An Integrated Python Toolkit for Foundational Network Security Assessment

Nagasai Kandhukuri
School of Computer Science and Engineering
VIT-AP University
Amaravati, India
nagasai.kandhukuri.22bce20404@vitapstudent.ac.in

Arava Hithesh Prem
School of Computer Science and Engineering
VIT-AP University
Amaravati, India
hithesh.22bce20145@vitapstudent.ac.in

Rupesh Malisetty
School of Computer Science and Engineering
VIT-AP University
Amaravati, India
rupesh.22bce9097@vitapstudent.ac.in

Rakesh Aswanth
School of Computer Science and Engineering
VIT-AP University
Amaravati, India
aswanth.22bce7778@vitapstudent.ac.in

Dr. D. Santhadevi
Assistant Professor Sr. Gr. 1
School of Computer Science and Engineering
VIT-AP University
Amaravati, India
d.santhadevi@vitap.ac.in

Abstract—**Network security necessitates continuous assessment, yet sophisticated tools can be complex. This paper introduces an integrated, command-line network security toolkit developed in Python, designed for accessibility, foundational analysis, and educational purposes. It consolidates three core modules: (1) a Basic Vulnerability Scanner checking common ports against a predefined list of potential weaknesses; (2) a Network Traffic Analyzer leveraging Pandas and Scikit-learn (Isolation Forest) to detect common DDoS flood patterns (e.g., SYN, UDP, ICMP) and general anomalies within offline CSV traffic data; and (3) an Interactive Port Scanner providing user-configurable TCP Connect and SYN scans (using Scapy, if available) alongside basic TTL-based OS detection. The toolkit emphasizes modularity, ease of use, and demonstrates the application of standard Python libraries combined with specialized packages like Scapy for practical network security tasks. It serves as a platform for preliminary security exploration and understanding fundamental network assessment techniques.**

Index Terms—**Network Security, Port Scanning (Technique), Vulnerability Assessment (Technique), DDoS Detection (Technique), Anomaly Detection (Technique), Traffic Analysis (Technique), Python (Mechanism), Scapy (Mechanism), Socket Programming (Mechanism), Multithreading (Mechanism), Thresholding (Mechanism), Isolation Forest (Mechanism), Vulnerability Scanner (Module), Traffic Analyzer (Module), Port Scanner (Module).**

I. Introduction

The ubiquitous nature of computer networks makes their security paramount for individuals, businesses, and critical infrastructure. Threats such as unauthorized access, data breaches, denial-of-service attacks, and malware propagation pose significant risks [1]. Effective network defense requires ongoing monitoring, assessment, and the application of appropriate security tools and practices.

A wide array of powerful network security tools exists. Nmap is the de facto standard for network discovery and port scanning. Wireshark provides deep packet inspection capabilities for traffic analysis. Intrusion Detection Systems (IDS) like Snort or Suricata monitor traffic for malicious patterns in real-time. While highly capable, these tools are often specialized, may possess steep learning curves, or can be resource-intensive [2]. Furthermore, integrating findings across multiple disparate tools can be cumbersome for initial assessments or educational exploration.

To address the need for an accessible, integrated platform for fundamental network security tasks, we developed a command-line toolkit entirely in Python. This toolkit combines three essential functionalities into a single, user-friendly interface (see Fig. 1):

1) **Basic Vulnerability Scanner:** Performs rapid checks on common TCP ports, mapping open ports to a static list of potential, high-level vulnerabilities often associated with those services.
2) **Network Traffic Analyzer:** Analyzes pre-captured network traffic data (CSV format) to detect common Distributed Denial of Service (DDoS) flood attacks using heuristic thresholds and identifies general anomalies using the Isolation Forest machine learning algorithm.
3) **Interactive Port Scanner:** Allows users to perform targeted TCP Connect or SYN scans (if Scapy is available and permissions allow) on specific port ranges, including a basic OS detection feature based on ICMP TTL analysis.

This integration facilitates a streamlined workflow for preliminary network checks and provides a practical environment for learning core network security concepts.

The primary objectives of this work are:

- To design and implement an integrated, multi-functional network security toolkit using Python.
- To demonstrate the combination of Python's standard libraries (e.g., ‘socket‘, ‘threading‘) with external packages (e.g., ‘pandas‘, ‘scikit-learn‘, ‘scapy‘) for practical security assessment tasks.

- To provide an accessible tool suitable for educational purposes and basic network reconnaissance and analysis.
- To implement and evaluate specific techniques including different port scanning methods (TCP Connect, SYN), heuristic-based DDoS detection, and unsupervised anomaly detection in network traffic data.

This paper details the toolkit's design, implementation, and functionality. Section II reviews related work. Section III describes the methodology and implementation of each module. Section IV discusses the toolkit's capabilities and limitations, and Section V concludes the paper.

II. Literature Review

This section reviews existing literature relevant to the techniques and tools employed in our toolkit.

A. Port Scanning

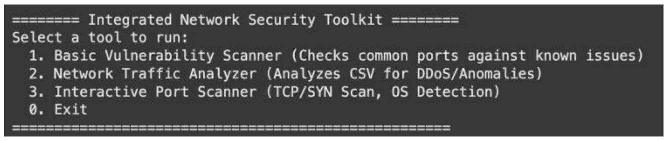
Port scanning is a fundamental technique for network reconnaissance, identifying open ports and potential services running on a target host. Nmap is the most comprehensive open-source scanner, offering numerous scan types, including TCP Connect (-sT), SYN (-sS), UDP (-sU), and various stealth techniques, along with OS detection and service version identification. Our toolkit implements the basic TCP Connect and SYN scans. TCP Connect completes the three-way handshake and is reliable but easily logged. SYN scan, often called ”half-open” scanning, sends only a SYN packet and analyzes the response (SYN/ACK for open, RST for closed), making it faster and stealthier but requiring raw socket privileges [3]. While effective, our implementation lacks the advanced timing, evasion, and scripting capabilities of Nmap.

B. DDoS Detection

Distributed Denial of Service (DDoS) attacks aim to overwhelm target resources. Common types include volumetric floods like SYN floods, UDP floods, and ICMP floods [4]. Detection mechanisms range from simple thresholding (flagging traffic exceeding predefined packet rates or connection counts) to statistical analysis and machine learning approaches [5]. Thresholding, as used in our traffic analyzer for specific flood types, is computationally inexpensive but requires careful tuning and can be prone to false positives/negatives [6]. Our implementation uses static thresholds based on packet counts per source IP for various protocols/patterns identified in offline CSV data.

C. Network Traffic Analysis and Anomaly Detection

Analyzing network traffic is crucial for security monitoring and forensics. Tools like Wireshark and tcpdump allow detailed packet capture and inspection. For automated analysis of large datasets, techniques often involve feature extraction and statistical or machine learning models [7]. Our toolkit analyzes CSV exports, extracting basic features like packet length, protocol, ports, and flow duration. For general anomaly detection, we employ the Isolation Forest algorithm , [8]. It is an unsupervised method effective at identifying outliers (anomalies) in data without prior labeling, making it suitable for finding unusual patterns that might not fit specific DDoS signatures [9]. However, its effectiveness depends on appropriate feature selection and interpretation of results, as statistical outliers are not always malicious.



```
======== Integrated Network Security Toolkit ========
Select a tool to run:
  1. Basic Vulnerability Scanner (Checks common ports against known issues)
  2. Network Traffic Analyzer (Analyzes CSV for DDoS/Anomalies)
  3. Interactive Port Scanner (TCP/SYN Scan, OS Detection)
  0. Exit
=================================================
```

Fig. 1. The main menu of the Integrated Network Security Toolkit, presenting the three tool options and exit command.

D. Python in Network Security

Python's extensive standard library (e.g., ‘socket‘) and powerful third-party packages make it popular for network programming and security tasks. Scapy is a cornerstone library, enabling packet crafting, manipulation, sending, and sniffing, essential for tasks like SYN scanning and custom protocol interaction. Pandas is invaluable for data analysis, particularly with structured data like CSV traffic logs. Scikit-learn provides a wide range of machine learning algorithms applicable to security problems like anomaly detection. Our toolkit leverages these libraries to provide its integrated functionality, demonstrating Python's suitability for developing such tools.

Our work contributes by integrating these diverse techniques (scanning, heuristic detection, ML anomaly detection) into a single, accessible Python tool, primarily aimed at education and preliminary assessments, distinguishing it from highly specialized, professional-grade standalone applications.

III. Methodology and Implementation

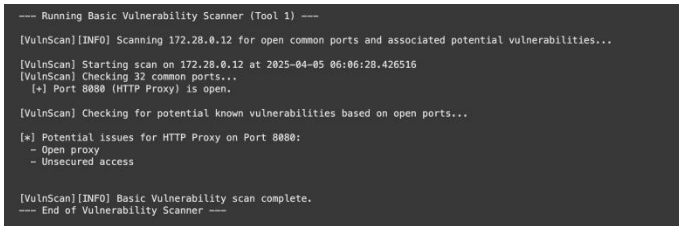
This section details the design, algorithms, and implementation of the three core modules within the integrated toolkit.

A. Toolkit Architecture and Environment

The toolkit operates as a command-line application driven by a text-based menu, as shown in Fig. 1. It is implemented in Python 3 and utilizes standard libraries along with Pandas, Scikit-learn, and optionally Scapy. Execution requires Python 3, and relevant libraries installed via pip (`pip install pandas scikit-learn scapy matplotlib`). Advanced features (SYN scan, OS detection) require Scapy and root/administrator privileges. The code is structured modularly, with dedicated functions for each tool and sub-task. Error handling is included for file operations, network issues, and permissions.

B. Tool 1: Basic Vulnerability Scanner

1) Methodology: This module performs a quick scan against a predefined dictionary (`COMMON_PORTS_VULN`) mapping common TCP port numbers (e.g., 21, 80, 443, 3306) to potential associated service names and generic vulnerability



```
--- Running Basic Vulnerability Scanner (Tool 1) ---

[VulnScan][INFO] Scanning 172.28.0.12 for open common ports and associated potential vulnerabilities...

[VulnScan] Starting scan on 172.28.0.12 at 2025-04-05 06:06:28.426516
[VulnScan] Checking 32 common ports...
  [+] Port 8080 (HTTP Proxy) is open.

[VulnScan] Checking for potential known vulnerabilities based on open ports...

[*] Potential issues for HTTP Proxy on Port 8080:
  - Open proxy
  - Unsecured access


[VulnScan][INFO] Basic Vulnerability scan complete.
--- End of Vulnerability Scanner ---
```

Fig. 2. Example output from the Basic Vulnerability Scanner (Tool 1), showing detection of an open port (8080) and associated potential vulnerabilities.

descriptions (e.g., ”FTP” - ”Anonymous access”, ”Plain-text passwords”). It iterates through the ports in this dictionary, attempting a TCP connection to the target IP using `socket.socket` and `connect_ex()` with a 1-second timeout. If `connect_ex()` returns 0, the port is considered open.

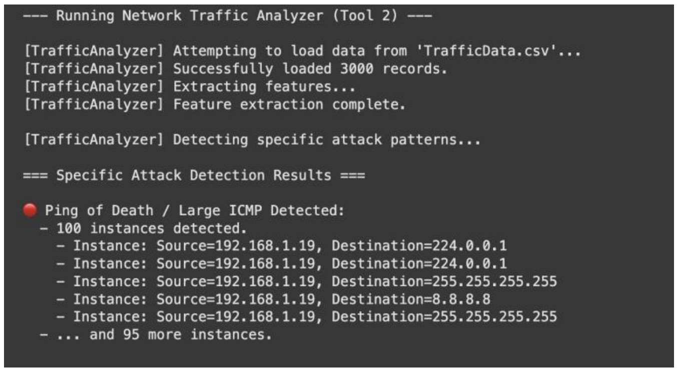
2) Implementation: The `vuln_scan_port()` function handles the connection attempts. If a port is open, it retrieves the corresponding service description and potential vulnerabilities from the `COMMON_PORTS_VULN` dictionary. It also tries to resolve the official service name using `socket.getservbyport()`. The `check_vulnerabilities()` function formats and prints the findings for open ports. Fig. 2 shows typical output from this module, highlighting an open port and its associated potential issues.

3) Limitations: This scanner is basic: it relies on a static, predefined list of ports and generic vulnerabilities, performs no version detection or actual vulnerability confirmation, and only scans TCP ports listed in its internal dictionary.

C. Tool 2: Network Traffic Analyzer

1) Methodology: This tool analyzes network traffic data from a CSV file. It first loads the data using Pandas (`load_csv()`), performing basic cleaning (filling NaNs). It then extracts features (`extract_features()`) potentially useful for detecting attacks, including packet length, source/destination ports (heuristically extracted), protocol type, and flow duration (time delta between packets from the same source). DDoS detection relies on heuristics: counting packets per source IP for specific protocols (UDP, ICMP, DNS) or patterns (TCP SYN flags in ’Info’, HTTP methods, specific ports like 53, 80, 443) and comparing against hardcoded thresholds (e.g., `detect_udp_flood()`, `detect_syn_flood()`). It also checks for oversized ICMP packets (Ping of Death) and long-duration TCP flows (potential Slowloris). General anomalies are detected using Scikit-learn’s `IsolationForest` (`detect_anomalies()`) trained on numerical features (packet length, flow duration, ports).

2) Implementation: Functions like `detect_udp_flood()`, `detect_syn_flood()`, `detect_slowloris()`, etc., implement the specific heuristic checks by filtering the Pandas DataFrame and counting occurrences per source. `detect_anomalies` selects numeric columns, trains the `IsolationForest`



```
--- Running Network Traffic Analyzer (Tool 2) ---

[TrafficAnalyzer] Attempting to load data from 'TrafficData.csv'...
[TrafficAnalyzer] Successfully loaded 3000 records.
[TrafficAnalyzer] Extracting features...
[TrafficAnalyzer] Feature extraction complete.

[TrafficAnalyzer] Detecting specific attack patterns...

=== Specific Attack Detection Results ===

🔴 Ping of Death / Large ICMP Detected:
  - 100 instances detected.
    - Instance: Source=192.168.1.19, Destination=224.0.0.1
    - Instance: Source=192.168.1.19, Destination=224.0.0.1
    - Instance: Source=192.168.1.19, Destination=255.255.255.255
    - Instance: Source=192.168.1.19, Destination=8.8.8.8
    - Instance: Source=192.168.1.19, Destination=255.255.255.255
  - ... and 95 more instances.
```

Fig. 3. Example output from the Network Traffic Analyzer (Tool 2), indicating detection of numerous large ICMP packets potentially related to a Ping of Death attack.

model, predicts anomalies (score=-1), and optionally visualizes the score distribution using Matplotlib and saves anomalies to a file. Fig. 3 illustrates the detection of a potential Ping of Death attack pattern.

3) Limitations: Analysis is offline and depends entirely on the input CSV’s quality and format. Heuristic thresholds are static and may require tuning. Feature extraction (especially ports) is basic. Anomaly detection identifies statistical outliers, which require context for interpretation.

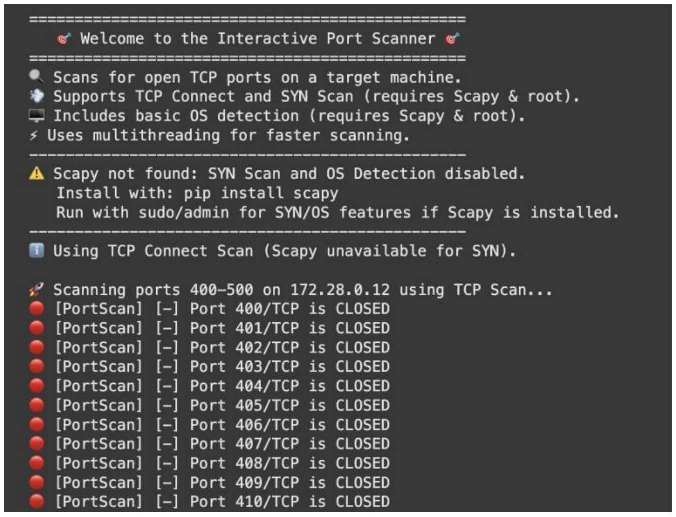
D. Tool 3: Interactive Port Scanner

1) Methodology: This module allows interactive scanning of a user-defined TCP port range on a target IP. Users can select between:

- **TCP Connect Scan:** Uses `socket.connect_ex()` to establish a full connection. Reliable but easily logged.
- **SYN Scan:** Requires Scapy and root privileges. Sends a TCP SYN packet (e.g., `IP(dst=target)/TCP(dport=port, flags=’S’)`) and analyzes the response flags (SYN/ACK for open, RST/ACK for closed) using `sr1()`. It sends an RST packet upon receiving SYN/ACK to close the connection prematurely.

Basic OS detection (optional, requires Scapy/root) sends an ICMP echo request (e.g., `IP(dst=target)/ICMP()`) and analyzes the TTL of the reply to make a broad guess (e.g., TTL ≤ 64 Linux/Unix, 64 TTL ≤ 128 Windows). Multithreading (`threading.Thread`) is used to scan ports concurrently for speed, with a `Lock` protecting shared access to the `results` dictionary.

2) Implementation: The `run_interactive_scanner()` function handles user input (target, ports, scan type, verbosity). Based on the choice and Scapy availability, it assigns either `scan_port_tcp()` or `scan_port_syn()` as the target function for worker threads. Each thread scans a single port. The `detect_os()` function implements the TTL-based fingerprinting. Results (open ports, service names, OS guess) are collected and can be saved to JSON and TXT files (`save_results()`). Fig. 4 shows the scanner in action, indicating closed ports during a TCP Connect scan when Scapy is unavailable.



```
==============================================
    🎯 Welcome to the Interactive Port Scanner 🎯
==============================================
🔍 Scans for open TCP ports on a target machine.
💨 Supports TCP Connect and SYN Scan (requires Scapy & root).
🖥 Includes basic OS detection (requires Scapy & root).
⚡ Uses multithreading for faster scanning.
----------------------------------------------
⚠️ Scapy not found: SYN Scan and OS Detection disabled.
   Install with: pip install scapy
   Run with sudo/admin for SYN/OS features if Scapy is installed.
----------------------------------------------
ℹ️ Using TCP Connect Scan (Scapy unavailable for SYN).

🚀 Scanning ports 400-500 on 172.28.0.12 using TCP Scan...
🔴 [PortScan] [-] Port 400/TCP is CLOSED
🔴 [PortScan] [-] Port 401/TCP is CLOSED
🔴 [PortScan] [-] Port 402/TCP is CLOSED
🔴 [PortScan] [-] Port 403/TCP is CLOSED
🔴 [PortScan] [-] Port 404/TCP is CLOSED
🔴 [PortScan] [-] Port 405/TCP is CLOSED
🔴 [PortScan] [-] Port 406/TCP is CLOSED
🔴 [PortScan] [-] Port 407/TCP is CLOSED
🔴 [PortScan] [-] Port 408/TCP is CLOSED
🔴 [PortScan] [-] Port 409/TCP is CLOSED
🔴 [PortScan] [-] Port 410/TCP is CLOSED
```

Fig. 4. Example output from the Interactive Port Scanner (Tool 3) running a TCP Connect scan. It shows the scan progress and indicates closed ports. The warning about Scapy being unavailable is also visible.

3) Limitations: SYN scan and OS detection depend on Scapy and elevated privileges. OS detection is rudimentary and unreliable. Lacks UDP scanning and advanced firewall evasion techniques. Performance might lag behind optimized tools like Nmap for large scans.

IV. Evaluation and Discussion

The developed toolkit successfully integrates three fundamental network security assessment functionalities into a single Python application.

A. Strengths

- **Integration and Accessibility:** Provides a unified interface for diverse tasks, lowering the barrier to entry compared to using multiple specialized tools. Python implementation enhances accessibility for understanding and modification.
- **Educational Utility:** Serves as a practical tool for learning about port scanning mechanisms (TCP vs. SYN), DDoS attack signatures, traffic analysis concepts, and the application of libraries like Scapy and Pandas.
- **Modularity:** The distinct separation of the three tools allows for independent use and facilitates future extensions.
- **Combined Analysis Approach:** The traffic analyzer employs both specific heuristic detection for known patterns and general anomaly detection for potentially unknown threats.

B. Weaknesses and Limitations

Despite its utility, the toolkit has notable limitations:

- **Superficiality:** Vulnerability assessment is based only on open ports, not actual configuration or software versions. OS detection is basic.
- **Heuristic Fragility:** DDoS detection thresholds are static and environment-dependent. Feature extraction from CSV is simplified.
- **Offline Analysis:** Traffic analysis operates on static CSV files, not live traffic.
- **Scalability and Performance:** Python's performance, even with threading, may not match optimized C-based tools for large-scale scanning or analysis of massive datasets.
- **Dependencies and Privileges:** Advanced features rely on Scapy and root access, which may not always be available or permissible.
- **Evasion:** Lacks mechanisms to bypass sophisticated firewalls or intrusion detection systems.

C. Use Cases

The toolkit is well-suited for:

- Network security education and training environments.
- Initial reconnaissance and basic assessment of small networks or lab setups.
- Quick analysis of packet capture CSV exports for obvious flood attacks or anomalies.
- A starting point or framework for developing more customized network security scripts.

D. Future Work

Potential enhancements include:

- Adding service version detection and banner grabbing (Tool 1 & 3).
- Integrating with external vulnerability databases (e.g., CVE lookup).
- Implementing real-time traffic analysis using Scapy/Pcapy (Tool 2).
- Adding support for more DDoS types and refining detection thresholds/models (Tool 2).
- Incorporating UDP scanning and more robust OS fingerprinting (Tool 3).
- Developing a graphical user interface (GUI).
- Allowing configuration of parameters (ports, thresholds) via external files.

V. Conclusion

We have presented an integrated network security toolkit developed in Python, combining basic vulnerability scanning, network traffic analysis for DDoS/anomalies, and interactive port scanning capabilities. By leveraging standard Python libraries alongside powerful packages like Pandas, Scikit-learn, and Scapy, the toolkit provides an accessible platform for performing foundational security assessments and serves as a valuable educational resource. While lacking the depth and robustness of professional-grade, specialized tools, its integration, ease of use, and modular design offer a practical starting point for network exploration and analysis. Future work can address current limitations by incorporating more advanced techniques, improving performance, and enhancing user interaction.

References

[1] M. Ahmed, A. N. Mahmood, and J. Hu, ”A survey of network anomaly detection techniques,” *Journal of Network and Computer Applications*, vol. 60, pp. 19-31, 2016.

[2] I. Ghafir et al., ”Security threats to critical infrastructure: the human factor,” *Journal of Supercomputing*, vol. 74, no. 10, pp. 4986-5002, 2018.

[3] Z. Durumeric, M. Bailey, and J. A. Halderman, ”An Internet-wide view of Internet-wide scanning,” *USENIX Security Symposium*, pp. 65-78, 2014.

[4] S. T. Zargar, J. Joshi, and D. Tipper, ”A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks,” *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046-2069, 2013.

[5] T. Hirakawa, K. Ogura, B. B. Bista, and T. Takata, ”A defense method against distributed slow HTTP DoS attack,” *19th International Conference on Network-Based Information Systems*, pp. 152-158, 2016.

[6] A. Marnerides, A. Schaeffer-Filho, and A. Mauthe, ”Traffic anomaly detection and classification in virtual networks,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 752-766, 2016.

[7] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, ”A detailed analysis of the KDD CUP 99 data set,” *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1-6, 2009.

[8] F. T. Liu, K. M. Ting, and Z. H. Zhou, ”Isolation Forest,” *IEEE International Conference on Data Mining*, pp. 413-422, 2008.

[9] V. Chandola, A. Banerjee, and V. Kumar, ”Anomaly detection: A survey,” *ACM Computing Surveys*, vol. 41, no. 3, pp. 1-58, 2009.

[10] A. X. Liu, ”Firewall policy verification and troubleshooting,” *IEEE Transactions on Network and Service Management*, vol. 7, no. 4, pp. 231-241, 2010.

[11] D. Kennedy, J. O’Gorman, D. Kearns, and M. Aharoni, *Metasploit: The Penetration Tester’s Guide*, No Starch Press, 2011.

[12] R. Sommer and V. Paxson, ”Outside the closed world: On using machine learning for network intrusion detection,” *IEEE Symposium on Security and Privacy*, pp. 305-316, 2010.

[13] S. Khan, A. Gani, A. W. A. Wahab, M. Shiraz, and I. Ahmad, ”Network forensics: Review, taxonomy, and open challenges,” *Journal of Network and Computer Applications*, vol. 66, pp. 214-235, 2016.

[14] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, ”Surveying port scans and their detection methodologies,” *The Computer Journal*, vol. 54, no. 10, pp. 1565-1581, 2011.

[15] C. Yin, Y. Zhu, J. Fei, and X. He, ”A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21954-21961, 2017.

[16] B. Rhodes and J. Goerzen, *Foundations of Python Network Programming*, 3rd ed., Apress, 2014.

[17] S. S. Kolahi, K. Treseangrat, and B. Sarrafpour, ”Analysis of UDP DDoS flood cyber attack and defense mechanisms on web server with Linux Ubuntu 13,” *International Conference on Communications, Signal Processing, and their Applications*, pp. 1-5, 2015.

[18] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, ”Robust network traffic classification,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1257-1270, 2015.

[19] R. Beverly, ”A robust classifier for passive TCP/IP fingerprinting,” *International Workshop on Passive and Active Network Measurement*, pp. 158-167, 2004.

[20] H. Holm, T. Sommestad, J. Almroth, and M. Persson, ”A quantitative evaluation of vulnerability scanning,” *Information Management Computer Security*, vol. 19, no. 4, pp. 231-247, 2011.