CREATING WEALTH
FOR WELLBEING

NLC INDIA LIMITED

"NAVRATNA – A GOVERNMENT OF INDIA ENTERPRISE"

INTERNSHIP REPORT ON

WebSec_AutoGuard

A Basic Vulnerability Scanner Using Python

SUBMITTED BY

NIRANJANA V

B.E., COMPUTER SCIENCE AND ENGINEERING

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND ENGINEERING

(A State University Accredited with 'A+' Grade by NAAC)
Annamalainagar-608 002, Chidambaram, Tamil Nadu

UNDER THE GUIDANCE OF

Mr. SHALU MATHEWS JOHN
DEPUTY CHIEF MANAGER/COMPUTER SCIENCE
CORPORATE OFFICE/ NLC INDIA LIMITED

CREATING WEALTH
FOR WELLBEING

NLC INDIA LIMITED

"NAVRATNA – A GOVERNMENT OF INDIA ENTERPRISE"
NEYVELI– 607-801, TAMILNADU

BONAFIDE CERTIFICATE

Certified that the Internship Training report titled "**WebSec_AutoGuard***":* A BASIC VULNERABILITY SCANNER USING PYTHON" is the bonafide work of **NIRANJANA V (2336010063)** studying Bachelor in Computer science and Engineering at Annamalai University Faculty of Engineering and Technology, Chidambaram, done during the period from 16-06-2025 to 11-07-2024 at Corporate Office, NLC INDIA LIMITED, NEYVELI

Her performance, conduct and attendance during the period was

Mr. SHALU MATHEWS JOHN
DEPUTY CHIEF MANAGER/COMPUTER SCIENCE
CORPORATE OFFICE/ NLC INDIA LIMITED

DATE: 09-07-2025
PLACE: NEYVELI

Permitted to submit the internship report to college/university authorities

DATE: 09-07-2025
PLACE: NEYVELI

DEPUTY CHIEF MANAGER
LEARNING AND DEVELOPMENT CENTRE
NLC INDIA LIMITED

DECLARATION

We hereby declare that the Internship training report titled **"CYBERSECURITY"** is the original work done by us under the guidance of **Shri. SHALU MATHEWS JOHN, DEPUTY CHIEF MANAGER COMPUTER SCIENCE, CORPORATE OFFICE**, NLC India Limited, Neyveli.

This internship report is for reference only and no part of the report will be published copied anywhere without the written permission from officials of NLCIL, Neyveli.

Signature of the student

NIRANJANA V (2336010063)

ACKNOWLEDGEMENT

I express our sincere thanks to **NLC INDIA LIMITED** for providing us the opportunity to carry out the Internship Training.

I also express our sincere thanks to **Shri. A Kr. Saravanabhavan, Unit Head /L&DC, Shri. Prabakaran K, Deputy General Manager / L&DC and the Executives & staffs of Industry Institution Interaction Section / Learning & Development Centre** for providing us the opportunity to carry out the Internship Training in NLC India Limited, Neyveli.

I sincerely thank **Mr. Shalu Mathews John, DEPUTY CHIEF MANAGER/COMPUTER SCIENCE CORPORATE OFFICE/ NLC INDIA LIMITED** for providing the guidance during the internship training programme from 16-06-2025 to 11-07-2025.

I express our gratitude to the Unit Head, **Shri. XXXXXXXXX, Chief General Manager, Mine-I** for granting us permission to undergo internship training in the Unit.

I sincerely thank **Dr. C. KARTHIKEYAN, PROFESSOR & DEAN, Department of CHEMICAL ENGINEERING** at Annamalai University Faculty of Engineering and Technology for providing us support and guidance for the Internship.

I sincerely thanks **Dr. R. BHAVANI, PROFFESOR AND HEAD, CSE Dept, Annamalai University Faculty of Engineering and Technology** for encouraging us to successfully complete the internship training.

INDEX

1.1. About NLC India Limited

NLC India Limited (formerly Neyveli Lignite Corporation Limited) is a Navratna Government of India Enterprise under the administrative control of the Ministry of Coal.

Established in 1956, NLCIL is a pioneer in the field of lignite mining and thermal power generation. With its headquarters located in Neyveli, Tamil Nadu, NLCIL has significantly contributed to the energy sector by operating large-scale opencast lignite mines and power stations across multiple states.

Over the decades, NLCIL has expanded into renewable energy sectors, including solar and wind power, thus embracing a more sustainable approach to energy production.

It owns and operates several power projects with thousands of megawatts of installed capacity, supplying electricity to multiple southern states and contributing immensely to the national grid.

The company is known for its structured organizational setup, eco-conscious initiatives, and a strong focus on community welfare. Through its Learning and Development Centres, NLCIL provides internship and training opportunities to students and professionals, aiming to foster technical expertise, innovation, and real-world industry exposure.

NLCIL has been actively adopting sustainable mining and power generation practices. It is one of the few energy companies in India that is significantly investing in renewable energy projects, including solar and wind power, aiming to reduce its carbon footprint and support India's clean energy goals.

The organization houses a dedicated R&D wing that focuses on improving operational efficiency, adopting new-age technologies in mining, and exploring alternate energy solutions. Its continuous innovation efforts contribute to technological advancement and energy security in India.

1.2 About the Concern Unit (Corporate Office, Neyveli)

The Corporate Office of NLC India Limited, located at Neyveli, Tamil Nadu, serves as the central administrative and strategic hub of the organization.

It plays a pivotal role in coordinating the activities of various departments, including finance, human resources, planning, marketing, legal, IT, and administration.

The Corporate Office ensures smooth communication and alignment among the operational units, government bodies, and stakeholders.

During the internship period at the Corporate Office, the exposure includes interactions with departments such as IT Services, Digital Infrastructure, ERP Systems, and Cybersecurity Compliance.

This unit also governs the company-wide digital transformation initiatives and project planning for future expansions, making it a knowledge-intensive and high-level management division.

The work environment promotes discipline, professionalism, and a problem-solving mindset.

Interns working here are introduced to corporate ethics, documentation protocols, reporting standards, and inter-departmental coordination, enabling them to understand how a major public-sector enterprise functions at the strategic level.

The Corporate Office is responsible for framing critical policies related to project execution, safety, employee welfare, procurement, and IT governance. It acts as the decision-making authority for compliance with central government guidelines and ensures that all units operate under unified standards and strategic direction

The office integrates advanced tools such as ERP (Enterprise Resource Planning) systems, digital workflow automation, and real-time performance dashboards to improve transparency and accountability.

1.3 Objectives of the Study

The primary objective of the study was to conceptualize, design, and implement a lightweight, modular, and user-friendly web vulnerability scanner, primarily focused on identifying **Cross-Site Scripting (XSS)** vulnerabilities within HTML forms. The project aimed to bridge academic learning with practical cybersecurity applications by developing a working prototype named **WebSec_AutoGuard**. The study was conducted as part of the internship program at **NLC India Limited – Corporate Office, Neyveli**, and focused on understanding how digital tools can be used to enhance web application security.

1. **To understand the nature of client-side vulnerabilities, especially Cross-Site Scripting (XSS):**
   The project began with a theoretical understanding of how web vulnerabilities are introduced during input handling and how XSS attacks exploit these flaws. This knowledge was essential in guiding the detection logic of the tool.

2. **To develop a Python-based scanner capable of detecting XSS vulnerabilities in web forms:**
   A key goal was to design a script that can programmatically send HTTP requests, retrieve webpage content, extract HTML forms, and simulate user input with malicious payloads to identify vulnerabilities.

3. **To automate the parsing of form elements from target websites:**
   The tool needed to intelligently identify <form> tags and their associated input fields using HTML parsing libraries like BeautifulSoup. Automating this step ensures scalability and consistency in vulnerability testing.

4. **To simulate real-world attack vectors using malicious payloads:**
   By injecting XSS payloads into various form inputs, the tool mimics attacker behavior to test whether the web application reflects unsanitized input back to the user, which could lead to exploitation.

5. **To generate detailed and structured PDF-based vulnerability reports:**
   A critical aspect was to translate technical results into a formal and readable format. The project included developing a reporting module that uses the fpdf library to create downloadable PDF reports summarizing findings with risk classifications.

6. **To design a user-friendly dashboard using Flask Web Framework:**
   Recognizing the need for accessibility, the study aimed to build a responsive and simple user interface that allows non-technical users to input URLs and view results without using a terminal or coding knowledge.

7. **To implement a logging mechanism for transparency and debugging:**
   Every scan session, including errors and results, is logged into a .log file. This not only assists in debugging but also helps track historical scans and ensure accountability.

8. **To ensure modular architecture for maintainability and extensibility:**
   By breaking down the project into independent modules—scanner, parser, detector, and reporter—the design allows for easy updates, testing, and potential integration with larger systems.

9. **To expose the student to real-world project development using Git & GitHub:**
   Version control and collaborative coding are essential industry practices. As part of the project objectives, the tool was fully documented and published on GitHub with a README file, LICENSE, and demo video.

10. **To explore how cybersecurity aligns with digital transformation in public sector enterprises:**
    Through the internship at NLC India Ltd., the study also sought to understand the importance of cybersecurity tools in enhancing the safety of digital systems used in government and enterprise settings.

WebSec_AutoGuard: A Basic Vulnerability Scanner Using Python

2.1 Project Overview

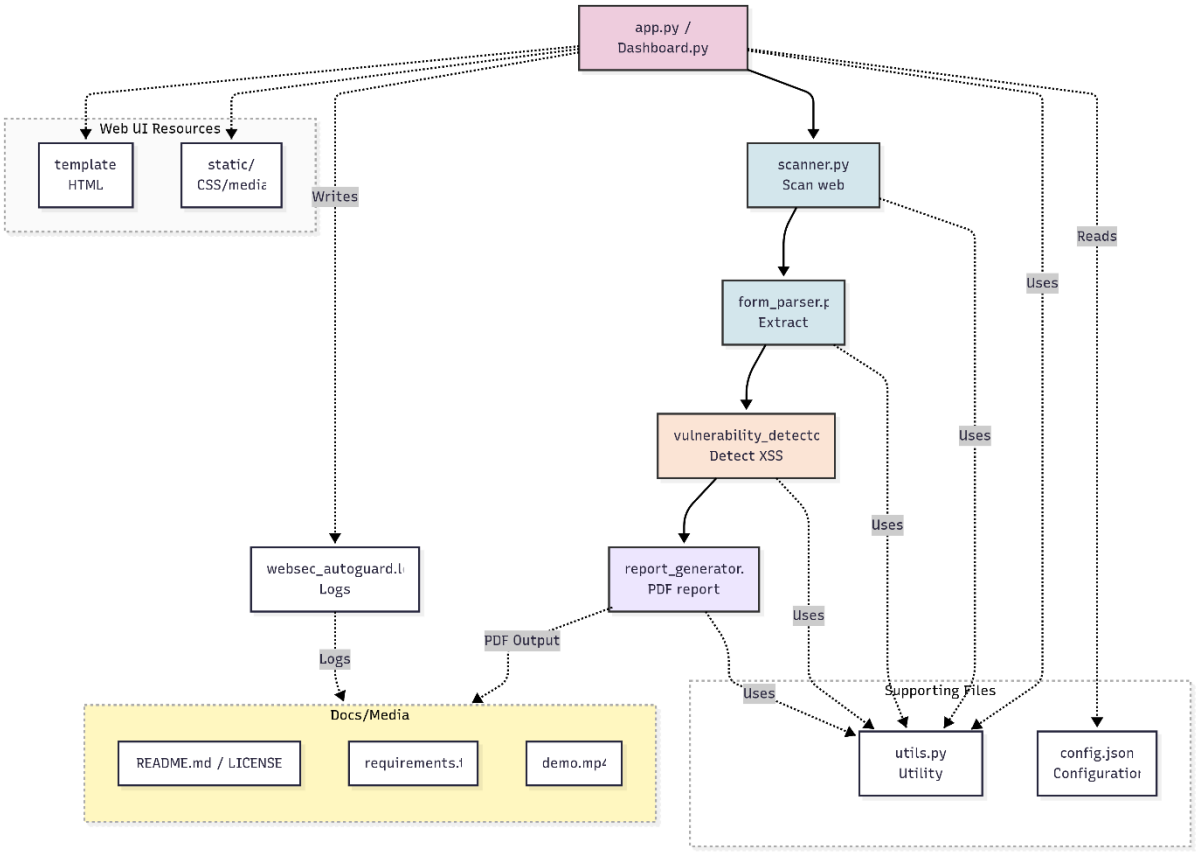
- The project titled **WebSec_AutoGuard: A Basic Vulnerability Scanner Using Python** was developed as an academic initiative aimed at understanding the fundamental principles of web application security, particularly Cross-Site Scripting (XSS) vulnerabilities. The tool is a compact yet powerful scanner designed to analyze HTML forms on websites and detect client-side security loopholes that could be exploited by attackers.
- The development process followed a structured approach, beginning with identifying the problem statement—web applications often fail to sanitize user input, leading to XSS attacks. The project aimed to create a modular, Python-based tool that could automate vulnerability detection and provide user-friendly reports.
- The key strength of the project lies in its modular architecture, with clearly defined components for scanning (fetching HTML), parsing (extracting form elements), detecting vulnerabilities, and generating reports. Each module was written independently, ensuring ease of testing and maintainability. Additionally, a Flask-based Web User Interface (WebUI) was implemented to allow even non-technical users to perform scans and generate reports without needing command-line expertise.
- The use of technologies such as Python, Flask, BeautifulSoup, requests, and fpdf played a significant role in achieving the project objectives. The scanner was tested on various local and sample websites, and the results were validated through manually crafted payloads to ensure accuracy.
- One of the major outcomes of the project is its ability to simulate real-world attacks using XSS payloads and immediately reflect whether the application is secure or not. Furthermore, the inclusion of a downloadable PDF report, execution logs, and open-source availability via GitHub enhances the credibility and usability of the tool.
- In conclusion, WebSec_AutoGuard not only fulfilled the defined objectives but also laid the foundation for future developments like adding detection for other vulnerabilities (SQLi, CSRF, etc.). It demonstrates the practical application of cybersecurity principles and serves as a strong academic and skill-building exercise in both web security and Python development.

2.2 Objectives

- To **develop a Python-based web vulnerability scanner** that focuses on detecting Cross-Site Scripting (XSS) attacks in web applications.
- To **extract HTML form elements automatically** from target websites using parsing techniques with BeautifulSoup.
- To **simulate malicious payload injections** and check whether the application reflects these inputs, indicating potential vulnerabilities.
- To **build a modular codebase** comprising components like scanner, parser, detector, reporter, and UI for better maintainability and extensibility.
- To **implement a PDF report generation system** using the fpdf library, summarizing vulnerabilities, input fields affected, and associated risk levels.
- To **design a Flask-based Web User Interface (WebUI)** that enables non-technical users to input URLs and view scan results easily.
- To **enable both command-line interface (CLI) and web-based access**, ensuring flexibility for different types of users.
- To **create a logging system** that records each scanning session in a log file (websec_autoguard.log) for debugging and auditing purposes.
- To **maintain clean documentation**, including a README file, license information, and usage instructions for ease of understanding and collaboration.
- To **publish the complete project on GitHub**, making it open-source for academic review, future contribution, and version control.
- To **enhance cybersecurity awareness** by demonstrating common form-based vulnerabilities and encouraging secure coding practices.
- To **provide a reusable project structure** that can be extended in the future for detecting other vulnerabilities like SQL Injection (SQLi), CSRF, etc.

2.3 ARCHITECTURE OF WEBSEC_AUTOGUARD

```
WebSec_AutoGuard/
 ├── scanner.py              # Performs HTTP scanning and fetches HTML
 ├── form_parser.py          # Parses HTML and extracts forms
 ├── vulnerability_detector.py  # Checks forms for XSS vulnerabilities
 ├── report_generator.py     # Creates vulnerability reports in PDF format
 ├── utils.py                # Reusable utility functions
 ├── config.json             # Configuration file
 ├── templates/              # HTML templates for WebUI
 │    ├── index.html         # Dashboard UI
 │    └── result.html        # Results display UI
 ├── static/                 # Optional CSS or JS files for WebUI
 ├── app.py / Dashboard.py   # Flask-based Web interface
 ├── README.md, LICENSE      # Documentation and licensing
 ├── requirements.txt        # Required Python libraries
 ├── demo.mp4                # Demo video showing functionality
 ├── WebSec_Report.pdf       # Sample generated report
 ├── websec_autoguard.log    # Logs
```



2.4 Features

- **Modular Codebase**: The project is structured into independent, logically separated modules for scanning, parsing, detection, reporting, and UI. This ensures maintainability and scalability.

- **XSS Detection Engine**: Uses simulated input attacks to test for potential **Cross-Site Scripting (XSS)** vulnerabilities in form elements, a common and dangerous web security flaw.

- **PDF Report Generation**: After scanning, the tool compiles findings into a clean, downloadable **PDF report** that summarizes each vulnerability with its risk level.

- **User-Friendly Web Dashboard**: Built with Flask and Bootstrap, the UI allows users to easily input a URL, select scanning options, and view results without using the terminal.

- **Logging Mechanism**: Automatically logs each scanning session, errors, and findings into websec_autoguard.log, useful for debugging and record-keeping.

- **CLI and Web Support**: Offers flexibility by supporting both **Command-Line Interface (CLI)** for advanced users and **Web Interface** for non-technical users.

- **Lightweight and Customizable**: Uses minimal third-party packages and allows developers to expand detection logic for other vulnerabilities.

- **Real-Time Feedback on Scan Progress**: The tool provides immediate feedback during scanning, helping users understand which forms or payloads are being processed.

- **Configurable Scanning Options**: Through a config.json file, users can adjust scanning parameters such as target payloads, timeouts, or logging levels.

- **Open-Source and Educational Use**: Designed with clarity and simplicity, the tool is perfect for learning purposes and can be used in workshops, academic demonstrations, or student projects.

- **Cross-Platform Compatibility**: Built with Python and Flask, the tool runs smoothly on Windows, macOS, and Linux environments without modification.

- **Secure Input Handling**: Implements safe handling of URLs and payloads to prevent unintended behaviors during scanning.

- **Extensive Form Coverage**: Capable of detecting and scanning various form types including hidden fields, multi-step forms, and nested form structures

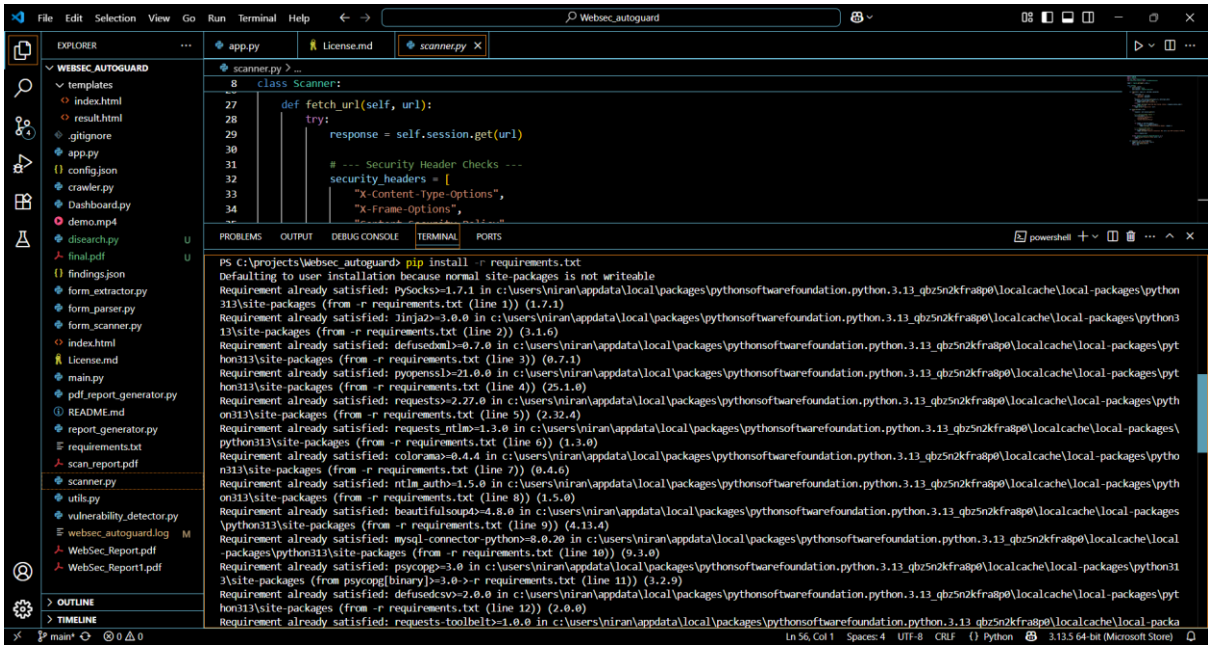
2.5 Technologies Used

Technology	Purpose
Python	Core language for building scanner logic, detection engine, and utilities.
Flask	Lightweight web framework for building the interactive WebUI.
BeautifulSoup	HTML parsing and form element extraction from webpages.
requests	Sending HTTP requests to fetch HTML content from target URLs.
fpdf	Generating structured and printable PDF reports.
Git & GitHub	Version control and publishing the complete project online.
Bootstrap	Styling and responsive design for the web interface.

2.6 Step-by-Step Implementation

2.6.1 Step 1: Setup & Environment Configuration

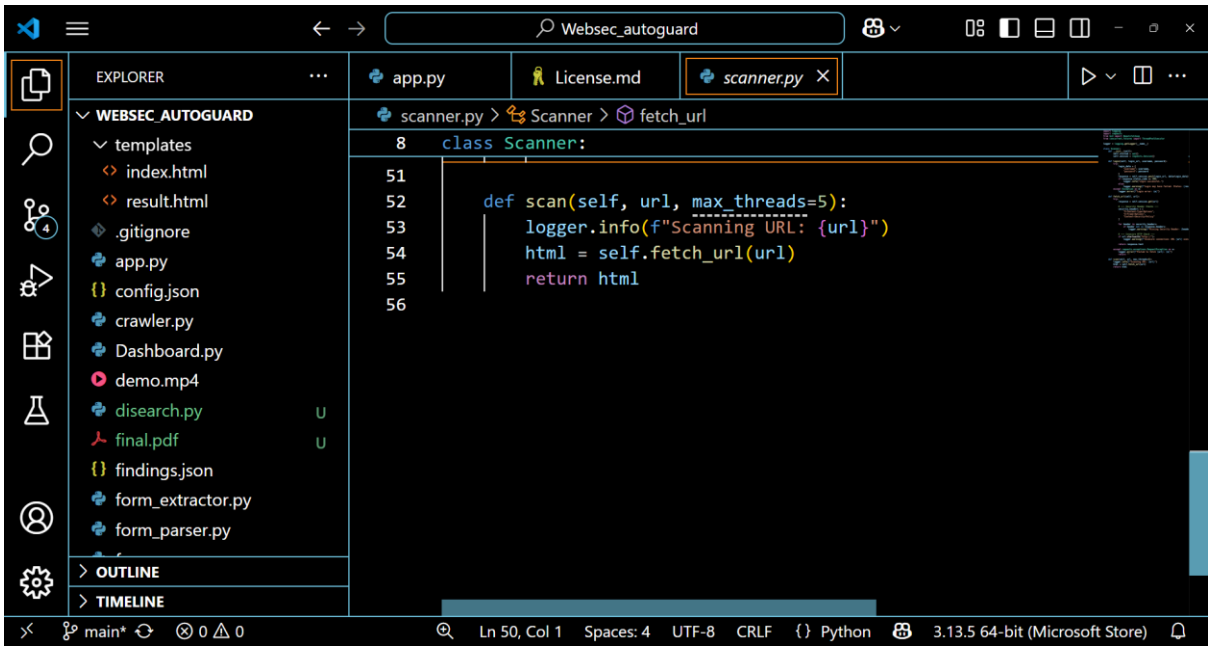
- The development of ***WebSec_AutoGuard*** began with the setup of a dedicated Python development environment to ensure a clean and conflict-free workspace.
- A virtual environment was created using Python's built-in venv module. This helped isolate the dependencies required for the project, preventing interference with system-wide packages or other Python projects.
- Once the environment was activated, all essential libraries and modules were installed using the pip install -r requirements.txt command.
- This requirements file included packages such as Flask for the web interface, BeautifulSoup for HTML parsing, requests for sending HTTP requests, and fpdf for PDF report generation.
- This setup stage was crucial as it laid the foundation for the entire tool, ensuring that all components function smoothly and are easily portable across systems.



2.6.2 Step 2: Built the Scanner Module

- In this step, the core functionality of fetching and analyzing webpage content was implemented through the development of the scanner.py module.
- This module is responsible for retrieving the HTML source code of the target website, which serves as the primary input for the vulnerability scanning process.
- The requests library was used to send HTTP GET requests to the user-specified URL.

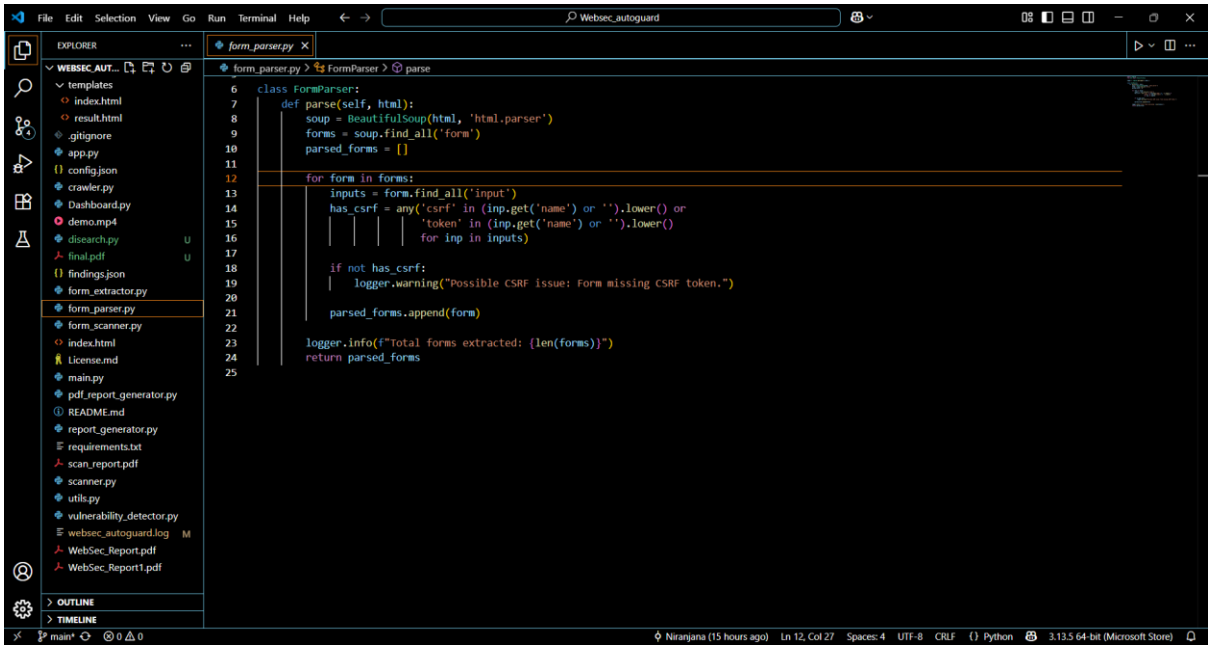
- It handles network communication and error management to ensure successful responses are obtained from the server. Proper exception handling was incorporated to manage issues like timeouts, broken URLs, or access errors.
- The module not only retrieves the page but also logs the response status and saves the raw HTML content for further analysis. This scanner acts as the foundation of the entire tool, feeding clean and structured data to the form parser and vulnerability detector in the later stages. Thus, this component plays a crucial role in ensuring that the tool begins the scan with reliable and accurate web data



2.6.3 Step 3: Form Parsing

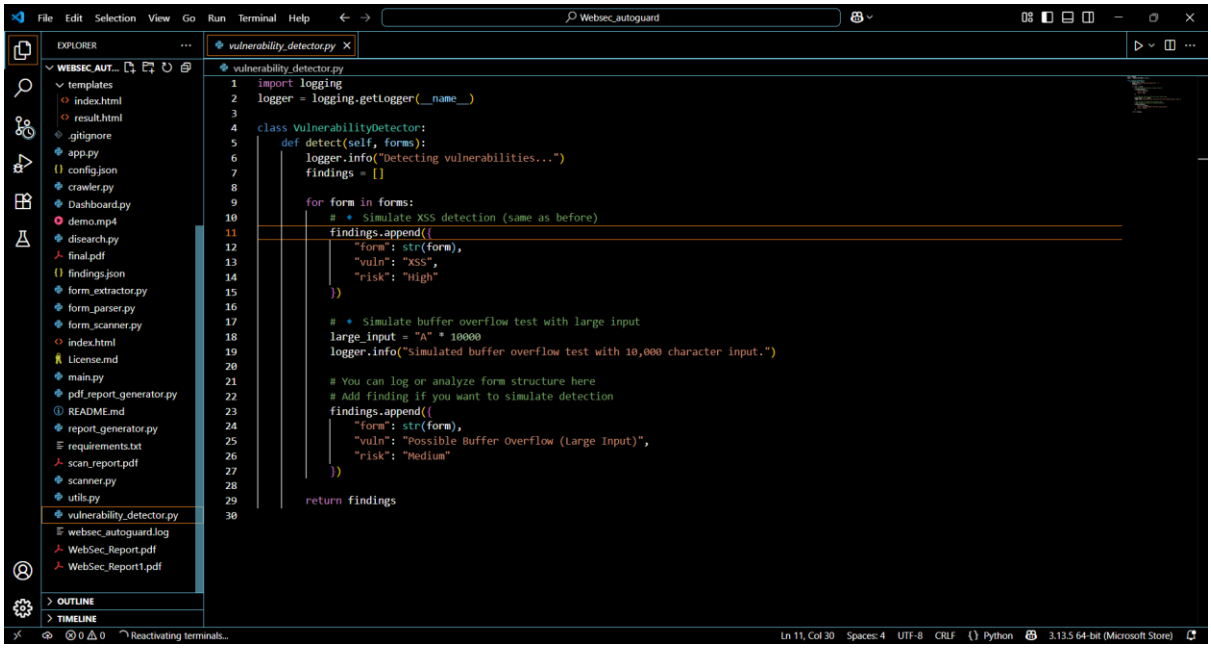
- After successfully retrieving the HTML content from the target website, the next critical phase involved extracting and analyzing form elements.
- This was accomplished by developing the form_parser.py module, which utilizes the BeautifulSoup library to parse the raw HTML content.
- The primary objective of this step was to locate all form tags embedded in the webpage and identify their associated input fields, such as textboxes, passwords, hidden fields, buttons, and more.
- Each form's action, method (GET or POST), and input parameters were collected and structured into a format suitable for further analysis.
- The parser filters out irrelevant tags and focuses only on those that could potentially accept user input—crucial for identifying vulnerable endpoints.

- This structured form data is essential for simulating XSS attacks in the next step, as it determines where and how payloads will be injected. By isolating user-interactive elements, this module bridges the gap between raw HTML content and targeted vulnerability testing, making it an indispensable part of the WebSec_AutoGuard architecture.



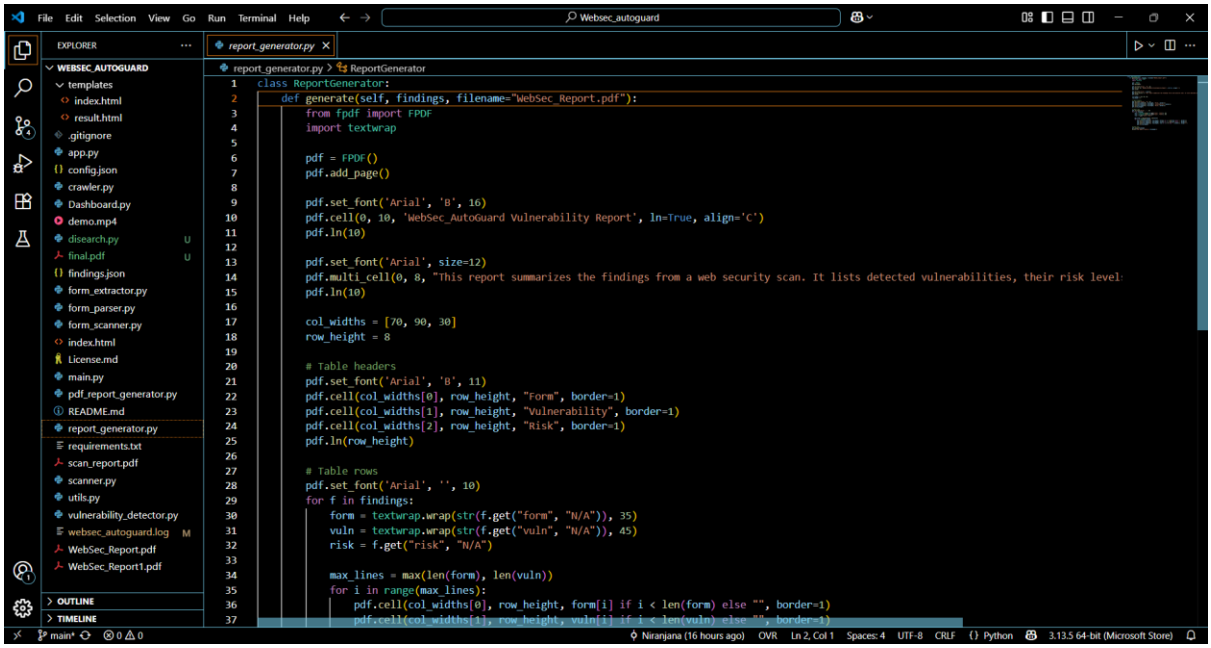
2.6.4 Step 4: Vulnerability Detection

- In this critical phase of the project, the focus shifted to actively identifying security flaws—specifically, Cross-Site Scripting (XSS) vulnerabilities.
- Using the form data parsed in the previous step, the vulnerability_detector.py module was developed to simulate real-world attack scenarios.
- This involved injecting a set of predefined XSS payloads into the input fields of the detected forms. These payloads typically included harmless scripts like <script>alert('XSS')</script> that, if improperly handled by the web application, would be reflected back in the response HTML.
- The detector then analyzed the server's response to check if the injected payload appeared unencoded, indicating a vulnerability.
- This method helps identify websites that do not sanitize or validate user inputs properly, making them susceptible to client-side attacks. By automating this testing process, the module could quickly and accurately flag insecure input handling.
- This stage represents the heart of the scanner’s security logic and transforms WebSec_AutoGuard from a passive analysis tool into an active vulnerability detection engine.



2.6.5 Step 5: Report Generation

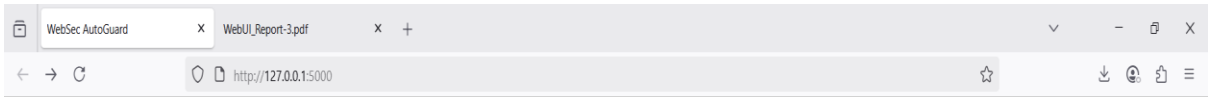
- Once the vulnerability scanning and detection process was complete, the next essential step was to generate a well-structured and professional report summarizing the findings.
- This was implemented using the report_generator.py module, which utilized the fpdf library to create downloadable PDF reports.
- These reports contained critical details such as the type of vulnerabilities detected (e.g., XSS), their associated risk levels (Low, Medium, or High), and contextual explanations for each.
- The module was designed to present the results in a clean and readable format, making it suitable for submission in academic projects or for preliminary audits in real-world applications.
- Each entry in the report included both the affected form details and the vulnerability type detected, helping the reader understand where the issue lies.
- This step enhanced the tool's usability by providing a tangible, shareable output that can be archived, reviewed, or presented to security teams or faculty advisors.



2.6.6 Step 6: Web UI Development

- To make the tool accessible and user-friendly for individuals with little to no coding experience, a simple and interactive web-based interface was developed using the Flask framework.

- The backend logic was embedded in app.py, which managed routing, form submissions, and communication with the scanning modules. On the frontend, two primary HTML templates were created: index.html served as the dashboard where users could enter a target URL and select scanning preferences (e.g., XSS detection), while result.html displayed the scanned results, including a list of vulnerabilities found and a link to download the generated PDF report.
- This visual interface was styled using Bootstrap to enhance usability and ensure a responsive layout across different screen sizes.
- By integrating the Web UI, the project allowed users to initiate scans, view real-time results, and retrieve reports directly from their browser—eliminating the need for command-line operations.
- The Flask app also handled data flow between user input and the scanning engine, making the process smooth and intuitive.
- This step significantly increased the practicality of the tool for non-technical audiences, such as students or project reviewers.



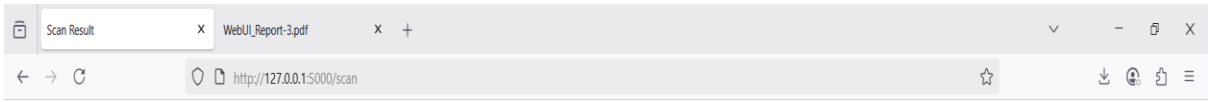
Welcome to WebSec AutoGuard

Enter URL:

http://testphp.vulnweb.com

XSS Scan

Start Scan



Scan Result for http://testphp.vulnweb.com

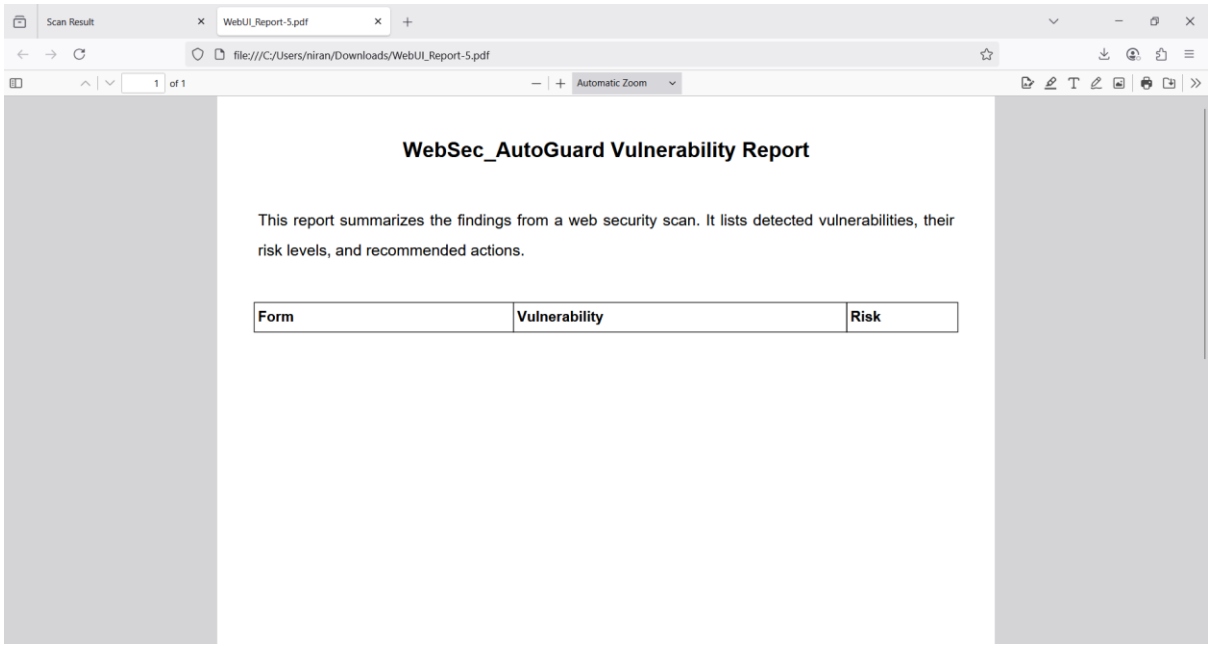
Vulnerabilities:

No vulnerabilities found.

Report File:

WebUI_Report.pdf

Go Back



Screenshot of dashboard form, scan results and pdf report.

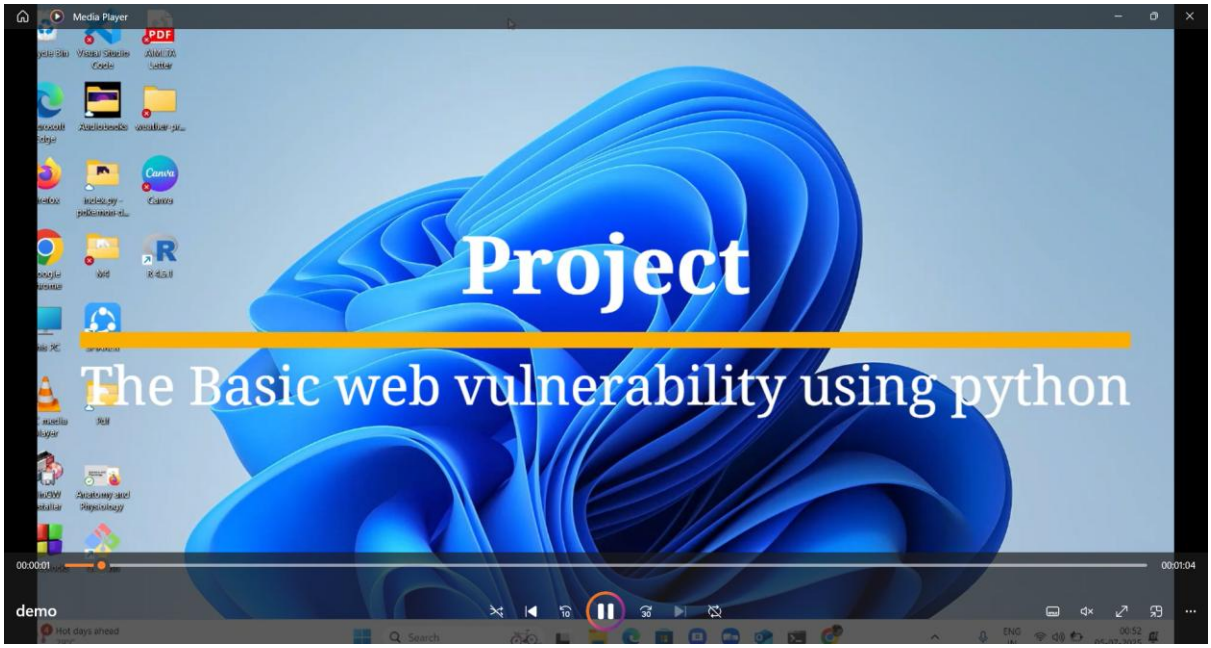
2.6.7 Step 7: Demo Video Recording

- To visually demonstrate the functionality of the WebSec_AutoGuard tool, a screen-recorded demo video was prepared and included as part of the project deliverables.

- This video captures the complete usage of the tool—starting from launching the web dashboard, entering the URL to be scanned, initiating the vulnerability scan, and finally downloading and viewing the PDF report generated by the tool.

- Instead of a voiceover, **detailed subtitles were added** throughout the video to explain each action being performed. These subtitles describe the purpose and functionality of each step clearly and are synchronized with the visuals, making the video accessible and easy to follow for viewers from both technical and non-technical backgrounds.

- The use of subtitles makes the video presentation flexible for use in silent academic environments, formal presentations, and for users who prefer reading explanations over listening. This visual documentation enhances the understanding of the project and complements the written README file and user guide. The final video is also uploaded to the GitHub repository as demo.mp4.

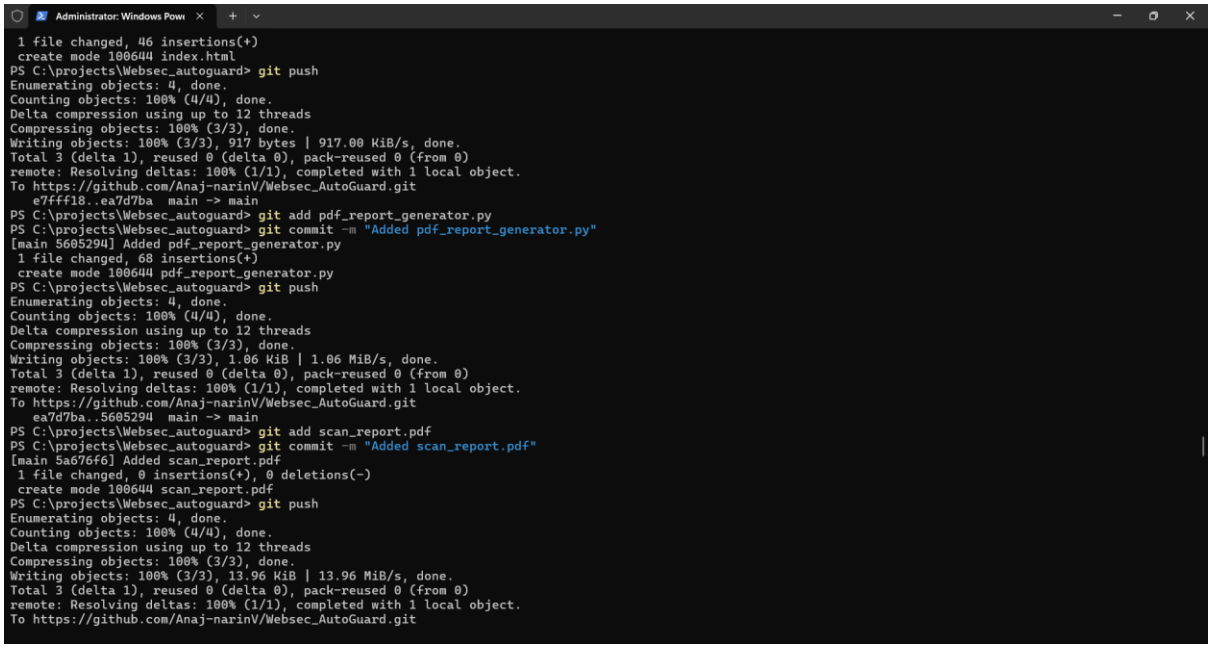


2.6.8 Step 8: GitHub Publishing

After completing the development and testing of **WebSec_AutoGuard,** the entire project was organized and published on GitHub to ensure open access, version control, and professional presentation. GitHub serves as a collaborative platform where developers and reviewers can access the source code, view documentation, and contribute to the project if needed.

- **Repository Initialization**: A new Git repository was created locally and initialized with git init. A remote repository was then connected using GitHub.
- **.gitignore File**: This file was added to ensure that unnecessary files (like __pycache__, .log files, and .env folders) are excluded from version control, keeping the repository clean.
- **README.md**: A well-documented README.md was created, containing the project introduction, features, setup instructions, usage guide, screenshots, and GitHub badges. This serves as the landing page for visitors.
- **LICENSE File**: An MIT License was added to allow others to use, modify, and distribute the project freely with proper attribution.
- **requirements.txt**: This file includes a list of required Python libraries, allowing users to easily install dependencies with a single command.
- **Final Commit & Push**: After staging and committing the final versions of all project files—including scanner.py, report_generator.py, app.py, templates, static files, and demo video—the content was pushed to the GitHub repository by using simple commands in terminals.

The final GitHub repository now contains everything needed for future reference or collaborative development. It demonstrates good software development practices and ensures the project can be revisited, reused, or upgraded at any point.



2.7 Demo Video With Subtitles

Demo File Name: “demo.mp4”
Availability: This video file is available in the GitHub repository under the root directory for easy access and download.

As part of the project deliverables, a **demo video** was created to visually explain the working of the WebSec_AutoGuard tool. This video serves as a guide for evaluators, developers, and end-users who wish to understand the functionality without diving into the codebase.

No Voiceover – Only Subtitles: To make it universally understandable and simple, the video does not contain a voiceover. Instead, **subtitles are added** throughout the video to narrate each action being performed.

2.8 Final Deliverables

The following files and documents were prepared and submitted as part of the complete project package for **WebSec_AutoGuard**. These deliverables ensure that the tool is well-documented, easy to use, and demonstrable for academic or practical purposes.

Source Code Files

- scanner.py: Core scanning logic that fetches web content.
- form_parser.py: Extracts form elements from HTML using BeautifulSoup.

- vulnerability_detector.py: Contains logic for injecting and detecting XSS payloads.
- report_generator.py: Builds structured, printable PDF reports using fpdf.
- utils.py: Utility functions that support scanning and parsing.

Web Interface

- app.py: Flask-based application that serves as the backend for the dashboard.
- templates/index.html: HTML form interface where users input target URLs.
- templates/result.html: Displays detected vulnerabilities and report download link.
- static/: Directory containing optional CSS or JS assets for the UI.

Supporting Files

- config.json: Stores configuration details like payloads or settings.
- requirements.txt: List of dependencies required to run the project.
- .gitignore: Specifies files and folders to exclude from Git tracking (e.g., logs, environment files).
- websec_autoguard.log: Runtime log file containing session info, errors, and activity history.

Documentation & Media

- README.md: Provides a comprehensive overview, setup instructions, and screenshots of the project on GitHub.
- LICENSE: Licensing file (MIT) for open-source distribution.
- WebSec_Report.pdf: Sample vulnerability report generated after a scan.
- demo.mp4: A screen-recorded demo of the entire scanning process with subtitles, not voice.

GitHub Repository

- All the above files were committed, version-controlled, and pushed to the GitHub repository.
- The repository also contains screenshots and a properly structured README to help users quickly understand and deploy the tool.

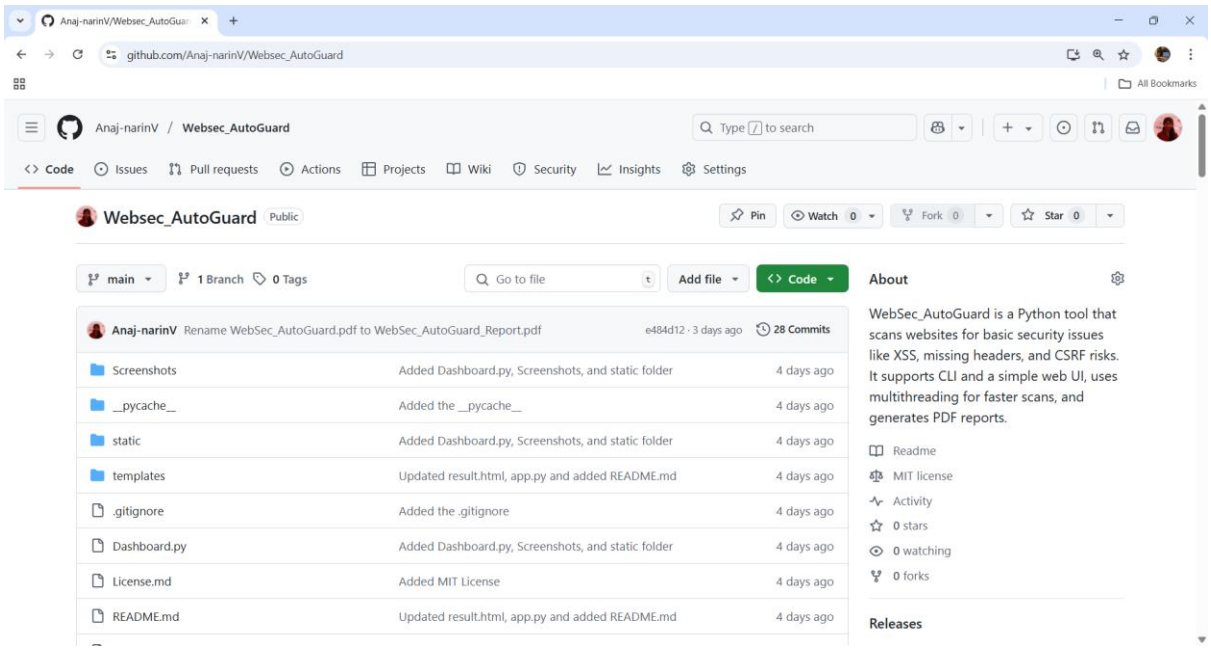
Together, these components form a complete deliverable kit for the **WebSec_AutoGuard** project, covering both functionality and documentation.

2.9 GitHub Repository

The entire **WebSec_AutoGuard** tool was published on GitHub.

GitHub Repository URL:

https://github.com/Anaj-narinV/Websec_AutoGuard



3.1 Conclusion

The **WebSec_AutoGuard** project was conceptualized, designed, and implemented as a lightweight and educational web vulnerability scanning tool. It showcases the power of Python and open-source libraries to address real-world security challenges, particularly Cross-Site Scripting (XSS) vulnerabilities in HTML forms.

Throughout the development process, the focus was on modular design, clarity of code, and user-friendliness. From creating a robust backend scanning engine to building a sleek Flask-based frontend, each component was carefully crafted and integrated. The tool's ability to generate downloadable PDF reports and log scanning activities adds to its professionalism.

This project not only meets academic expectations but also reflects practical relevance in web security auditing. With proper documentation, open-source publication, and a demo video, **WebSec_AutoGuard** stands as a foundational step in understanding vulnerability assessment and secure software development.

It serves as a reference for students, cybersecurity enthusiasts, and developers aiming to understand the basics of input validation, XSS exploitation, and secure coding practices.

3.2 References

1. **OWASP (Open Web Application Security Project)**
   https://owasp.org/www-community/attacks/xss/

2. **Python Official Documentation**
   https://docs.python.org/3/

3. **Flask Framework Documentation**
   https://flask.palletsprojects.com/

4. **BeautifulSoup Documentation**
   https://www.crummy.com/software/BeautifulSoup/bs4/doc/

5. **requests Library Documentation**
   https://docs.python-requests.org/

6. **fpdf Documentation for PDF Generation**
   https://pyfpdf.github.io/fpdf2/

7. **Bootstrap 5 Documentation**
   https://getbootstrap.com/

8. **Tutorials and Articles from:**
   - GeeksforGeeks
   - Medium Blogs (Python Security)
   - Stack Overflow

9. **GitHub Repository for WebSec_AutoGuard**
   https://github.com/Anaj-narinV/Websec_AutoGuard

10. **OpenAI ChatGPT** – Used for conceptual guidance, code troubleshooting, and documentation support.
    https://chat.openai.com/

3.3 Feedback About My Internship

Working on the *WebSec_AutoGuard* project has been an incredibly enriching experience. I gained hands-on exposure to core cybersecurity concepts, particularly how web vulnerabilities like Cross-Site Scripting (XSS) are detected and mitigated. Unlike theoretical study, building this project allowed me to **apply my learning practically**, which made the concepts much clearer and easier to understand.

Developing and testing each module—scanner, parser, detector, and report generator—taught me how modular code is structured and how real-world tools are built from scratch. I also learned how to create a user-friendly interface using Flask, which improved my confidence in web development and backend integration.

Additionally, the process of pushing the project to GitHub and documenting it thoroughly gave me exposure to professional development practices like version control, open-source publishing, and technical writing.

Throughout the project, I was guided by helpful online resources and AI-based tools like ChatGPT for solving errors and understanding certain programming logic.

The **self-paced learning** combined with structured implementation made this project a valuable step in my academic and career journey.

I would also like to express my sincere gratitude to my **internship guide**, whose support and encouragement played a vital role throughout this project. Their timely suggestions and consistent feedback helped me stay on the right track and shaped the success of this work