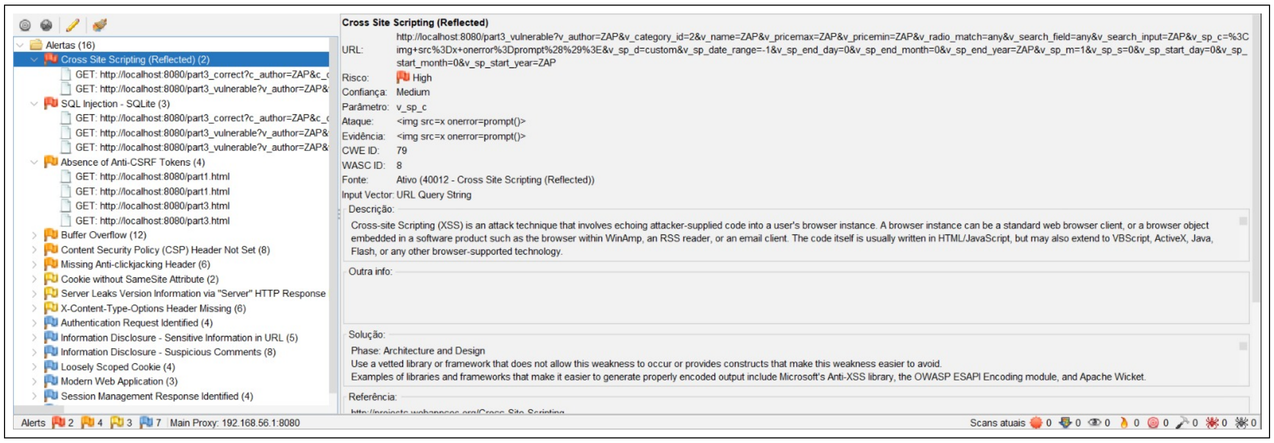
This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

```
URL to attack:            http://localhost:8080
Use traditional spider:   ☑
Use ajax spider:          ☑ with Chrome Headless
                          Attack   Stop
Progress:                 Attack complete - see the Alerts tab for details of any issues found
```

Attack complete - see the Alerts tab for details of any issues found

Figure 10: ZAP Scan Setup

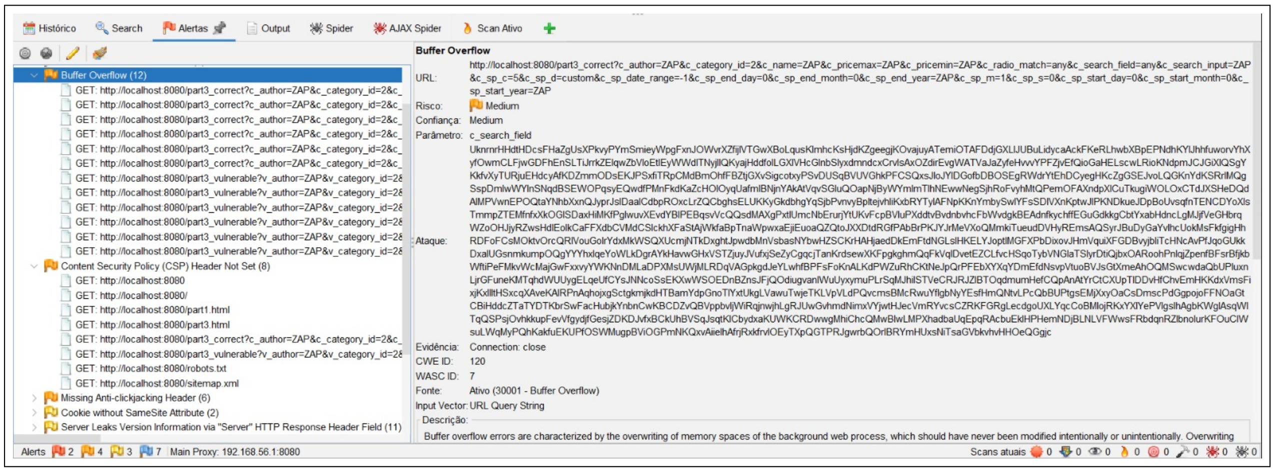


```
Alertas (16)
  Cross Site Scripting (Reflected) (2)
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_c
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&
  SQL Injection - SQLite (3)
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_c
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&
  Absence of Anti-CSRF Tokens (4)
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part3.html
    GET: http://localhost:8080/part3.html
  Buffer Overflow (12)
  Content Security Policy (CSP) Header Not Set (8)
  Missing Anti-clickjacking Header (6)
  Cookie without SameSite Attribute (2)
  Server Leaks Version Information via "Server" HTTP Response
  X-Content-Type-Options Header Missing (6)
  Authentication Request Identified (4)
  Information Disclosure - Sensitive Information in URL (5)
  Information Disclosure - Suspicious Comments (8)
  Loosely Scoped Cookie (4)
  Modern Web Application (3)
  Session Management Response Identified (4)
```

```
Cross Site Scripting (Reflected)
             http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&v_name=ZAP&v_pricemax=ZAP&v_pricemin=ZAP&v_radio_match=any&v_search_field=any&v_search_input=ZAP&v_sp_c=%3C
URL:         img+src%3Dx+onerror%3Dprompt%28%29%3E&v_sp_d=custom&v_sp_date_range=-1&v_sp_end_day=0&v_sp_end_month=0&v_sp_end_year=ZAP&v_sp_m=1&v_sp_s=0&v_sp_start_day=0&v_sp_
             start_month=0&v_sp_start_year=ZAP
Risco:       High
Confiança:   Medium
Parâmetro:   v_sp_c
Ataque:      <img src=x onerror=prompt()>
Evidência:   <img src=x onerror=prompt()>
CWE ID:      79
WASC ID:     8
Fonte:       Ativo (40012 - Cross Site Scripting (Reflected))
Input Vector: URL Query String
Descrição:
  Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object
  embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java,
  Flash, or any other browser-supported technology.
Outra info:
Solução:
  Phase: Architecture and Design
  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
  Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.
Referência:
```

```
Alerts 2 4 3 7   Main Proxy: 192.168.56.1:8080                Scans atuais 0 0 0 0 0 0 0 0
```

Figure 11: ZAP Results

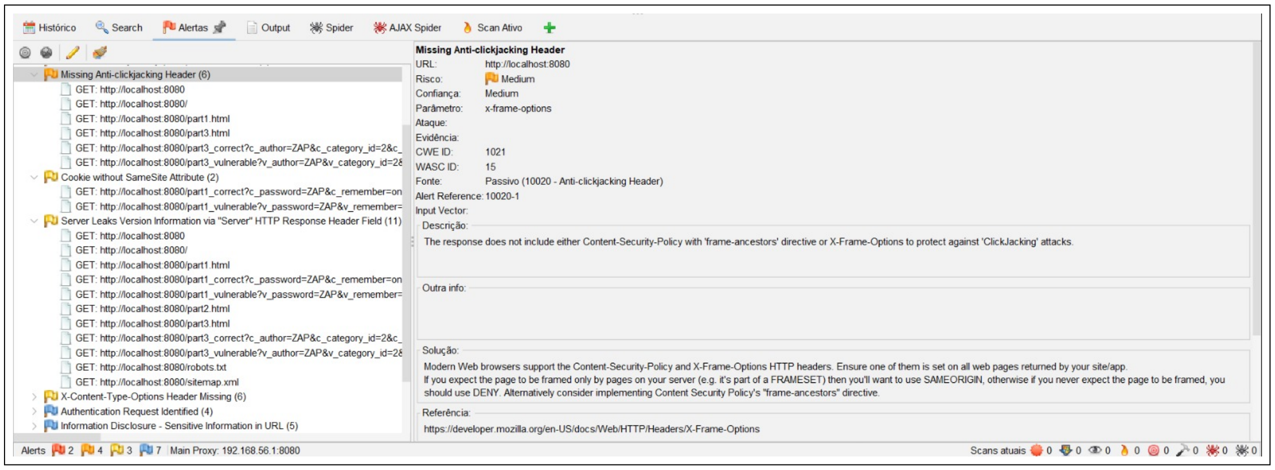


```
Buffer Overflow (12)
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
Content Security Policy (CSP) Header Not Set (8)
  GET: http://localhost:8080
  GET: http://localhost:8080/
  GET: http://localhost:8080/part1.html
  GET: http://localhost:8080/part3.html
  GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
  GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
  GET: http://localhost:8080/robots.txt
  GET: http://localhost:8080/sitemap.xml
Missing Anti-clickjacking Header (6)
Cookie without SameSite Attribute (2)
Server Leaks Version Information via "Server" HTTP Response Header Field (11)
```

```
Buffer Overflow
             http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_name=ZAP&c_pricemax=ZAP&c_pricemin=ZAP&c_radio_match=any&c_search_field=any&c_search_input=ZAP
URL:         &c_sp_c=5&c_sp_d=custom&c_sp_date_range=-1&c_sp_end_day=0&c_sp_end_month=0&c_sp_end_year=ZAP&c_sp_m=1&c_sp_s=0&c_sp_start_day=0&c_sp_start_month=0&c_
             sp_start_year=ZAP
Risco:       Medium
Confiança:   Medium
Parâmetro:   c_search_field
             UknrnrHHdtHDcsFHaZgUsXPkvyPYmSmieyWpgFxnJOWvrXZfijlVTGwXBoLqusKlmhcKsHjdKZgeegjKOvajuyATemiOTAFDdjGXLIJUBuLidycaAckFKeRLhwbXBpEPNdhKYIJhhfuworvYhX
             yfOwmCLFjwGDFhEnSLTiJrrkZElqwZbVloEtlEyWWdlTNyjllQKyajHddfolLGXlVHcGlnbSlyxdmndcxCrvlsAxOZdirEvgWATVaJaZyfeHvvvYPFZjvEfQioGaHELscwLRioKNdpmJCJGiXlQSgY
             KkfvXyTURjuEHdcyAfKDZmmODsEKJPSxfiTRpCMdBmOhfFBZtjGXvSigcotxyPSvDUSqBVUVGhkPFCSQxsJloJYIDGofbDBOSEgRWdrYtEhDCyegHKcZgGSEJvoLQGKnYdKSRrlMQg
             SspDmlwWYInSNqdBSEWOPqsyEQwdfPMnFkdKaZcHOlOyqUafmlBNjnYAkAtVqvSGluQOapNjByWYmlmTlhNEwwNegSjhRoFvyhMtQPemOFAXndpXlCuTkugiWOLOxCTdJXSHeDQd
             AlMPVwnEPOQtaYNhbXxnQJyprJslDaalCdbpROxcLrZQCbghsELUKKyGkdbhgYqSjbPvnvyBpltejvhliKxbRYTylAFNpKKnYmbySwlYFsSDlVXnKptwJlPKNDkueJDpBoUvsqfnTENCDYoXls
             TmmpZTEMfnfxXkOGlSDaxHiMKfPglwuvXEvdYBIPEBqsvVcQQsdMAXgPxtlUmcNbErurjYtUKvFcpBVluPXddtvBvdnbvhcFbWvdgkBEAdnfkychffEGuGdkkgCbtYxabHdncLgMJjfVeGHbrq
             WZoOHJjyRZwsHdlEolkCaFFXdbCVMdCSlckhXFaStAjWkfaBpTnaWpwxaEjiEuoaQZQtoJXXDtdRGfPAbBrPKJYJrMeVXoQMmkiTueudDVHyREmsAQSyrJBuDyGaYvlhcUokMsFkfgigHh
Ataque:      RDFoFCsMOktvOrcQRlVouGolrYdxMkWSQXUcmjNTkDxghtJpwdbMnVsbasNYbwHZSCKrHAHjaedDkEmFtdNGLslHKELYJoptlMGFXPbDixovJHmVquiXFGDBvyjbliTcHNcAvPfJqoGUkk
             DxalUGsnmkumpOQgYYYhxlqeYoWLkDgrAYkHavwGHxVSTZjuyJVufxjSeZyCgqcjTanKrdsewXKFpgkghmQqFkVqlDvetEZCLfvcHSqoTybVNGlaTSlyrDtiQjbxOARoohPnlqjZpenfBFsrBfjkb
             WftiPeFMkvWcMajGwFxxvyYWKNnDMLaDPXMsUWjMLRDqVAGpkgdJeYLwhfBPFsFoKnALKdPWZuRhCKtNeJpQrPFEbXYXqYDmEfdNsvpVtuoBVJsGtXmeAhOQMSwcwdaQbUPluxn
             LjrGFuneKMTqhdWUUygELqeUfCYsJNNcoSsEKXwWSOEDnBZnsJFjQOdiugvanlWuUyxymuPLrSqMJhilSTVeCRJRJZlBTOqdmumHefCQpAnAtYrCtCXUpTIDDvHfChvEmHKKdxVmsFi
             xjKxlIltHSxcqXAveKAlRPnAqhojxgSctgkmjkdHTBamYdpGnoTlYxtUkgLVawuTwjeTKLVpVLdPQvcmsBMcRwuYflgbNyYEsfHmQNtvLPcQbBUPtgsEMjXxyOaCsDmscPdGgpojoFFNOaGt
             CBiHddcZTaTYDTKbrSwFacHubjkYnbnCwKBCDZvQBVppbvljWiRqjnwjhLgRJUwGvhmdNimxVYjwtHJecVmRYvcsCZRKFGRgLecdgoUXLYqcCoBMlojRKxYXlYePVlgslhAgbKWglAsqWl
             TqQSPsjOvhkkupFevVfgydjfGesjZDKDJvfxBCkUhBVSqJsqtKlCbydxaKUWKCRDwwgMhiChcQMwBlwLMPXhadbaUqEpqRAcbuEklHPHemNDjBLNLVFWwsFRbdqnRZlbnolurKFOuClW
             suLWqMyPQhKakfuEKUPfOSWMugpBViOGPmNKQxvAiielhAfrjRxkfrvlOEyTXpQGTPRJgwrbQOrlBRYmHUxsNiTsaGVbkvhvHHOeQGgjc
Evidência:   Connection: close
CWE ID:      120
WASC ID:     7
Fonte:       Ativo (30001 - Buffer Overflow)
Input Vector: URL Query String
Descrição:
  Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally. Overwriting
```

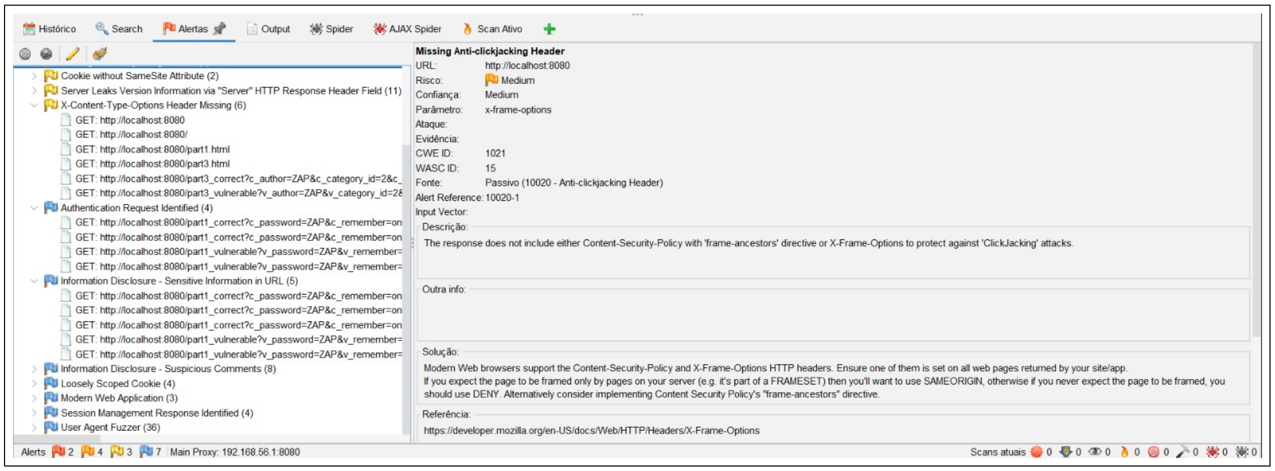
```
Alerts 2 4 3 7   Main Proxy: 192.168.56.1:8080                Scans atuais 0 0 0 0 0 0 0 0
```

Figure 12: ZAP Results



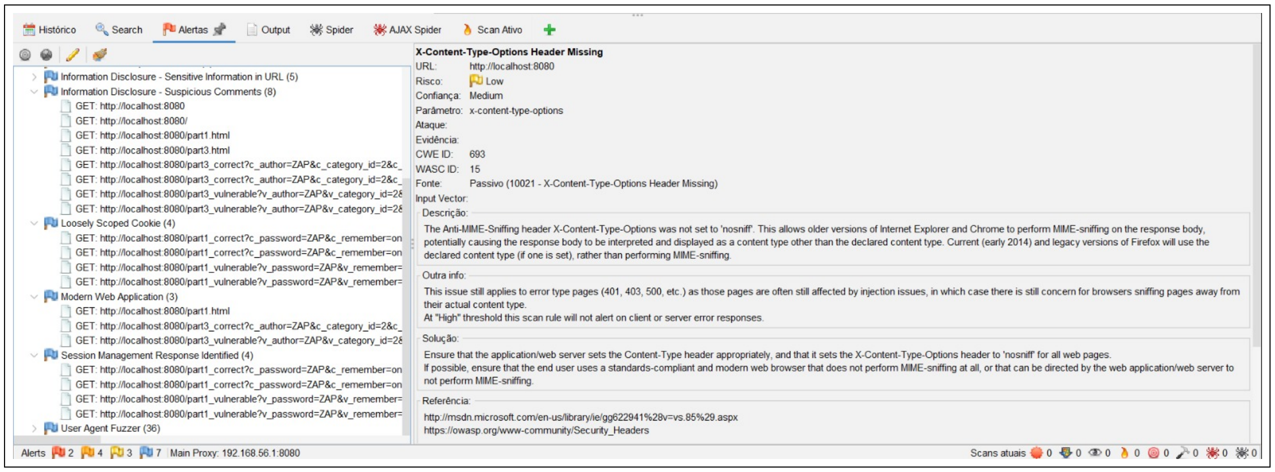
```
Histórico   Search   Alertas   Output   Spider   AJAX Spider   Scan Ativo

Missing Anti-clickjacking Header (6)
    GET: http://localhost:8080
    GET: http://localhost:8080/
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part3.html
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
Cookie without SameSite Attribute (2)
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
Server Leaks Version Information via "Server" HTTP Response Header Field (11)
    GET: http://localhost:8080
    GET: http://localhost:8080/
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
    GET: http://localhost:8080/part2.html
    GET: http://localhost:8080/part3.html
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
    GET: http://localhost:8080/robots.txt
    GET: http://localhost:8080/sitemap.xml
X-Content-Type-Options Header Missing (6)
Authentication Request Identified (4)
Information Disclosure - Sensitive Information in URL (5)

Missing Anti-clickjacking Header
URL:             http://localhost:8080
Risco:           Medium
Confiança:       Medium
Parâmetro:       x-frame-options
Ataque:
Evidência:
CWE ID:          1021
WASC ID:         15
Fonte:           Passivo (10020 - Anti-clickjacking Header)
Alert Reference: 10020-1
Input Vector:
Descrição:
 The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
Outra info:
Solução:
 Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.
 If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you
 should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.
Referência:
 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Alerts 2  4  3  7  Main Proxy: 192.168.56.1:8080          Scans atuais 0  0  0  0  0  0  0  0
```

Figure 13: ZAP Results



```
Histórico   Search   Alertas   Output   Spider   AJAX Spider   Scan Ativo

Cookie without SameSite Attribute (2)
Server Leaks Version Information via "Server" HTTP Response Header Field (11)
X-Content-Type-Options Header Missing (6)
    GET: http://localhost:8080
    GET: http://localhost:8080/
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part3.html
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
Authentication Request Identified (4)
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
Information Disclosure - Sensitive Information in URL (5)
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
Information Disclosure - Suspicious Comments (8)
Loosely Scoped Cookie (4)
Modern Web Application (3)
Session Management Response Identified (4)
User Agent Fuzzer (36)

Missing Anti-clickjacking Header
URL:             http://localhost:8080
Risco:           Medium
Confiança:       Medium
Parâmetro:       x-frame-options
Ataque:
Evidência:
CWE ID:          1021
WASC ID:         15
Fonte:           Passivo (10020 - Anti-clickjacking Header)
Alert Reference: 10020-1
Input Vector:
Descrição:
 The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
Outra info:
Solução:
 Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.
 If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you
 should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.
Referência:
 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Alerts 2  4  3  7  Main Proxy: 192.168.56.1:8080          Scans atuais 0  0  0  0  0  0  0  0
```

Figure 14: ZAP Results



```
Histórico   Search   Alertas   Output   Spider   AJAX Spider   Scan Ativo

Information Disclosure - Sensitive Information in URL (5)
Information Disclosure - Suspicious Comments (8)
    GET: http://localhost:8080
    GET: http://localhost:8080/
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part3.html
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
Loosely Scoped Cookie (4)
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
Modern Web Application (3)
    GET: http://localhost:8080/part1.html
    GET: http://localhost:8080/part3_correct?c_author=ZAP&c_category_id=2&c_
    GET: http://localhost:8080/part3_vulnerable?v_author=ZAP&v_category_id=2&
Session Management Response Identified (4)
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_correct?c_password=ZAP&c_remember=on
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
    GET: http://localhost:8080/part1_vulnerable?v_password=ZAP&v_remember=
User Agent Fuzzer (36)

X-Content-Type-Options Header Missing
URL:        http://localhost:8080
Risco:      Low
Confiança:  Medium
Parâmetro:  x-content-type-options
Ataque:
Evidência:
CWE ID:     693
WASC ID:    15
Fonte:      Passivo (10021 - X-Content-Type-Options Header Missing)
Input Vector:
Descrição:
 The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body,
 potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the
 declared content type (if one is set), rather than performing MIME-sniffing.
Outra info:
 This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from
 their actual content type.
 At "High" threshold this scan rule will not alert on client or server error responses.
Solução:
 Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
 If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to
 not perform MIME-sniffing.
Referência:
 http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx
 https://owasp.org/www-community/Security_Headers

Alerts 2  4  3  7  Main Proxy: 192.168.56.1:8080          Scans atuais 0  0  0  0  0  0  0  0
```

Figure 15: ZAP Results

The automated scan results revealed multiple security vulnerabilities in the application running on http://localhost:8080. Key findings from the scan are summarized below:

- Cross-Site Scripting (Reflected): The scan has detected at least two instances where user input is reflected back in the response without proper sanitization, making it susceptible to XSS attacks. This is where an attacker can inject client-side scripts into web pages viewed by other users. It's listed under GET requests to endpoints like /part3_correct and /part3_vulnerable.

- SQL Injection - SQLite: There are three instances identified that suggest the application's SQL queries could be vulnerable to injection attacks, where an attacker could manipulate the queries to access or manipulate the database. This is indicated by GET requests to /part3_correct and /part3_vulnerable.

- Absence of Anti-CSRF Tokens: Four instances the application might not be using anti-CSRF tokens, necessary to prevent Cross-Site Request Forgery attacks, where unauthorized commands are transmitted from a user that the web application trusts. The specific parts triggered are as follows: GET /localhost:8080/part1.html, GET /localhost:8080/part3.html.

- Buffer Overflow Vulnerabilities: The buffer overflow vulnerabilities are found in the application at multiple endpoints. These are areas where the application may accept more data than it is safely configured to handle, which can lead to overwriting valid data, causing crashes or code execution vulnerabilities. The specific endpoints where buffer overflow issues were detected are:
  - GET /localhost:8080/part3_correct
  - GET /localhost:8080/part3_vulnerable

- Content Security Policy (CSP) Header Not Set: The absence of CSP headers has been noted in responses from the application, which increases the risk of XSS and other client-side attacks. The CSP header is important because it helps prevent the execution of unauthorized scripts by declaring which dynamic resources are allowed to load. The specific resources where the CSP header is not set include:
  - The root directory GET localhost:8080/
  - HTML pages GET /localhost:8080/part1.html and GET /localhost:8080/part3.html
  - Resource directories GET /localhost:8080/robots.txt and GET /localhost:8080/sitemap.xml

  These points indicate that the application lacks a crucial security header across these resources, which should ideally be included in the HTTP response to enforce client-side security policies.

- Missing Anti-clickjacking Header: Anti-clickjacking headers, like X-Frame-Options, are HTTP response headers used to prevent clickjacking attacks, where a user's interaction is hijacked to click on something different from what the user perceives. The absence of this header means the application could be embedded into a frame or iframe, which can be abused for clickjacking attacks. The specific resources lacking this header are:
  - The root directory GET localhost:8080/
  - HTML pages GET /localhost:8080/part1.html and GET /localhost:8080/part3.html
  - Pages with parameter GET /localhost:8080/part3_correct and GET /localhost:8080/part3_vulnerable

- Cookie without SameSite Attribute: The SameSite attribute on a cookie is a security measure that prevents the browser from sending this cookie along with cross-site requests. The primary goal is to mitigate the risk of cross-origin information leakage. It also provides some protection against cross-site request forgery attacks. Resources missing the SameSite attribute in their cookies are:

  - GET /localhost:8080/part1_vulnerable
  - GET /localhost:8080/part1_correct

- Server Leaks Version Information via 'Server' HTTP Response Header: This alert indicates that the HTTP response headers contain information about the server's software version, which could be used by an attacker to exploit known vulnerabilities specific to the version of the software being used. The resources where the server version information is disclosed include:
  - The root directory GET `localhost:8080/`)
  - HTML pages GET /localhost:8080/part1.html and GET /localhost:8080/part3.html
  - Pages with parameters GET /localhost:8080/part1_correct, GET /localhost:8080/part1_vulnerable, GET /localhost:8080/part3_correct and GET /localhost:8080/part3_vulnerable
  - Resource directories GET /localhost:8080/robots.txt and GET /localhost:8080/sitemap.xml

- X-Content-Type-Options Header Missing: This header is used to protect against MIME type sniffing where the browser may ignore the server-provided MIME type and try to detect the MIME type itself. If this header is missing, attackers can exploit this behavior to deliver a malicious payload that the browser executes. The alerts for this missing header are found in the responses for:
  - The root directory GET /localhost:8080/
  - HTML pages GET /localhost:8080/part1.html
  - Pages with parameters: GET /localhost:8080/part3_correct and GET /localhost:8080/part3_vulnerable

- Authentication Request Identified: This suggests that the scan has picked up on pages or requests that involve authentication mechanisms. If these requests are not properly secured, they could be vulnerable to attacks like credential stuffing or brute force attacks. The specific requests identified are:
  - Pages with parameters GET /localhost:8080/part1_correct and GET /localhost:8080/part1_vulnerable

- Information Disclosure - Sensitive Information in URL: Sensitive information should not be included in URLs since URLs are logged in various places such as browser history, web server logs, and proxy server logs. This can lead to exposure of sensitive data like passwords. The scan has detected this type of information disclosure in:
  - Pages with parameters GET /localhost:8080/part1_correct and GET /localhost:8080/part1_vulnerable

  Authentication mechanisms should ensure secure transmission, such as HTTPS, to prevent the exposure of credentials. Finally, sensitive information should never be passed in the URL; instead, it should be transmitted in a more secure manner such as HTTP POST requests or using encryption.

- Information Disclosure - Suspicious Comments: Suspicious comments in the source code can sometimes reveal internal information that should not be exposed to users or potential attackers. This includes comments that developers might leave in the code, which could give insights into the application's backend, such as file paths, TODOs, or debugging information. The alerts for suspicious comments were identified in:
  - The root directory GET /localhost:8080/
  - HTML pages GET /localhost:8080/part1.html and GET /localhost:8080/part3.html
  - Pages with parameters GET /localhost:8080/part3_correct and GET /localhost:8080/part3_vulnerable

- Loosely Scoped Cookie: Cookies should be scoped as narrowly as possible. If a cookie is set for a broad scope of domains or paths, it can be sent across different applications or areas where it's not needed or intended, potentially leading to security risks. The scan has detected loosely scoped cookies in requests for:
  - Pages with parameters GET /localhost:8080/part1_correct and GET /localhost:8080/part1_vulnerable

- Modern Web Application: This alert suggests that the application uses modern web application techniques, such as AJAX or single-page applications. While this is not a vulnerability per se, it implies that traditional security tools may not effectively audit the application without additional configuration or techniques that understand these technologies. The specific requests identified are:
  - Pages with parameters GET /localhost:8080/part1_correct and GET /localhost:8080/part3_vulnerable

- Session Management Response Identified: Proper session management is critical for the security of web applications. This alert indicates that the scan has identified issues with how sessions are managed, potentially exposing the application to risks such as session hijacking or fixation. The specific endpoints identified are:
  - Pages with parameters GET /localhost:8080/part1_correct and GET /localhost:8080/part1_vulnerable

- User Agent Fuzzer: This is likely an indication that the scanner used a technique called fuzzing, where a large number of random or unexpected inputs (in this case, user agent strings) are sent to the application to see how it behaves. This can help identify vulnerabilities like buffer overflows or error handling issues. The specific endpoints where fuzzing was performed are not detailed in the alert summary provided in the image. These identified issues highlight the need for thorough code reviews to remove unnecessary comments, proper scoping of cookies, appropriate handling of modern web application technologies, robust session management practices, and comprehensive testing to ensure the application can handle unexpected inputs gracefully.

After analyzing all alerts, it is important to note that the ZAP tool did not register any vulnerabilities in part 2. This is because if the program tried to access that part, it would be redirected to the login form due to the session mechanism, which only allows access once a user is authenticated. Consequently, the program was incapable of detecting SQL injections, XSS, or other vulnerabilities in the part2_vulnerable.

These vulnerabilities highlight the importance of thorough security testing and the need for immediate remediation to enhance the application's resilience against potential cyber-attacks.

6 Conclusion

The project successfully demonstrates the importance of secure coding practices. By comparing vulnerable and corrected versions of the same functionality, it highlights how vulnerabilities can be introduced and, more importantly, how they can be mitigated or prevented. The use of tools like Bandit and ZAP further emphasizes the role of testing and analysis in ensuring application security.