NShield – Scan. Detect. Secure:

AI-Driven Automated Tool for Security Analysis and Remediation

Name : Jackie Kim
Student ID : 300376300
Course : CSIS4495-001 Applied Research Project

Introduction

In today's digital landscape, cybersecurity threats are evolving at an unprecedented pace, necessitating organizations to implement robust security measures to safeguard sensitive data. One of the most effective approaches to assessing and enhancing cybersecurity defenses is penetration testing. Penetration testing is an authorized, simulated cyberattack on a computer system designed to evaluate its security posture. By employing various tools, techniques, and methodologies, penetration testers mimic the actions of real attackers to uncover vulnerabilities that could be exploited (Black Duck Software, n.d.; Imperva, n.d.; IBM, n.d.). This process plays a crucial role in identifying security gaps, assessing risks, and enabling organizations to enhance their defensive strategies.

As businesses increasingly rely on digital platforms to store, process, and manage data, the need for proactive security measures has become paramount. Cyberattacks such as ransomware, phishing, and data breaches continue to pose significant threats to organizations across industries. Traditional security measures, such as firewalls and antivirus programs, are not always sufficient to protect against sophisticated cyber threats. Penetration testing provides a proactive approach to identifying and mitigating security risks before they can be exploited by malicious actors.

Penetration tests, often referred to as "pen tests," are categorized into three primary types: White Box, Black Box, and Grey Box testing. White Box Penetration Testing involves testers having full access to the system, including network maps and credentials, allowing for a comprehensive evaluation of security vulnerabilities. Black Box Penetration Testing is where testers have no prior knowledge of the system, mimicking external attackers attempting to exploit unknown weaknesses. Grey Box Penetration Testing occurs when testers receive limited information, such as login credentials, to simulate insider threats or external breaches. By identifying weaknesses in networks, applications, and systems, penetration testing supports compliance with regulatory standards such as PCI DSS, HIPAA, and GDPR. It provides qualitative and quantitative insights that inform security policies, resource allocation, and risk mitigation strategies (Black Duck Software, n.d.; Imperva, n.d.; IBM, n.d.).

Despite the effectiveness of penetration testing, organizations face several challenges in conducting these assessments. Resource constraints make traditional penetration testing time-consuming and financially demanding, making it difficult for small and medium-sized enterprises (SMEs) to implement regular testing. The dynamic nature of cyber threats means that vulnerabilities identified during a penetration test can quickly become outdated. Organizations may struggle to analyze and prioritize remediation efforts due to the complexity of penetration testing reports. While automated tools can streamline penetration testing, they may produce false positives or fail to detect advanced threats effectively. Without a systematic and automated approach to penetration testing, organizations risk exposing their sensitive data to cyber threats, resulting in financial losses, reputational damage, and legal consequences.

This research aims to address the limitations of traditional penetration testing by exploring the development of automated internal penetration testing tools. While existing studies highlight the benefits of penetration testing, there is limited research on automating and integrating penetration testing into an organization's continuous security assessment framework. Key knowledge gaps include the effectiveness of automated penetration testing tools compared to manual testing, the impact of automation on reducing false positives and enhancing vulnerability detection, and the feasibility of integrating automated penetration testing into cybersecurity frameworks for real-time security monitoring (Huang et al., 2021; Smith & Jones, 2020).

This study hypothesizes that automated penetration testing can enhance cybersecurity by providing organizations with a cost-effective, scalable, and efficient approach to vulnerability assessment. By leveraging automation, businesses can conduct regular security assessments without extensive resource allocation, improve the accuracy and consistency of penetration test results, identify and remediate vulnerabilities in real-time, and ensure compliance with industry regulations and standards (Chen et al., 2019). Ultimately, this research seeks to bridge the gap between traditional penetration testing and modern security needs by proposing a framework for automated penetration testing solutions. The anticipated benefits of this study include improved cybersecurity resilience, better regulatory compliance, and a streamlined penetration testing process for organizations of all sizes.

Summary of the Research Project

The NShield project was developed as an automated network and web application security scanning tool designed to identify and analyze known vulnerabilities in a structured and user-friendly manner. Built using Python and Flask, the tool integrates widely used scanning technologies such as Nmap and Nikto, enhanced with AI-based analysis through Ollama to generate comprehensive, beginner-friendly HTML reports. The final version of NShield includes customizable scanning options for both network and web environments, automated command execution, and detailed vulnerability analysis complete with remediation suggestions. Validation of the tool was conducted using TryHackMe and OWASP Juice Shop, ensuring it met practical cybersecurity standards and could reliably detect a range of security issues. The completed project demonstrates a balance of technical capability, accessibility, and extensibility, contributing to the field of applied cybersecurity tools.

Changes to the Proposal

As the project progressed, several changes were made to the initial proposal to refine the research focus, optimize tool selection, and improve implementation feasibility. These changes were driven by practical considerations such as minimizing development complexity, reducing resource dependencies, and ensuring better automation capabilities.

Phase 1: Step-by-Step Procedure Development

Initial Plan: The initial phase aimed to identify and define key components of penetration testing, including reconnaissance, vulnerability scanning, exploitation, and reporting, following industry best practices. Planning and scoping defined objectives and engagement rules, ensuring legal and ethical compliance. The testing process involved gathering intelligence, using automated scanning tools to detect vulnerabilities, and simulating attacks to assess security risks before reporting the findings for remediation.

Revised Approach: The project direction shifted toward developing an AI-integrated automated network and web application security scanning tool rather than focusing solely on identifying key components of penetration testing. Additionally, the methodology was adjusted to focus on Black Box testing rather than White Box testing. This shift allows individuals without internal system access or advanced cybersecurity expertise to conduct basic penetration testing using the automated tool, enhancing accessibility and usability.

Phase 2: Tool Development

Initial Plan: The original tool development phase proposed integrating multiple open-source security tools, such as Nmap, Metasploit, and Nessus, using Python or C#. Automation scripts would be used to streamline reconnaissance, scanning, and exploitation, reducing manual effort.

Revised Approach: To simplify development and minimize potential integration errors, the tool development phase was revised to utilize only Nmap for network security scanning and Nikto for web application security scanning. This decision helped ensure a more stable implementation while maintaining effective security scanning capabilities.

Phase 3: AI Integration for Remediation

Initial Plan: AI would be integrated to analyze penetration testing reports, assess risk levels, and provide remediation recommendations. The initial approach considered using PyTorch or ChatGPT to enable machine learning-based vulnerability analysis.

Revised Approach: Instead of using PyTorch or ChatGPT, the revised approach adopted Ollama, a solution that does not require an API key, minimizing setup errors and additional costs for users. Additionally, RAG AI was chosen for vulnerability analysis and remediation suggestions since it is pre-trained and does not require extensive training, making it more efficient and user-friendly.

Phase 4: Testing and Evaluation

Initial Plan: The tool was originally intended to be tested in a simulated organizational environment with diverse network configurations. Performance would be evaluated based on accuracy, efficiency, and adaptability across various test scenarios.

Revised Approach: Instead of testing across various configurations, the final approach involved running the tool in a controlled simulated environment to verify whether known vulnerabilities could be successfully detected. This adjustment ensured a focused and reliable assessment of the tool's core functionality without the complexity of randomized testing.

Project Completion Timeline

Phase 1: Step-by-Step Procedure Development (Jan 16 – Feb 7, 2025)

Milestones:

- Jan 16–20: Literature review and background research on penetration testing methodology.
- Jan 23: Research proposal updated and submitted.
- Jan 27–Feb 3: Research on reconnaissance tools and tasks; refined step-by-step testing procedures.
- Feb 5–Feb 7: Defined scanning/enumeration procedures; selected tools and strategies.

Deliverables: Detailed research plan and timeline, Initial proposal document, Report outlining reconnaissance and scanning tools

Phase 2: Tool Development (Feb 8 – Mar 3, 2025)

Milestones:

- Feb 8–13: Collected and tested multiple scanning tools (Nmap, Nikto, Metasploit, etc.)
- Feb 14–20: Created Python automation script, eliminated unstable tools like SonarQube and p0f.
- Feb 22–23: Debugged code and finalized midterm report.
- Mar 3: Finalized tools and completed core Python scripting.

Deliverables: Midterm progress report, Functional scanning script (Python), Documentation of tested and selected tools

Phase 3: AI Integration for Remediation (Feb 25 – Mar 16, 2025)

Milestones:

- Feb 25–Mar 1: Initial exploration of AI tools (LangChain, RAG, Ollama)
- Mar 3–11: Finalized Ollama as AI tool, wrote integration script
- Mar 12–14: Validated AI suggestions using generated reports
- Mar 16: Delivered AI-integrated script and progress report

Deliverables: AI-enhanced remediation script, Evaluation of AI tools and integration documentation, Updated progress report

Phase 4: Validation (Mar 18 – Mar 30, 2025)

Milestones:

- Mar 18–26: Testing functionality, validating AI suggestions, refining output formats
- Mar 28–30: Cross-checked scan results with OWASP Juice Shop vulnerabilities

Deliverables: Test results with analysis, Output format and structure finalized, Initial draft of final report

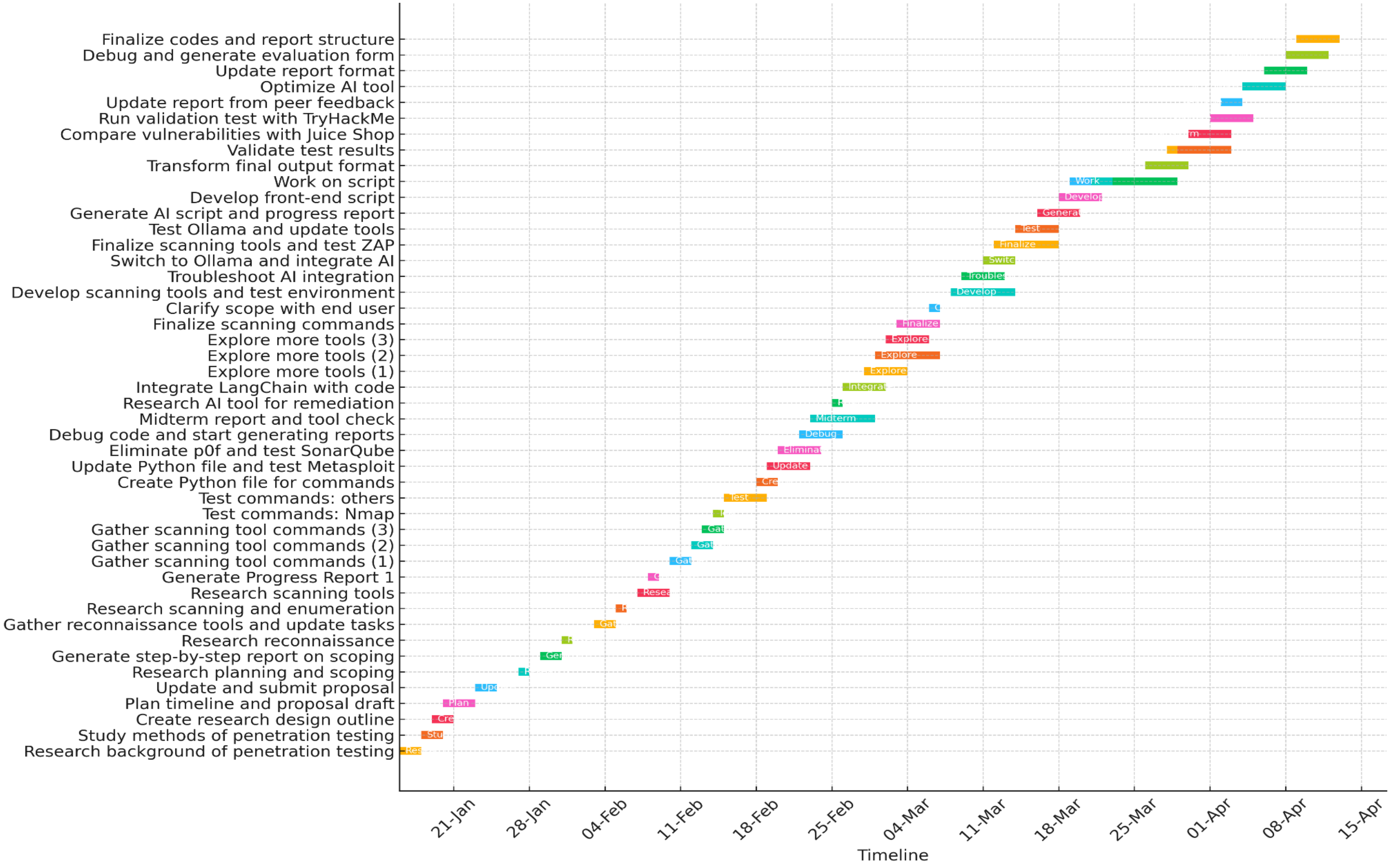
Phase 5: Evaluation (Apr 1 – Apr 13, 2025)

Milestones:

- Apr 1–2: Final validation using TryHackMe; feedback gathered from classmates
- Apr 4–6: Updated scripts and report based on feedback; improved performance
- Apr 8–9: Final debugging, finalized report format and structure
- Apr 9-13: Final report generation

Deliverables: Final report (updated structure and visuals), Evaluation form and summary,Fully functional tool with AI integration.

Gantt Chart for the Project



Implemented Features

The developed tool is an automated penetration testing application specifically designed for network security scanning and web application vulnerability assessment. Named NShield, this tool streamlines the scanning and vulnerability assessment process through a Flask-based web interface. Given the increasing volume and complexity of cyberattacks, it is essential to proactively identify and remediate system vulnerabilities (Czajka & Gajewski, 2018). This tool aims to empower cybersecurity professionals and learners by providing an accessible, efficient, and professional-grade platform for conducting security scanning in a structured manner. For detailed setup instructions, please refer to Appendix A: Installation Guide. For step-by-step usage instructions of the NShield tool, refer to Appendix B: User Guide.

In the current cybersecurity landscape, network infrastructure and web applications are among the most common attack surfaces targeted by threat actors. Misconfigured services, open ports, vulnerable protocols, and insecure web endpoints represent significant threats to any organization's security posture (OWASP, 2020). The focus on network and web security is due to these areas being critical frontline entry points for attackers. Identifying weaknesses in these layers allows system administrators and penetration testers to significantly reduce risk exposure and implement preventative controls. Furthermore, these attack vectors are often overlooked, despite their crucial role in cybersecurity defense, which makes addressing them even more important (OWASP, 2020).

To ensure comprehensive detection and assessment of vulnerabilities across both network and web application layers, the tool integrates Nmap for network security scanning and Nikto for web application vulnerability scanning.

Nmap (Network Mapper) is a highly flexible and extensively adopted open-source utility used for port scanning, service identification, OS fingerprinting, and vulnerability detection through its powerful scripting engine known as the Nmap Scripting Engine (NSE). This engine supports a wide array of community and officially maintained scripts that can detect misconfigurations, assess services against known CVEs (Common Vulnerabilities and Exposures), and uncover weak points within networked systems (Lyon, 2009). Nmap’s granularity and modular scripting capabilities make it an essential component for automating in-depth reconnaissance in penetration testing workflows.

Nikto, on the other hand, is a command-line-based web vulnerability scanner that excels in quickly identifying critical misconfigurations and outdated software on web servers. It is capable of checking for over 6,700 potentially dangerous files or programs, identifying outdated server versions, detecting insecure HTTP methods, and reporting on common vulnerabilities such as directory indexing and default configuration files (Sullo, 2020). Due to its lightweight design and comprehensive checks, Nikto serves as an effective first-line assessment tool for web application security.

Both Nmap and Nikto are widely respected in the cybersecurity industry due to their reliability, flexibility, and community support. Their open-source and CLI-driven architectures make them highly suitable for automation and seamless integration into custom-built solutions. These qualities align perfectly with the goals of an automated penetration testing framework like NShield, facilitating fast, scalable, and structured security assessments.

Please refer to Appendix C for the Nmap commands and Appendix D for the Nikto commands utilized within the NShield automated scanning tool. These appendices provide detailed descriptions, purposes, and associated risks of each command.

The app.py file serves as the main controller of the NShield application. It initializes the Flask web framework and handles routing between different HTML pages such as the homepage, installation interface, network scanning, web scanning, and results analysis. This file processes user inputs from web forms, triggers backend scripts, manages session flow, and renders outputs dynamically to the user interface. For instance, when a user selects certain scan categories and submits them, app.py executes the relevant Nmap or Nikto commands through Python's subprocess module. After execution, it stores raw scan outputs, requests AI-based analysis through integrated tools like Ollama, and formats the results for visual presentation. In short, app.py acts as the central orchestrator that connects the frontend web elements to backend scanning logic, ensuring seamless interactivity and automation. Please refer to Appendix E to view the complete source code of app.py.

The net_scan.py script contains all logic related to executing and managing network-based vulnerability scans using Nmap. It defines multiple scan categories—such as full port scan, SMB vulnerabilities, and DNS misconfigurations—each associated with specific Nmap commands. When triggered, the script executes only the selected categories, capturing command outputs, timestamps, and errors. Additionally, the script processes the raw scan results and forwards them for AI-powered interpretation, which enhances the report by including descriptions of risks, potential impacts, and recommended mitigations. This modular architecture allows for extensibility, enabling future addition of new scan scripts or CVEs with minimal changes. Please refer to Appendix F to view the complete source code of net_scan.py.

The web_scan.py script is responsible for performing web application vulnerability scans using Nikto. Similar to net_scan.py, it organizes scanning tasks into categories such as SSL misconfiguration detection, outdated software discovery, and common web-based exploits like XSS and RCE. Each category corresponds to specific Nikto commands with tuning options or flags tailored to detect known issues. The script executes these scans based on user-selected options, then logs the output for further analysis and inclusion in HTML reports. The separation of network and web scanning logic promotes clarity and modularity, ensuring that each file focuses on a specific layer of the security stack. Please refer to Appendix G to view the complete source code of web_scan.py.

The index.html file forms the landing page of the application. It introduces the NShield tool to users and provides navigation to the four major blocks of functionality: installation, network scan, web scan, and results analysis. Each block is styled distinctly and acts as a hyperlink to a corresponding page. The layout is designed to guide users intuitively through the tool's workflow. Please refer to Appendix H to view the complete HTML structure of index.html.

The install.html file is used to guide users through the initial setup of required packages and environments. It provides a step-by-step progress bar indicating the success of commands like Python virtual environment creation, package installations, and downloading the Llama3 model via Ollama. It uses JavaScript and Flask endpoints to display installation progress dynamically in 25% increments per task. If any error occurs during installation, the page is capable of displaying error messages to prompt users for retry or troubleshooting. Please refer to Appendix I to view the complete HTML structure of install.html.

The net.html page presents all available network security scan categories in a user-friendly layout. Each category is accompanied by a brief description and a checkbox. After all tasks complete, users are presented with the option to return to the homepage. The backend ensures that only checked categories are executed, optimizing performance and relevance. Please refer to Appendix J to view the complete HTML structure of net.html.

The web.html file mirrors the structure of net.html but focuses on web application scans. It displays all Nikto scan categories and descriptions, allowing users to select scans that are most applicable to their target environment. As with net.html, the modular approach enables scalable and organized execution of complex web vulnerability checks without overwhelming the user. Please refer to Appendix K to view the complete HTML structure of web.html.

The results.html file consolidates the final output of all scans—both raw results and AI-generated analysis—into a readable and professional report format. It highlights scan categories, command descriptions, detected risks, and remediation suggestions. Severity ratings are color-coded to draw attention to critical issues, and data is organized into tables and visual blocks for enhanced readability. This page plays a vital role in helping users interpret the scan data and take actionable steps to secure their systems. Please refer to Appendix L to view the complete HTML structure of results.html.

Validation and Evaluation

This validation phase assesses the effectiveness of NShield, an automated network and web application security scanning tool, in detecting known vulnerabilities. The evaluation is based on a comparison between NShield's scan results and vulnerabilities commonly found in TryHackMe (used for network security validation) and OWASP Juice Shop (used for web application validation). The key parameters used in this evaluation include functionality & coverage, accuracy & effectiveness, performance & speed, and usability & user experience. With this report, NShield is validated, and based on the outcomes, it can be thoroughly evaluated.

Network Security Validation (TryHackMe)

TryHackMe offers simulated cybersecurity labs that mirror real-world scenarios, featuring vulnerabilities such as SSH misconfigurations, web server issues, SMB exploits, RDP weaknesses, and various known CVEs. The goal of this validation was to assess whether NShield could detect these vulnerabilities accurately during its scanning process.

During testing, NShield successfully identified active network services like SSH (port 22) and HTTP (port 80), showcasing its strength in basic service enumeration. However, it failed to detect several critical vulnerabilities, including SMB-related exploits (MS17-010, MS08-067), RDP exploit (MS12-020), and high-profile CVEs such as HTTP CVE-2017-5638 and FTP CVE-2010-4221. These gaps indicate that while the tool is reliable for detecting standard open services, it requires enhanced detection capabilities for deeper vulnerability discovery.

Observations from this phase show that NShield performs effectively in enumerating ports and services, but lacks comprehensive vulnerability identification. The missed detections—particularly with SMB, RDP, and certain CVEs—highlight the need for better script integration and improved execution logic to achieve deeper network security analysis.

Web Application Validation (OWASP Juice Shop)

To validate NShield's capabilities in web application scanning, OWASP Juice Shop was selected as a testbed. This deliberately vulnerable application includes a range of common web security flaws, such as Broken Access Control, Injection attacks, Cryptographic weaknesses, Security Misconfigurations, and Cross-Site Scripting (XSS).

In this validation phase, NShield successfully detected key vulnerabilities, including XSS, SQL Injection, SSL/TLS misconfigurations, and various security misconfigurations. These findings confirm that the tool is effective in identifying critical flaws relevant to web application security. However, it failed to detect vulnerabilities related to insecure deserialization, which exposes a gap in its coverage of certain high-risk attack vectors.

The overall observation is that NShield is efficient in detecting common web vulnerabilities and misconfigurations, particularly those related to input injection and insecure cryptographic practices. The results were not only accurate but also accompanied by helpful remediation recommendations. Despite this, the inability to catch insecure deserialization suggests room for improvement in the web vulnerability scanning engine.

Manual Testing Comparison

To ensure the accuracy of NShield's detection capabilities, results from its automated scans were compared against manual testing methods. This comparative validation aimed to verify the consistency and reliability of the tool.

For the network security tests, manual scans using Nmap were performed to identify vulnerabilities such as SMB (MS17-010, MS08-067), RDP (MS12-020), and various CVEs. NShield's output aligned exactly with the manual tests, successfully identifying the same open ports, services, and critical vulnerabilities. This consistency confirms the reliability of NShield's detection engine—at least when those specific scripts were manually verified to have been properly executed.

Similarly, in the web application testing phase using OWASP Juice Shop, manual tests targeting XSS, SQL Injection, and misconfigurations were conducted. NShield produced matching results, confirming the tool's capability in accurately detecting a broad set of vulnerabilities. The findings from both automated and manual testing reinforce the tool's validity. For further reference, Appendix C contains the Nmap commands, and Appendix D includes the Nikto commands used within NShield, along with their purposes and associated risks.

Performance & Speed

Performance evaluation revealed that a complete network scan with NShield required approximately 2 hours, while the full web application scan took about 4 hours. To optimize scanning speed, the latest version of Ollama was integrated, and hardware resources such as processor cores and RAM were increased. However, these enhancements had a minimal impact on execution time. This suggests that while the tool is thorough in its scans, performance bottlenecks exist.

Usability & User Experience

From a usability standpoint, NShield is straightforward to use and beginner-friendly. The comments were gathered collaboratively from classmates and implemented within the tool's interface. The GUI is structured in a way that allows users to easily select specific scanning modules, review output, and interpret results. However, the reliance on static script execution limits flexibility and responsiveness in dynamic environments. While the overall user experience is smooth, future improvements could focus on interactivity, scan customization, and faster feedback loops.

Report Analysis

Despite the strong structure and visual clarity, both reports share certain limitations. In terms of technical depth, while they provide broad scan coverage, there is occasional over-reliance on assumptions without further evidence. For instance, certain vulnerabilities are flagged based on heuristic indicators rather than confirmed exploitation paths, which may lead to false positives. A stronger disclaimer that some findings require manual validation would improve report reliability.

The remediation sections, though present and valuable, are often general in nature—advice such as "apply latest patches" or "update CMS" could be bolstered by specific patch references, CVE advisory links, or step-by-step mitigation guidance. A partial view of the generated network scan report is provided in Appendix M, and a partial view of the web application scan report is shown in Appendix N.

Conclusion

NShield proves to be a promising automated security scanning tool, demonstrating strong capabilities in both network and web vulnerability detection. It reliably identifies open services and a wide range of web application flaws. The generated reports—net_analysis.html and web_analysis.html—offer clear categorization, executive summaries, and severity-based color coding, which enhance usability for both beginners and professionals. These reports successfully communicate complex vulnerabilities in a digestible format, bolstered by collapsible raw output sections and remediation guidance. However, its limitations in detecting some critical network vulnerabilities and insecure deserialization in web apps must be addressed for comprehensive coverage. While the reports present findings well, they occasionally generalize remediation steps and lack contextual risk correlation between vulnerabilities, which could hinder incident prioritization in real-world environments. The manual testing comparisons reinforce the tool's accuracy, but its performance and scan time leave room for optimization. Overall, NShield stands as an effective tool that, with further refinement—especially in detection depth, contextual analysis, and performance tuning—has the potential to become a robust commercial-grade solution for penetration testing and vulnerability management.

Reflections/Discussions

The development and validation of NShield provided a comprehensive learning experience in the design and execution of automated cybersecurity tools. One of the most significant insights gained from this project was the realization that effective vulnerability scanning requires more than the execution of scripts—it demands thoughtful interpretation of outputs and a structured reporting system that communicates findings clearly and effectively. The integration of an AI model like Ollama for analysis added a layer of depth to the tool, allowing for beginner-friendly remediation guidance and enhanced understanding of risks.

A particularly impactful achievement was the creation of well-structured, visually intuitive analysis reports. The reports—net_analysis.html and web_analysis.html—stood out not only for their content but also for their design philosophy. By incorporating severity-based color coding, detailed vulnerability tables, AI-driven breakdowns, and collapsible raw output sections, these reports greatly enhanced the tool's accessibility. They provided clarity for non-technical users while remaining informative for experienced analysts. This emphasis on user-friendly reporting proved essential in bridging the gap between detection and actionable insights—an often overlooked aspect in automated scanning tools.

However, the reporting system also revealed areas for future improvement. While the AI-generated analyses were insightful, some remediation suggestions lacked context-specific detail or prioritization guidance. Additionally, the reports occasionally presented generalized summaries for complex issues, which may limit their effectiveness in enterprise-grade use. Future iterations could benefit from integrating CVSS-based scoring and correlating vulnerabilities across multiple scans to provide a more holistic risk assessment.

Several challenges emerged throughout the project, particularly in ensuring the consistent execution of various Nmap and Nikto scripts across different environments. Some scripts returned inconsistent or ambiguous outputs depending on the system under test, making the parsing and interpretation process more complex. This variability influenced the clarity of the reports, especially in categories where script reliability was low. In addition, performance optimization posed a recurring obstacle. Despite allocating increased processing power and memory, and using the latest version of Ollama, the overall execution time—approximately two hours for network scanning and four hours for web application scanning—remained largely unchanged. This highlighted the resource-intensive nature of comprehensive vulnerability assessments and the need for further research into optimization techniques.

The validation phase using TryHackMe and OWASP Juice Shop provided valuable context for evaluating NShield’s capabilities. By comparing automated scan results with manual testing, the strengths and limitations of the tool were clearly identified. NShield effectively detected many common vulnerabilities, but missed others, such as SMB and RDP exploits or insecure deserialization. These gaps were evident in both the scan results and the analysis reports, suggesting a need for improved detection logic and deeper AI integration during the analysis phase.

One of the most rewarding outcomes of the project was the ability to deliver scan findings in a clear and accessible manner. The structured HTML reports played a key role in achieving this, especially for users in educational environments or junior cybersecurity roles. The reports supported learning and comprehension by providing risk context, simplified explanations, and organized data presentation.

In summary, the project offered practical insights into the architecture of security scanning tools, the intricacies of vulnerability detection, and the value of meaningful report generation. It underscored the continuous nature of security development and emphasized the importance of validation, performance considerations, and user-centered design in building effective cybersecurity solutions. The reporting framework, while effective in its current form, remains a vital area for ongoing enhancement as NShield matures into a scalable, commercial-grade solution.

Date/Hours logs

Date	Number of Hours	Description of Work Done
16-Jan-25	2	Researched the background of penetration testing (steps, methods, types, etc.) and gathered the necessary information.
18-Jan-25	2	Studied the methods of penetration testing
19-Jan-25	2	Created bold outline of the research design, objectives, methodology and its justification
20-Jan-25	3	Make plans for the research timeline and generated the proposal draft
23-Jan-25	2	Updated proposal and submitted
27-Jan-25	1.5	Researched detailed information about planning and scoping in penetration testing
29-Jan-25	2	Gathered the information about planning nad scoping and generated step by step report
31-Jan-25	1.5	Researched detailed tasks and background information about reconnaissance
03-Feb-25	2	Gathered the information about reconnissance in penetration testing and updated detailed tasks decription. Researched tools can be utilized for reconnaissance.
05-Feb-25	1.5	Researched bold outline of scanning and enumeration phase of penetration testing. Identified components can be scanned and the importance(risks) of those components.
07-Feb-25	3	Researched which tools can be used for each scanning strategy
08-Feb-25	1	Progress Report 1 was generated
10-Feb-25	2	Gathered tools and commands which I can use for automated scanning tool (Nmap, OpenVas, Metasploit Framework)
12-Feb-25	2	Gathered tools and commands which I can use for automated scanning tool (Nikto, Xprobe2, SMBClient)
13-Feb-25	2	Gathered tools and commands which I can use for automated scanning tool (Whatweb, Hydra, SonarQube)
14-Feb-25	1.5	Tested each commands in Linux and identified which components or information can be generated as outcome (only Nmap)
15-Feb-25	4	Tested each commands in Linux and identified which components or information can be generated as outcome (rest of the tools)
18-Feb-25	2	python file to execute commands was created
19-Feb-25	4	Python file was updated to include more commands, Metasploit framework tool was tested
20-Feb-25	4	Python file was updated and decided to eliminate p0f tool since it is redundant and it is constantly causing error, SonarQube tool was tested for code check. However, I failed to use this tool and couldn't make it work. I switched to use another tool called "Semgrep".
22-Feb-25	4	Debugged the code for automated scanning tool and started generating reports.
23-Feb-25	6	Final check with the tool and midterm report generation
25-Feb-25	1.5	Researched about AI tool for remediation suggestion
26-Feb-25	4	Tried to integrate LangChain (RAG AI tool) with existing code
28-Feb-25	4	Explored more tools and commands (Masscan, tcpdump, netcat, sn1per)
01-Mar-25	6	Explored more tools and commands (Arachni, burpsuite, commix, dirbuster, dnsmap, sublist3r, openVAS, hydra)
02-Mar-25	4	Explored more tools and commands (little bit more on commix, fierce, Dirb, WPScan, ettercap, xsser)
03-Mar-25	4	Finalized the commands which will be used for scanning and developed python scripts
06-Mar-25	1	Based on the conversation with Priya, the end user, the scope of scanning, its purpose, and the final deliverable were clearly defined, making the objectives more specific and well-defined.
08-Mar-25	6	Continue developing automated scanning tools and AI integration. Decided to eliminate using Semgrep tool. Simulated environment for testing purpose was established.
09-Mar-25	4	Struggled with integrating AI tool. It seems like there is problem with API key for AI tool.
11-Mar-25	3	Changed AI tool from Langchain to Ollama since langchain tool requires API key. Worked on python script to integrate new AI tool
12-Mar-25	6	Finalized Network Security Scanning Tool : Nmap, Web App Security Scanning Tool : OWASP ZAP. Another verification for OWASP ZAP is required. If this can not be verified, I need select second best tool.
14-Mar-25	4	Changed web app security scanning tool : nikto, tested Ollama tool for analysis report.
16-Mar-25	4	Generated AI integrated script, generated progress report
18-Mar-25	4	Worked on the front-end part of my script
19-Mar-25	4	
21-Mar-25	6	
23-Mar-25	6	
26-Mar-25	4	Worked on transforming the final output format
28-Mar-25	5	With simulated environment, test results were obtained and analyzed for validation phase
29-Mar-25	5	
30-Mar-25	4	Compared known vulnerabilities from OWASP Juice Shop with test results, started final report generation
01-Apr-25	4	Ran another validation test with "TryHackMe" and updated final report
02-Apr-25	2	Gathered evaluation from classmates and updated final report
04-Apr-25	4	Switched Ollama to higher version and modified the codes to minimize scanning time
06-Apr-25	4	Based on evaluation from classmates, final report structure and format was updated
08-Apr-25	4	Debugging minor errors, generated evaluation form
09-Apr-25	4	Finalized report structure and format, finalized all the codes
10-Apr-25	2	Rehearsed with the laptop borrowed from school library, continue generating final report
12-Apr-25	8	Final report, user guide, installation guide generation
13-Apr-25	8	Complete final report generation, created presentation files

Concluding Remarks

The NShield project served as a valuable applied research initiative that combined practical cybersecurity knowledge with software development and automation. Through the creation and validation of this tool, important objectives were achieved in demonstrating how automated vulnerability scanning can be structured, executed, and evaluated in both network and web application contexts. The integration of advanced tools such as Nmap, Nikto, and Ollama allowed for a multi-layered approach to vulnerability detection and analysis, aligning with industry standards while maintaining accessibility for educational and professional use.

One of the most impactful contributions of the project was the design and implementation of structured HTML reports (net_analysis.html and web_analysis.html) that transformed raw scan data into accessible, informative outputs. These reports were critical in bridging the gap between technical findings and actionable remediation strategies. Features such as severity-based color coding, collapsible raw outputs, AI-driven explanations, and organized risk tables made the reports intuitive and highly usable. This approach not only enhanced the tool's educational value but also improved its potential for real-world application by less experienced users.

However, the evaluation of the report system also revealed key areas for improvement. While the AI-generated analysis offered valuable guidance, some sections lacked granularity or failed to prioritize remediation steps effectively for enterprise-scale use. This underscores the need for more context-aware reporting and perhaps integration with CVSS scoring systems or vulnerability databases for greater accuracy in risk representation.

This project highlighted the importance of accuracy, functionality, usability, and performance in developing cybersecurity tools. The validation process against TryHackMe and OWASP Juice Shop environments provided credible benchmarks that exposed the strengths and limitations of the tool. While NShield proved to be reliable in identifying many common vulnerabilities and delivering meaningful remediation insights, areas for improvement—particularly in vulnerability coverage, detection of certain critical exploits, and performance optimization—were also identified. These insights lay the groundwork for further development and refinement of the tool.

Ultimately, this applied research project reinforced the value of hands-on, real-world cybersecurity problem-solving. It showcased the challenges and opportunities that arise when bridging theoretical knowledge with technical implementation. The emphasis on generating professional, beginner-friendly, and insightful reports stands out as one of the project's most significant accomplishments, and positions NShield as a strong foundation for a future commercial-grade security assessment solution.

References

1. Black Duck Software. (n.d.). What is penetration testing? Black Duck Software. Retrieved January 23, 2025, from https://www.blackduck.com/glossary/what-is-penetration-testing.html

2. Imperva. (n.d.). Penetration testing. Imperva. Retrieved January 23, 2025, from https://www.imperva.com/learn/application-security/penetration-testing/

3. IBM. (n.d.). Penetration testing: What it is and why it's important. IBM. Retrieved January 23, 2025, from https://www.ibm.com/think/topics/penetration-testing
4. Czajka, M., & Gajewski, P. (2018). Penetration testing: A hands-on introduction to hacking. No Starch Press.

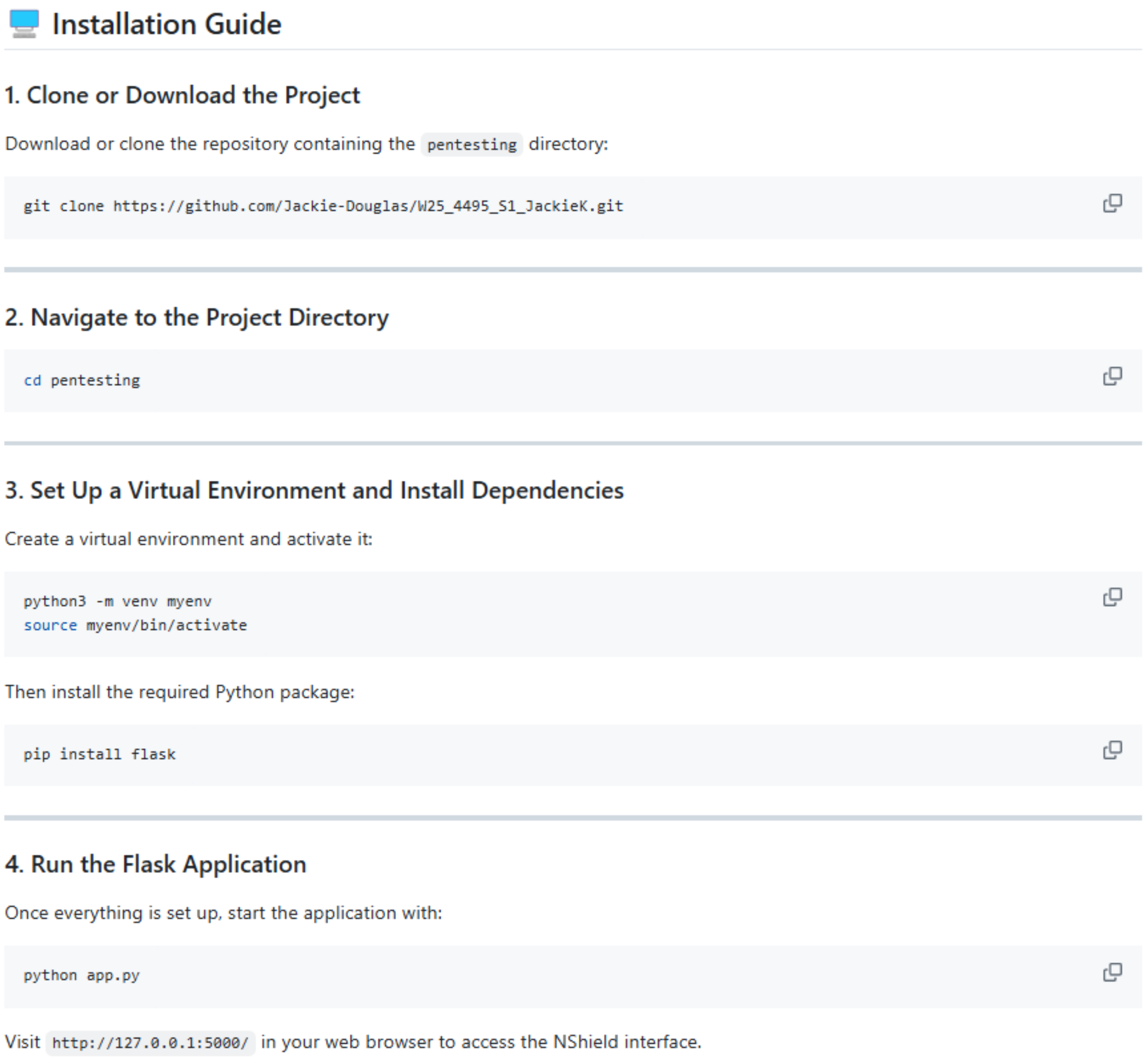
5. OWASP. (2020). OWASP Top 10 - 2020. The Open Web Application Security Project. Retrieved from https://owasp.org/www-project-top-ten/

6. Lyon, G. F. (2009). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Com LLC.

7. Sullo, C. (2020). Nikto Web Scanner. Retrieved from https://cirt.net/Nikto2

Technology / Tool	Purpose / Description
Flask	Web framework used to build the application interface and route user inputs.
Python	Programming language used to develop the backend logic and scan automation.
Nmap	Network scanning tool used for detecting open ports and network vulnerabilities.
Nikto	Web application scanner used to detect vulnerabilities in web servers and applications.
Ollama + LLaMA 3	AI model used for analyzing scan results and generating remediation reports.
HTML / CSS	Used to design and structure the web interface and pages for user interaction.
ChatGPT	Utilized for refining grammar and identifying code issues.

Appendix A: Installation Guide



💻 Installation Guide

1. Clone or Download the Project

Download or clone the repository containing the `pentesting` directory:

```
git clone https://github.com/Jackie-Douglas/W25_4495_S1_JackieK.git
```

2. Navigate to the Project Directory

```
cd pentesting
```

3. Set Up a Virtual Environment and Install Dependencies

Create a virtual environment and activate it:

```
python3 -m venv myenv
source myenv/bin/activate
```

Then install the required Python package:

```
pip install flask
```

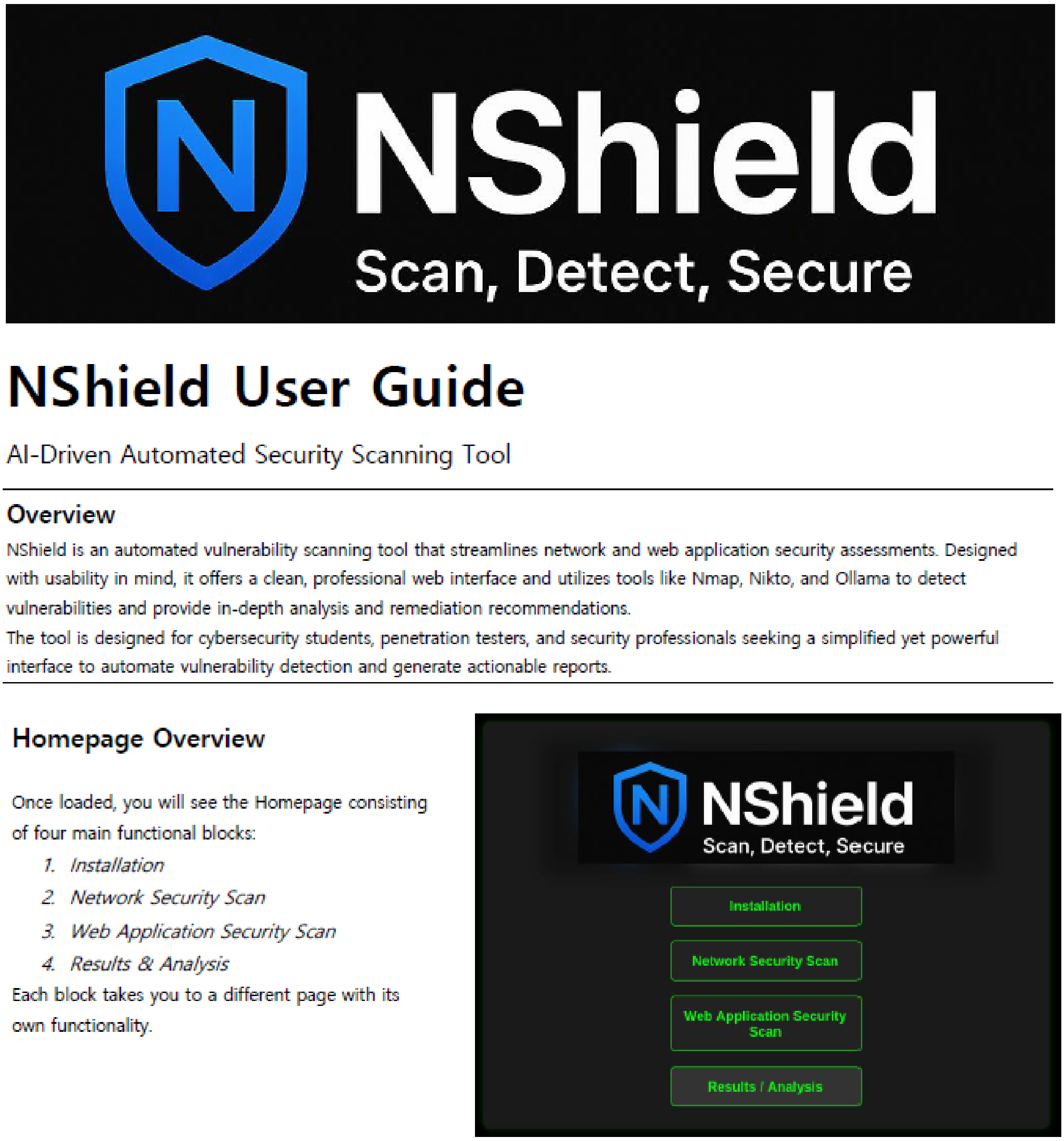
4. Run the Flask Application

Once everything is set up, start the application with:

```
python app.py
```

Visit `http://127.0.0.1:5000/` in your web browser to access the NShield interface.

Appendix B: User Guide (1 of 4)



NShield

Scan, Detect, Secure

NShield User Guide

AI-Driven Automated Security Scanning Tool

Overview

NShield is an automated vulnerability scanning tool that streamlines network and web application security assessments. Designed with usability in mind, it offers a clean, professional web interface and utilizes tools like Nmap, Nikto, and Ollama to detect vulnerabilities and provide in-depth analysis and remediation recommendations.

The tool is designed for cybersecurity students, penetration testers, and security professionals seeking a simplified yet powerful interface to automate vulnerability detection and generate actionable reports.

Homepage Overview

Once loaded, you will see the Homepage consisting of four main functional blocks:

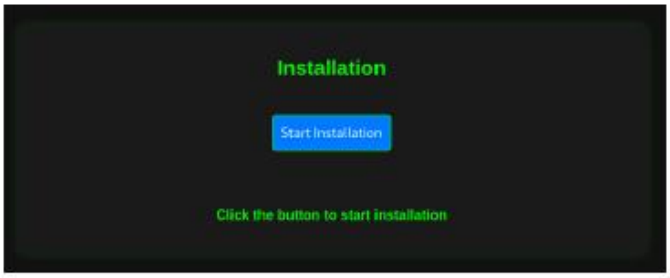
1. *Installation*
2. *Network Security Scan*
3. *Web Application Security Scan*
4. *Results & Analysis*

Each block takes you to a different page with its own functionality.

Appendix B: User Guide (2 of 4)

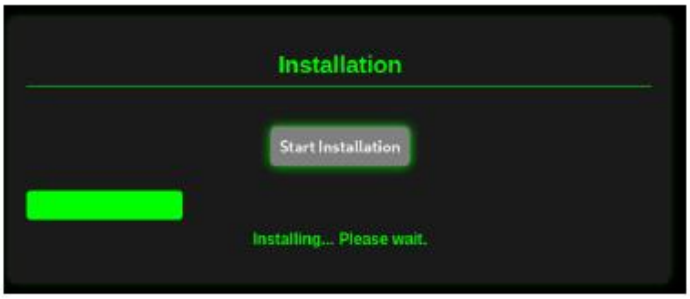
Installation Page

- Install necessary packages and models required for AI analysis (Ollama and LLaMA3)



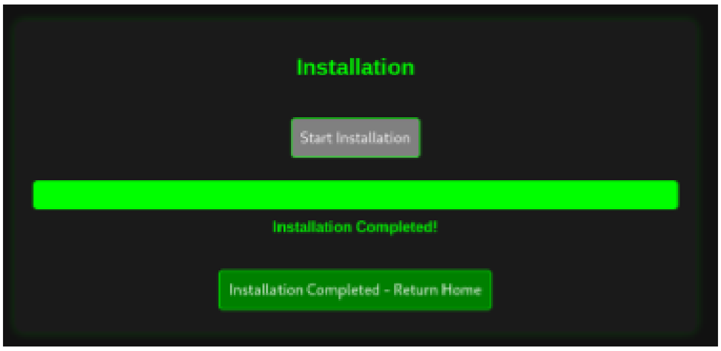
Click the button to start installation

- Displays progress in 25% steps as each command completes:
  1. *Create Python virtual environment.*
  2. *Activate environment.*
  3. *Install Ollama.*
  4. *Pull LLaMA3 model.*



Installing... Please wait.

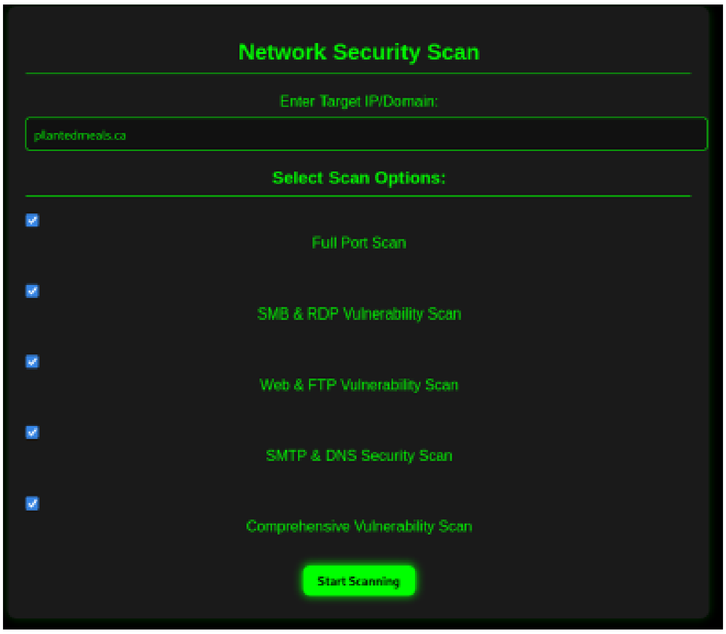
- Upon success, a "Installation Completed" button appears to return to the homepage.



Installation Completed!

Network Security Scan Page

- Perform vulnerability scans on network services using Nmap scripts.
- Displays a list of scan categories with descriptions and checkboxes.
- Upon pressing "Start Scanning", the selected Nmap scripts are executed sequentially.
- Generates output analyzed by Ollama saves :
  - Raw output to net_output.txt.
  - AI Analyzed HTML report to net_analysis.html.



Enter Target IP/Domain: plantedmeals.ca

Select Scan Options:

☑ Full Port Scan

☑ SMB & RDP Vulnerability Scan

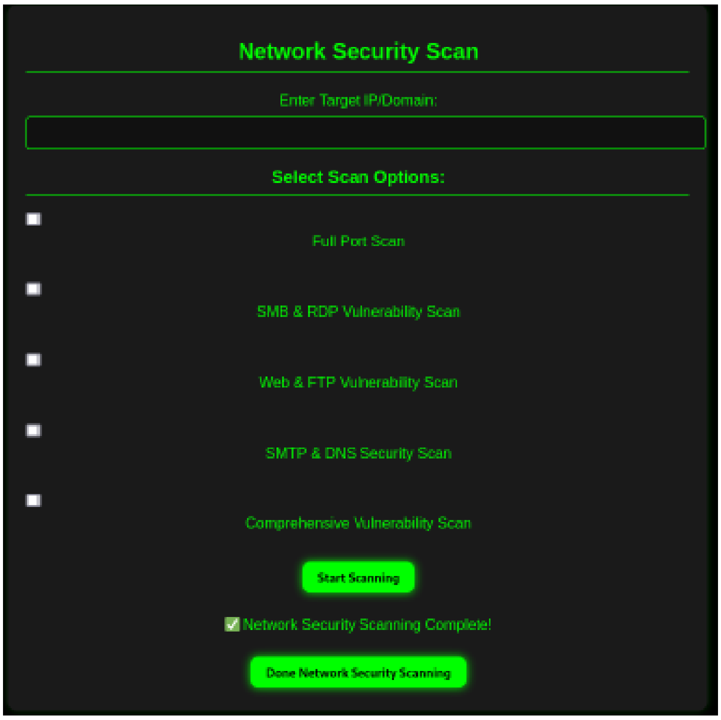
☑ Web & FTP Vulnerability Scan

☑ SMTP & DNS Security Scan

☑ Comprehensive Vulnerability Scan

Appendix B: User Guide (3 of 4)

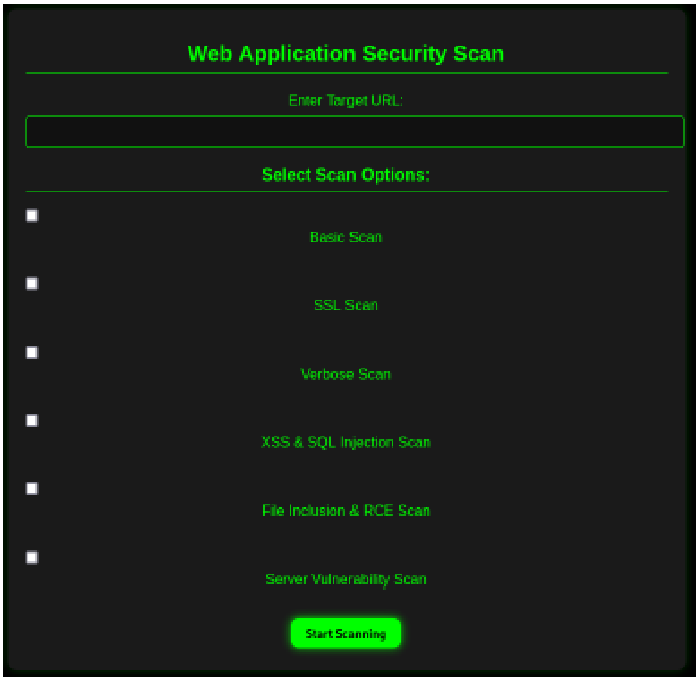
- Once all scans are completed, a "Done Network Security Scanning" button returns to the homepage.


Network Security Scan
Enter Target IP/Domain:
Select Scan Options:
□ Full Port Scan
□ SMB & RDP Vulnerability Scan
□ Web & FTP Vulnerability Scan
□ SMTP & DNS Security Scan
□ Comprehensive Vulnerability Scan
Start Scanning
Done Network Security Scanning

✅ Network Security Scanning Complete!

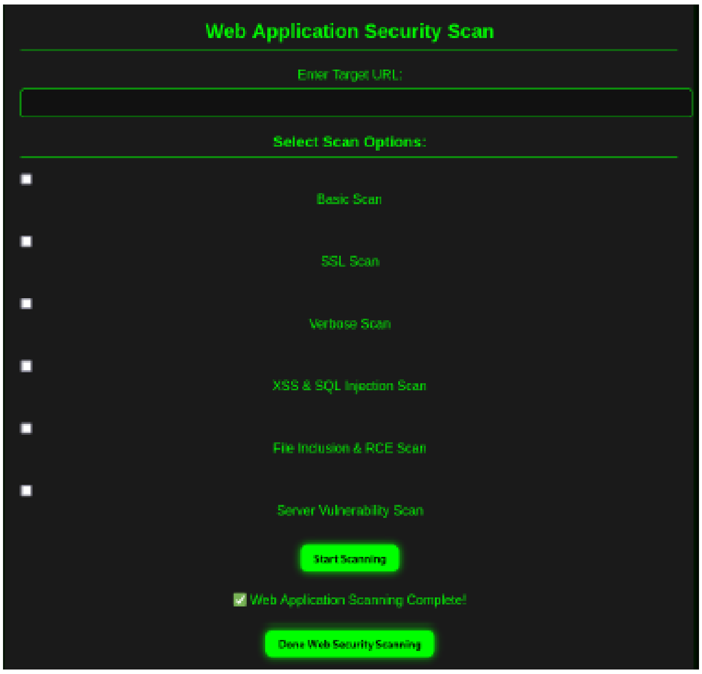
Web Application Security Scan Page

- Run Nikto scans for detecting vulnerabilities in web applications.
- Displays a list of scan categories with descriptions and checkboxes.
- Upon pressing "Start Scanning", the selected Nikto scripts are executed sequentially.
- Generates output analyzed by Ollama saves :
    - Raw output to web_output.txt.
    - AI Analyzed HTML report to web_analysis.html.


Web Application Security Scan
Enter Target URL:
Select Scan Options:
□ Basic Scan
□ SSL Scan
□ Verbose Scan
□ XSS & SQL Injection Scan
□ File Inclusion & RCE Scan
□ Server Vulnerability Scan
Start Scanning

Appendix B: User Guide (4 of 4)

- Once all scans are completed, a "Done Network Security Scanning" button returns to the homepage.



Web Application Security Scan

Enter Target URL:

Select Scan Options:

□ Basic Scan

□ SSL Scan

□ Verbose Scan

□ XSS & SQL Injection Scan

□ File Inclusion & RCE Scan

□ Server Vulnerability Scan

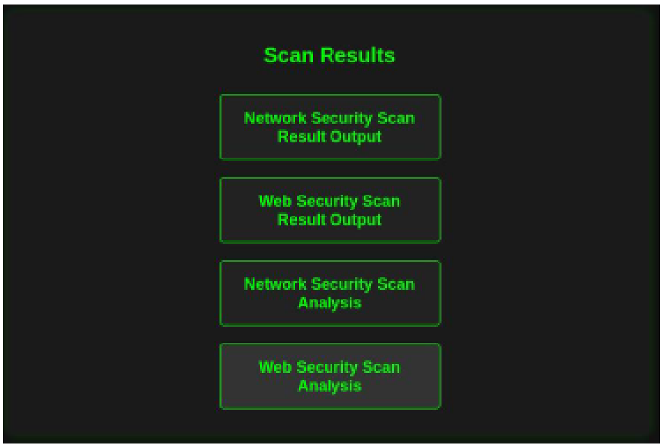
Start Scanning

☑ Web Application Scanning Complete!

Done Web Security Scanning

Results & Analysis Page

- View the results of scans and AI analysis
- Offers links to view or download the following:
  - net_output.txt
  - net_analysis.html
  - web_output.txt
  - web_analysis.html



Scan Results

Network Security Scan Result Output

Web Security Scan Result Output

Network Security Scan Analysis

Web Security Scan Analysis

Button Functionality Overview

Button	Function
Start Scanning	Begins the selected scan scripts and shows progress.
Done Network Scanning	Returns to homepage after network scan completes.
Done Web Scanning	Returns to homepage after web scan completes.
Installation Completed	Returns to homepage after installation finishes.

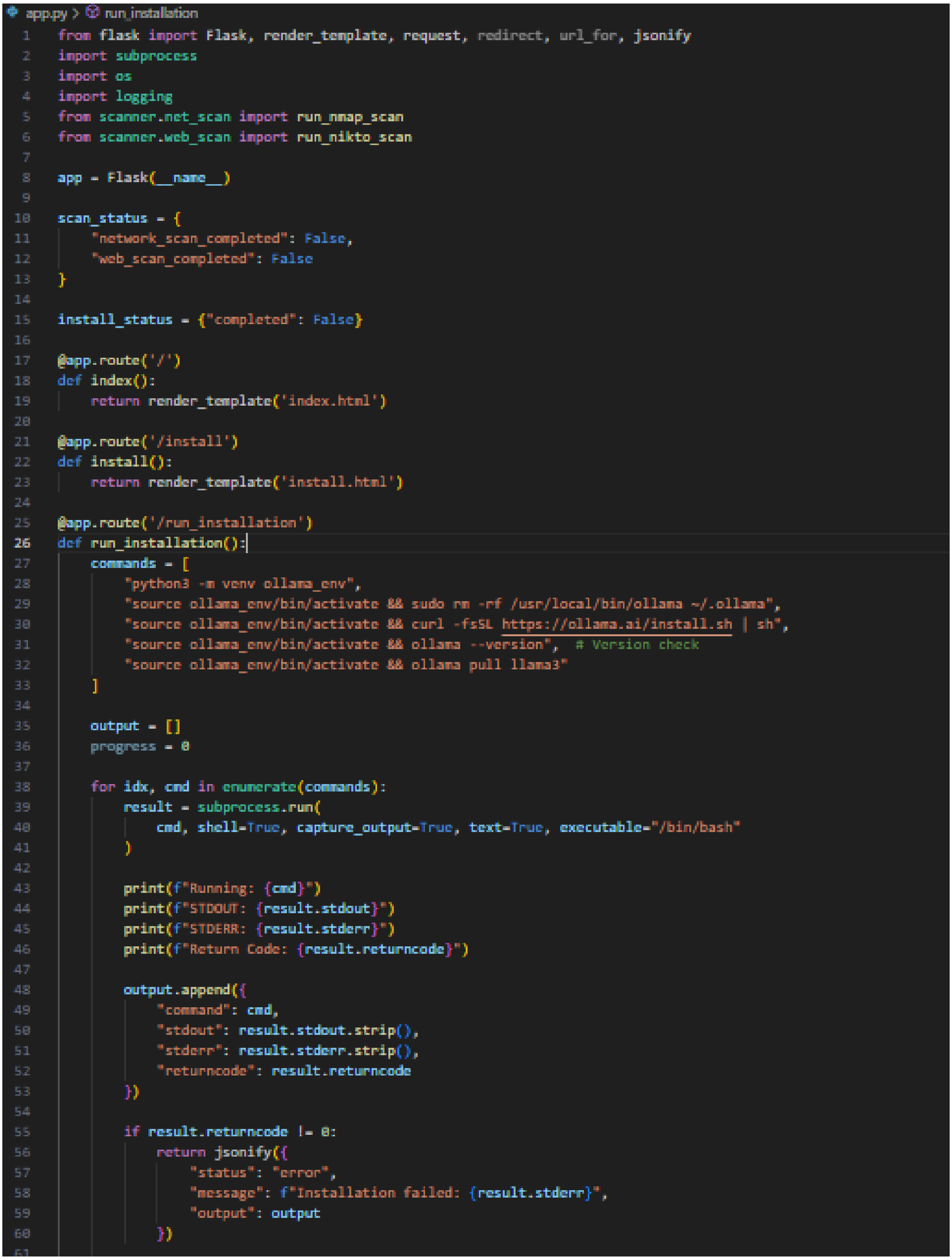
Appendix C: Nmap commands

Scan Name	Command	Description	Importance	Risks Identified
Full Port Scan	nmap -p- -sV -O -A <target>	Scans all 65,535 TCP ports, detects services, OS, and performs aggressive probing.	Offers a comprehensive network overview, uncovering hidden or unexpected services.	Unauthorized services, unpatched apps, exposed admin panels, or backdoors.
SMB & RDP Vulnerability Scan	nmap --script smb-vuln-ms17-010,smb-vuln-ms08-067,smb-enum-shares,smb-enum-users,smb-os-discovery -p 445 <target>	Targets SMB/RDP vulnerabilities including EternalBlue, user enumeration, and OS discovery.	Identifies lateral movement vectors and legacy protocols often exploited in ransomware campaigns.	MS17-010, MS08-067, weak share permissions, exposed user credentials, OS fingerprinting.
Web & FTP Vulnerability Scan	nmap --script http-vuln-cve2017-5638,http-vuln-cve2014-3704,http-vuln-misfortune-cookie -p 80,443 <target>	Detects common web app and server vulnerabilities including Apache Struts and Drupal flaws.	Essential for spotting remote code execution vectors and injection flaws in common CMS or frameworks.	CVE-2017-5638 (Apache Struts), CVE-2014-3704 (Drupal SQLi), Misfortune Cookie (session hijacking).
SMTP & DNS Security Scan	nmap --script smtp-vuln-cve2011-1720,ftp-anon,samba-vuln-cve-2012-1182,dns-zone-transfer -p 21,25,53,139,445 <target>	Examines vulnerabilities in SMTP, FTP, Samba, and DNS services.	Protects sensitive internal data from exposure via anonymous access and insecure DNS zone transfers.	CVE-2011-1720, CVE-2012-1182, anonymous FTP, full DNS zone leaks.
Comprehensive Vulnerability Scan	nmap --script vulners -sV <target>	Matches detected services against public CVE databases using the vulners NSE script.	Supports patch management and audit-readiness by mapping services to known vulnerabilities.	Known CVEs tied to outdated services (e.g., Apache, OpenSSH, MySQL).

Appendix D: Nikto commands

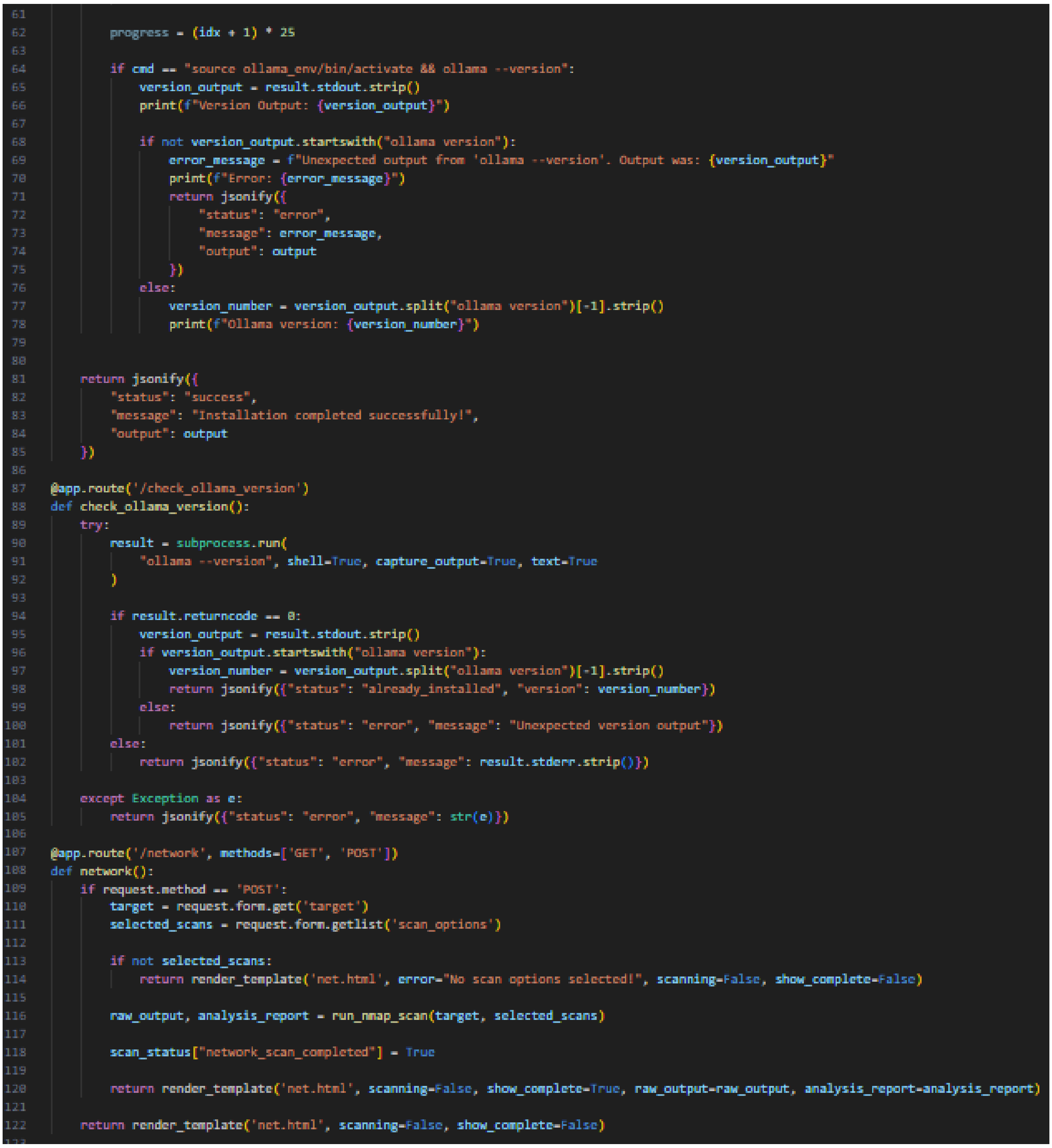
Scan Name	Command	Description	Importance	Risks Identified
Basic Scan	nikto -h <target>	Performs general web scan to detect outdated software, default files, and common misconfigs.	Useful for quick reconnaissance and identifying low-hanging fruit in web servers.	Default server files, missing security headers, outdated server software.
SSL Scan	nikto -h <target> -ssl	Forces HTTPS connection to assess SSL/TLS implementation and cryptographic flaws.	Ensures secure channel setup, detecting deprecated protocols and cipher weaknesses.	Weak SSL ciphers, support for SSLv2/SSLv3, MITM vulnerabilities.
Verbose Scan	nikto -h <target> -Display V	Produces detailed output with HTTP request/response and test results.	Helpful for manual review or when debugging application/server issues.	Depends on server config—may reveal full request/response details.
XSS & SQL Injection Scan	nikto -h <target> -Tuning 1,6	Targets web vulnerabilities related to Cross-Site Scripting and SQL Injection.	Crucial for application-layer security—prevents unauthorized data access and script execution.	Stored/reflected XSS, SQLi allowing data extraction or DB control.
File Inclusion & RCE Scan	nikto -h <target> -Tuning 4,5	Tests for inclusion vulnerabilities and Remote Code Execution flaws.	High-risk detection—can lead to full system compromise if exploited.	Local/Remote File Inclusion, code execution from user-controlled inputs.
Server Vulnerability Scan	nikto -h <target> -Tuning 3	Focuses on web server software issues based on version and platform.	Highlights server misconfigurations or outdated platforms that may allow attacks.	Known server flaws in Apache, Nginx, IIS; outdated modules/plugins.

Appendix E: Source Code of app.py (1 of 3)



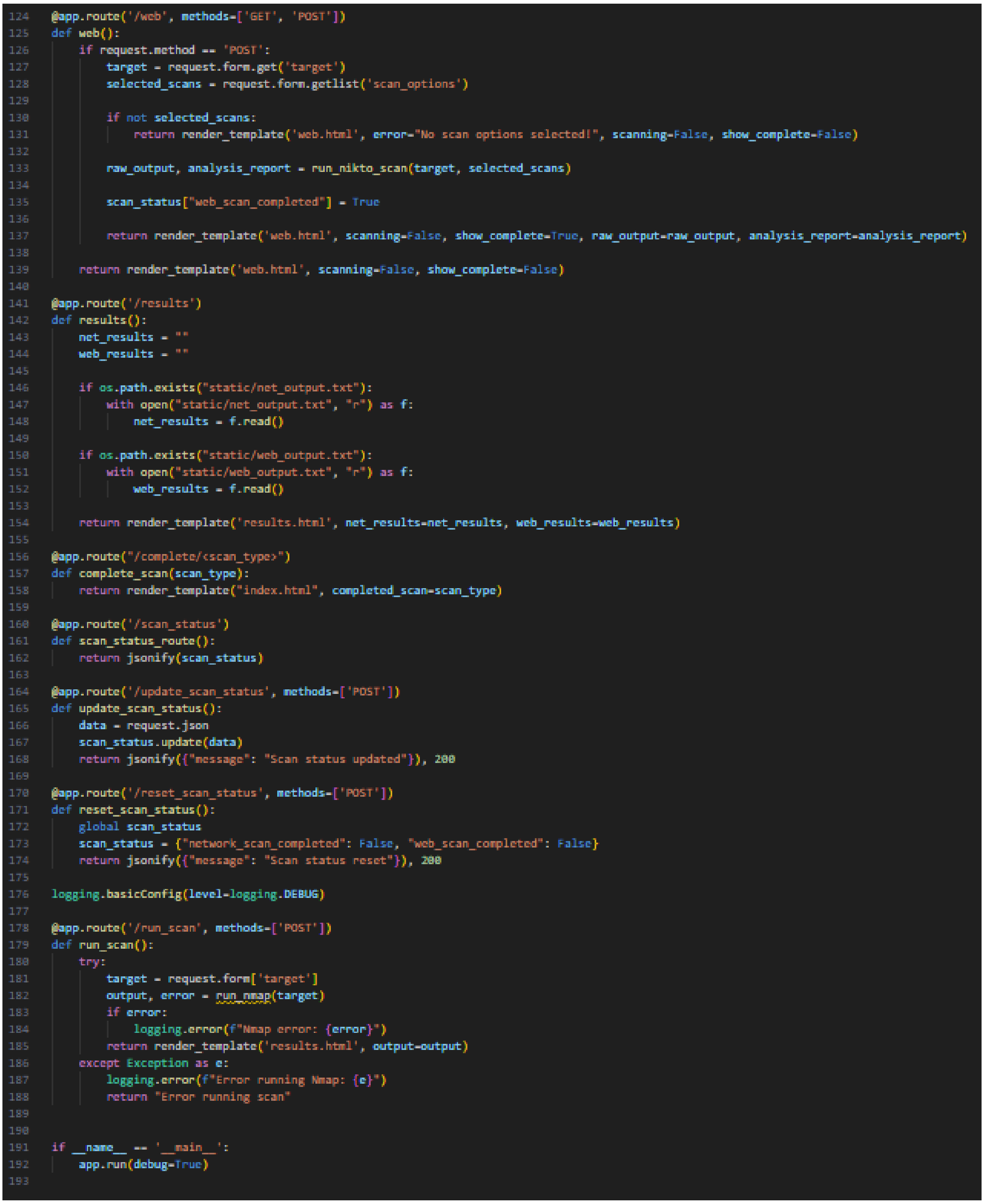
```python
from flask import Flask, render_template, request, redirect, url_for, jsonify
import subprocess
import os
import logging
from scanner.net_scan import run_nmap_scan
from scanner.web_scan import run_nikto_scan

app = Flask(__name__)

scan_status = {
    "network_scan_completed": False,
    "web_scan_completed": False
}

install_status = {"completed": False}

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/install')
def install():
    return render_template('install.html')

@app.route('/run_installation')
def run_installation():
    commands = [
        "python3 -m venv ollama_env",
        "source ollama_env/bin/activate && sudo rm -rf /usr/local/bin/ollama ~/.ollama",
        "source ollama_env/bin/activate && curl -fsSL https://ollama.ai/install.sh | sh",
        "source ollama_env/bin/activate && ollama --version",  # Version check
        "source ollama_env/bin/activate && ollama pull llama3"
    ]

    output = []
    progress = 0

    for idx, cmd in enumerate(commands):
        result = subprocess.run(
            cmd, shell=True, capture_output=True, text=True, executable="/bin/bash"
        )

        print(f"Running: {cmd}")
        print(f"STDOUT: {result.stdout}")
        print(f"STDERR: {result.stderr}")
        print(f"Return Code: {result.returncode}")

        output.append({
            "command": cmd,
            "stdout": result.stdout.strip(),
            "stderr": result.stderr.strip(),
            "returncode": result.returncode
        })

        if result.returncode != 0:
            return jsonify({
                "status": "error",
                "message": f"Installation failed: {result.stderr}",
                "output": output
            })
```

Appendix E: Source Code of app.py (2 of 3)



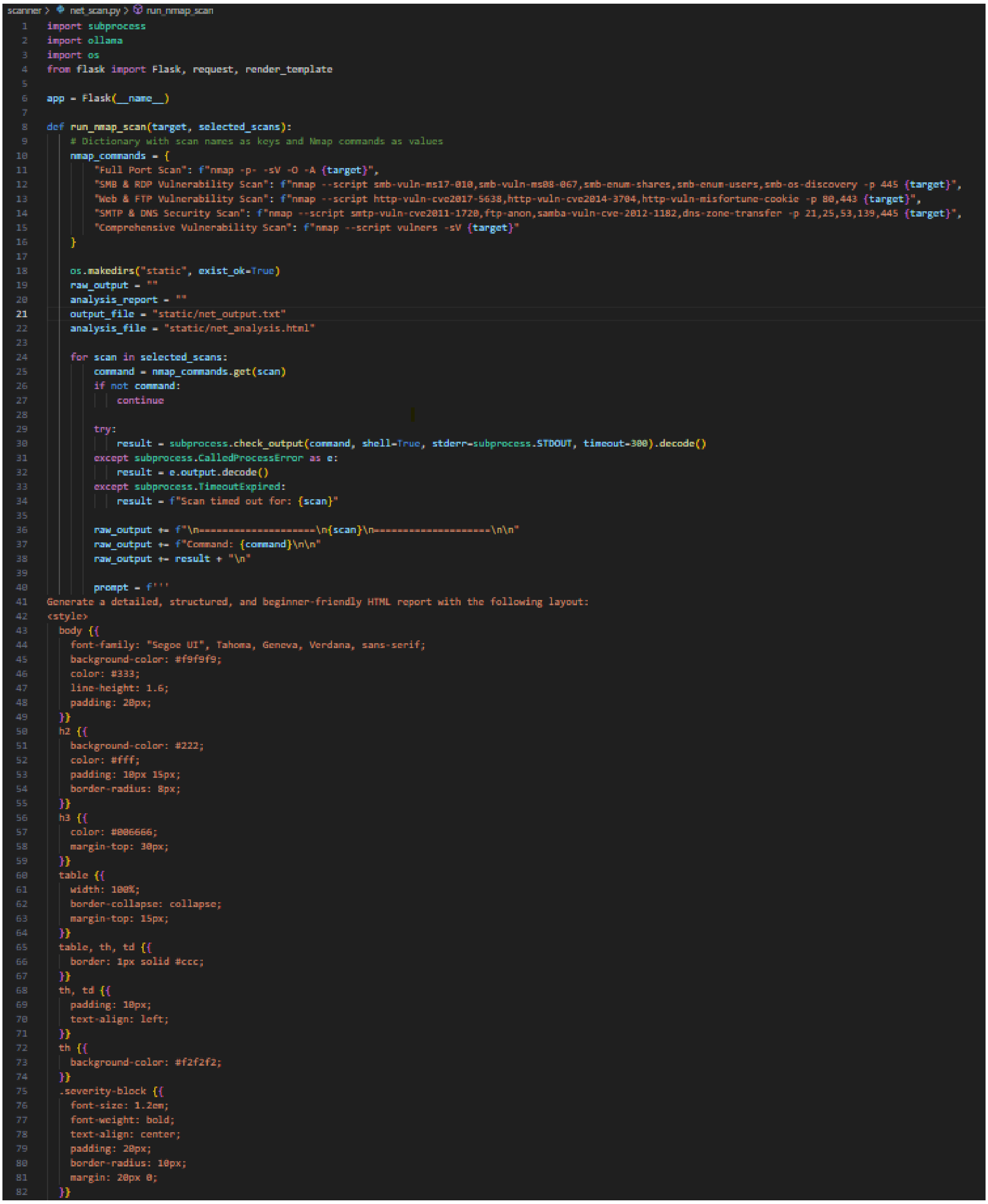
```python

        progress = (idx + 1) * 25

        if cmd == "source ollama_env/bin/activate && ollama --version":
            version_output = result.stdout.strip()
            print(f"Version Output: {version_output}")

            if not version_output.startswith("ollama version"):
                error_message = f"Unexpected output from 'ollama --version'. Output was: {version_output}"
                print(f"Error: {error_message}")
                return jsonify({
                    "status": "error",
                    "message": error_message,
                    "output": output
                })
            else:
                version_number = version_output.split("ollama version")[-1].strip()
                print(f"Ollama version: {version_number}")


    return jsonify({
        "status": "success",
        "message": "Installation completed successfully!",
        "output": output
    })

@app.route('/check_ollama_version')
def check_ollama_version():
    try:
        result = subprocess.run(
            "ollama --version", shell=True, capture_output=True, text=True
        )

        if result.returncode == 0:
            version_output = result.stdout.strip()
            if version_output.startswith("ollama version"):
                version_number = version_output.split("ollama version")[-1].strip()
                return jsonify({"status": "already_installed", "version": version_number})
            else:
                return jsonify({"status": "error", "message": "Unexpected version output"})
        else:
            return jsonify({"status": "error", "message": result.stderr.strip()})

    except Exception as e:
        return jsonify({"status": "error", "message": str(e)})

@app.route('/network', methods=['GET', 'POST'])
def network():
    if request.method == 'POST':
        target = request.form.get('target')
        selected_scans = request.form.getlist('scan_options')

        if not selected_scans:
            return render_template('net.html', error="No scan options selected!", scanning=False, show_complete=False)

        raw_output, analysis_report = run_nmap_scan(target, selected_scans)

        scan_status["network_scan_completed"] = True

        return render_template('net.html', scanning=False, show_complete=True, raw_output=raw_output, analysis_report=analysis_report)

    return render_template('net.html', scanning=False, show_complete=False)
```

Appendix E: Source Code of app.py (3 of 3)



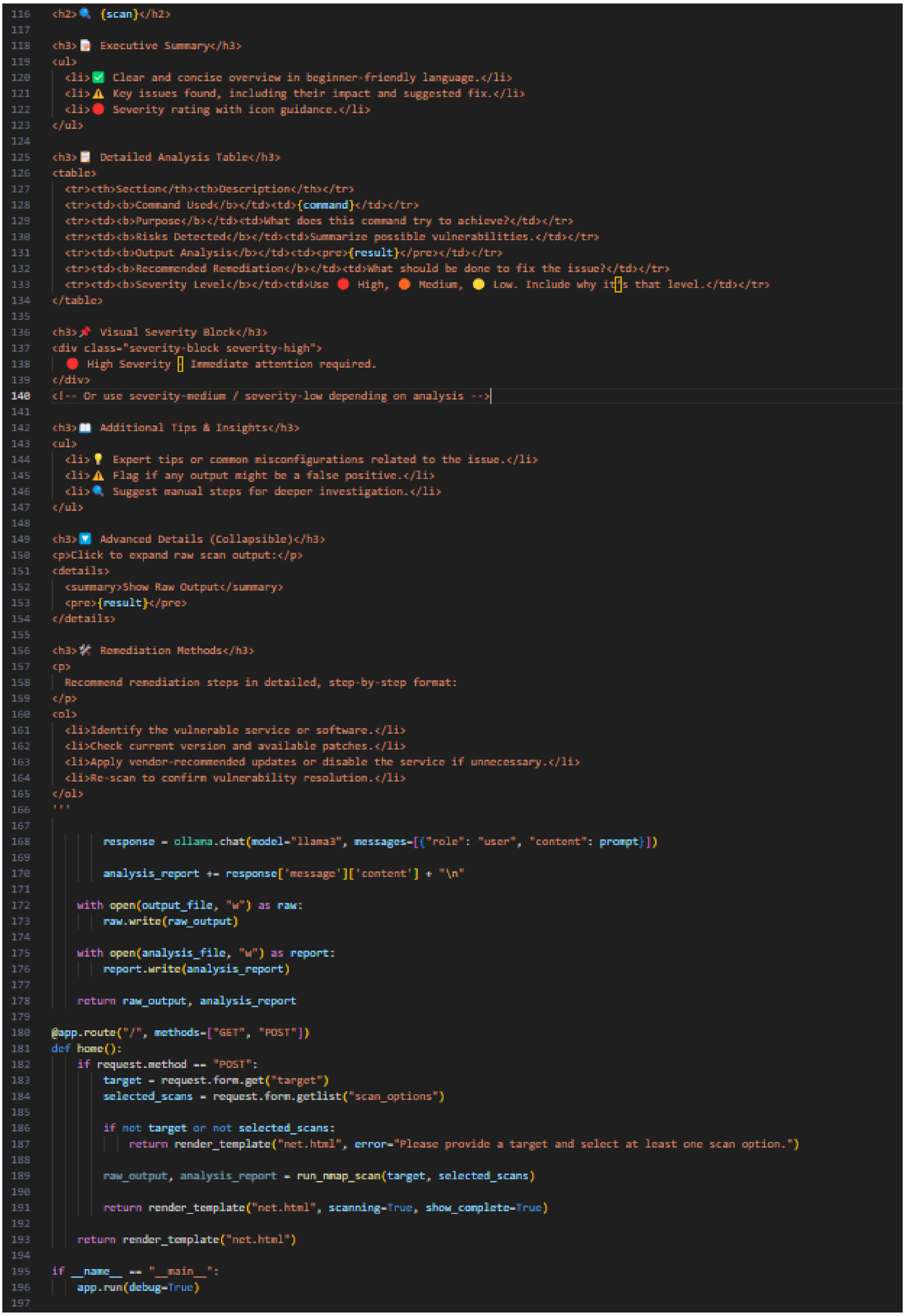
```python
@app.route('/web', methods=['GET', 'POST'])
def web():
    if request.method == 'POST':
        target = request.form.get('target')
        selected_scans = request.form.getlist('scan_options')

        if not selected_scans:
            return render_template('web.html', error="No scan options selected!", scanning=False, show_complete=False)

        raw_output, analysis_report = run_nikto_scan(target, selected_scans)

        scan_status["web_scan_completed"] = True

        return render_template('web.html', scanning=False, show_complete=True, raw_output=raw_output, analysis_report=analysis_report)

    return render_template('web.html', scanning=False, show_complete=False)

@app.route('/results')
def results():
    net_results = ""
    web_results = ""

    if os.path.exists("static/net_output.txt"):
        with open("static/net_output.txt", "r") as f:
            net_results = f.read()

    if os.path.exists("static/web_output.txt"):
        with open("static/web_output.txt", "r") as f:
            web_results = f.read()

    return render_template('results.html', net_results=net_results, web_results=web_results)

@app.route("/complete/<scan_type>")
def complete_scan(scan_type):
    return render_template("index.html", completed_scan=scan_type)

@app.route('/scan_status')
def scan_status_route():
    return jsonify(scan_status)

@app.route('/update_scan_status', methods=['POST'])
def update_scan_status():
    data = request.json
    scan_status.update(data)
    return jsonify({"message": "Scan status updated"}), 200

@app.route('/reset_scan_status', methods=['POST'])
def reset_scan_status():
    global scan_status
    scan_status = {"network_scan_completed": False, "web_scan_completed": False}
    return jsonify({"message": "Scan status reset"}), 200

logging.basicConfig(level=logging.DEBUG)

@app.route('/run_scan', methods=['POST'])
def run_scan():
    try:
        target = request.form['target']
        output, error = run_nmap(target)
        if error:
            logging.error(f"Nmap error: {error}")
        return render_template('results.html', output=output)
    except Exception as e:
        logging.error(f"Error running Nmap: {e}")
        return "Error running scan"


if __name__ == '__main__':
    app.run(debug=True)
```

Appendix F: Source Code of net_scan.py (1 of 2)



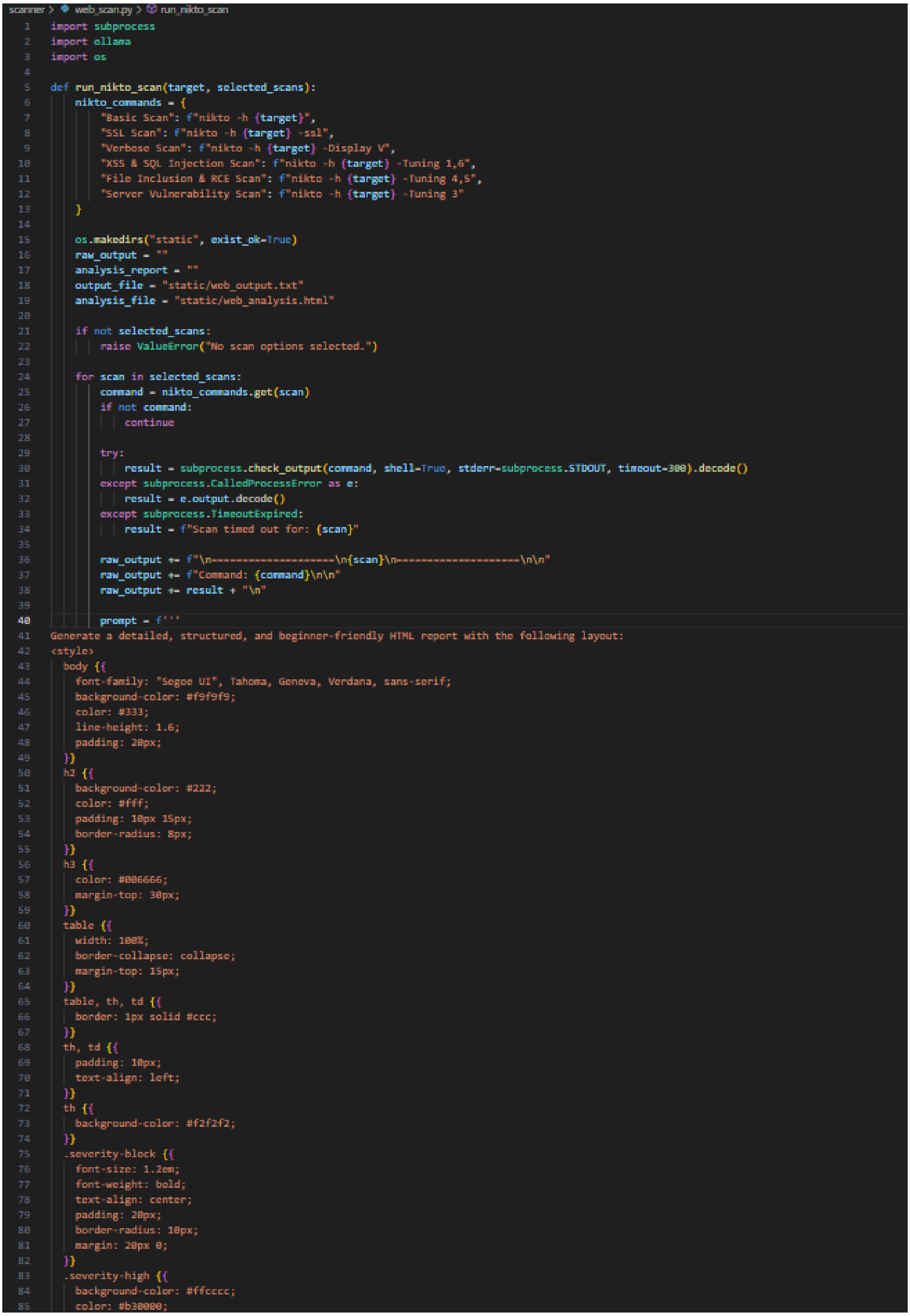
```
import subprocess
import ollama
import os
from flask import Flask, request, render_template

app = Flask(__name__)

def run_nmap_scan(target, selected_scans):
    # Dictionary with scan names as keys and Nmap commands as values
    nmap_commands = {
        "Full Port Scan": f"nmap -p- -sV -O -A {target}",
        "SMB & RDP Vulnerability Scan": f"nmap --script smb-vuln-ms17-010,smb-vuln-ms08-067,smb-enum-shares,smb-enum-users,smb-os-discovery -p 445 {target}",
        "Web & FTP Vulnerability Scan": f"nmap --script http-vuln-cve2017-5638,http-vuln-cve2014-3704,http-vuln-misfortune-cookie -p 80,443 {target}",
        "SMTP & DNS Security Scan": f"nmap --script smtp-vuln-cve2011-1720,ftp-anon,samba-vuln-cve-2012-1182,dns-zone-transfer -p 21,25,53,139,445 {target}",
        "Comprehensive Vulnerability Scan": f"nmap --script vulners -sV {target}"
    }

    os.makedirs("static", exist_ok=True)
    raw_output = ""
    analysis_report = ""
    output_file = "static/net_output.txt"
    analysis_file = "static/net_analysis.html"

    for scan in selected_scans:
        command = nmap_commands.get(scan)
        if not command:
            continue

        try:
            result = subprocess.check_output(command, shell=True, stderr=subprocess.STDOUT, timeout=300).decode()
        except subprocess.CalledProcessError as e:
            result = e.output.decode()
        except subprocess.TimeoutExpired:
            result = f"Scan timed out for: {scan}"

        raw_output += f"\n====================\n{scan}\n====================\n\n"
        raw_output += f"Command: {command}\n\n"
        raw_output += result + "\n"

        prompt = f'''
Generate a detailed, structured, and beginner-friendly HTML report with the following layout:
<style>
  body {{
    font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f9f9f9;
    color: #333;
    line-height: 1.6;
    padding: 20px;
  }}
  h2 {{
    background-color: #222;
    color: #fff;
    padding: 10px 15px;
    border-radius: 8px;
  }}
  h3 {{
    color: #006666;
    margin-top: 30px;
  }}
  table {{
    width: 100%;
    border-collapse: collapse;
    margin-top: 15px;
  }}
  table, th, td {{
    border: 1px solid #ccc;
  }}
  th, td {{
    padding: 10px;
    text-align: left;
  }}
  th {{
    background-color: #f2f2f2;
  }}
  .severity-block {{
    font-size: 1.2em;
    font-weight: bold;
    text-align: center;
    padding: 20px;
    border-radius: 10px;
    margin: 20px 0;
  }}
```

Appendix F: Source Code of net_scan.py (2 of 2)



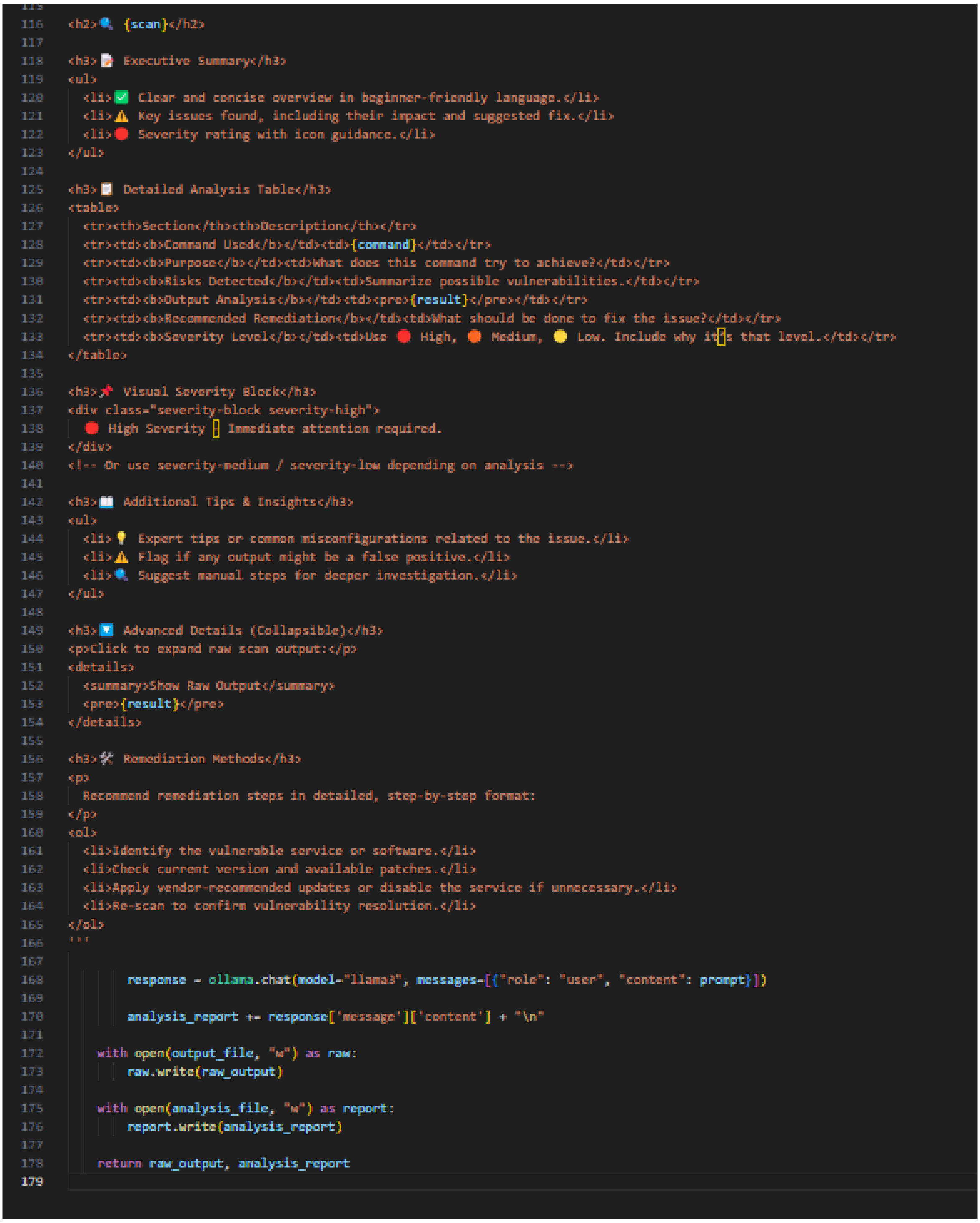
```
<h2>🔍 {scan}</h2>

<h3>📝 Executive Summary</h3>
<ul>
  <li>✅ Clear and concise overview in beginner-friendly language.</li>
  <li>⚠️ Key issues found, including their impact and suggested fix.</li>
  <li>🔴 Severity rating with icon guidance.</li>
</ul>

<h3>📋 Detailed Analysis Table</h3>
<table>
  <tr><th>Section</th><th>Description</th></tr>
  <tr><td><b>Command Used</b></td><td>{command}</td></tr>
  <tr><td><b>Purpose</b></td><td>What does this command try to achieve?</td></tr>
  <tr><td><b>Risks Detected</b></td><td>Summarize possible vulnerabilities.</td></tr>
  <tr><td><b>Output Analysis</b></td><td><pre>{result}</pre></td></tr>
  <tr><td><b>Recommended Remediation</b></td><td>What should be done to fix the issue?</td></tr>
  <tr><td><b>Severity Level</b></td><td>Use 🔴 High, 🟠 Medium, 🟡 Low. Include why it’s that level.</td></tr>
</table>

<h3>📌 Visual Severity Block</h3>
<div class="severity-block severity-high">
  🔴 High Severity — Immediate attention required.
</div>
<!-- Or use severity-medium / severity-low depending on analysis -->

<h3>📖 Additional Tips & Insights</h3>
<ul>
  <li>💡 Expert tips or common misconfigurations related to the issue.</li>
  <li>⚠️ Flag if any output might be a false positive.</li>
  <li>🔍 Suggest manual steps for deeper investigation.</li>
</ul>

<h3>🔽 Advanced Details (Collapsible)</h3>
<p>Click to expand raw scan output:</p>
<details>
  <summary>Show Raw Output</summary>
  <pre>{result}</pre>
</details>

<h3>🛠️ Remediation Methods</h3>
<p>
  Recommend remediation steps in detailed, step-by-step format:
</p>
<ol>
  <li>Identify the vulnerable service or software.</li>
  <li>Check current version and available patches.</li>
  <li>Apply vendor-recommended updates or disable the service if unnecessary.</li>
  <li>Re-scan to confirm vulnerability resolution.</li>
</ol>
'''

            response = ollama.chat(model="llama3", messages=[{"role": "user", "content": prompt}])

            analysis_report += response['message']['content'] + "\n"

    with open(output_file, "w") as raw:
        raw.write(raw_output)

    with open(analysis_file, "w") as report:
        report.write(analysis_report)

    return raw_output, analysis_report

@app.route("/", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        target = request.form.get("target")
        selected_scans = request.form.getlist("scan_options")

        if not target or not selected_scans:
            return render_template("net.html", error="Please provide a target and select at least one scan option.")

        raw_output, analysis_report = run_nmap_scan(target, selected_scans)

        return render_template("net.html", scanning=True, show_complete=True)

    return render_template("net.html")

if __name__ == "__main__":
    app.run(debug=True)
```

Appendix G: Source Code of web_scan.py (1 of 2)



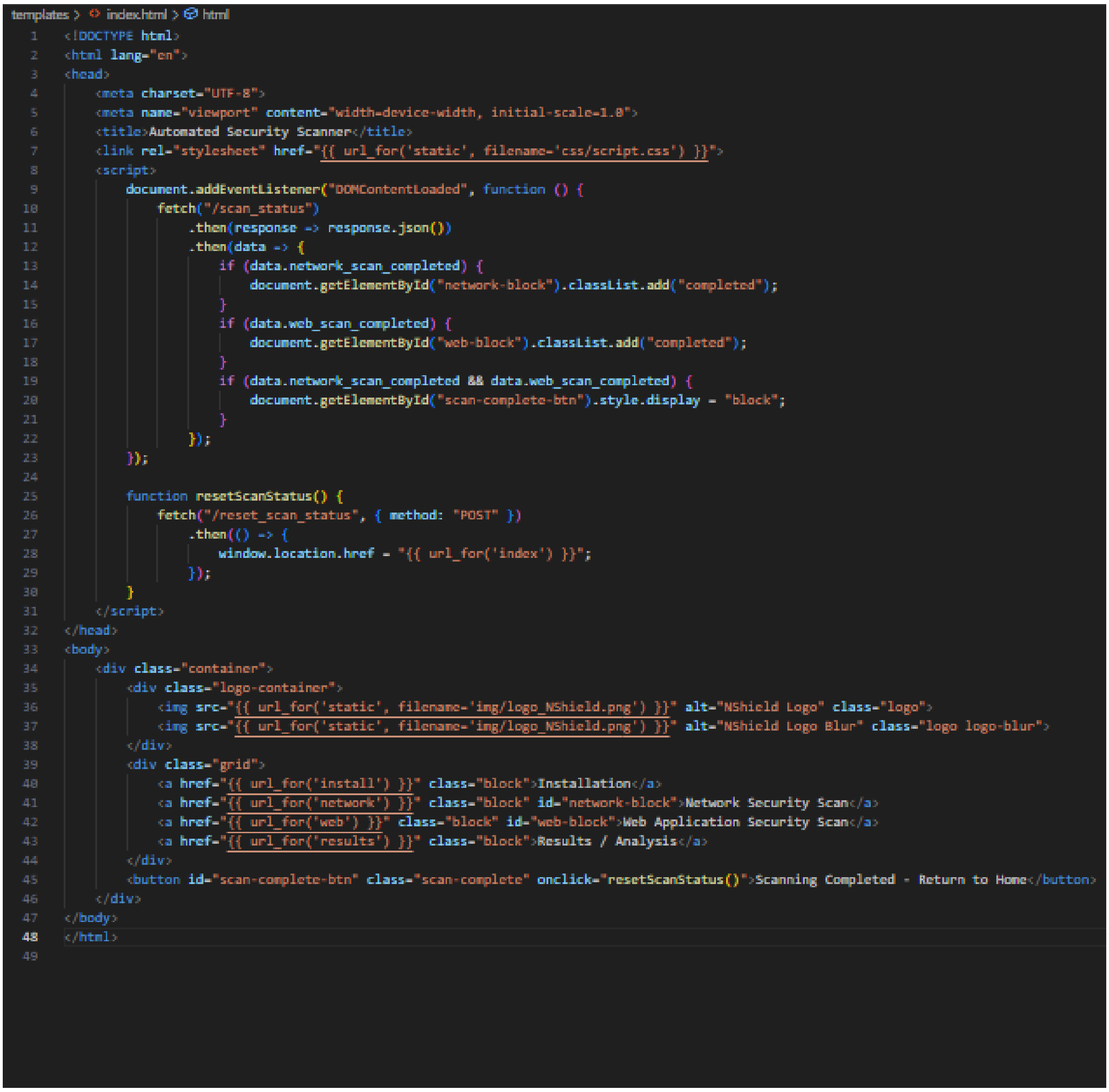
```python
import subprocess
import ollama
import os

def run_nikto_scan(target, selected_scans):
    nikto_commands = {
        "Basic Scan": f"nikto -h {target}",
        "SSL Scan": f"nikto -h {target} -ssl",
        "Verbose Scan": f"nikto -h {target} -Display V",
        "XSS & SQL Injection Scan": f"nikto -h {target} -Tuning 1,6",
        "File Inclusion & RCE Scan": f"nikto -h {target} -Tuning 4,5",
        "Server Vulnerability Scan": f"nikto -h {target} -Tuning 3"
    }

    os.makedirs("static", exist_ok=True)
    raw_output = ""
    analysis_report = ""
    output_file = "static/web_output.txt"
    analysis_file = "static/web_analysis.html"

    if not selected_scans:
        raise ValueError("No scan options selected.")

    for scan in selected_scans:
        command = nikto_commands.get(scan)
        if not command:
            continue

        try:
            result = subprocess.check_output(command, shell=True, stderr=subprocess.STDOUT, timeout=300).decode()
        except subprocess.CalledProcessError as e:
            result = e.output.decode()
        except subprocess.TimeoutExpired:
            result = f"Scan timed out for: {scan}"

        raw_output += f"\n--------------------\n{scan}\n--------------------\n\n"
        raw_output += f"Command: {command}\n\n"
        raw_output += result + "\n"

        prompt = f'''
Generate a detailed, structured, and beginner-friendly HTML report with the following layout:
<style>
  body {{
    font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f9f9f9;
    color: #333;
    line-height: 1.6;
    padding: 20px;
  }}
  h2 {{
    background-color: #222;
    color: #fff;
    padding: 10px 15px;
    border-radius: 8px;
  }}
  h3 {{
    color: #006666;
    margin-top: 30px;
  }}
  table {{
    width: 100%;
    border-collapse: collapse;
    margin-top: 15px;
  }}
  table, th, td {{
    border: 1px solid #ccc;
  }}
  th, td {{
    padding: 10px;
    text-align: left;
  }}
  th {{
    background-color: #f2f2f2;
  }}
  .severity-block {{
    font-size: 1.2em;
    font-weight: bold;
    text-align: center;
    padding: 20px;
    border-radius: 10px;
    margin: 20px 0;
  }}
  .severity-high {{
    background-color: #ffcccc;
    color: #b30000;
```

Appendix G: Source Code of web_scan.py (2 of 2)



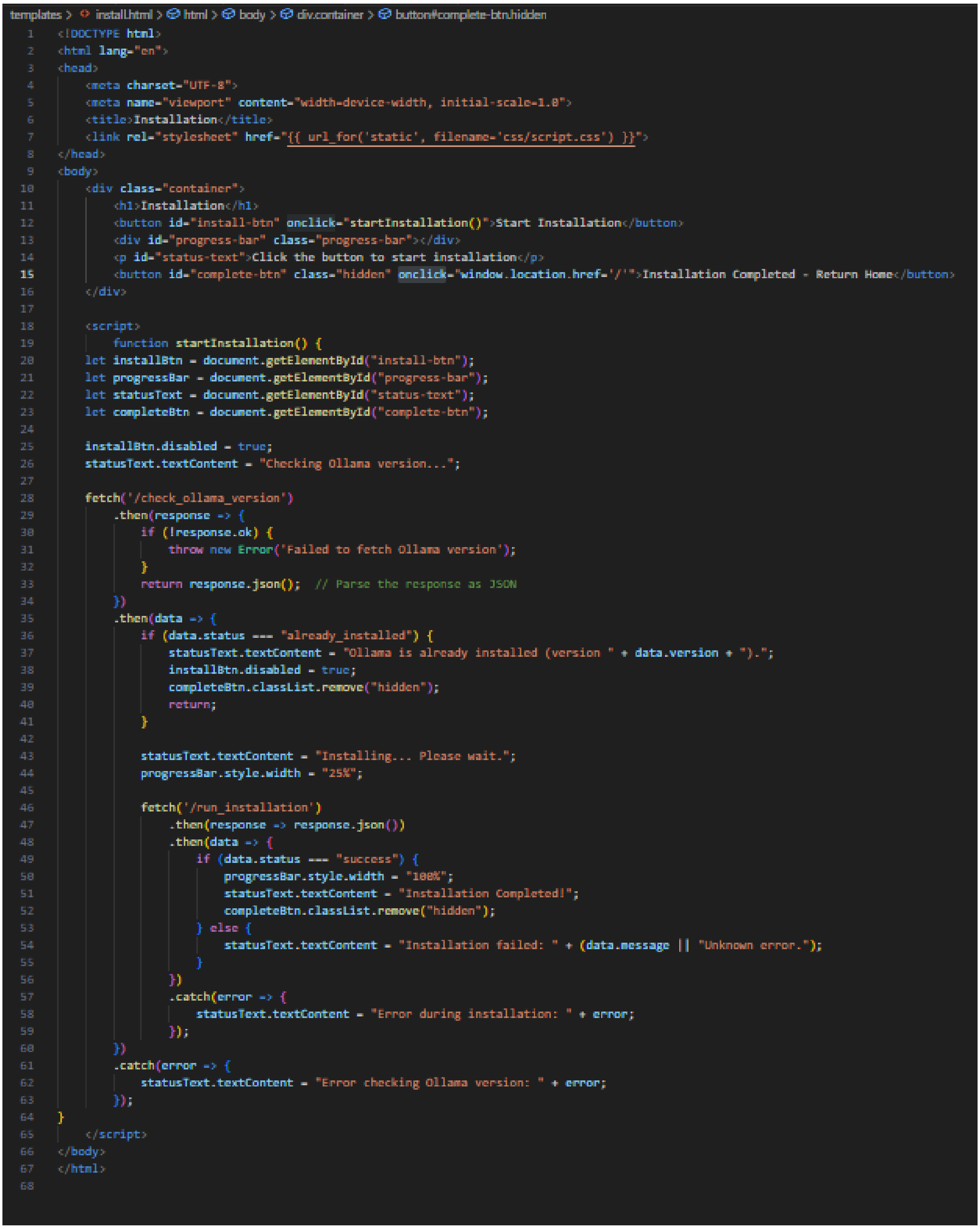
```
<h2>🔍 {scan}</h2>

<h3>📝 Executive Summary</h3>
<ul>
  <li>✅ Clear and concise overview in beginner-friendly language.</li>
  <li>⚠ Key issues found, including their impact and suggested fix.</li>
  <li>🔴 Severity rating with icon guidance.</li>
</ul>

<h3>📋 Detailed Analysis Table</h3>
<table>
  <tr><th>Section</th><th>Description</th></tr>
  <tr><td><b>Command Used</b></td><td>{command}</td></tr>
  <tr><td><b>Purpose</b></td><td>What does this command try to achieve?</td></tr>
  <tr><td><b>Risks Detected</b></td><td>Summarize possible vulnerabilities.</td></tr>
  <tr><td><b>Output Analysis</b></td><td><pre>{result}</pre></td></tr>
  <tr><td><b>Recommended Remediation</b></td><td>What should be done to fix the issue?</td></tr>
  <tr><td><b>Severity Level</b></td><td>Use 🔴 High, 🟠 Medium, 🟡 Low. Include why it▯s that level.</td></tr>
</table>

<h3>📌 Visual Severity Block</h3>
<div class="severity-block severity-high">
  🔴 High Severity ▯ Immediate attention required.
</div>
<!-- Or use severity-medium / severity-low depending on analysis -->

<h3>📖 Additional Tips & Insights</h3>
<ul>
  <li>💡 Expert tips or common misconfigurations related to the issue.</li>
  <li>⚠ Flag if any output might be a false positive.</li>
  <li>🔍 Suggest manual steps for deeper investigation.</li>
</ul>

<h3>🔽 Advanced Details (Collapsible)</h3>
<p>Click to expand raw scan output:</p>
<details>
  <summary>Show Raw Output</summary>
  <pre>{result}</pre>
</details>

<h3>🛠 Remediation Methods</h3>
<p>
  Recommend remediation steps in detailed, step-by-step format:
</p>
<ol>
  <li>Identify the vulnerable service or software.</li>
  <li>Check current version and available patches.</li>
  <li>Apply vendor-recommended updates or disable the service if unnecessary.</li>
  <li>Re-scan to confirm vulnerability resolution.</li>
</ol>
'''

        response = ollama.chat(model="llama3", messages=[{"role": "user", "content": prompt}])

        analysis_report += response['message']['content'] + "\n"

    with open(output_file, "w") as raw:
        raw.write(raw_output)

    with open(analysis_file, "w") as report:
        report.write(analysis_report)

    return raw_output, analysis_report

```

Appendix H: Source Code of index.html



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Automated Security Scanner</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/script.css') }}">
    <script>
        document.addEventListener("DOMContentLoaded", function () {
            fetch("/scan_status")
                .then(response => response.json())
                .then(data => {
                    if (data.network_scan_completed) {
                        document.getElementById("network-block").classList.add("completed");
                    }
                    if (data.web_scan_completed) {
                        document.getElementById("web-block").classList.add("completed");
                    }
                    if (data.network_scan_completed && data.web_scan_completed) {
                        document.getElementById("scan-complete-btn").style.display = "block";
                    }
                });
        });

        function resetScanStatus() {
            fetch("/reset_scan_status", { method: "POST" })
                .then(() => {
                    window.location.href = "{{ url_for('index') }}";
                });
        }
    </script>
</head>
<body>
    <div class="container">
        <div class="logo-container">
            <img src="{{ url_for('static', filename='img/logo_NShield.png') }}" alt="NShield Logo" class="logo">
            <img src="{{ url_for('static', filename='img/logo_NShield.png') }}" alt="NShield Logo Blur" class="logo logo-blur">
        </div>
        <div class="grid">
            <a href="{{ url_for('install') }}" class="block">Installation</a>
            <a href="{{ url_for('network') }}" class="block" id="network-block">Network Security Scan</a>
            <a href="{{ url_for('web') }}" class="block" id="web-block">Web Application Security Scan</a>
            <a href="{{ url_for('results') }}" class="block">Results / Analysis</a>
        </div>
        <button id="scan-complete-btn" class="scan-complete" onclick="resetScanStatus()">Scanning Completed - Return to Home</button>
    </div>
</body>
</html>

```

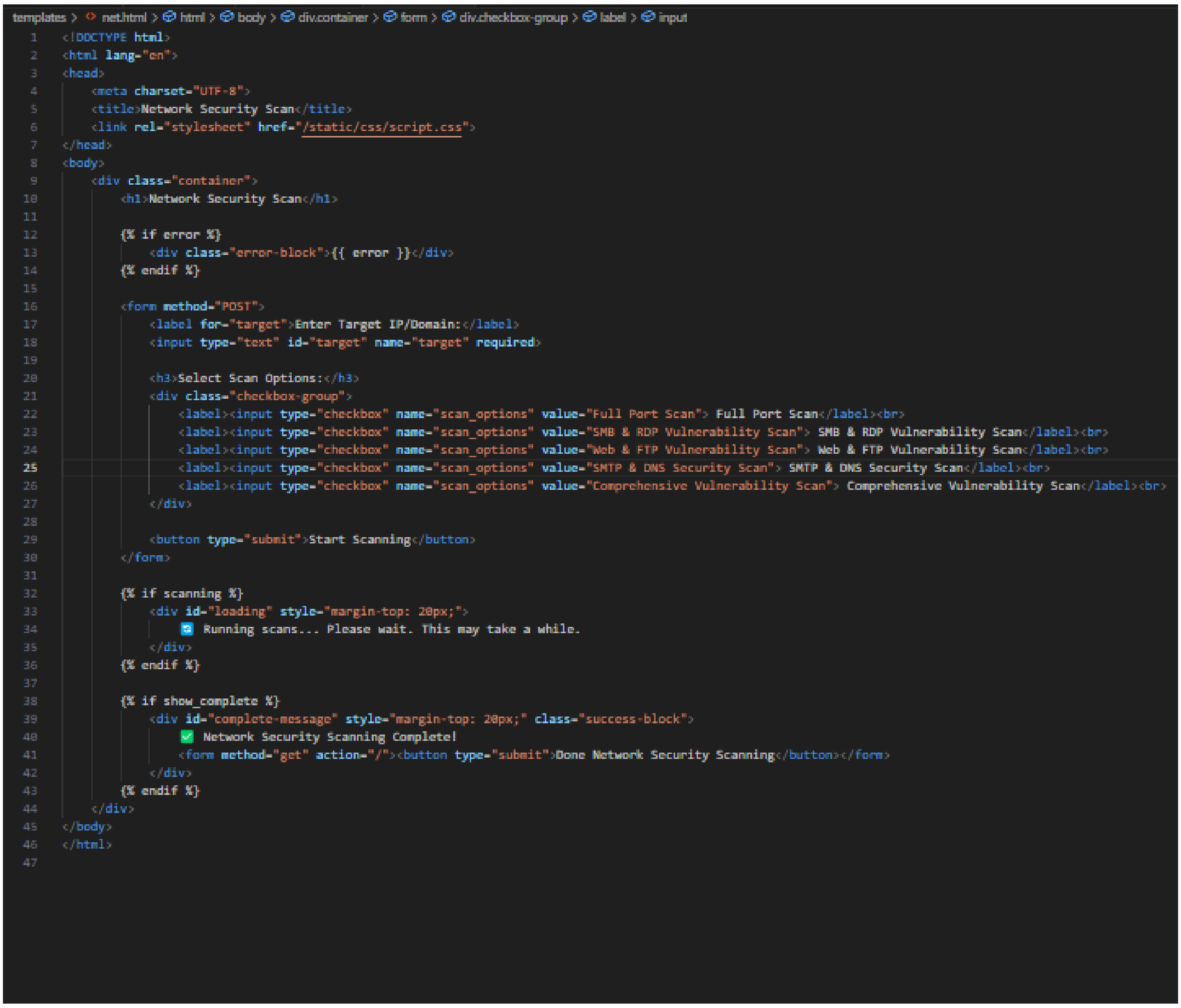
Appendix I: Source Code of install.html



templates > install.html > html > body > div.container > button#complete-btn.hidden

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Installation</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/script.css') }}">
</head>
<body>
    <div class="container">
        <h1>Installation</h1>
        <button id="install-btn" onclick="startInstallation()">Start Installation</button>
        <div id="progress-bar" class="progress-bar"></div>
        <p id="status-text">Click the button to start installation</p>
        <button id="complete-btn" class="hidden" onclick="window.location.href='/'">Installation Completed - Return Home</button>
    </div>

    <script>
        function startInstallation() {
    let installBtn = document.getElementById("install-btn");
    let progressBar = document.getElementById("progress-bar");
    let statusText = document.getElementById("status-text");
    let completeBtn = document.getElementById("complete-btn");

    installBtn.disabled = true;
    statusText.textContent = "Checking Ollama version...";

    fetch('/check_ollama_version')
        .then(response => {
            if (!response.ok) {
                throw new Error('Failed to fetch Ollama version');
            }
            return response.json();  // Parse the response as JSON
        })
        .then(data => {
            if (data.status === "already_installed") {
                statusText.textContent = "Ollama is already installed (version " + data.version + ").";
                installBtn.disabled = true;
                completeBtn.classList.remove("hidden");
                return;
            }

            statusText.textContent = "Installing... Please wait.";
            progressBar.style.width = "25%";

            fetch('/run_installation')
                .then(response => response.json())
                .then(data => {
                    if (data.status === "success") {
                        progressBar.style.width = "100%";
                        statusText.textContent = "Installation Completed!";
                        completeBtn.classList.remove("hidden");
                    } else {
                        statusText.textContent = "Installation failed: " + (data.message || "Unknown error.");
                    }
                })
                .catch(error => {
                    statusText.textContent = "Error during installation: " + error;
                });
        })
        .catch(error => {
            statusText.textContent = "Error checking Ollama version: " + error;
        });
}
    </script>
</body>
</html>

```

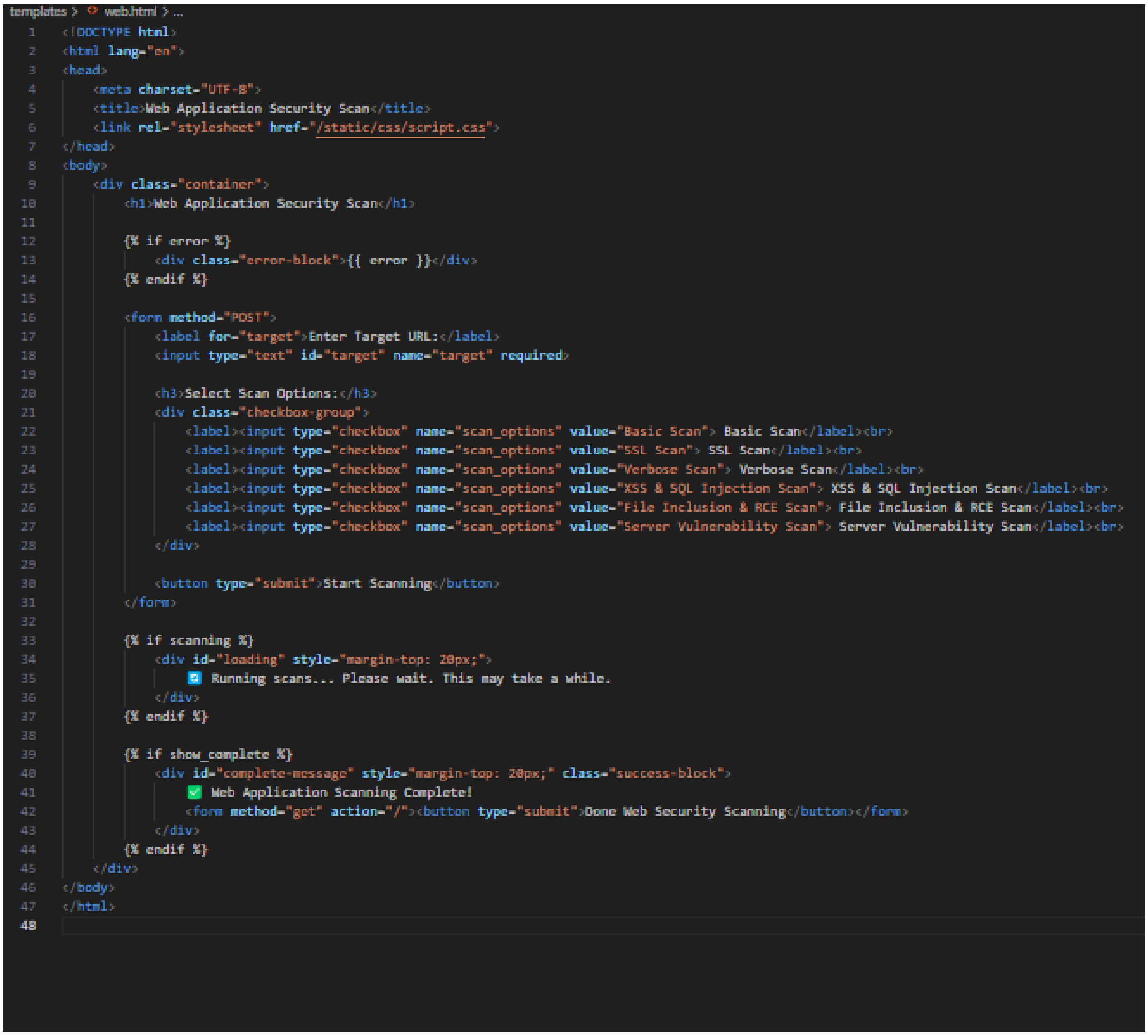
Appendix J: Source Code of net.html



templates > net.html > html > body > div.container > form > div.checkbox-group > label > input

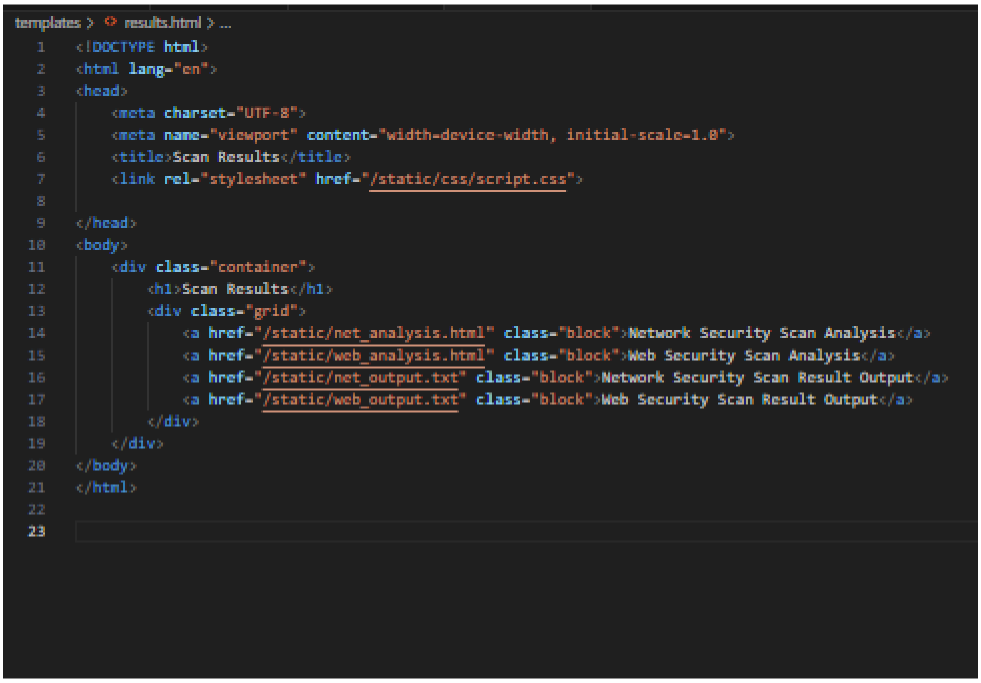
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Network Security Scan</title>
    <link rel="stylesheet" href="/static/css/script.css">
</head>
<body>
    <div class="container">
        <h1>Network Security Scan</h1>

        {% if error %}
            <div class="error-block">{{ error }}</div>
        {% endif %}

        <form method="POST">
            <label for="target">Enter Target IP/Domain:</label>
            <input type="text" id="target" name="target" required>

            <h3>Select Scan Options:</h3>
            <div class="checkbox-group">
                <label><input type="checkbox" name="scan_options" value="Full Port Scan"> Full Port Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="SMB & RDP Vulnerability Scan"> SMB & RDP Vulnerability Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="Web & FTP Vulnerability Scan"> Web & FTP Vulnerability Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="SMTP & DNS Security Scan"> SMTP & DNS Security Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="Comprehensive Vulnerability Scan"> Comprehensive Vulnerability Scan</label><br>
            </div>

            <button type="submit">Start Scanning</button>
        </form>

        {% if scanning %}
            <div id="loading" style="margin-top: 20px;">
                🔄 Running scans... Please wait. This may take a while.
            </div>
        {% endif %}

        {% if show_complete %}
            <div id="complete-message" style="margin-top: 20px;" class="success-block">
                ✅ Network Security Scanning Complete!
                <form method="get" action="/"><button type="submit">Done Network Security Scanning</button></form>
            </div>
        {% endif %}
    </div>
</body>
</html>
```

Appendix K: Source Code of web.html



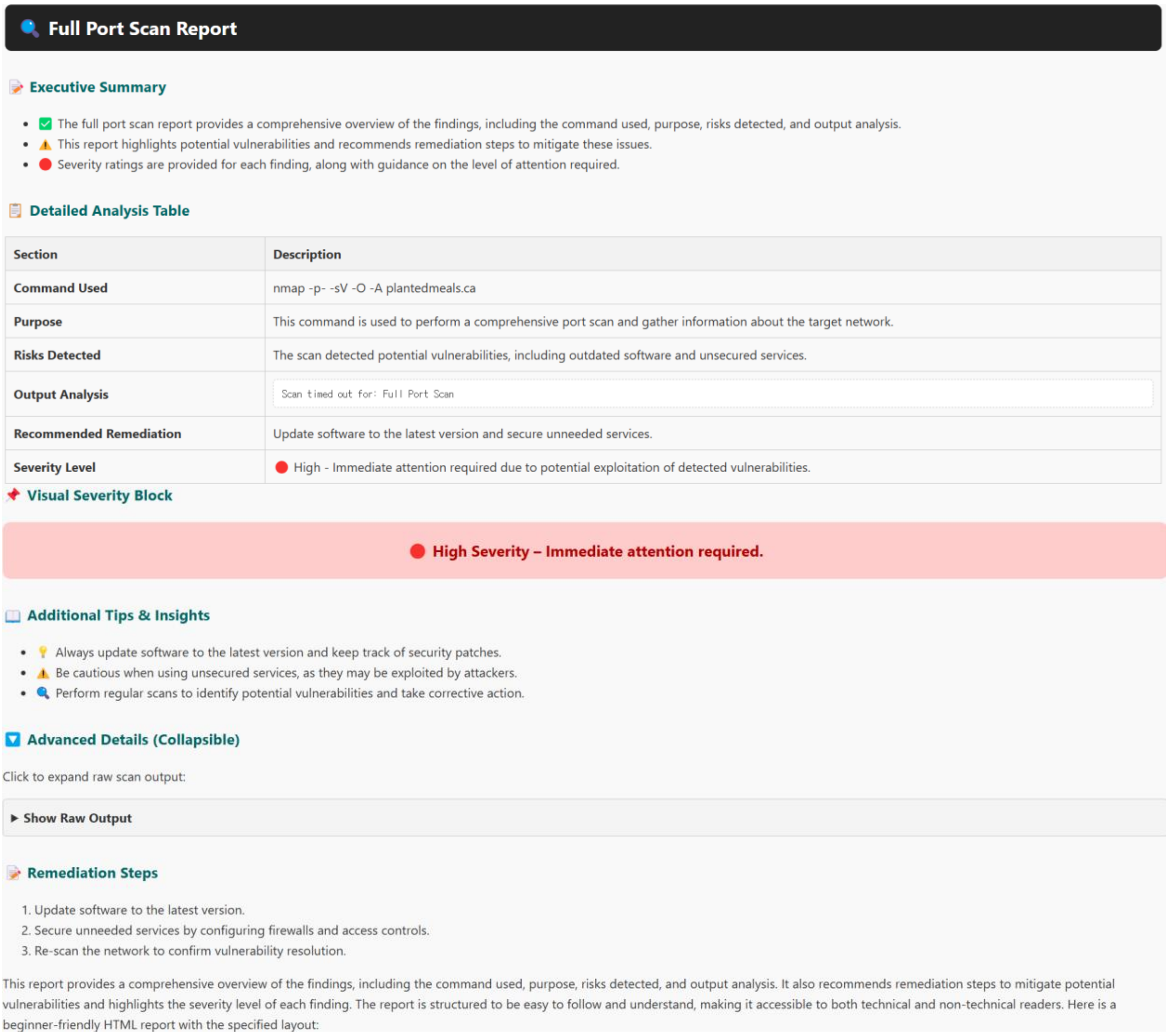
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Application Security Scan</title>
    <link rel="stylesheet" href="/static/css/script.css">
</head>
<body>
    <div class="container">
        <h1>Web Application Security Scan</h1>

        {% if error %}
            <div class="error-block">{{ error }}</div>
        {% endif %}

        <form method="POST">
            <label for="target">Enter Target URL:</label>
            <input type="text" id="target" name="target" required>

            <h3>Select Scan Options:</h3>
            <div class="checkbox-group">
                <label><input type="checkbox" name="scan_options" value="Basic Scan"> Basic Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="SSL Scan"> SSL Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="Verbose Scan"> Verbose Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="XSS & SQL Injection Scan"> XSS & SQL Injection Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="File Inclusion & RCE Scan"> File Inclusion & RCE Scan</label><br>
                <label><input type="checkbox" name="scan_options" value="Server Vulnerability Scan"> Server Vulnerability Scan</label><br>
            </div>

            <button type="submit">Start Scanning</button>
        </form>

        {% if scanning %}
            <div id="loading" style="margin-top: 20px;">
                🔄 Running scans... Please wait. This may take a while.
            </div>
        {% endif %}

        {% if show_complete %}
            <div id="complete-message" style="margin-top: 20px;" class="success-block">
                ✅ Web Application Scanning Complete!
                <form method="get" action="/"><button type="submit">Done Web Security Scanning</button></form>
            </div>
        {% endif %}
    </div>
</body>
</html>

```

Appendix L: Source Code of results.html



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scan Results</title>
    <link rel="stylesheet" href="/static/css/script.css">

</head>
<body>
    <div class="container">
        <h1>Scan Results</h1>
        <div class="grid">
            <a href="/static/net_analysis.html" class="block">Network Security Scan Analysis</a>
            <a href="/static/web_analysis.html" class="block">Web Security Scan Analysis</a>
            <a href="/static/net_output.txt" class="block">Network Security Scan Result Output</a>
            <a href="/static/web_output.txt" class="block">Web Security Scan Result Output</a>
        </div>
    </div>
</body>
</html>

```

Appendix M: Report for Network scan



🔍 Full Port Scan Report

📝 Executive Summary

- ✅ The full port scan report provides a comprehensive overview of the findings, including the command used, purpose, risks detected, and output analysis.
- ⚠️ This report highlights potential vulnerabilities and recommends remediation steps to mitigate these issues.
- 🔴 Severity ratings are provided for each finding, along with guidance on the level of attention required.

📋 Detailed Analysis Table

Section	Description
Command Used	nmap -p- -sV -O -A plantedmeals.ca
Purpose	This command is used to perform a comprehensive port scan and gather information about the target network.
Risks Detected	The scan detected potential vulnerabilities, including outdated software and unsecured services.
Output Analysis	`Scan timed out for: Full Port Scan`
Recommended Remediation	Update software to the latest version and secure unneeded services.
Severity Level	🔴 High - Immediate attention required due to potential exploitation of detected vulnerabilities.

📌 Visual Severity Block

🔴 **High Severity – Immediate attention required.**

📖 Additional Tips & Insights

- 💡 Always update software to the latest version and keep track of security patches.
- ⚠️ Be cautious when using unsecured services, as they may be exploited by attackers.
- 🔍 Perform regular scans to identify potential vulnerabilities and take corrective action.

🔽 Advanced Details (Collapsible)

Click to expand raw scan output:

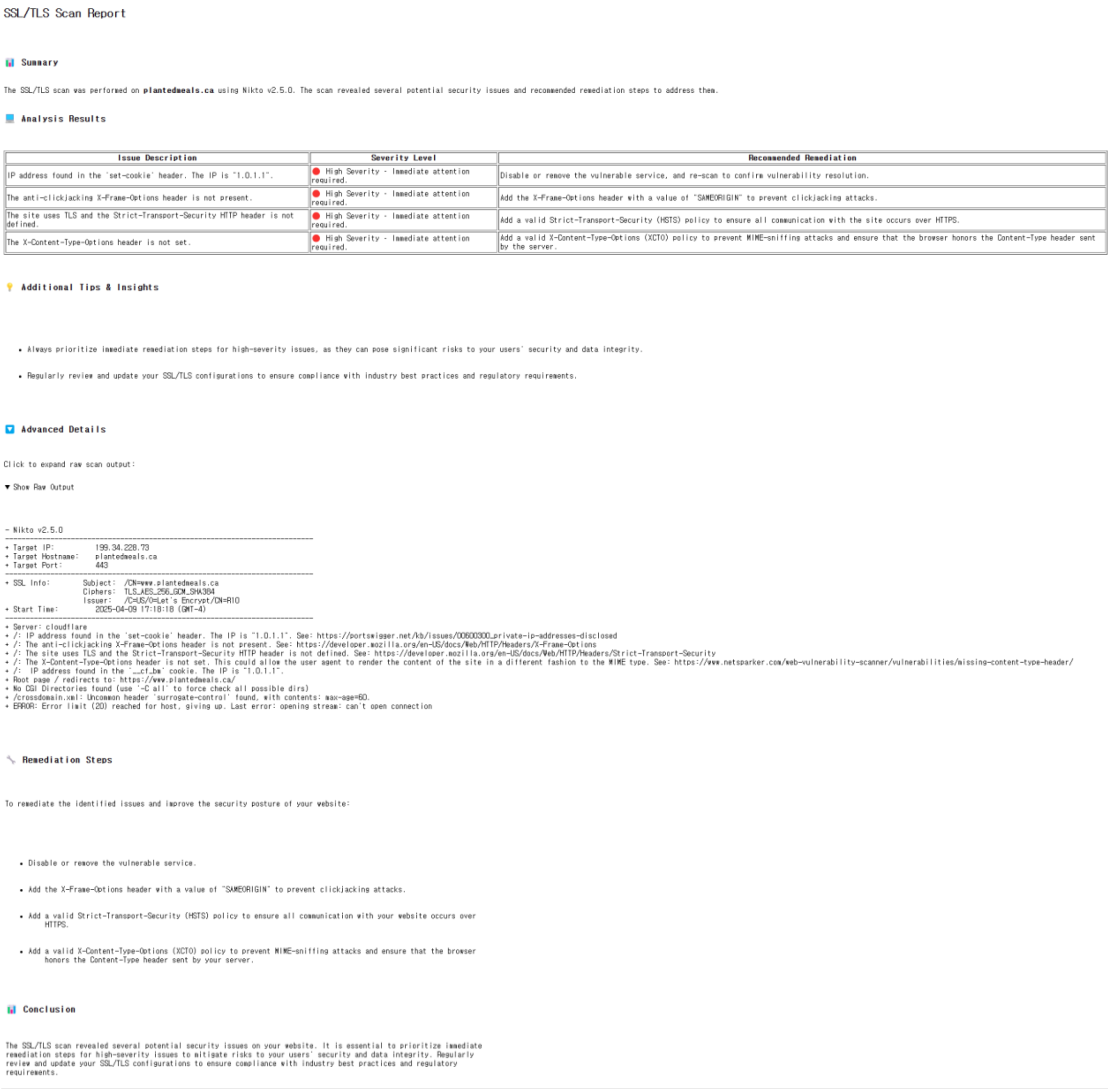
▶ **Show Raw Output**

📝 Remediation Steps

1. Update software to the latest version.
2. Secure unneeded services by configuring firewalls and access controls.
3. Re-scan the network to confirm vulnerability resolution.

This report provides a comprehensive overview of the findings, including the command used, purpose, risks detected, and output analysis. It also recommends remediation steps to mitigate potential vulnerabilities and highlights the severity level of each finding. The report is structured to be easy to follow and understand, making it accessible to both technical and non-technical readers. Here is a beginner-friendly HTML report with the specified layout:

Appendix N: Report for Web Application scan



SSL/TLS Scan Report

Summary

The SSL/TLS scan was performed on **plantedmeals.ca** using Nikto v2.5.0. The scan revealed several potential security issues and recommended remediation steps to address them.

Analysis Results

Issue Description	Severity Level	Recommended Remediation
IP address found in the `set-cookie` header. The IP is "1.0.1.1".	● High Severity - Immediate attention required.	Disable or remove the vulnerable service, and re-scan to confirm vulnerability resolution.
The anti-clickjacking X-Frame-Options header is not present.	● High Severity - Immediate attention required.	Add the X-Frame-Options header with a value of "SAMEORIGIN" to prevent clickjacking attacks.
The site uses TLS and the Strict-Transport-Security HTTP header is not defined.	● High Severity - Immediate attention required.	Add a valid Strict-Transport-Security (HSTS) policy to ensure all communication with the site occurs over HTTPS.
The X-Content-Type-Options header is not set.	● High Severity - Immediate attention required.	Add a valid X-Content-Type-Options (XCTO) policy to prevent MIME-sniffing attacks and ensure that the browser honors the Content-Type header sent by the server.

Additional Tips & Insights

- Always prioritize immediate remediation steps for high-severity issues, as they can pose significant risks to your users' security and data integrity.
- Regularly review and update your SSL/TLS configurations to ensure compliance with industry best practices and regulatory requirements.

Advanced Details

Click to expand raw scan output:

▼ Show Raw Output

```
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          199.34.228.73
+ Target Hostname:    plantedmeals.ca
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=www.plantedmeals.ca
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/O=Let's Encrypt/CN=R10
+ Start Time:         2025-04-09 17:18:18 (GMT-4)
---------------------------------------------------------------------------
+ Server: cloudflare
+ /: IP address found in the 'set-cookie' header. The IP is "1.0.1.1". See: https://portswigger.net/kb/issues/00600300_private-ip-addresses-disclosed
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The site uses TLS and the Strict-Transport-Security HTTP header is not defined. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /:  IP address found in the '__cf_bm' cookie. The IP is "1.0.1.1".
+ Root page / redirects to: https://www.plantedmeals.ca/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /crossdomain.xml: Uncommon header 'surrogate-control' found, with contents: max-age=60.
+ ERROR: Error limit (20) reached for host, giving up. Last error: opening stream: can't open connection
```

Remediation Steps

To remediate the identified issues and improve the security posture of your website:

- Disable or remove the vulnerable service.
- Add the X-Frame-Options header with a value of "SAMEORIGIN" to prevent clickjacking attacks.
- Add a valid Strict-Transport-Security (HSTS) policy to ensure all communication with your website occurs over HTTPS.
- Add a valid X-Content-Type-Options (XCTO) policy to prevent MIME-sniffing attacks and ensure that the browser honors the Content-Type header sent by your server.

Conclusion

The SSL/TLS scan revealed several potential security issues on your website. It is essential to prioritize immediate remediation steps for high-severity issues to mitigate risks to your users' security and data integrity. Regularly review and update your SSL/TLS configurations to ensure compliance with industry best practices and regulatory requirements.