Automated Port Scanner Python Script

Stephen Broadbridge

CMP320: Advanced Ethical Hacking

2023/24

Note that Information contained in this document is for educational purposes.

.

Abstract

This white paper details the creation of an automated port scanner designed to improve network security by identifying open ports more efficiently. The goal of the project was to streamline the vulnerability assessment process which is often performed manually using Nmap scan commands.

Utilising the Python language, the project integrated the Scapy library for ARP scanning to discover active devices and Nmap for comprehensive port scanning. Additionally, the script will then output the results into text files for further analysis and into a Word document as part of a larger report.

The scanner was successfully developed and tested, capable of detecting open ports and potential security vulnerabilities across active networked devices. While effective in intermediate scenarios, limitations do exist when handling more complex security threats.

.

Contents

.

1 INTRODUCTION

1.1 BACKGROUND

In the current digital landscape, it is becoming increasingly important for robust network security. Organisations and individuals are increasingly vulnerable to sophisticated cyber-attacks that could compromise sensitive information, disrupt operations and lead to substantial financial loss. “In 2022-23, Police Scotland recorded 14,890 cyber crimes. This has been broadly stable over the last 3 years but represents in a significant increase from the estimated 7,710 cyber crimes recorded in 2019-20” (The Scottish Government, 2023). One of the most critical vulnerabilities in network security is the presence of unsecured and open ports. “If ports are not properly configured, hackers can potentially access your computer or network, exploit software vulnerabilities, and gain control of the system” (*What are open ports? risks and security* 2024).

Port scanning is a method used to identify open ports and services available on devices connected to a network. This plays a critical role in vulnerability assessments. Traditional port scanning methods are often manual, which can be time-consuming and can cause significant delays and potential oversights when addressing vulnerabilities.

The automated port scanner developed within this project removes those typical negative aspects and allows the user to concentrate on other tasks while the Python script completes the task efficiently. The Python script automates the detection of open ports, streamlining the process of vulnerability assessments and enhancing the ability to quickly respond to potential threats.

1.2 AIM

The primary aims of this project were to:

- **Develop an automated port scanner tool –** To design and implement an automated port scanner that enhances efficiency when undertaking a vulnerability assessment.
- **Automate the results into a Word document –** The output of the automated port scanner should output the results into a Word document to make the report writing process more efficient.

2 PROGRAM AND DEVELOPMENT

2.1 OVERVIEW OF PROCEDURE

The automated port scanner tool developed as part of this project leverages two main Python libraries: Scapy for ARP scanning and Nmap for port scanning. This combination of libraries allows for the effective detection of live hosts on a network and a comprehensive examination of their open ports, services and potential vulnerabilities.

2.2 CODE DEVELOPMENT

2.2.1 Initial Setup

Before running the script, users must ensure that their system has Python installed and set up properly. The script depends on several external libraries which need to be installed via pip, Pythons package installer. Here are the setup instructions:

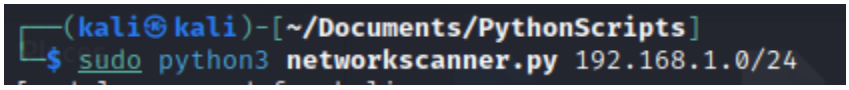
1. **Install Python:** Ensure Python 3 is installed on your system, this can be downloaded from the official Python website.
2. **Install required libraries:**
    a. Open your command line interface (CLI) and ensure pip is installed by running the following command:

    ```
    python -m ensurepip
    ```

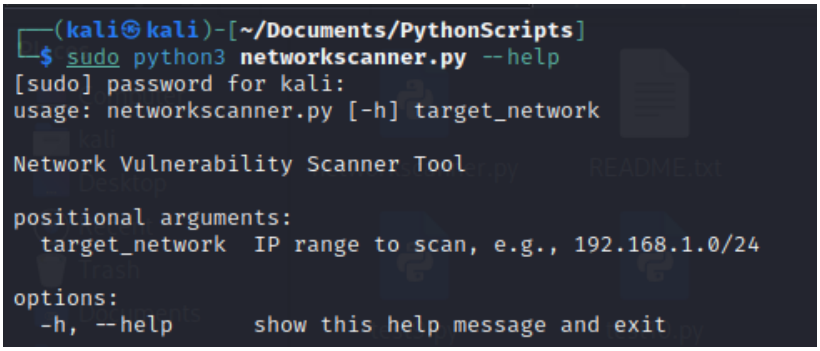
    b. Install the necessary python libraries using pip:

    ```
    pip install prettytable scapy-python3 python-nmap python-docx
    ```

3. **Script Configuration:** Before running the script, you may need to configure the target network or IP range within the script. This can be set when starting the script.
    a. When starting the script enter the target subnet you wish to scan. In this example 192.168.1.0/24 is used:

    b. The script does offer assistance with a “-h” or “--help” option which displays some useful information to users.

2.2.2 Importing Libraries

The script begins by importing essential libraries required for its operation.

```
# Import Libraries
import argparse
from prettytable import PrettyTable
from scapy.all import ARP, Ether, srp
import nmap
from docx import Document
from docx.shared import Pt
from docx.shared import Inches
```

- **Argparse:** Used to allow Command Line Interface (CLI) instructions to be entered by the user.
- **Pretty Table:** Used for creating readable formatted tables in the text files outputted, enhancing the presentation of the Nmap data.
- **Scapy:** Utilized for crafting and sending network packets, crucial for the ARP scanning process.
- **Nmap:** Integrated for conducting in-depth port scanning and vulnerability assessments.
- **Python-docx:** Used for creating and managing Word documents, allowing for professional reporting of scan results.

2.2.3 Network Discovery via ARP Scanning

To identify live hosts on a network the automated port scanner tool performs an Address Resolution Protocol (ARP) scan. This scan is the crucial first step as it maps out IP addresses to their corresponding MAC addresses within a local network segment.

```
# Function to perform an ARP scan on a target IP address
def arp_scan(target_ip):
    print("Performing ARP Scan ... ")
    ether = Ether(dst="ff:ff:ff:ff:ff:ff")  # Set Ethernet frame destination as broadcast
    arp = ARP(pdst=target_ip)  # Create ARP request for the target IP
    result = srp(ether/arp, timeout=3, verbose=False)[0]  # Send the packet and get response

    ip_addresses = [received.psrc for sent, received in result]  # Extract IP addresses from responses
    print("IP Addresses found:")
    for ip in ip_addresses:
        print(ip)
    return ip_addresses
```

The function in the script called **'arp_scan'** is critical for network discovery. It creates an ARP request packet using the Scapy library's **'Ether'** and **'ARP'** classes. The Ethernet frame is set to broadcast (dst="ff:ff:ff:ff:ff:ff") to ensure that the ARP request reaches all devices within the subnet specified by the **'target_ip'** parameter. The '**scp'** function then sends these packets and captures any responses, extracting the source IP addresses. This method effectively identifies active devices within the target subnet by collecting responses from devices that acknowledge the ARP request.

```
┌──(kali㉿kali)-[~/Documents/PythonScripts]
└─$ sudo python3 networkscanner.py 192.168.1.0/24
[sudo] password for kali:
Performing ARP Scan ...
IP Addresses found:
192.168.1.1
192.168.1.93
192.168.1.95
```

2.2.4 Nmap Port Scanning and Report Generation

The **'nmap_scan_and_output'** function performs scans on each IP address, scanning for all its ports, service versions and any known vulnerabilities. This function leverages the Nmap library to thoroughly scan each IP address from the list **'ip_addresses'**, which contains all IP addresses identified as active by the initial ARP scan.

```
# Function to perform an Nmap scan on discovered IP addresses and output results to a document
def nmap_scan_and_output(ip_addresses):
    nm = nmap.PortScanner()  # Create a new PortScanner object
    document = Document()  # Create a new Word document
    document.add_heading('Vulnerability Assessment Report', 0).alignment = 1  # Add main heading
    document.add_page_break()  # Add a page break
    document.add_heading('Table of Contents', level=1)  # Add a table of contents heading
    document.add_paragraph('Contents here... \nNote: Update this section manually or with an automated tool.')
    document.add_page_break()  # Add another page break

    for ip in ip_addresses:
        print(f"Scanning {ip} with Nmap Full Scan and Vulnerability Script...")
        nm.scan(ip, arguments='-p 1-65535 -sV --script=vuln')  # Perform an Nmap scan
        output_filename = f"nmap_full_vuln_scan_{ip}.txt"
        extract_service_info_to_file(ip, nm, output_filename)  # Extract info to text file
        extract_service_info_to_document(ip, nm, document)  # Extract info to Word document

    report_filename = "vulnerability_assessment_report.docx"
    document.save(report_filename)  # Save the Word document
    print(f"White paper report generated: {report_filename}")
```

This function initiates a **'nmap'** PortScanner object and configures it to perform a scan on each IP address using specified arguments **'-p 1-65535'** scans all ports, **'-sV'** enables service detection and **'--script=vuln'** runs vulnerability scripts to detect any known vulnerabilities. The results are both saved into a text file for each IP address and documented in a detailed Word report creating a record of the networks open ports.

A Word document is prepared using the python-docx library, where the scan results are systematically documented. Each IP address's results are formatted into sections within the document, ensuring the information is presented in a readable format for reporting. Additionally, results are simultaneously output into text files for immediate reference.

2.2.5 Data Formatting and Document Generation

After completing the network port scans the script transitions to formatting the gathered data for both quick reference and detailed reporting. This involves using two functions designed to handle text and document outputs respectively.

2.2.5.1 Function: extract_service_to_file

This function is designed to create a text file of the data gathered from the nmap scan results and display them in a readable format using the PrettyTable library.

```
# Function to extract service information to a text file
def extract_service_info_to_file(ip, nm, filename):
    table = PrettyTable()  # Create a new PrettyTable object
    table.field_names = ["IP", "PROTOCOL", "PORT", "STATE", "SERVICE", "VERSION", "VULNERABILITIES"]
    populate_table(ip, nm, table)  # Populate table with scan results
    with open(filename, 'w') as file:
        file.write(str(table))  # Write table to file
```

The function begins by creating the new **'PrettyTable'** object. This object is designed to display data in a table format in the console or text file. The headers for the table are set to categorize crucial information about each scanner port of the device.

The **'populate_table'** function is invoked, which iterates over the scan results stored in the **'nmap'** object for the specific IP address. It then fills the data about each open port, including the protocol used, the state of the port (open or closed), the service running on the port and any vulnerabilities identified.

IP	PROTOCOL	PORT	STATE	SERVICE	VERSION	VULNERABILITIES
192.168.1.93	TCP	21	open	ftp	Golden ftpd 5.00	N/A
192.168.1.93	TCP	135	open	msrpc	Microsoft Windows RPC	N/A
192.168.1.93	TCP	139	open	netbios-ssn	Microsoft Windows netbios-ssn	N/A
192.168.1.93	TCP	445	open	microsoft-ds		N/A
192.168.1.93	TCP	902	open	vmware-auth	VMware Authentication Daemon 1.10	
192.168.1.93	TCP	912	open	vmware-auth	VMware Authentication Daemon 1.0	N/A
192.168.1.93	TCP	5040	open			N/A
192.168.1.93	TCP	5357	open	http	Microsoft HTTPAPI httpd 2.0	
192.168.1.93	TCP	49676	open	msrpc	Microsoft Windows RPC	N/A

2.2.5.2 Function: extract_service_to_document

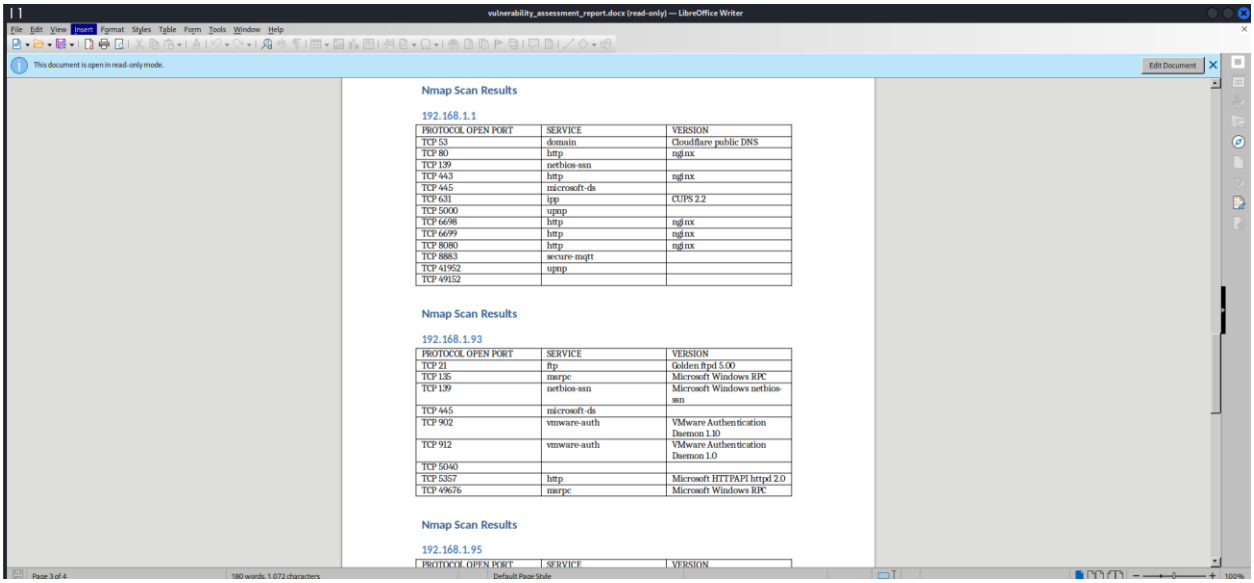
Simultaneously, the script prepares a report template in a Word document using the python-docx library, which provides functionalities to create, edit and save Word documents.

```python
# Function to extract service information to a Word document
def extract_service_info_to_document(ip, nm, document):
    document.add_heading(f"Nmap Scan Results", level=1)  # Add a heading for Nmap results
    document.add_heading(f"{ip}", level=2)  # Add a subheading for the IP address
    table = document.add_table(rows=1, cols=3)  # Create a new table in the document
    table.style = 'Table Grid'
    hdr_cells = table.rows[0].cells
    hdr_cells[0].text = "PROTOCOL OPEN PORT"
    hdr_cells[1].text = "SERVICE"
    hdr_cells[2].text = "VERSION"
    for host in nm.all_hosts():
        for proto in nm[host].all_protocols():
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                port_info = nm[host][proto][port]
                state = port_info['state']
                if state == 'open':
                    service = port_info['name']
                    version = port_info.get('product', '') + " " + port_info.get('version', '').strip()
                    row_cells = table.add_row().cells
                    row_cells[0].text = f"{proto.upper()} {port}"
                    row_cells[1].text = service
                    row_cells[2].text = version
```

A heading is added to the Word document for each IP address being reported on, clearly segmenting the document into sections per device. Following this, a table is then created in the document to display open ports found during the Nmap scan. This table is styled using **'Table Grid'** to improve the readability of the document.

Data is inserted into the document via the **'populate_table_for_document'**. For each open port, it adds a new row to the table with detailed information about the protocol, service and any known vulnerabilities detected.

```python
# Function to populate the PrettyTable with data from Nmap scan results
def populate_table(ip, nm, table):
    for host in nm.all_hosts():
        for proto in nm[host].all_protocols():
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                port_info = nm[host][proto][port]
                service = port_info['name']
                state = port_info['state']
                version = port_info.get('product', '') + " " + port_info.get('version', '').strip()
                script_info = port_info.get('script', '')
                vuln_info = script_info.get('vuln', '') if script_info else 'N/A'
                if state == 'open':
                    table.add_row([ip, proto.upper(), str(port), state, service, version, vuln_info])
```



Nmap Scan Results

192.168.1.1

PROTOCOL OPEN PORT	SERVICE	VERSION
TCP 53	domain	Cloudflare public DNS
TCP 80	http	nginx
TCP 139	netbios-ssn	
TCP 443	http	nginx
TCP 445	microsoft-ds	
TCP 631	ipp	CUPS 2.2
TCP 5000	upnp	
TCP 6698	http	nginx
TCP 6699	http	nginx
TCP 8080	http	nginx
TCP 8883	secure-mqtt	
TCP 41952	upnp	
TCP 49152		

Nmap Scan Results

192.168.1.93

PROTOCOL OPEN PORT	SERVICE	VERSION
TCP 21	ftp	Golden ftpd 5.00
TCP 135	msrpc	Microsoft Windows RPC
TCP 139	netbios-ssn	Microsoft Windows netbios-ssn
TCP 445	microsoft-ds	
TCP 902	vmware-auth	VMware Authentication Daemon 1.10
TCP 912	vmware-auth	VMware Authentication Daemon 1.0
TCP 5040		
TCP 5357	http	Microsoft HTTPAPI httpd 2.0
TCP 49676	msrpc	Microsoft Windows RPC

Nmap Scan Results

192.168.1.95

2.2.6 '_ _Main_ _' Section

This section of the Python script initiates by parsing command line arguments using the 'arparse' Python library to define the target network IP range facilitating user-specified network scans. Next, the script incorporates the 'arp_scan' function to detect active devices by IP addresses to assess vulnerabilities and service details with results organized into text files and a Word document. This section effectively combines network discovery and reporting, streamlining the vulnerability assessment workflow.

```python
# Main execution block
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Network Vulnerability Scanner Tool')
    parser.add_argument('target_network', help='IP range to scan, e.g., 192.168.1.0/24')
    args = parser.parse_args()

    discovered_ips = arp_scan(args.target_network)
    if discovered_ips:
        nmap_scan_and_output(discovered_ips)
    else:
        print("No IP addresses discovered.")
```

3 DISCUSSION

3.1 GENERAL DISCUSSION

The developed Python script successfully meets the two primary aims outlined in the project aims section: The creation of an automated portscanner tool and the automation of results documentation into a Word document. This section will detail the performance and outcomes of the script confirming its efficiency and effectiveness in vulnerability assessments.

The script begins with an ARP scan of the specified target IP range, effectively identifying active IP addresses within the network. This initial scan is crucial as it filters out inactive or non-responsive devices, focusing the subsequent port scans on relevant targets. The script leverages the Scapy library for the ARP scan, which proved efficient, completing the scan within a few seconds and identifying active hosts.

Following the ARP scan the script utilizes the Nmap PortScanner module to perform scans on each discovered IP address. The Nmap scan is configured to check all 65545 ports, detect service versions, and identify any known vulnerabilities. This comprehensive scan provides a detailed view of open ports, running services, their versions, and potential vulnerabilities. All of which are valuable information for any vulnerability assessment report.

The efficiency of the script extends into its documentation capabilities. Once the port scanning section is complete, the script automates the generation of a Microsoft Word report document. The report includes a title page, table of contents, and a dedicated section for each IP address that includes tables summarising the Nmap scan results. Each table in the document is well-structured, containing columns for protocol, open port, service, and version.

The script's ability to output directly into a Word document significantly streamlines the report-writing process. By automating this step, the script minimizes the need for manual data entry and formatting, which are typically time-consuming and prone to errors. This automation ensures that the results are not only quickly available but also presented in a consistent and professional format, ready for further analysis and distribution.

In conclusion, the Python script has successfully achieved its objectives, demonstrating robust performance in scanning and reporting. The automated port scanner efficiently identifies and evaluates network vulnerabilities, while the automated reporting feature enhances productivity by simplifying the documentation process.

3.2 FUTURE WORK

The successful implementation of the automated port scanning and reporting tool has laid a strong foundation for further enhancements. To continue improving its functionality and user experience, several developments are proposed for further iterations of the script.

3.2.1 Further Nmap Scan Capabilities

The Python script in its current form utilizes a fixed set of Nmap scanning parameters which focus on collecting information on service, versions and detecting known vulnerabilities. To enhance the automated port scanners capabilities the integration of a broader range of Nmap scanning options would increase its effectiveness. This will enable users to tailor their scans to specific security requirements and scenarios, ideally simulating the experience of operating Nmap natively on a Linux system. Proposed scanning options could include:

- **Quick Scan:** For fast and efficient vulnerability assessments.
- **OS Detection:** To identify operating systems and associated vulnerabilities.
- **SYN Scans:** Utilising SYN stealth scans to reduce detection risk.
- **UDP Scans:** Addressing the scanning of UDP ports, often used by essential services by scanning much less.

These enhancements would be implemented by using the 'argparse' Python library, enabling users to select or customise their scanning techniques via command line arguments.

3.2.2 Enhanced Document Layout and Content

While the current script successfully generates a Word document and displays the scan results, there is scope to enhance the layout and content of the report to cover a broader range of vulnerability assessment aspects. Future versions of the script could include the following headings: Executive Summary, Methodology, Results/Findings and Appendices.

These enhancements will not only make the document more comprehensive but will also improve its efficiency in speeding up the report-writing process.

REFERENCES

For URLs, Blogs:

Scapy's Documentation (no date) Welcome to Scapy's documentation! - Scapy 2.5.0 documentation. Available at: https://scapy.readthedocs.io/en/latest/index.html (Accessed: 05 February 2024).

GeeksforGeeks (2020) Network scanning using Scapy Module - Python, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/network-scanning-using-scapy-module-python/ (Accessed: 06 February 2024).

Python-nmap Documentation (no date) PyPI Python-Nmap. Available at: https://pypi.org/project/python-nmap/ (Accessed: 22 February 2024)

Python PrettyTable Documentation (no date) PyPI. Available at: https://pypi.org/project/prettytable/ (Accessed: 25 February 2024).

Python Docx Documentation (no date) Python Docx. Available at: htts://python-docx.readthedocs.io/en/latest/ (Accessed: 01 March 2024).

Argparse - parser for command-line options, arguments and sub-commands (no date) Python documentation. Available at: https://docs.python.org/3/library/argparse.html (Accessed: 18 April 2024).

What are open ports? risks and security (2024) NordVPN. Available at: https://nordvpn.com/blog/what-are-open-ports/#:~:text=If%20ports%20are%20not%20properly,gain%20control%20of%20the%20system. (Accessed: 18 April 2024).

The Scottish Government (2023) Taking stock: Report on progress towards a cyber resilient scotland, Scottish Government. Available at: https://www.gov.scot/publications/taking-stock-report-progress-towards-cyber-resilient-scotland/pages/3/ (Accessed: 19 April 2024).

APPENDICES

APPENDIX A

Networkscanner.py

```
# Import Libraries
import argparse
from prettytable import PrettyTable
from scapy.all import ARP, Ether, srp
import nmap
from docx import Document
from docx.shared import Pt
from docx.shared import Inches

# Function to perform an ARP scan on a target IP address
def arp_scan(target_ip):
    print("Performing ARP Scan...")
    ether = Ether(dst="ff:ff:ff:ff:ff:ff")  # Set Ethernet frame destination as broadcast
    arp = ARP(pdst=target_ip)  # Create ARP request for the target IP
    result = srp(ether/arp, timeout=3, verbose=False)[0]  # Send the packet and get response

    ip_addresses = [received.psrc for sent, received in result]  # Extract IP addresses from
responses
    print("IP Addresses found:")
    for ip in ip_addresses:
        print(ip)
    return ip_addresses

# Function to perform an Nmap scan on discovered IP addresses and output results to a document
def nmap_scan_and_output(ip_addresses):
    nm = nmap.PortScanner()  # Create a new PortScanner object
    document = Document()  # Create a new Word document
    document.add_heading('Vulnerability Assessment Report', 0).alignment = 1  # Add main heading
    document.add_page_break()  # Add a page break
    document.add_heading('Table of Contents', level=1)  # Add a table of contents heading
    document.add_paragraph('Contents here...\nNote: Update this section manually or with an
automated tool.')
    document.add_page_break()  # Add another page break

    for ip in ip_addresses:
        print(f"Scanning {ip} with Nmap Full Scan and Vulnerability Script...")
        nm.scan(ip, arguments='-p 1-65535 -sV --script=vuln')  # Perform an Nmap scan
        output_filename = f"nmap_full_vuln_scan_{ip}.txt"
        extract_service_info_to_file(ip, nm, output_filename)  # Extract info to text file
        extract_service_info_to_document(ip, nm, document)  # Extract info to Word document

    report_filename = "vulnerability_assessment_report.docx"
    document.save(report_filename)  # Save the Word document
    print(f"White paper report generated: {report_filename}")

# Function to extract service information to a text file
def extract_service_info_to_file(ip, nm, filename):
    table = PrettyTable()  # Create a new PrettyTable object
    table.field_names = ["IP", "PROTOCOL", "PORT", "STATE", "SERVICE", "VERSION",
"VULNERABILITIES"]
    populate_table(ip, nm, table)  # Populate table with scan results
    with open(filename, 'w') as file:
```

```
        file.write(str(table))  # Write table to file

# Function to extract service information to a Word document
def extract_service_info_to_document(ip, nm, document):
    document.add_heading(f"Nmap Scan Results", level=1)  # Add a heading for Nmap results
    document.add_heading(f"{ip}", level=2)  # Add a subheading for the IP address
    table = document.add_table(rows=1, cols=3)  # Create a new table in the document
    table.style = 'Table Grid'
    hdr_cells = table.rows[0].cells
    hdr_cells[0].text = "PROTOCOL OPEN PORT"
    hdr_cells[1].text = "SERVICE"
    hdr_cells[2].text = "VERSION"
    for host in nm.all_hosts():
        for proto in nm[host].all_protocols():
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                port_info = nm[host][proto][port]
                state = port_info['state']
                if state == 'open':
                    service = port_info['name']
                    version = port_info.get('product', '') + " " + port_info.get('version',
'').strip()
                    row_cells = table.add_row().cells
                    row_cells[0].text = f"{proto.upper()} {port}"
                    row_cells[1].text = service
                    row_cells[2].text = version

# Function to populate the PrettyTable with data from Nmap scan results
def populate_table(ip, nm, table):
    for host in nm.all_hosts():
        for proto in nm[host].all_protocols():
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                port_info = nm[host][proto][port]
                service = port_info['name']
                state = port_info['state']
                version = port_info.get('product', '') + " " + port_info.get('version',
'').strip()
                script_info = port_info.get('script', '')
                vuln_info = script_info.get('vuln', '') if script_info else 'N/A'
                if state == 'open':
                    table.add_row([ip, proto.upper(), str(port), state, service, version,
vuln_info])

# Main execution block
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Network Vulnerability Scanner Tool')
    parser.add_argument('target_network', help='IP range to scan, e.g., 192.168.1.0/24')
    args = parser.parse_args()

    discovered_ips = arp_scan(args.target_network)
    if discovered_ips:
        nmap_scan_and_output(discovered_ips)
    else:
        print("No IP addresses discovered.")
```

APPENDIX B - SUGGESTIONS FOR FORMATTING FIGURES/TABLES/SCREENSHOTS IN THE BODY OF THE TEXT