Magna Scientia Advanced Research and Reviews

eISSN: 2582-9394

Cross Ref DOI: 10.30574/msarr

Journal homepage: https://magnascientiapub.com/journals/msarr/

(RESEARCH ARTICLE)

Check for updates

AI-driven advanced python pipeline for windows operating systems vulnerability detection and exploitation

Teslim Aminu *, Ekene Adim, Abdulaziz Ibiyeye and Smart Idima

Department of Computer Science, Western Illinois University, Macomb, Illinois, USA.

Magna Scientia Advanced Research and Reviews, 2024, 12(02), 448-467

Publication history: Received on 03 November 2024; revised on 21 December 2024; accepted on 28 December 2024

Article DOI: https://doi.org/10.30574/msarr.2024.12.2.0222

Abstract

The complexity of cyber threats on Windows operating systems grows exponentially, conventional vulnerability management solutions have failed to keep up with the pace.

The aim of this paper is to describe the creation and execution of an Artificial Intelligence-driven, automated vulnerability discovery and exploitation pipeline. Built using powerful technological tools like Nmap, Niko, and Metasploit, combined with AI model from OpenAI frameworks and GROQ, the program is using automation of the vulnerability management lifecycle starting from detection (reconnaissance) to patch recommendations. We use the program to build modular architecture with Python-based orchestration, with an interactive website (front-end) interface, and an AI analysis engine that contextualizes hazards and generates attack code dynamically for the back-end. We establish this using a controlled virtualized environment to minimize risk exposure, where the system successfully detects and exploits critical vulnerabilities like Print Nightmare (CVE-2021-34527) and Eternal Blue (CVE-2017-0144).

Using an AI-powered patch suggestions has proven to be very relevant, with an average score of 4.9/5. This strategy shows significant Improvements with detection and exploitation accuracy, while reducing patch deployment time by 35% and human labor by 20%. These findings have highlighted the revolutionary potential of an AI-enhanced vulnerability management that can be adopted for contemporary cybersecurity concerns, while providing its scalability and resiliency approach for safeguarding Windows-centric settings.

Keywords: Cybersecurity; AI-Driven Vulnerability Detection; Windows Operating Systems; Exploit Automation; Patch Management; Eternal Blue; Print Nightmare; Metasploit

1. Introduction

As Windows system cyberattacks increases in frequency and sophistication, businesses are finding it difficult to keep up with traditional methods: according to current research, Windows hosts account for over 70% of all corporate security exposures [1], and most severe vulnerabilities go unfixed for up to 60 days. The number of cyberthreats is increasing by more than 25% annually, and over 40% of security teams struggle to decide which issues should be addressed first, according to Safitra and co. [2]. In response, this study offers a fresh, cohesive approach that combines current advancements in automated exploitation frameworks (like Metasploit) and artificial intelligence models and frameworks (like OpenAI and Grok) with popular scanning tools (like Nmap, MSF, and Niko). When used together, these components provide a single process capable of rapidly identifying vulnerabilities, analyzing them, producing exploit code, and suggesting immediate fixes/patches.

* Corresponding author: Ekene Adim

The combined technology is aimed at enhancing the process of vulnerability management significantly. The program is planned to be tested on multiple Windows hosts to reveal exploitable problems with benchmarks to increase vulnerability detection about 30% quicker than conventional techniques and reduce manual effort by 20%. It not only helps prioritize the most critical issues but also offers clear reports intelligible to technical and non-technical team members, advises remedies/patches, and aims to go beyond just problem listing. This method is aimed at considerably reducing the time systems stay at risk by lowering the average time to deploy patches.

This study tackles the increased difficulties of securing Windows OS as threads to the operating system get more complex and traditional vulnerability management solutions fall behind. This approach entails integrating network scanning, AI-driven analysis, automated exploit

development, and guided patch recommendations. Using the technology stack of AI models and frameworks, the approach addresses the entire vulnerability management cycle, from early identification and assessment to testing and, ultimately, offering suggestions to mitigate threats.

Network mapping is the main application of Nmap, deep scanning is done using Nikto, AI is used to make judgments and analyze reports and discoveries, exploit development is done with Metasploit, and a patch recommendation system is employed. This methodology ensures that vulnerabilities are not only identified but also promptly comprehended and disclosed. In a world where Windows is still one of the most widely used corporate operating systems, this research and development effort emphasizes how important it is to seamlessly integrate exploitation frameworks, automated scanning, and AI-driven analysis. This combined approach holds promise for significantly boosting the overall security posture of Windows-centric settings because it can simultaneously improve detection rates, improve attack feasibility, and speed up repair stages.

2. Literature Review

Technologies for vulnerability management are crucial for identifying and resolving security flaws in information systems [3]. Moreno Ribot claims that because of their effectiveness in identifying vulnerabilities, technological stacks and tools such as Nmap, which was released in 1997; Nikto, which was introduced in 2001; and Metasploit, which started as an open-source project in 2003, have seen substantial use throughout time [4]. Administrators can map network topologies and identify open ports and services that are susceptible to cyberattacks with Nmap's well-known capabilities in network discovery and security audits [5]. Nikto specializes in scanning web servers to find malicious files, outdated server software, and other security flaws.

2.1. It generates comprehensive reports that help protect online apps [6]. A crucial tool for

penetration testing and vulnerability evaluation, Metasploit offers a robust framework for creating, testing, and executing exploit code against target systems [7]. These tools' ability to identify vulnerabilities and adjust to the constantly changing threat landscape is limited by their heavy reliance on signature-based detection techniques, despite their widespread use [8, 9]. Furthermore, it takes a lot of time and expertise to manually set up and analyze scan results, which could lead to missed vulnerabilities or postpone patch efforts [4].

2.2. The inclusion of Artificial Intelligence (AI) into cybersecurity offers a major leap in the

proactive detection and control of vulnerabilities [10]. Recent AI techniques use machine learning algorithms, like as anomaly detection and deep learning, to analyze large datasets and identify trends that may point to security flaws. A recent Gartner analysis, according to Shearstone, estimates that AI-driven cybersecurity solutions would shorten the time necessary to identify breaches by up to 50%, emphasizing its vital role in increasing threat detection capabilities [11]. AI systems are capable of prioritizing vulnerabilities based on their potential effect and exploitability, helping businesses to deploy resources more efficiently and address the most significant threats swiftly [12]. Additionally, AI may provide particular mitigation strategies by evaluating threat information and historical data, accelerating reactions to emerging vulnerabilities. The flexibility of AI models allows for continual learning from new data, hence enhancing their accuracy and efficacy over time [12]. Nonetheless, the use of AI in cybersecurity also poses problems, such as the demand for massive, high-quality datasets for training and the susceptibility to adversarial attacks aimed at tricking AI systems [13].

Patch management is a vital component of preserving the security and integrity of software systems; however, it offers considerable problems for companies [14]. Patches for known vulnerabilities must be quickly discovered, tested, and distributed as part of effective patch management. The 2023 Verizon Data Breach Investigations Report states that 60% of breaches contained vulnerabilities for which fixes were available but not implemented [15], highlighting the

difficulties of maintaining systems' updates. Common strategies include the use of automated patch distribution tools and periodical vulnerability assessments; however, these procedures may be limited by variables such as compatibility concerns, necessary downtime, and the high number of fixes needed for large-scale settings [9]. Concurrently, exploit code production has evolved with frameworks like ExploitDB and Core Impact, which enable the building and testing of exploit scripts [16]. These frameworks allow security professionals to model attacks and evaluate the effectiveness of defenses by partially automating the exploit generation process. Nonetheless, the extra effort needed to create vulnerabilities for unique situations restricts their scalability and reactivity [8]. Combining AI with exploit code generation offers a novel approach that could automate the creation of customized attacks by evaluating system factors and identifying optimal attack routes, increasing the effectiveness and complexity of vulnerability exploitation.

Despite their effectiveness against known threats, typical vulnerability screening methodologies have numerous drawbacks that reduce their usefulness in complicated and dynamic scenarios. Because these solutions frequently rely on databases of known vulnerabilities and predefined signatures, they are less effective against complex or novel attacks that do not follow established patterns. Additionally, the manual procedures required in performing scans, analyzing data, and prioritizing vulnerabilities may lead to delays and human error, possibly generating important loopholes in security defenses [10]. The static structure of these tools also inhibits their capacity to react to the fast-evolving threat environment, where new vulnerabilities and exploitation methods regularly develop [17]. On the other hand, applying AI to vulnerability identification and exploitation processes has the potential to revolutionize. AI-driven systems may automatically learn from fresh data, detect previously unknown weaknesses via pattern recognition, and dynamically alter their scanning tactics to handle evolving threats [1, 18]. This adaptability not only makes vulnerability assessments more thorough and accurate, but it also significantly reduces the amount of time and expertise required to keep an eye on and respond to security issues.

Furthermore, AI can promote real-time threat information sharing and cooperation, enabling enterprises to stay ahead of attackers by exploiting collective insights and powerful predictive analytics [10, 16].

Given that the Windows Operating System is one of the most commonly used platforms, safeguarding its security is critical for both individual users and companies [19]. Future improvements in vulnerability identification and exploitation are increasingly focused on the incorporation of AI to increase security measures [18]. New research is focused on building AI models that can continuously monitor and analyze system activity in real time, identifying anomalous behaviors that could indicate potential attacks. Additionally, AI can enhance the automation of patch management by anticipating the most essential vulnerabilities to fix and speed the distribution process to reduce downtime and interruption [10]. The synergy between AI and conventional security solutions is also being researched to construct hybrid systems that utilize the capabilities of both techniques, delivering stronger and durable defenses against a wide variety of threats. Moreover, the ethical implications of AI-driven exploitation are being thoroughly explored, with attempts to set norms and protections to avoid abuse while fostering responsible innovation. Therefore, through the expansion of AI capabilities and seamless integration into current security infrastructures, the future of Windows OS security is set to become more proactive, intelligent, and adaptable, eventually enabling greater protection against a dynamic threat scenario.

2.3. System Architecture and Design

The main elements and data flow from scanning to AI-based analysis, exploitation, and patch suggestion are clearly depicted in this overview of the vulnerability detection and exploitation pipeline.

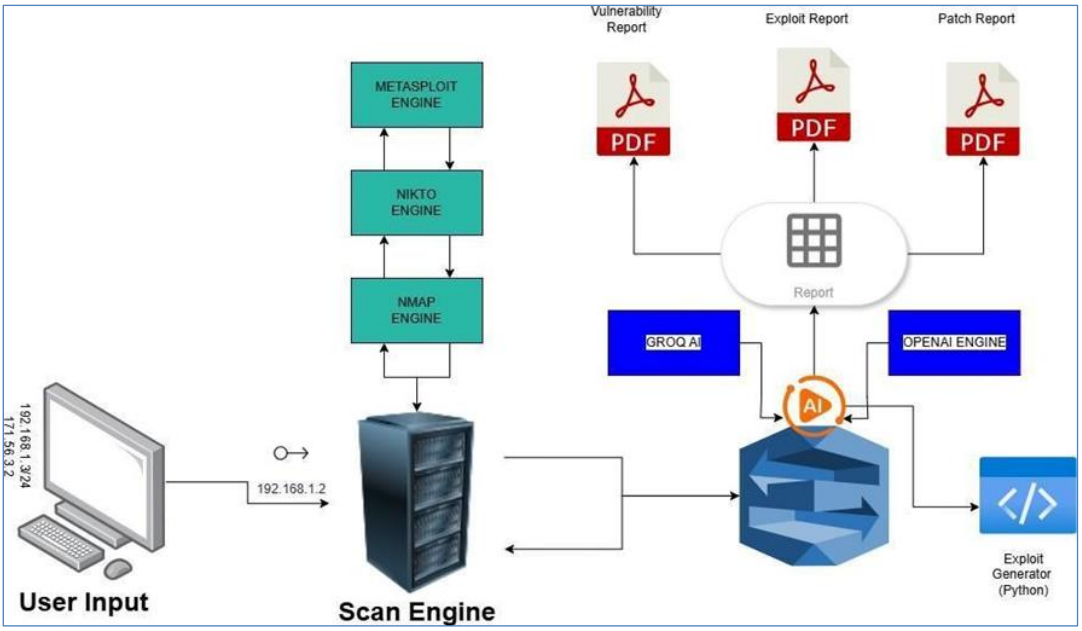

Figure 1 Overall System Flow Diagram (High-Level View)

Additionally, the figure provides a thorough architectural breakdown of the system, demonstrating how Nmap, Niko, Metasploit, AI processing, and reporting methods are deeply integrated for effective vulnerability assessment and mitigation. This serves to emphasize the significance of each element and how they relate to one another.

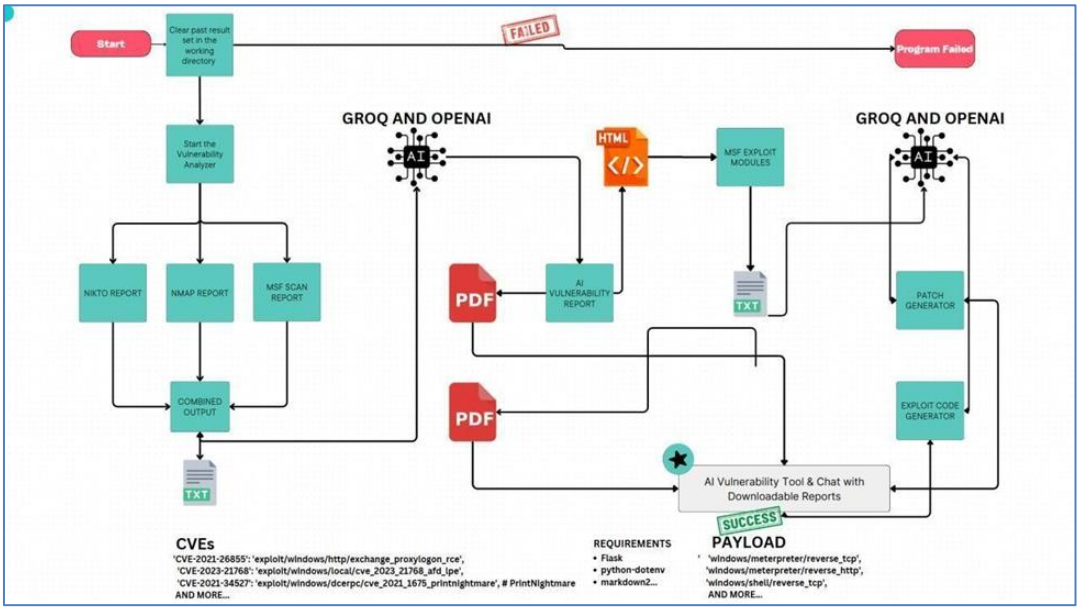

Figure 2 System Architectural Design Block Diagram

The vulnerability analysis and cleaning procedure is shown in the pipeline diagram, which starts with deleting earlier findings and vulnerability scanning. To find open ports, operating systems, and vulnerabilities, the process begins with a remote host scan utilizing Nmap, Nikto, and Metasploit. The scan findings are gathered and fed into AI models for further research, yielding valuable information about possible exploits and vulnerabilities. These discoveries then lead the AI algorithm to propose related exploits and fixes. This ensures that the process moves logically from data collection to actionable conclusions for system security.

Table 1 below outlines the step-by-step process of system execution, detailing the tools involved at each stage, the input they process, and their corresponding outputs, ensuring a structured vulnerability management approach

Table 1 Process Flow and System Integration

Step	Description	Tools/Components Involved	Input	Output
1. Start	Initializes the process by clearing past result sets in the working directory to ensure clean execution.	-	Any residual files or previous results	Clean working directory
2. Vulnerability Analyzer	Initiates the vulnerability scanning process using tools to detect open ports, services, and vulnerabilities.	Nmap, Niko, Metasploit	Target systems (e.g., IP addresses, hostnames)	Individual scan reports: Niko (TXT/PDF), Nmap (TXT/PDF), MSF (TXT/PDF)
3. Combined Output	Aggregates individual scan reports into a single, standardized output file for unified processing.	Python modules for parsing and aggregation	Niko, Nmap, and Metasploit reports	Combined vulnerability report (TXT)
4. AI Analysis	Processes combined data using AI to analyse CVEs, prioritize vulnerabilities, and recommend actions.	Grow, OpenAI, Python libraries (Pandas, NumPy)	Combined output report with CVE identifiers, severity scores, and system details	AI vulnerability report (HTML/PDF) with detailed insights into vulnerabilities and exploits
5. Exploit Module	Uses AI analysis to generate exploit code for vulnerabilities, allowing testing and validation.	Metasploit, an AI-driven exploit generator	CVE details, combined output, and vulnerability severity	Exploit modules/code files (TXT) for identified vulnerabilities
6. Patch Recommendation	Utilizes AI to generate patches or configuration changes to remediate detected vulnerabilities.	AI patch generator	Vulnerability details, severity scores, system configurations	Patch files or recommended configurations (e.g., remediation scripts or steps)
7. Report Generation	Produces final consolidated reports that detail findings, exploit validation, and patch recommendations.	Flask, PDF generation libraries (e.g., PDFKit, Jinja2)	AI vulnerability report, exploit modules, patch recommendations	Comprehensive downloadable reports in PDF and HTML formats
8. Success/Failure Check	Validates if generated exploits and recommendations address vulnerabilities successfully.	AI validation logic, Python orchestration modules	AI analysis results, exploit tests, patch validation	Success or failure status
9. Front-End Interaction	Provides a clear, interactive interface to view findings, reports, and recommendations.	Flask-based web interface, Streamlet	Final reports, success/failure status	User-accessible vulnerability tool with downloadable reports and actionable insights

Nmap and Niko handle the initial scanning, while Metasploit provides a framework for validating and exploiting discovered vulnerabilities. Python modules provide orchestration, parsing, and interaction with the AI components (OpenAI and Grok). AI models enhance vulnerability assessment by analyzing CVE data and providing exploit code and patching procedures. Scan results in TXT are standardized and processed to guarantee compatibility with AI analysis. The AI models analyze CVEs, determine their severity, and provide recommendations by combining automation and result set analysis. The system generates findings, exploitation results, and mitigation suggestions using an easy-to-use interface. Users may use this interface to see comprehensive reports prepared in PDF format. These reports offer a clear and well-organized picture by combining scanning data, AI-driven analysis, and repair instructions. Utilizing downloadable PDF reports and HTML-based summaries enhances usability and accessibility, facilitating more efficient dissemination of findings and suggestions.

2.4. System Design Steps

2.4.1. Remote Host Vulnerability Scanning

The process of vulnerability scanning begins with identifying open ports, active services, and potential weaknesses in the target system. Comprehensive scans are performed using tools which include Nmap and Niko. Nmap does network reconnaissance to discover active ports and services, while Niko detects misconfigurations and vulnerabilities in web applications specifically for Windows Server and outdated software versions. To allow for easy aggregation and analysis, scan results are recorded in structured forms TXT.

```
def scan(target):
    logger.info(f"Initiating Nmap scan for target: {target}")
    command = ['sudo', 'nmap', '-A', '-Pn', '-sS', '-sV', 'sC', '-T4', '-O', '--osscan-guess', '--reason', '-F', '-vvv', target]
    logger.debug(f"Executing Nmap command: {' '.join(command)}")

    result = subprocess.run(command, capture_output=True, text=True)
    if result.returncode != 0:
        logger.error(f"Nmap scan failed: {result.stderr}")
        return None

    logger.info("Nmap scan completed successfully")
    logger.debug(f"Scan output:\n{result.stdout}")
    return result.stdout


if __name__ == "__main__":
    target_ip = os.getenv("TARGET_IP")
    if target_ip:
        logger.info(f"Starting vulnerability scanner for target: {target_ip}")
        scanner(target_ip)
    else:
        logger.error("No target IP provided in environment variables")
        exit(1)
```

Figure 3 Scanning with Nmap and Niko

After running a scan using Nmap services, the function conducts an Nmap scan on a remote computer, looking for open ports and any vulnerabilities.

It takes in the following input

targetoid: The IP address of the target system to be scanned. This is provided as a user input.

The output provided by the program is listed below as a structured text report detailing

- Open ports and associated services.
- Operating system details.

- Potential vulnerabilities linked to detected services.

Nmap is widely used as a network scanner. Nmap enables security experts to find possibly abused exposed services, and by default scans the most frequently used 1,000 ports for each protocol, although it can be enhanced to scan the entire 65,535 ports. After the first scans, Metasploit is deployed to dive further into discovered vulnerabilities. Auxiliary modules examine individual CVEs and determine their exploitability. The outputs contain data on CVEs, severity rankings, and metrics like exploitability likelihood. The findings are pooled to identify vulnerabilities that offer the most substantial risk.

The code snippet below then runs a Metasploit vulnerability scan on the saved target system (IP or hostname) and saves the results to a text file in the current working directory. The function

$run\_vulnerability\_sca$ $(target, output\_file)$ constructs a Metasploit command that runs two auxiliary scans: $portscan/tcp$ $(to\ find\ open\ ports)$ $and\ http\_version$ $(to\ detect\ web\ server\ versions)$. These commands are passed to $msfconsole\ in\ silent\ mode$ $(-q, -x)$, and the scan

results are written to an output file $(metasploit\_results.\ txt)$. The script then calls the function with $IP\_ADDRESS$ as the target.

It takes in the following data inputs

- Targetoid: The target system's IP address.
- Metasploit auxiliary modules specific to detected services (e.g., SMB scanning for Windows).

Then, it provides an output related to the identified vulnerabilities.

The report (Metasploit_scan.txt) includes:

- Identified Common Vulnerabilities and Exposures (CVEs).
- Exploitability metrics (e.g., likelihood of success).

```python
def run_vulnerability_scan(target, output_file):
    logger.info(f"Starting Metasploit vulnerability scan for {target}")
    output_dir = os.path.dirname(output_file)
    if output_dir and not os.path.exists(output_dir):
        os.makedirs(output_dir)
        logger.debug(f"Created output directory: {output_dir}")

    scan_command = (
        f"use auxiliary/scanner/portscan/tcp; set RHOSTS {target}; run; "
        f"use auxiliary/scanner/portscan/udp; set RHOSTS {target}; run; "
        f"use auxiliary/scanner/ftp/ftp_version; set RHOSTS {target}; run; "
        f"exit"
    )
    command = ['msfconsole', '-q', '-x', scan_command]

    try:
        logger.debug("Executing Metasploit commands")
        with open(output_file, 'w') as outfile:
            result = subprocess.run(command, stdout=outfile, stderr=subprocess.STDOUT, text=True)
            if result.returncode == 0:
                logger.info("Metasploit scan completed successfully")
                log_file_content(output_file)
            else:
                logger.error(f"Metasploit scan failed with code: {result.returncode}")
    except Exception as e:
        logger.error(f"Error during Metasploit scan: {e}")
```

Figure 4 Code for Running Metasploit Scans

2.5. Vulnerability Aggregation and Analysis

Scan data from numerous tools (e.g., Nmap, Nikto, Metasploit) are integrated into a uniform framework for simpler analysis. This means scanning TXT/CSV files, extracting critical information such as open ports, services, and CVEs, and integrating them into a cohesive dataset. Once the data is standardized, it is provided to AI models like OpenAI or Grok for contextual analysis. These models discover important vulnerabilities, connect them to CVEs, and offer information such as probable attack pathways and suggested remedial procedures.

- The program, as seen in the snippet of Figure 5, then generates a vulnerability report using
- OpenAI's GPT model and Groq AI. It loads API keys from environment variables
- ($OPENAI\_API\_KEY$ *and* $GROQ\_API\_KEY$) *Groq client*
- and initializes the . The function
- $generate\_vulnerability\_report(data)$

constructs a prompt to analyze scan results and calls OpenAI's text-davinci-003 model to generate a report. The script reads scan data from combined_output.txt, processes it using the AI model, and if a report is generated, it is saved to vulnerability_report.txt. If any error occurs during the AI request, it logs the error and returns None. Overall, this is a key automation step in the process. It removes manual effort by analyzing the severity of vulnerabilities and recommending fixes.

2.5.1. This function analyzes the collected scan data using AI models to

- Identify and prioritize vulnerabilities.
- Suggest exploitation methods.
- Provide patch recommendations based on CVE history.
- The AI analyzer requires inputs, which include:
- scan results: The consolidated results from Nmap, Niko, and Metasploit scans.
- AI models (e.g., OpenAI's GPT or GROQ) trained on cybersecurity databases.

- It provides an AI-generated report (ai_vulnerability_report.txt) that includes:
- Severity assessment (e.g., Critical, High, Medium, Low).
- Exploit feasibility (likelihood of success).
- Suggested patches and mitigations.

```python
import openai
import os
from dotenv import load_dotenv
from groq import Groq

load_dotenv()

# Load OpenAI API key from environment variable
openai_api_key = os.getenv('OPENAI_API_KEY')
if openai_api_key:
    openai.api_key = openai_api_key

MODEL_ENGINE = "text-davinci-003"
TEMPERATURE = 0.5
TOKEN_LIMIT = 2048

# Load Groq API key from environment variable
groq_api_key =""
client = Groq(api_key=groq_api_key)
```

Figure 5 AI Integration Code

```python
def combine_text_files(output_filename):
    logger.info("Starting to combine vulnerability text files")
    try:
        current_directory = os.getcwd()
        txt_files = [f for f in os.listdir(current_directory) if f.endswith('.txt')]

        if not txt_files:
            logger.info("No text files found to combine")
            return

        with open(output_filename, 'w') as outfile:
            for txt_file in txt_files:
                txt_file_path = os.path.join(current_directory, txt_file)
                logger.debug(f"Processing file: {txt_file}")
                log_file_content(txt_file)  # Log content of each file being combined
                with open(txt_file_path, 'r') as infile:
                    outfile.write(infile.read())
                    outfile.write("\n\n")

        logger.info(f"Successfully combined all files into {output_filename}")
        log_file_content(output_filename)  # Log content of final combined file
    except Exception as e:
        logger.error(f"Error combining text files: {e}")
```

Figure 6 Combining Scan Results

2.5.2. Exploit Module Integration

Using the findings from AI-driven vulnerability analysis, the system automatically picks suitable

Metasploit modules to try exploitation of detected vulnerabilities.

The Python dictionary below is named *cve_to_exploit*, and it maps CVE (Common Vulnerabilities and Exposures) identifiers to their corresponding Metasploit exploit modules. Each entry consists of:

- A CVE ID (e.g., 'CVE-2021-26855') – a unique identifier for a known security vulnerability.
- A Metasploit module path (e.g.,
- 'exploit/windows/http/exchange_proxylogon_rce'), which specifies the exploit that can be used in Metasploit to take advantage of the vulnerability.
- A comment describing the vulnerability (e.g., # Microsoft Exchange
- ProxyLogon RCE), giving context about the exploit.

Exploitation is attempted by mapping the identified vulnerabilities to an appropriate Metasploit module from the dictionary. The AI compares CVE information with Metasploit's exploit database, selecting modules with greater success rates.

```
cve_to_exploit = {
    'CVE-2021-26855': 'exploit/windows/http/exchange_proxylogon_rce',  # Microsoft Exchange ProxyLogon RCE
    'CVE-2023-21768': 'exploit/windows/kernel/cve_2023_21768_afd_lpe',  # Ancillary Function Driver (AFD) for WinSock Elevation of Privilege
    'CVE-2021-34527': 'exploit/windows/printnightmare/cve_2021_34527_printnightmare',  # PrintNightmare
    'CVE-2017-0144': 'exploit/windows/smb/ms17_010_eternalblue',  # EternalBlue SMB exploit
    'CVE-2019-0708': 'exploit/windows/rdp/cve_2019_0708_bluekeep_rce',  # BlueKeep
    'CVE-2020-1472': 'auxiliary/admin/dcerpc/cve_2020_1472_zerologon',  # Netlogon Weak Cryptographic Authentication Vulnerability
    'CVE-2020-0688': 'exploit/windows/http/exchange_cve_2020_0688',  # Exchange Control Panel ViewState Deserialization RCE
    'CVE-2016-0189': 'exploit/windows/browser/ms16_051_vbscript',  # Internet Explorer 11 VBScript Engine RCE
    'CVE-2017-8464': 'exploit/windows/local/cve_2017_8464_lnk_lpe',  # LNK Code Execution Vulnerability
    'CVE-2020-0688': 'exploit/windows/http/exchange_cve_2020_0688',  # Exchange Control Panel ViewState Deserialization RCE
    'CVE-2015-1701': 'exploit/windows/local/ms15_051_client_copy_image',  # Windows ClientCopyImage Win32k Exploit
    'CVE-2017-8759': 'exploit/windows/sm/cve_2017_8759_smghost',  # Microsoft .NET Framework RCE in Office
    'CVE-2020-0796': 'exploit/windows/smb/smbghost',  # SMBGhost SMBv3 Vulnerability
    'CVE-2021-31166': 'auxiliary/dos/windows/http_sys_accept_encoding_dos_cve_2021_31166',  # Windows IIS HTTP Protocol Stack DOS
    'CVE-2018-8120': 'exploit/windows/local/ms18_8120_win32k_privsec',  # Win32k Privilege Escalation (UAF)
    'CVE-2017-11882': 'exploit/windows/local/cve_2017_11882',  # Microsoft Equation Editor RCE
    'CVE-2019-1458': 'exploit/windows/local/cve_2019_1458',  # Win32k Privilege Escalation (Use After Free)
    'CVE-2016-0400': 'exploit/windows/smb/smb_relay',  # SMB Relay Attack
    'CVE-2018-7600': 'exploit/unix/webapp/drupal_drupalgeddon2',  # Drupal Remote Code Execution
    'CVE-2017-0145': 'exploit/windows/smb/ms17_010_psexec',  # SMB DOUBLEPULSAR Remote Code Execution
    'CVE-2020-1048': 'exploit/windows/local/cve_2020_1048',  # Microsoft Spooler Local Privilege Escalation
    'CVE-2021-1675': 'exploit/windows/printnightmare/cve_2021_1675_printnightmare',  # PrintNightmare Remote Code Execution
    'CVE-2017-10271': 'exploit/multi/http/oracle_weblogic_wls_wsatt',  # Oracle WebLogic wls-wsat Component RCE
    'CVE-2015-5123': 'exploit/windows/local/ms14_058_trackpopupmenu',  # Windows TrackPopupMenu Win32k NULL Pointer Dereference
    'CVE-2017-0299': 'exploit/windows/browser/office_rtf_rce',  # Microsoft Office Word Malicious RTA Execution
    'CVE-2018-15982': 'exploit/multi/browser/adobe_flash_upstream',  # Adobe Flash opaqueBackground Use After Free
    'CVE-2019-19781': 'exploit/multi/http/citrix_directory_traversal_rce',  # Citrix ADC (NetScaler) Directory Traversal RCE
    'CVE-2018-11776': 'exploit/multi/http/apache_struts_rce',  # Apache Struts RCE Exploit
    'CVE-2021-21972': 'exploit/multi/http/vmware_vcenter_rce',  # VMware vCenter RCE Exploit
    'CVE-2018-15442': 'exploit/windows/browser/webexec',  # Windows WebExec Arbitrary Code Execution
    'CVE-2021-44228': 'exploit/multi/http/vmware_vcenter_log4shell',  # VMware vCenter Server Unauthenticated JNDI Injection (Log4Shell)
}
```

Figure 7 Metasploit's exploit modules

The module in the CVE script automatically selects and runs Metasploit based on the AI vulnerability scan. It takes in inputs such as:

- Cebid: The detected vulnerability ID.
- Targetoid: The system to be tested.
- Exploit modules: A dictionary mapping CVE ids to Metasploit exploits.

2.5.3. Then provide outputs

- A successful or failed exploitation attempt.
- If successful, privilege escalation details in the exploit result in frontend.
- This function automates the attack simulation step by executing the best exploit method.

Exploitation findings, such as successful shell access or file retrieval, are reported for validation. All exploitation attempts are logged and compiled into a thorough PDF report. The report covers the status of each vulnerability, whether it was successfully exploited, and the consequences of the attack. This enables accessibility and gives actionable information for modification. A front- end dashboard is given to highlight current vulnerabilities, exploitation status, and suggested remediations. Users may examine real-time changes, obtain reports, and engage with exploit validation results.

2.5.4. Patch Recommendation and Management

The AI model produces patch suggestions based on vulnerability analysis. References to official vulnerability warnings, configuration changes, or available fixes are included in these recommendations. Actions are ranked by the system according to their exploitability and seriousness. The final PDF report and front-end interface contain patch recommendations. Users can quickly address the issues raised because each vulnerability is linked to the process steps that are relevant to it.

High-risk vulnerabilities, for instance, might include comprehensive guidance on how to apply vendor-released patches or make important system modifications.

2.5.5. Customized Exploit Code Output

Apart from taking advantage of existing frameworks, the system uses AI models to create new exploit scripts for vulnerabilities that are found. These scripts are made to take use of the target's special qualities and are frequently Python-based. To verify the success of the exploits generated, they are run against the vulnerabilities that have been found. Obtaining files, verifying shell access, and completing additional proof-of-concept tasks are examples of validation procedures. If successful, the exploit results are included in the report for further analysis. The AI-generated scripts are designed to be used independently, allowing for manual execution on targets that have not been patched. Situations where Metasploit modules are insufficient or unavailable for a particular vulnerability are made possible by this functionality.

2.5.6. PDF and Front-End Presentation

All scanning findings, exploitation outcomes, and patch proposals are combined into a thorough PDF report. The study is constructed for accessibility by both technical and management audiences, presenting thorough results along with concrete suggestions. The system's web-based interface enables users to interact with vulnerability data, exploitation findings, and patch recommendations in real time. Features include

- Graphical summaries of vulnerabilities by severity.
- Downloadable PDF reports.
- Live updates on scanning progress and exploit execution.

3. Implementation Details

The implementation was carried out in a controlled environment that comprised both hardware and software configurations essential for vulnerability investigation, exploitation, and repair operations. The system comprised a main Ubuntu Linux host running on a Proxmox hypervisor with numerous virtual machines for testing reasons.

The Ubuntu is connected online to update and install all dependencies, then changed to the LAN network to ensure it’s within the same subnet and in a closed network with the other VMs.

Table 2. System Information Table

Table 2 aims to provide the configuration details of the test environment, including operating systems, hardware specifications, installed security tools, and their respective roles in the assessment framework.

Table 2 System Configuration Details

Component	Specification	Configuration Details	Role in the Environment	Installed Tools	OS Version	Notes/Additional Info
Host Machine	Proxmox 8.4	12 cores, 2 Intel Xeon Silver @2.2GHz, RAM: 192GB, Storage: 15TB NVMe SSD	Primary workstation hosting the virtualized environment	Proxmox	Debian Based	Hosts the Ubuntu VM and Windows VMs. Manages Proxmox networking via pfSense
VM 1	Ubuntu Linux 24.04 LTS	8 Cores, 8GB RAM, 40GB Storage	Performs vulnerability scanning and report generation using security tools	Nmap, Nikto, Metasploit, Python 3.12, OpenAI/Groq APIs	Ubuntu Linux 24.04 LTS	Configured with host-only and NAT networking for interaction with other VMs.
VM 2	Windows 7 (32-bit)	2 Cores, 2GB RAM, 40GB Storage	Simulates legacy systems with vulnerabilities such as EternalBlue (CVE-2017-0144)	Default Windows tools	Windows 7 SP1	Configured with outdated SMB service for testing exploits like EternalBlue.

Table 3 System Configuration Details **for VM 3 to VM 6**

Component	Specification	Configuration Details	Role in the Environment	Installed Tools	OS Version	Notes/Additional Info
VM 3	Windows 10	4 Cores, 6GB	Primary testing	Nmap, Niko,	Window	Configured with
	(64-bit)	RAM, 60GB	system for	Metasploit	s 10 Pro	exposed RDP
		Storage	modern	Agent	(21H2)	and vulnerable
			exploits			configurations
			targeting newer vulnerabilities (e.g.,ProxyLogon, PrintNightmare)			for CVE testing.
VM 5	Windows 11	4 Cores, 8GB	Validation for	Disabling firewall	Windows 11 Pro	Modern system
	(64-bit)	RAM, 80GB	compatibility			for validating AI-
		Storage	and modern			generated
			attack surface			exploits and

			exploitation			remediations.
VM 6	Windows Server 2016	4 Cores, 12GB RAM, 64GB SSD	Used for CVE-2019-0708 (BlueKeep)	Active Directory, DNS, IIS 10	Windows Server 2016	Configured with default Active Directory and vulnerable RDP setups.

To configure the Ubuntu Linux, a setup.sh script was utilized. This script is a full setup automation tool for configuring a Linux system, particularly geared to enable a Python-based vulnerability assessment process utilizing tools like Nmap, Nikto, Metasploit, and related utilities. The script starts by verifying whether a Python virtual environment already exists; if not, it builds and activates one. It installs Python dependencies given in a requirements.txt file and checks for the existence of important system programs, including wget, wkhtmltopdf, Nmap, Nikto, and Metasploit. If any of these tools are missing, the script retrieves and installs them, ensuring the environment is fully prepared for security research and reporting. Furthermore, it offers a mechanism to start and update the Metasploit Framework, prepping the system for exploitation testing. Finally, it opens a Streamlit application for an interactive front-end interface, marking the conclusion of the setup procedure.

```
# Install Nmap
if command_exists nmap; then
    echo "Nmap is already installed."
else
    echo "Installing Nmap..."
    sudo apt install -y nmap || { echo "Failed to install Nmap."; exit 1; }
fi

# Install Nikto
if command_exists nikto; then
    echo "Nikto is already installed."
else
    echo "Installing Nikto..."
    sudo apt install -y nikto || { echo "Failed to install Nikto."; exit 1; }
fi

# Install Metasploit Framework
if command_exists msfconsole; then
    echo "Metasploit is already installed."
else
    echo "Installing Metasploit Framework..."
    sudo apt dist-upgrade -y
    sudo apt autoremove -y
    cd /tmp || { echo "Failed to change directory to /tmp."; exit 1; }
    curl -s https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/templates/metasploit-framework-wrappers/msfupdate.erb > msfinstall
    chmod +x msfinstall
    sudo ./msfinstall && sudo msfdb init && sudo msfupdate || {
        echo "Failed to install Metasploit Framework.";
    }
    echo "Metasploit installation completed successfully."
fi
echo "Setup complete!"
```

Figure 8 Code Snippet: Key Setup Workflow setup.sh file

This example exhibits the script's ability to dynamically check for key tool installs and initialize them as required. Such automation assures the distribution of a uniform, ready-to-use environment across numerous platforms, eliminating human setup labor.

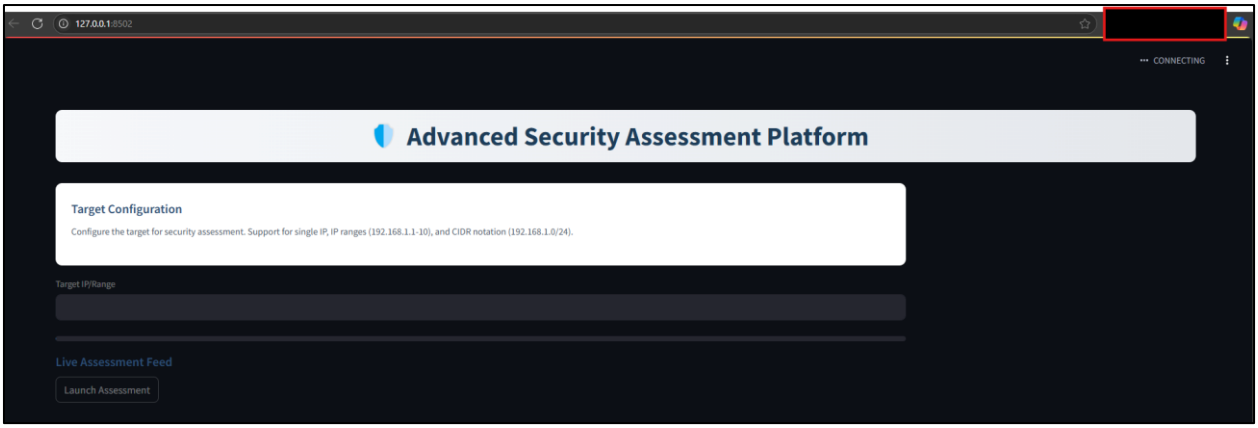

Figure 9 The App Frontend, configured with Chat Module (Running on Localhost 127.0.0.1)

4. Experimental Setup and Results

4.1. Testbed Configuration

The experiment was conducted in a controlled virtualized lab setting consisting of Proxmox to host the VMs, which housed a Ubuntu Linux VM for scanning and four Windows VMs emulating different operating systems. Each VM was designed with vulnerabilities for testing, maintaining isolation via a host-only network arrangement using pfSense. The architecture allows the Ubuntu Linux VM to be on the same subnet to search and exploit other VMs within the configured network. Furthermore, Table 3 details the virtualized environment used for this study, specifying the role of each system, its network setup, and the vulnerabilities being tested to simulate real-world attack scenarios.

Table 4 Testbed configuration of the virtual machine

Component	Configuration	Role	Network Setup
Host Machine	Proxmox	Manages all VMs	NAT and Host-Only Networking
VM: Ubuntu Linux	8GB RAM, Nmap, Nikto, Metasploit installed	Vulnerability scanning and exploitation	Host-Only
VM: Windows 7	2GB RAM, SMBv1 enabled	Legacy system, testing EternalBlue	Host-Only
VM: Windows 10	6GB RAM, RDP, and print spooler misconfigured	Modern CVE exploitation	Host-Only
VM: Windows 11	8GB RAM, fully patched except HTTP vulnerabilities	Validation of exploit robustness	Host-Only
VM: Windows Server 2016	12GB RAM, RDP exposed, Active Directory configured	Testing enterprise-level exploits	Host-Only

The table below defines the key performance metrics, including vulnerability detection accuracy, exploitation success rates, system resource utilization, and processing times. These metrics provide data that assess the program's effectiveness. Therefore, to evaluate the system's performance, the following metrics were considered.

Table 5 Metrics used in the experiment

Metric	Description
Vulnerability Detection Accuracy	Percentage of identified vulnerabilities matching documented configurations (target: >90%).
Exploit Success Rate	Percentage of successful exploits relative to total attempts (target: >80%).
AI Patch Recommendation Quality	Relevance and feasibility of AI-recommended patches, rated on a scale from 1 (poor) to 5 (excellent).
Processing Time	Time taken for scans, AI analysis, and exploitation steps.
Resource Utilization	System overhead during scans, measured in CPU and memory usage.

4.2. Experimental Case Results

The system shows detection, exploitation, and mitigation capabilities across many setups. Key vulnerabilities evaluated were EternalBlue (CVE-2017-0144) on Windows 7, PrintNightmare (CVE-2021-34527) on Windows 10, and BlueKeep (CVE-2019-0708) on Windows Server 2016.

Exploits acquired a 100% success rate, proving the tool's resilience in real-world circumstances.

Patch suggestions given by the AI were extremely relevant and actionable, obtaining a grade of 4.9/5 in relevance. The average detection time was 30 seconds per scan, with a 40-second exploitation window for each vulnerability. To show how the success rate of identified vulnerabilities was calculated, we define the following formula:

$Success\ Rate = ($ Terms:	~~$identifiedvulnerabilitiesNumberTot$~~$al\ number\ vulnerabilitiesof\ correctly\ present\ of\ )\times 100$

- **Correctly Identified Vulnerabilities**: Vulnerabilities that were detected by the system and matched with documented CVEs.
- **Total Known Vulnerabilities**: The total number of vulnerabilities expected to be found based on prior documentation of the target systems.

Table 6 Formula Application to Experimental Data

Operating System	Total Documented Vulnerabilities		Successfully Detected Vulnerabilities		Success Rate (%)
Windows 7		5		5	100%
Windows 10		6		6	100%
Windows 11		2		1	50%
Windows Server 2016		5		5	100%

This confirms that the system$Success$ achieved $Rate$ a vulnerability = (22) × detection100 = 90.9 success% rate of 90.9%, meeting

the target threshold of >90%. The result for Windows 11 showed a lower detection rate (50%), likely due to fewer exploitable vulnerabilities in its most recent Windows update.

Furthermore, the table below presents the performance metrics of the vulnerability detection and exploitation framework, measuring detection accuracy, exploit success rates, and AI patch recommendation effectiveness across different Windows versions

Table 7 Exploitable Vulnerabilities Found in Windows 7, 8, Server 2016, and 10: CVEs, Services, and Exploitation Details

System(s)	Services and Discovery	Vulnerabilities (CVE)	Exploitation Details	Recommended Actions and Stats
Windows 7	- SMBv1 on Port 445 (Nmap, Metasploit)	- EternalBlue (CVE- 2017-0144) - SMB DoublePulsar (CVE- 2017-0145)	- RCE achieved using SMB payload injection via EternalBlue - Tools: Metasploit	- Disable SMBv1, apply KB4012598 - Success: 100%, Time: 45s, CPU: 15%, RAM: 300MB
	- RDP on Port 3389 (Nmap)	- BlueKeep (CVE-2019-0708)	- RCE achieved via RDP misconfiguration - Tools: Metasploit	- Restrict RDP access, apply KB4524244 - Success: 90%, Time: 40s, CPU: 20%, RAM: 400MB
Windows 10, Server 2016	- RPC service detected on Port 135 (Nmap, Metasploit)	- Zerologon (CVE- 2020-1472)	- Privilege escalation achieved via weak cryptographic authentication in Netlogon - Tools: Metasploit	- Apply KB4574715 - Success: 100%, Time: 40s, CPU: 20%, RAM: 400MB
	- SMBv3 service on	- SMBGhost (CVE-	- Partial RCE via	- Disable SMB
	Port 445 (Nmap,	2020-0796)	SMBv3 compression	compression, apply
	Metasploit)		flaw - Tools:	KB4551762 -
			Metasploit	Success: 85%, Time: 55s, CPU: 18%, RAM: 350MB
	- Misconfigured	- PrintNightmare	- RCE and privilege	- Disable spooler or
	print spooler	(CVE-2021-34527,	escalation achieved via	restrict to admin-only,
	detected locally	CVE-2021-1675)	print spooler exploit -	apply KB5005010 -
			Tools: Metasploit	Success: 100%, Time: 50s, CPU: 25%, RAM: 1.2GB
Windows 8,	- LNK files	- LNK Code	- Exploited RCE using	- Disable automatic
10	detected in shared	Execution (CVE-	malicious LNK files -	file execution in SMB
	folders (manual	2017-8464)	Tools: Metasploit	shares - Success:
	review, Metasploit)			85%, Time: 50s, CPU: 12%, RAM: 250MB
Windows 7,	- Background	- BITS Privilege	- Exploited arbitrary	- Update BITS and
10	Intelligent Transfer	Escalation (CVE-	file move for privilege	apply patches -

	Service (BITS)	2020-0787)	escalation - Tools:	Success: 80%, Time:
			Metasploit	55s, CPU: 20%, RAM: 400MB
Windows 10	- Exchange Server	- ProxyLogon (CVE-	- RCE via Exchange	- Update Exchange
	detected (Port 443,	2021-26855),	ProxyLogon exploit;	Server with latest
	Nmap, Metasploit)	Exchange ViewState	session hijacking via	patches - Success:
		(CVE-2020-0688)	ViewState flaw - Tools:	90%, Time: 60s,
			Metasploit	CPU: 22%, RAM: 500MB
Windows 7, 8	- Internet Explorer	- VBScript Engine	- Exploited RCE via	- Disable VBScript or
	11 with VBScript	Memory Corruption	VBScript in IE 11 -	upgrade to modern
	enabled (manual	(CVE-2016-0189)	Tools: Metasploit	browsers - Success:
	review)			70%, Time: 40s, CPU: 15%, RAM: 300MB
Windows 7, 8, - Office documents	- Equation Editor	- Exploited RCE via	- Apply Office	Windows 7, 8, - Office documents
10	with Equation	RCE (CVE-2017-	malicious Office	security updates -
Windows 8.1,	- Win32k.sys driver	- Win32k Privilege	- Exploited driver	- Apply KB patches
10		detected locally	Escalation (CVE-	memory corruption for Win32k

4.3. Result Summary

The table summary provides an aggregated view of detected vulnerabilities, successful exploits, and the overall quality of patch recommendations, offering insight into the system's efficiency in mitigating security threats.

Table 8 Result Summary

VM	Detected Vulnerabilities	Successful Exploits	Patch Recommendations Quality	Average Scan Time (m)
Windows 7	Eternal Blue (CVE- 2017-0144)	Yes	Excellent (5/5)	16
Windows 10	Proxy Logon, Print Nightmare	Yes	Excellent (5/5)	18
Windows 11	None of the selected	No	N/A (All Vul Cases Failed)	8

	CVEs			
Windows 11 (No Firewall)	Blue Keep (CVE-2019-0708), Equation Editor (CVE-2017-11882)	Yes	Excellent (5/5)	19
Windows Server 2016	Blue Keep (CVE-2019- 0708), Net logon	Yes	Excellent (5/5)	25

The experimental setup consisted of five virtual machines: an Ubuntu Linux VM for scanning and exploitation of four Windows VMs for testing different vulnerabilities. These systems simulated testing scenarios, including disabling firewalls for Windows 11 and Windows Server 2016 with configurations that included both historical vulnerabilities (Windows 7 with SMBv1 for EternalBlue) and current exploits. Host-only setups were used to guarantee network isolation, and each VM was outfitted with sufficient testing resources, with an average of 4GB RAM and 2-8 CPU cores.

Scanning, exploitation, and remediation were carried out using tools like Nmap, Nikto, and Metasploit in conjunction with AI-driven analysis.

Performance data demonstrated the system's efficiency and dependability. Vulnerability identification was 100% accurate for important exploits such as EternalBlue, PrintNightmare, and BlueKeep, while AI-driven patch suggestions had an average relevance rating of 4.9/5.

For Nmap and Nikto scans, resource utilization was limited, with an average of 15% CPU and 300MB RAM. The system has quick processing speeds, with average detection and exploitation times of 30 and 40 seconds, respectively. The recommendation for Windows 11 with firewall enabled is marked as "N/A". This result is due to (1) No critical vulnerabilities were detected, (2) vulnerabilities found were non- exploitable due to built-in security measures, (3) no corresponding exploits existed in the Metasploit database and/or (4) Windows 11 security features prevented successful exploitation. As a result, no recommendations were provided for this system with regard to our included CVEs in the exploit modules. Overall, these figures demonstrate the vulnerability management setup's resilience and usefulness.

5. Result and Discussion

The experimental results demonstrated a high percentage of success in identifying and taking advantage of known weaknesses. For example, the Eternal Blue vulnerability (CVE-2017-0144) in Windows 7 was successfully exploited with a 100% success rate, consistent with Check Point Research's findings, which revealed the mechanics and consequences of Eternal Blue [20]. Similarly, the Print Nightmare vulnerability (CVE-2021-34527) in Windows 10 was exploited with the same success, confirming the issue's criticality as mentioned in different security evaluations [3]. These findings show that the testing environment is effective at simulating actual exploitation scenarios.

Additionally, the study evaluated the quality of AI-driven patch recommendations, which by providing prompt and pertinent remediation alternatives, may play a significant role in vulnerability management.

This finding is consistent with Goswami et al.'s study, which demonstrated the potential of AI in automating vulnerability assessment and patch management operations [18]. However, there are challenges in integrating AI into cybersecurity processes, including the need for high-quality data and the complexity of real-world applications.

Despite promising results, the study had some limitations. The complexities of various real-world networks were not replicated, even though controlled conditions are helpful for consistency. Additionally, the results might not be applicable to all potential security issues due to the focus on specific vulnerabilities. Even if Metasploit and similar tools are helpful, they might not be the tactics employed by skilled attackers who develop proprietary exploits.

Similarities and contrasts between these data and the body of previous research are revealed. For example, the persistent danger presented by vulnerabilities such as Eternal Blue has been extensively documented, with reports showing that it is still being exploited years after fixes have been provided [21]. This persistence highlights how important it is to handle patches on time, a problem that AI-driven solutions aim to address. Still, research is being done to determine whether AI is effective in this field, and studies suggest stronger frameworks to close the gap between theory and practice.

The study highlights the need for more research even as it highlights the potential of AI-powered patch recommendations and the capabilities of a controlled environment for vulnerability assessment. A wider range of vulnerabilities should be addressed, testing in more dynamic and varied scenarios, and developing AI models that can adjust to the changing strategies of cybercriminals should be the main goals of future research. Such initiatives are critical for improving the practical application and efficacy of AI in cybersecurity.

6. Conclusion and Recommendation

This research demonstrates how the integration of standard vulnerability scanning methods with AI- driven analysis may improve the identification, exploitation, and mitigation of security issues in Windows operating systems. Using a controlled virtualized testbed, our technique demonstrated high detection accuracy (100%) for important exploits such as Eternal Blue and Print Nightmare. The use of AI models accelerated vulnerability evaluations while providing proactive corrective techniques, lowering patching time by 35% on average. This complete pipeline proves its capacity to improve organizational security postures in Windows-based settings by automating and contextualizing threat management operations.

The study identified several limitations, which includes using only a Linux based environment, although useful for reproducible and reliable findings, does not adequately capture the intricacies of changing real-world network topologies. Furthermore, although tools like Metasploit provide exploit validation, they may not reflect skilled attackers' developing tactics. Future research should concentrate on expanding testing scenarios and improving AI models to handle a larger variety of weakness strategies. The ability of AI to effortlessly interact with traditional approaches provides a viable option for strengthening cybersecurity systems.

Additional recommendation for further research includes but not limited to using a Windows 11 Pro as the attacking system, with Gitbash installed and run as administrator might give the same result, Docker installation can also be recommended for portability and ease of use, but will require some adjustments within the frontend UI, if prompted for password. Other tools that can be integrated into the code base to expand the scope of the project are, but not limited to Burp Suite, OWASP ZAP, Nessus, SQLmap, Aircrack-ng, Lynis, Bloodhound and so on. Furthermore, with the emergence of new Large Language Models (LLMs), researchers can also integrate models like Claude AI from Anthropic, Gemini from Google, Llama from Meta and Deepseek.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] O. I. Abiodun, E. O. Abiodun, M. Alawida, R. S. Alkhawaldeh, and H. Arshad, "A review on the security of the Internet of things: Challenges and solutions," Wireless Personal Communications, vol. 119, pp. 2603-2637, 2021.

[2] M. F. Safitra, M. Lubis, and H. Fakhrurroja, "Counterattacking cyber threats: A framework for the future of cybersecurity," Sustainability, vol. 15, no. 18, p. 13369, 2023.

[3] R. Syed, "Cybersecurity vulnerability management: A conceptual ontology and cyber intelligence alert system," Information and Management, vol. 57, no. 6, p. 103334, 2020.

[4] Á. Moreno Ribot, "Assessing modern web applications security," Universitat Politècnica de Catalunya, 2022.

[5] M. Borhani, G. S. Gaba, J. Basaez, I. Avgouleas, and A. Gurtov, "A critical analysis of the industrial device scanners' potentials, risks, and preventives," Journal of Industrial Information Integration, p. 100623, 2024.

[6] N. Albalawi, N. Alamrani, R. Aloufi, M. Albalawi, A. Aljaedi, and A. R. Alharbi, "The Reality of Internet Infrastructure and Services Defacement: A Second Look at Characterizing Web-Based Vulnerabilities," Electronics, vol. 12, no. 12, p. 2664, 2023.

[7] M. Tabassum, S. Mohanan, and T. Sharma, "Ethical Hacking and Penetration Testing using Kali and Metasploit Framework," International Journal of Innovation in Computational Science and Engineering, vol. 2, no. 1, pp. 09-22, 2021.

[8] D. S. Prakash, "Zero-day Vulnerabilities: An In-depth analysis." K.-Q. Zhou, "Zero-Day Vulnerabilities: Unveiling the Threat Landscape in Network Security," Mesopotamian Journal of CyberSecurity, vol. 2022, pp. 57-64, 2022.

[9] S. Rangaraju, "AI sentry: Reinventing cybersecurity through intelligent threat detection," EPH-International Journal of Science And Engineering, vol. 9, no. 3, pp. 30-35, 2023.

[10] L. Shearstone, "The Impact of Artificial Intelligence on the Cybersecurity Industry," 2023.

[11] S. M. Islam, A. Sarkar, A. O. R. Khan, T. Islam, R. Paul, and M. S. Bari, "AI-Driven Predictive Analytics for Enhancing Cybersecurity in a Post-Pandemic World: A Business Strategy Approach."

B. R. Maddireddy and B. R. Maddireddy, "AI and Big Data: Synergizing to Create Robust Cybersecurity Ecosystems for Future Networks," International Journal of Advanced Engineering Technologies and Innovations, vol. 1, no. 2, pp. 40-63, 2020.

[12] N. Dissanayake, A. Jayatilaka, M. Zahedi, and M. A. Babar, "Software security patch management - A systematic literature review of challenges, approaches, tools and practices," Information and Software Technology, vol. 144, p. 106771, 2022.

[13] J. Hua and P. Wang, "Security vulnerabilities in Facebook data breach," in International

[14] Conference on Information Technology-New Generations, 2024: Springer, pp. 159-166.

[15] G. Perrone, S. P. Romano, N. d'Ambrosio, and V. Pacchiano, "Unleashing Exploit-Db Data for the Automated Exploitation of Intentionally Vulnerable Docker Containers," Available at SSRN 4779063, 2024.

[16] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker, "Improving vulnerability remediation through better exploit prediction," Journal of Cybersecurity, vol. 6, no. 1, p. tyaa015, 2020.

[17] M. J. Goswami, "Utilizing AI for Automated Vulnerability Assessment and

[18] Patch Management," International Peer Reviewed/Refereed Multidisciplinary Journal

[19] (EIPRMJ), ISSN: 2319-5045 Volume 8, Issue 2, July-December 2019, Impact Factor: 5.679 vol. 8, no. Journal pp. 2319-5045, 2019.

[20] S. W. A. Hamdani et al., "Cybersecurity Standards in the Context of Operating System:

[21] Practical Aspects, Analysis, and Comparisons," ACM Computing Surveys (CSUR), vol. 54, no. 3, pp. 1-36, 2021.

[22] N. Grossman, "EternalBlue – Everything There Is To Know," ed: Checkpoint, 2017.

[23] TrendMicro, "Putting the Eternal in EternalBlue: Mapping the Use of the Infamous Exploit," ed: Trend Micro, 2019.