

NETWORK VULNERABILITY SCANNING TOOL

NAME: K PRABHU
REG NO: 22BCE8733

ABSTRACT: Abstract This report presents the development and functionality of a Network Vulnerability Scanner designed for assessing the security of networked systems. It provides real-time visualization of scanning progress, identifies open ports, highlights potential vulnerabilities, and suggests mitigation measures. The tool is implemented using Python and leverages threading for efficiency and matplotlib for dynamic plotting.

INTRODUCTION: The Network Vulnerability Scanner is designed to detect open ports and assess the associated vulnerabilities on a target network. Network security is a critical aspect of modern systems, as unprotected or weakly protected ports can be exploited by attackers to gain unauthorized access. This tool aims to provide an efficient means of detecting potential weaknesses before they are exploited by malicious entities.

EXISTING OR RELATED TOOLS: Existing network vulnerability scanners include Nmap, Nessus, and OpenVAS. These tools provide comprehensive scanning capabilities but are often resource-intensive and may lack real-time visualization during scanning.

DRAWBACKS OF EXISTING TOOLS:

- High resource consumption, especially for large networks.
- Lack of real-time progress visualization.
- Complexity in configuration for novice users.
- Some tools are limited by licensing restrictions.

PROPOSED TOOL:

What the Tool is Used For:

The proposed tool is a Python-based network vulnerability scanner with real-time visualization capabilities. It scans for open ports, identifies associated services, checks for known vulnerabilities, and provides mitigation recommendations.

ALGORITHM OR PSEUDOCODE:

1. Accept user inputs for target IP, port range, threading count, and timeout.
2. Initialize scanning with user-defined settings.
3. For each thread, scan a subset of ports.
4. Record open ports and check against known vulnerabilities.
5. Display real-time progress using matplotlib plotting.
6. Provide a detailed report upon completion.

BEGIN

INITIALIZE color output for terminal

DEFINE global variables:

- scan_results (stores scan status for ports)
- active_threads (tracks active scanning threads)
- total_ports_scanned (counter for scanned ports)
- scan_start_time (records scan start time)
- is_scanning (boolean to control scanning)
- vuln_ports (dictionary of common vulnerable ports)
- VULNERABILITIES (dictionary of known vulnerabilities)
- port_history (stores real-time scan data)
- time_history (stores scan time updates)

DEFINE Class: NetworkVulnerability

ATTRIBUTES: port, service, description, risk_level, mitigation

METHODS:

- to_string(): Return formatted vulnerability details
- get_report(): Return structured vulnerability report

FUNCTION scan_port(ip, port, timeout)

CREATE socket connection to (ip, port)

SET socket timeout

ATTEMPT connection

IF connection successful

RECORD port as "open"

IDENTIFY service running on port

CHECK for known vulnerabilities

UPDATE scan_results and port_history

ELSE

RECORD port as "closed"

CLOSE socket

INCREMENT total_ports_scanned

FUNCTION worker(ip, ports, timeout)

FOR each port in ports

IF is_scanning is False THEN BREAK

CALL scan_port(ip, port, timeout)

DECREMENT active_threads

FUNCTION scan_network(ip, port_range, threads, timeout)

RESET scan tracking variables

SET scan_start_time

SET is_scanning = True

PARSE port range

```

DISTRIBUTE ports among threads
FOR each port group:
    WHILE active_threads >= max_threads WAIT
    INCREMENT active_threads
    START new thread calling worker(ip, port_group, timeout)

WHILE active_threads > 0 AND is_scanning:
    WAIT for threads to finish
    UPDATE real-time time_history

RETURN scan duration

FUNCTION stop_scan()
    SET is_scanning = False

FUNCTION generate_report(ip, scan_duration)
    COLLECT open ports from scan_results
    COLLECT detected vulnerabilities

    FORMAT report header with target IP and scan summary
    IF open ports exist:
        DISPLAY open ports list
    ELSE:
        DISPLAY "No open ports found"

    IF vulnerabilities exist:
        DISPLAY vulnerability details with mitigation steps
    ELSE:
        DISPLAY "No critical vulnerabilities detected"

    RETURN formatted report

FUNCTION update_plot(frame)
    CLEAR previous plot
    FILTER relevant ports from port_history
    SORT and limit displayed ports
    PLOT horizontal bar chart for open/closed ports
    ADD scan progress details
    RETURN plot elements

FUNCTION main()
    PARSE command-line arguments:
        - target IP
        - port range
        - thread count
        - timeout

```

- simulate mode flag

VALIDATE port range format
SET UP real-time visualization
DISPLAY scan initiation details

IF simulate mode enabled:
 RUN simulated vulnerability detection

TRY:
 CALL scan_network with given parameters
 GENERATE report using scan results
 DISPLAY scan completion message
 KEEP visualization window open

HANDLE exceptions:
 - KeyboardInterrupt → STOP scan and exit
 - Socket errors → DISPLAY appropriate messages
 - Cleanup → STOP ongoing scan if necessary

IF script is executed directly:
 CALL main()

END

SYSTEM REQUIREMENT SPECIFICATION(MINIMUM):

- Python 3.x
- Packages: socket, threading, time, datetime, argparse, os, sys, random, colorama, matplotlib
- Operating System: Windows, Linux, or macOS
- RAM: 4 GB or higher
- Processor: Dual-core or higher

CODE:

```
import threading
import socket
import time
import datetime
import argparse
import os
import sys
import random
from colorama import Fore, Style, init
```

```

import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from collections import defaultdict, deque

# Initialize colorama for cross-platform colored terminal output
init()

# Global variables for tracking scan results and status
scan_results = {}
active_threads = 0
total_ports_scanned = 0
scan_start_time = None
is_scanning = False
vuln_ports = {
    21: "FTP",
    22: "SSH",
    23: "Telnet",
    25: "SMTP",
    53: "DNS",
    80: "HTTP",
    110: "POP3",
    137: "NetBIOS",
    139: "NetBIOS",
    443: "HTTPS",
    445: "SMB",
    3306: "MySQL",
    3389: "RDP",
    5432: "PostgreSQL",
    8080: "HTTP-Alt"
}

# For real-time plotting
port_history = defaultdict(lambda: deque(maxlen=30))
time_history = deque(maxlen=30)

# Class representing the vulnerability
class NetworkVulnerability:
    def __init__(self, port, service, description, risk_level, mitigation):
        self.port = port
        self.service = service
        self.description = description
        self.risk_level = risk_level
        self.mitigation = mitigation

```

```
def __str__(self):
    return f"{Fore.RED}[VULNERABILITY]{Style.RESET_ALL} {self.service} on port {self.port} - {self.description} (Risk: {self.risk_level})"
```

```
def get_report(self):
    return {
        "port": self.port,
        "service": self.service,
        "description": self.description,
        "risk_level": self.risk_level,
        "mitigation": self.mitigation
    }
```

Define common vulnerabilities

```
VULNERABILITIES = {
    21: NetworkVulnerability(21, "FTP", "Anonymous FTP access allowed", "High",
        "Disable anonymous access or use SFTP instead"),
    22: NetworkVulnerability(22, "SSH", "Weak SSH configuration (allows old protocols)",
        "Medium",
        "Update SSH configuration to use only strong encryption algorithms"),
    23: NetworkVulnerability(23, "Telnet", "Unencrypted remote access protocol",
        "Critical",
        "Replace Telnet with SSH"),
    25: NetworkVulnerability(25, "SMTP", "Open relay mail server", "High",
        "Configure proper authentication for mail relay"),
    80: NetworkVulnerability(80, "HTTP", "Unencrypted web traffic", "Medium",
        "Implement HTTPS with valid certificates"),
    445: NetworkVulnerability(445, "SMB", "Legacy SMB protocol vulnerable to
        ransomware", "Critical",
        "Update to latest SMB version and disable SMBv1"),
    3306: NetworkVulnerability(3306, "MySQL", "Database exposed to network", "High",
        "Restrict database access to local connections or use VPN"),
    3389: NetworkVulnerability(3389, "RDP", "Remote Desktop exposed to internet",
        "High",
        "Use VPN and implement Network Level Authentication")
}
```

Scan a single port

```
def scan_port(ip, port, timeout=1):
    global total_ports_scanned
```

```
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(timeout)
```

```

result = sock.connect_ex((ip, port))

if result == 0:
    try:
        service = socket.getservbyport(port)
    except:
        service = vuln_ports.get(port, "Unknown")

    # Check if this is a known vulnerability
    vulnerability = VULNERABILITIES.get(port, None)

    scan_results[port] = {
        "status": "open",
        "service": service,
        "vulnerability": vulnerability
    }

    # Update real-time plotting data
    port_history[port].append(1) # 1 indicates open
else:
    port_history[port].append(0) # 0 indicates closed

sock.close()
total_ports_scanned += 1

# Thread worker function
def worker(ip, ports, timeout):
    global active_threads

    for port in ports:
        if not is_scanning:
            break
        scan_port(ip, port, timeout)

    active_threads -= 1

# Main scanning function
def scan_network(ip, port_range, threads=100, timeout=1):
    global active_threads, scan_results, total_ports_scanned, scan_start_time, is_scanning,
    time_history

    # Reset global tracking variables
    scan_results = {}
    total_ports_scanned = 0

```

```

active_threads = 0
scan_start_time = time.time()
is_scanning = True

start_port, end_port = port_range
ports = list(range(start_port, end_port + 1))

# Initialize plotting data
current_time = datetime.datetime.now().strftime("%H:%M:%S")
time_history.append(current_time)
for port in ports:
    if port not in port_history:
        port_history[port].append(0)

# Distribute ports among threads
total_ports = end_port - start_port + 1
ports_per_thread = max(1, total_ports // threads)

port_groups = [ports[i:i + ports_per_thread] for i in range(0, len(ports),
ports_per_thread)]

for port_group in port_groups:
    if not is_scanning:
        break

while active_threads >= threads:
    time.sleep(0.1)

active_threads += 1
threading.Thread(target=worker, args=(ip, port_group, timeout)).start()

# Wait for all threads to complete
while active_threads > 0 and is_scanning:
    time.sleep(0.1)
    current_time = datetime.datetime.now().strftime("%H:%M:%S")
    time_history.append(current_time)

scan_duration = time.time() - scan_start_time
return scan_duration

# Function to stop ongoing scan
def stop_scan():
    global is_scanning
    is_scanning = False

```

```
def generate_report(ip, scan_duration):
    open_ports = [port for port, data in scan_results.items() if data["status"] == "open"]
    vulnerabilities = [data["vulnerability"] for port, data in scan_results.items()
    if data["status"] == "open" and data["vulnerability"] is not None]
```

```
{Fore.CYAN} ||  
||  
|| NETWORK VULNERABILITY REPORT ||  
||  
|| {Style.RESET_ALL}
```

```
Target IP: {Fore.GREEN}{ip}{Style.RESET_ALL}
Scan completed in: {Fore.GREEN}{scan_duration:.2f} seconds{Style.RESET_ALL}
Total ports scanned: {Fore.GREEN}{total_ports_scanned}{Style.RESET_ALL}
Open ports found: {Fore.GREEN}{len(open_ports)}{Style.RESET_ALL}
Vulnerabilities detected: {Fore.RED}{len(vulnerabilities)}{Style.RESET_ALL}
```

```
{Fore.CYAN} ||=====
||=====
|| OPEN PORTS SUMMARY ||
||=====
||===== {Style.RESET_ALL}
|||||
```

```
if open_ports:
for port in sorted(open_ports):
data = scan_results[port]
report += f"{Fore.YELLOW}[+]{Style.RESET_ALL} Port {port} ({data['service']}) is {Fore.GREEN}open{Style.RESET_ALL}\n"
else:
report += f"{Fore.YELLOW}[!]{Style.RESET_ALL} No open ports found\n"
```

```
if vulnerabilities:
    report += f"""\n
{Fore.CYAN} ||=====
||=====
|| VULNERABILITY DETAILS ||
||=====
||=====
{Style.RESET_ALL}
"""\n
```

```
for vuln in vulnerabilities:
    report += f"{'':100}"
    {vuln}
```

```
{Fore.BLUE}[i] Mitigation:{Style.RESET_ALL} {vuln.mitigation}
```

```
"""
```

```
report += f"""
```

```
{Fore.CYAN} ||
```

```
|| SECURITY RECOMMENDATIONS ||
```

```
|| {Style.RESET_ALL}
```

```
"""
```

```
if vulnerabilities:
```

```
risk_levels = {"Critical": [], "High": [], "Medium": [], "Low": []}
```

```
for vuln in vulnerabilities:
```

```
risk_levels[vuln.risk_level].append(vuln)
```

```
for level, vulns in risk_levels.items():
```

```
if vulns:
```

```
if level == "Critical":
```

```
color = Fore.RED
```

```
elif level == "High":
```

```
color = Fore.YELLOW
```

```
elif level == "Medium":
```

```
color = Fore.BLUE
```

```
else:
```

```
color = Fore.GREEN
```

```
report += f"\n{color}[{level} Priority]{Style.RESET_ALL}\n"
```

```
for vuln in vulns:
```

```
report += f"- {vuln.service} (Port {vuln.port}): {vuln.mitigation}\n"
```

```
else:
```

```
report += f"{Fore.GREEN}[✓] No immediate vulnerabilities
```

```
detected.{Style.RESET_ALL}\n"
```

```
report += "Recommendation: Continue regular security audits and keep all services  
updated.\n"
```

```
return report
```

```
# Real-time visualization update function
```

```
def update_plot(frame):
```

```
plt.clf()
```

```
# Only show open ports and a few most recently scanned ports
```

```

relevant_ports = [port for port, history in port_history.items()
if any(history) or len(history) > 0]

# Sort ports to show open ones first
relevant_ports.sort(key=lambda p: -sum(port_history[p]))

# Limit to top 10 for visibility
relevant_ports = relevant_ports[:10]

if not relevant_ports:
plt.text(0.5, 0.5, "Scanning in progress...\nNo open ports detected yet.",
ha='center', va='center', fontsize=14)
return []

# Create horizontal bar chart
y_pos = range(len(relevant_ports))
colors = []
labels = []

for port in relevant_ports:
if sum(port_history[port]) > 0:
# Port is open - check if it's vulnerable
if port in VULNERABILITIES:
colors.append('red')
service = VULNERABILITIES[port].service
labels.append(f"Port {port} - {service} (VULNERABLE)")
else:
colors.append('green')
service = vuln_ports.get(port, "Unknown")
labels.append(f"Port {port} - {service}")
else:
colors.append('grey')
service = vuln_ports.get(port, "Unknown")
labels.append(f"Port {port} - {service}")

# Calculate bar values (latest status)
values = [list(port_history[port])[-1] if port_history[port] else 0 for port in
relevant_ports]

# Plot horizontal bars
bars = plt.barh(y_pos, values, align='center', alpha=0.7, color=colors)
plt.yticks(y_pos, labels)
plt.xlabel('Status (Open=1, Closed=0)')
plt.title('Real-time Port Scanning Results')

```

```
current_time = time.time()
if scan_start_time:
    elapsed = current_time - scan_start_time
    plt.figtext(0.5, 0.01, f"Time elapsed: {elapsed:.1f}s | Ports scanned: {total_ports_scanned}",
    ha="center", fontsize=10, bbox={"facecolor": "lightgrey", "alpha": 0.5, "pad": 5})
return bars
```

```
def main():
    parser = argparse.ArgumentParser(description='Network Vulnerability Scanner with Real-time Tracking')
    parser.add_argument('-t', '--target', help='Target IP address', required=True)
    parser.add_argument('-p', '--ports', help='Port range (e.g., 1-1024)', default='1-1024')
    parser.add_argument('-n', '--threads', help='Number of threads', type=int, default=100)
    parser.add_argument('--timeout', help='Connection timeout', type=float, default=1.0)
    parser.add_argument('--simulate', help='Simulate finding vulnerabilities',
                        action='store_true')
    args = parser.parse_args()

    try:
        start_port, end_port = map(int, args.ports.split('-'))
        port_range = (start_port, end_port)
    except ValueError:
        print(f"{Fore.RED}Error: Invalid port range format. Use start-end format (e.g., 1-1024){Style.RESET_ALL}")
        sys.exit(1)
```

```
plt.figure(figsize=(10, 6))
ani = FuncAnimation(plt.gcf(), update_plot, interval=500)
plt.tight_layout()
```

```
plt.ion()
plt.show()
```

[illegible]

```

print(f"\nTarget: {Fore.GREEN}{args.target}{Style.RESET_ALL}")
print(f"Port range: {Fore.GREEN}{start_port}-{end_port}{Style.RESET_ALL}")
print(f"Threads: {Fore.GREEN}{args.threads}{Style.RESET_ALL}")
print(f"Timeout: {Fore.GREEN}{args.timeout}s{Style.RESET_ALL}")

if args.simulate:
    print(f"\n{Fore.YELLOW}[!] Running in simulation mode - vulnerabilities will be injected{Style.RESET_ALL}")

    print(f"\n{Fore.CYAN}[*] Starting scan at {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}{Style.RESET_ALL}")
    print(f"{Fore.CYAN}[*] Real-time visualization window opened{Style.RESET_ALL}")
    print(f"{Fore.CYAN}[*] Scanning in progress... Please wait.{Style.RESET_ALL}")

    # If simulating, inject some "vulnerabilities"
    if args.simulate:
        def simulate_findings():
            global scan_results
            time.sleep(2) # Wait a bit to simulate scanning

            # Randomly select a few ports to mark as "open"
            vulnerable_ports = random.sample(list(VULNERABILITIES.keys()), min(3, len(VULNERABILITIES)))
            other_open_ports = random.sample(range(1, 1025), 2)

            # Add the simulated results
            for port in vulnerable_ports:
                scan_results[port] = {
                    "status": "open",
                    "service": vuln_ports.get(port, "Unknown"),
                    "vulnerability": VULNERABILITIES[port]
                }
                port_history[port].append(1)

            for port in other_open_ports:
                if port not in scan_results:
                    scan_results[port] = {
                        "status": "open",
                        "service": vuln_ports.get(port, "Unknown"),
                        "vulnerability": None
                    }
                port_history[port].append(1)

        threading.Thread(target=simulate_findings).start()

```

```

try:
# Start the actual scan
scan_duration = scan_network(args.target, port_range, args.threads, args.timeout)

# Print the report
report = generate_report(args.target, scan_duration)
print(report)

print(f"\n{Fore.CYAN}[*] Scan completed in {scan_duration:.2f}
seconds{Style.RESET_ALL}")
print(f"{Fore.CYAN}[*] Keeping visualization window open... Press Ctrl+C to
exit{Style.RESET_ALL}")

# Keep the plot open until user interrupts
plt.ioff()
plt.show()

except KeyboardInterrupt:
print(f"\n{Fore.YELLOW}[!] Scan interrupted by user{Style.RESET_ALL}")
stop_scan()
except socket.gaierror:
print(f"\n{Fore.RED}[!] Hostname could not be resolved{Style.RESET_ALL}")
except socket.error:
print(f"\n{Fore.RED}[!] Could not connect to server{Style.RESET_ALL}")
finally:
if is_scanning:
stop_scan()

if __name__ == "__main__":
main()

```

```

#python3 scanner.py -t 115.244.41.200 -p 1-1024 -n 200 --timeout 0.5 --simulate
#python3 scanner.py -t 192.168.1.1 -p 1-1024 -n 200 --timeout 0.5 --simulate
#python3 scanner.py -t 223.187.5.17 -p 1-1024 -n 200 --timeout 0.5 --simulate
#python3 scanner.py -t 172.225.201.29 -p 1-1024 -n 200 --timeout 0.5 --simulate

```

RESULTS AND DISCUSSIONS:

The tool was tested against various IP addresses, scanning common ports (1-1024) and identifying open ports and their associated vulnerabilities. The real-time plotting feature provided immediate feedback during scanning.

INPUTS/OUTPUT SCREENSHOTS:



CONCLUSION:The Network Vulnerability Scanner provides a lightweight, efficient, and user-friendly solution for detecting network vulnerabilities. Its real-time visualization and detailed reporting enhance usability and ensure comprehensive assessment of network security.

REFERENCES:

- Python Documentation
(<https://docs.python.org/3/>)(<https://docs.python.org/3/>)
- Matplotlib Documentation
(<https://matplotlib.org/stable/contents.html>)(<https://matplotlib.org/stable/contents.html>)
- Colorama Documentation
(<https://pypi.org/project/colorama/>)(<https://pypi.org/project/colorama/>)
- GitHub Repository: https://github.com/Prabhu007K/Network-Vulnerability_scanner