JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Development and Evaluation of a Lightweight Python and Scapy-Based Network Intrusion Detection System for Real-Time Detection of Network Attacks

[1]Vrishab.S.Ganganaboina, [2]Mr. Rashid Patel
[1]MSc CS specialization in cybersecurity, [2]Assistant Professor
[1,2] Department of Advanced Computing
[1,2] Nagindas Khandwala College
[1] University of Mumbai, Maharashtra, India

Abstract - Network Intrusion Detection Systems (NIDS) are used to monitor network traffic and identify suspicious or malicious activity. This research presents a lightweight, transparent NIDS implemented in Python using the Scapy packet-processing framework. The system captures live network traffic, preprocesses packet information, applies protocol-specific rules and configurable thresholds, generates severitybased alerts, stores events in SQLite and log files, and supports whitelist/blacklist response handling. The implementation focuses primarily on real-time detection of TCP SYN scanning, TCP Connect scanning, FIN/NULL scanning and threshold-based port-scanning behaviour, with additional tests for ICMP flooding, TCP SYN flooding, FTP connection activity and repeated SSH connection attempts. Controlled testing is performed in a laboratory network using Kali Linux and Windows/VMware hosts. Implementation findings show that reliable NIDS operation depends not only on detection conditions but also on packet-path validation, event ordering, whitelist precedence, persistent logging and correct response handling. The study therefore contributes a reproducible and explainable network-monitoring framework for cybersecurity education and controlled experimentation, while recognizing that the current evaluation does not establish production-scale detection accuracy.

Keywords: NIDS, Python, Scapy, port scanning, blacklist, whitelist.

1.Introduction

Modern computer networks are exposed to reconnaissance, unauthorized access attempts, denial-of-service activity, brute-force attempts and other forms of malicious traffic. A networkbased intrusion detection system monitors traffic at the network boundary or on a monitored interface and identifies patterns that may indicate an attack. NIDS research includes signaturebased, anomaly-based and learning-based approaches, with ongoing concerns involving deployment cost, explainability, data quality and real-time operation [1], [3], [7], [8].

For a practical cybersecurity laboratory, a transparent and lightweight implementation can be useful because detection logic can be inspected, modified and tested without training a machine-learning model. Python-based IDS research demonstrates the feasibility of implementing packet analysis with tools such as Scapy, while recent real-time traffic-analysis work also uses Scapy for continuous packet capture and heuristic detection [1], [3], [8].

Port scanning is an important reconnaissance activity because it can reveal reachable services and potential attack surfaces. Rule-based portscan research has shown that traffic counts, thresholds and state information can be used to identify scanning behaviour in real time [4]. Threshold selection is itself an important design issue because network conditions vary and static thresholds may require tuning [15]. The present work therefore focuses on a configurable, rule- and threshold-based NIDS that prioritizes TCP SYN, TCP Connect, FIN/NULL and thresholdbased port-scan detection, while also providing additional protocol-specific attack-detection modules, alert persistence and response handling.

2. Problem Statement

Many modern NIDS studies concentrate on machine learning or deep learning, which can require datasets, preprocessing pipelines, computational resources and careful evaluation [2], [5], [6]. Conversely, simple rule-based systems can be transparent but may be limited by manually selected conditions, short observation windows and rule maintenance. The practical problem addressed in this project is how to build a lightweight NIDS that remains understandable and configurable while detecting common portscanning behaviours and selected network attack behaviours in real time.

The project specifically addresses the need for a unified implementation that captures packets with Scapy, applies protocol-specific and threshold-based detection, records alerts persistently, and supports whitelist/blacklist handling. The system is designed for controlled laboratory validation rather than as a claim of production-scale detection performance.

3. Objectives

- Design and implement a real-time NIDS using Python and Scapy.
- Develop configurable rules and thresholds for multiple network attack behaviours.
- Detect TCP SYN scanning, TCP Connect scanning, FIN/NULL scanning and threshold-based port-scanning behaviour.
- Generate severity-based alerts and maintain traceable event records.
- Persist alerts and logs using SQLite and a text log file.
- Implement whitelist precedence and blacklist-based response handling.
- Validate detection through controlled traffic generation and packet-level verification.
- Provide a modular architecture that can be extended with adaptive thresholds, anomaly detection and additional monitoring features.

4. Literature Review

The literature was selected around four themes: Python/Scapy-based practical NIDS implementations [1], [3], [8]; rule- and threshold-based port-scan detection [4], [9], [15]; broader NIDS architectures, datasets and evaluation [5], [6], [7], [10], [12], [13]; and signature/rule modification and controlled attack-test methodology [14]. The reviewed work informs the design while also showing that the present project should be positioned as a practical integration and implementation study rather than as a new machine-learning algorithm.

Ref	Paper/Authors	Key findings	Project implementation gap
[1]	Hariyani et al. (2025)	Python-based IDS using Scapy/Pyshark demonstrates practical packet inspection.	Supports the Python implementation; this project emphasizes modular rules, thresholds, persistence and response.
[2]	Chinnasamy et al. (2025)	Systematic review highlights datasets, preprocessing and evaluation challenges in learningbased NIDS.	Motivates a lightweight, reproducible alternative for controlled laboratory testing.
[3]	Tamrakar et al. (2026)	Real-time traffic analysis uses Scapy, heuristic detection, packet capture and a monitoring dashboard.	Supports lightweight real-time Scapy analysis; this project focuses on configurable protocolspecific rules and SQLite/log persistence.
[4]	Kanlayasiri et al. (2000)	Rule-based state/threshold techniques can detect port scanning in real time.	Provides the conceptual basis for portcount and time-window detection.
[5]	Goldschmidt & Chudá (2025)	NIDS datasets vary in realism, quality and suitability for specific evaluations.	Supports controlled packet generation and explicit testcase documentation.
[6]	Z. Xu, et al. (2025)	Surveys deep-learning architectures and approaches used in intrusion detection systems.	Future integration of anomaly/ML detection.
[7]	N. Kalpani, N. Rodrigo, et al .(2025)	Reviews modern intelligent techniques for improving intrusion detection.	Future scope for intelligent detection techniques.
[8]	Wahal, Choudhury & Arora (2018)	Python-based IDS work demonstrates an implementationoriented approach to intrusion detection.	Supports practthe Python directthe present syswith extends and multiple modu persistent response handling.
[9]	C. Birkinshaw, E. Rouka & V. G. Vasilakis (2019)	Presents techniques for detecting port scanning and DoS attacks using real-time thresholdbased methods.	Real-time and detection threshold tuning.
[10]	Ahmed et al. (2009)	Network-based monitoring can detect attacks by inspecting	Supports the network-level monitoring

		network traffic rather than endpoint activity.	model used in the project.
[11]	Arifin et al.	Their work highlights the importance of data characteristics and class imbalance when developing and evaluating learningbased IDS solutions.	This is relevant to the broader evaluation challenges of NIDS and provides context for future datasetbased extensions of the proposed system
[12]	Oroian et al. (2024)	Anomaly NIDS requires capture, feature processing and structured evaluation.	Provides comparison with anomalybased workflows; the present system uses explici t rules instead.
[13]	Pittman (2023)	Port-scan detection research includes machinelear ning approaches and challenges in evaluation.	Supports portscan taxonomy and the need for realistic validation.
[14]	Guide et al. (2023)	Changes to signature IDS rules can alter detection behaviour and performance.	Supports careful rule design and threshold tuning.
[15]	Kim, Lee & Seo (2010)	Python-based IDS work demonstrates an implementation oriented approach to intrusion detection.	Supports the practical Python direction; the present system extends it with multiple modules and persistent response handling.

Table 4.1: Literature Review

and response logic in one reproducible laboratory framework [1], [3], [8].

The gap addressed by this research is therefore practical rather than a claim of a new detection theory: a lightweight NIDS is implemented so that packet capture, preprocessing, rule evaluation, alert generation, whitelist/blacklist handling, logging and database persistence operate as one traceable workflow. The project also documents implementation issues such as packet-path validation and response ordering that can be overlooked when detection is evaluated only from configuration settings.

5.1 Research Contribution

The main contribution is the integration of a Python/Scapy packet-capture layer with configurable rule and threshold modules, a severity-based alert manager, SQLite and textlog persistence, and whitelist/blacklist response handling. The primary research focus is portscan detection,

5. Research Gap

The reviewed literature shows that no single approach simultaneously optimizes simplicity, transparency, configurability and practical multiattack validation. Learning-based NIDS research can involve substantial data and evaluation requirements, while rule-based detection is easier to interpret but depends on carefully selected conditions and observation windows [4], [15], [2], [5], [12], [7], [6]. Practical Python/Scapy systems demonstrate that realtime packet inspection is feasible, but there remains value in a compact implementation that combines multiple protocol-specific rules, configurable thresholds, persistent alert storage

covering TCP SYN, TCP Connect, FIN/NULL and threshold-based scanning behaviour. Additional controlled tests cover ICMP flooding, TCP SYN flooding, FTP connection activity and repeated SSH connection attempts. The system is designed to be transparent: each detection condition can be inspected and modified without retraining amodel.

6. Proposed System

The proposed NIDS continuously monitors a selected network interface using Scapy. Captured packets are normalized into features such as source IP, destination IP, protocol, destination port, packet type, TCP flags and timestamp. The detection engine applies protocol-specific rules and threshold/window conditions, with primary emphasis on TCP SYN, TCP Connect,
FIN/NULL and multi-port scanning behaviour. When a condition is satisfied, an alert is assigned a severity, checked against the whitelist, optionally

handled through the blacklist/response mechanism, and persisted to SQLite and the alert log. Monitoring and reporting components can then present the stored events for investigation.

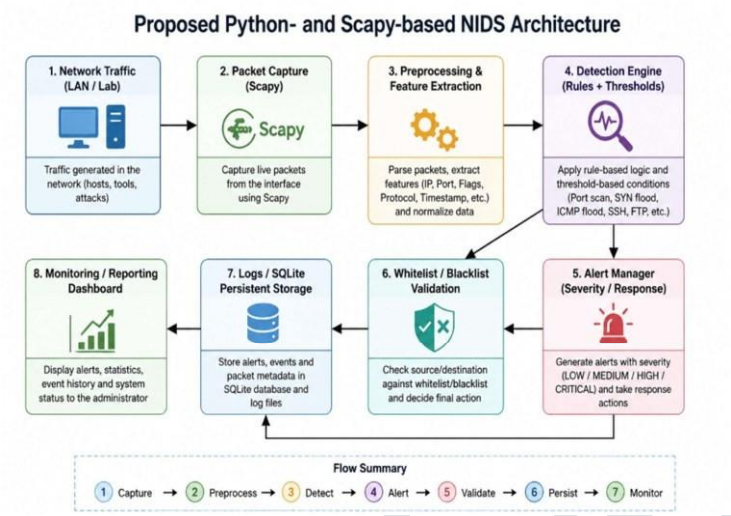

Figure 6.1: Proposed Python- and Scapy-based NIDS architecture.

Module	Main responsibility	Key implementation elements
Packet Capture	Capture live network packets from the monitored interface.	Scapy sniffing, interface selection, packet filtering
Preprocessing	Extract and normalize packet attributes for detection.	IP, protocol, ports, flags, timestamps, packet counts
Detection Engine	Evaluate attackspecific conditions.	Port-scan, ICMP, SYN, FTP and SSH rules/thresholds
Rule Manager	Store and tune detection conditions.	Threshold values, observation windows, severity
Alert Manager	Create and classify alerts.	Message, source/desti timestamp, nation, severity
Whitelist/Blacklist	Control trusted and blocked sources.	Whitelist precedence, blacklist updates and response
Database & Log Manager	Persist events for traceability.	SQLite database and alerts.log
Monitoring/Reporting	Display and review detected events.	Alert summaries, event history and reporting interface

Table 6.2: Module Specification

Detection module	Observed condition	Example threshold/window	Severity used in testing
TCP SYN scan	SYN packets directed to multiple destination ports from a source	Configurable; validated through port-scan tests	HIGH
TCP Connect scan	Repeated TCP connection attempts to multiple destination ports	Configurable; validated through port-scan tests	HIGH
FIN scan	FIN packets directed to multiple destination ports withou t normal connection establishment	Configurable	HIGH
NULL scan	TCP packets with no control flags directed to multiple destination ports	Configurable	HIGH
Port scanning	Multiple distinct destination ports observed from one source	20 ports within 30 seconds	HIGH
ICMP flood	Repeated ICMP echo requests	200 packets within 5 seconds	HIGH
SYN flood	Repeated TCP SYN packets	100 packets within 10 seconds	CRITICAL
FTP connection	Connection attempt to TCP port 21	Protocol/port condition	HIGH
Repeated SSH connection attempts	Repeated traffic toward TCP port 22	15 events within 60 seconds	HIGH

Table 6.2: Detection Conditions

7. Methodology

The methodology follows a controlled implementation-and-validation cycle. First, the project requirements were mapped to NIDS functions and attack behaviours. Second, the Python/Scapy environment and monitored interface were configured. Third, packetprocessing and detection modules were implemented and connected to the SQLite/logging layer. Fourth, controlled attack traffic was generated. Finally, alerts, database records and packet reachability were inspected to determine whether each test produced the expected result [1], [3], [4], [8], [15].

- Requirement analysis: identify attack behaviours, required packet features,

alert fields, storage requirements and response logic.

- Environment setup: configure Python, Scapy, SQLite and the monitored network interface on the Kali/VMware laboratory environment.
- Rule development: implement protocolspecific conditions and configurable thresholds/windows.
- Attack simulation: generate controlled TCP SYN, TCP Connect, FIN/NULL and threshold-based port-scanning traffic as the primary tests, with additional ICMP, SYN, FTP and SSH-related traffic tests using safe laboratory hosts and tools.
- Detection and response: capture packets, evaluate rules, generate severity-based alerts, apply whitelist/blacklist logic and persist events.
- Evaluation: compare expected and observed events, verify packet-path reachability and inspect stored records for traceability.

No.	Attack Type	Description	Detection Criteria
1	Port Scan	Multiple destination ports from one source	Port-count threshold within time window
2	ICMP Flood	Repeated ICMP echo requests	ICMP count threshold within time window
3	SYN Flood	High-rate TCP SYN traffic	SYN count threshold within time window
4	FTP Connection	Connection attempts to TCP/21	Protocol/portspecific alert condition
5	SSH Brute Force	Repeated traffic towards TCP/22	SSH threshold/window for repeated attempts.

Table 7.1: Controlled Attack Test Cases

8. Implementation and Experimental Findings

The implementation and troubleshooting phase produced several findings that are important for a practical NIDS. These findings describe the observed behaviour of the implemented system and are not presented as statistical performance results.

- FTP connection attempts to TCP port 21 were implemented as a dedicated protocol/port condition. Matching events were assigned HIGH severity and persisted in the alert log and SQLite database.
- Repeated traffic toward SSH port 22 was evaluated using a configurable threshold and time window. In the tested configuration, 15 events within 60 seconds could generate a HIGH alert; this condition identifies repeated SSH-related traffic rather than proving failed authentication.
- Port-scan detection used the number of distinct destination ports observed from a source within a defined time window. A threshold of 20 ports within 30 seconds was used in the test configuration.
- ICMP and SYN flooding conditions were implemented as repeated-packet thresholds. The tested configurations used 200 ICMP packets in 5 seconds and 100 SYN packets in 10 seconds respectively.
- Automatic blacklisting was associated with HIGH/CRITICAL events. Testing showed that response ordering matters because a newly written blacklist entry may affect subsequent packets rather than the packet that triggered the update.
- Whitelist processing was placed before blacklist enforcement and attack response so trusted sources are not incorrectly blocked.
- Persisting alerts in both a text log and SQLite improved traceability compared with console-only output.
- Packet-capture testing demonstrated that interface selection and packet reachability must be verified before interpreting a missing alert as a rule failure. A configured threshold is not evidence of detection unless the corresponding packets are actually observed on the monitored interface.

Step No.	Stage	Description	Key Activity / Actions
1	Event Detection	Packets / Traffic matched with rules & threshold.	Analyze captured packets / Match with detection rules and thresholds.
2	Alert Generation	Create alert with severity level (LOW / MEDIUM / HIGH / CRITICAL).	Generate alerts / Include relevant packet/traffic details

3	Whitelist / Blacklist check	Validate source and destination against whitelist and blacklist. Decide whether to allow, block or continue.	Check IP addresses against whitelist / Blacklist
4	Alert Classification	Classify alert based on severity and type, Determine priority for response.	Categorize alert (e.g., port scan, ICMP flood, etc.)
5	Response Decision	Select appropriate action based on Policy.	Choose response action / Apply predefined security policy
6	Response Action	Execute the selected action. Update internal state and counters.	Block IP, drop connection, rate limit, notify admin, etc. / Update system state and counters.
7	Monitoring & Review	Administrator reviews alerts in the dashboard.	View alerts an event history / Monitor system performance and effectiveness.

Table 8.1: NIDS Alert Processing and Response Workflow

Actions	Descriptions	Purpose
Block IP Address	Add source IP to Blacklist	Prevent further traffic from the malicious source
Drop Connection	Drop packets / terminate sessions	Stop the current malicious connection
Rate Limit	Limit traffic rate from source	Reduce the impact of High-rate traffic
Alert Administrator	Send notification via email / dashboard	Inform the administrator for further investigation action
Log only	Log event for further analysis	Keep record without taking immediate action.
Allow (Whitelist)	Allow traffic and continue monitoring	Permit trusted traffic from Whitelist sources.

Table 8.2: Possible Response Actions

9. Discussion

The implementation demonstrates that a practical NIDS is more than a collection of detection conditions. The complete event lifecycle— packet capture, preprocessing, rule matching, severity assignment, whitelist checking, response, persistence and monitoring—affects whether an alert is trustworthy. This complements the literature on rule-based portscan detection and threshold selection [4], [15] and the practical Scapy-based systems reported in recent work [1], [3], [8].

The use of configurable thresholds provides transparency and makes the system easy to tune for a laboratory network. However, the project does not claim that fixed thresholds are universally optimal. Kim et al. demonstrated the motivation for adaptive thresholding [15], so future versions should evaluate thresholds against baseline traffic rather than treating the current values as universal constants.

The project also demonstrates the importance of response ordering. A detection event can be logged correctly while the corresponding blacklist or whitelist action is applied at the wrong stage. Therefore, trusted-source validation should occur before blocking, and state changes should be clearly defined for the current and subsequent packets.

10. Evaluation Framework

The evaluation should report detection coverage for each test case, alert severity correctness, response correctness, alert persistence, detection latency, rule maintainability and packet-path validity. For a stronger quantitative study, a larger labelled dataset should be collected from repeated controlled experiments. Accuracy, precision, recall and F1-score should only be reported when ground-truth labels and sufficiently large test sets are available [2], [5], [7], [12].

For each test case, the recommended validation sequence is: (1) confirm source and destination addresses; (2) confirm the monitored interface; (3) generate the test traffic; (4) verify packet arrival with packet capture; (5) verify the generated alert; (6) verify SQLite/log persistence; and (7) verify whitelist/blacklist response. This sequence separates network-path problems from detection-rule problems.

11. Limitations

- The study is a controlled laboratory implementation and does not establish production-scale performance.
- The current rules depend on manually selected thresholds and time windows; these values may require tuning for different traffic environments.

- Slow, distributed and highly stealthy scans may require longer observation windows and crosssource correlation.
- The evaluation does not yet provide a statistically large labelled dataset for precision, recall or F1score.
- Packet visibility depends on correct interface configuration and network topology; traffic that does not reach the monitored interface cannot be detected.
- The blacklist/whitelist response is intended for controlled testing and requires additional safeguards before use in a production environment.

12. Future Work

- Introduce adaptive thresholding based on learned normal traffic baselines.
- Add long-window correlation for low-rate and stealthy scans.
- Add alert grouping and correlation to reduce duplicate alerts and alert fatigue.
- Improve blacklist/whitelist state handling and provide configurable response policies.
- Expand the monitoring dashboard with realtime charts, filtering and report export.
- Build a larger labelled test dataset and compare the system against baseline NIDS approaches.
- Evaluate distributed attack sources and multi-sensor correlation.
- Investigate automated rule validation and rule-generation assistance while retaining human review.

13. Conclusion

This research presents a lightweight Python- and Scapy-based NIDS for real-time monitoring and detection of common network attack and reconnaissance behaviours. The system combines packet capture, preprocessing, configurable protocol-specific rules, thresholdbased detection, severity-based alerts, whitelist/blacklist handling, SQLite persistence and logging in a single modular workflow. Controlled testing focused primarily on TCP SYN, TCP Connect, FIN/NULL and thresholdbased port scanning, with additional tests for ICMP flooding, SYN flooding, FTP connection activity and repeated SSH-related traffic. The implementation experience showed that reliable intrusion detection depends not only on rule conditions but also on packet-path validation, event ordering, trusted-source precedence and persistent evidence. The system is therefore positioned as a transparent and reproducible platform for cybersecurity education and controlled experimentation. Its main limitations—manual threshold tuning, limited quantitative evaluation and reduced coverage of slow or distributed attacks—provide clear directions for future research.

14. References

[1] D. Hariyani, Y. R. Reddy, K. Dave, T. Patel, D. Vekariya, and P. Khan, “Intrusion detection system:
A Python-based approach for network security,” IET Conference Proceedings, vol. 2025, no. 7, pp. 162– 169, 2025. https://doi.org/10.1049/icp.2025.1290

[2] R. Chinnasamy, M. Subramanian, S. V. Easwaramoorthy, and J. Cho, “Deep learningdriven methods for network-based intrusion detection systems: A systematic review,” ICT Express, vol. 11, no. 1, pp. 181–215, 2025. https://doi.org/10.1016/j.icte.2025.01.005

[3] A. Tamrakar, J. Chaturvedi, M. Poddar, and S. Kumari, “Network Traffic Analyzer: Real-Time Detection and Malicious Activity Identification,” International Journal on Advanced Computer Theory and Engineering, vol. 15, no. 1, pp. 263–268, 2026. https://doi.org/10.65521/ijacte.v15i1.3819

[4] U. Kanlayasiri, S. Sanguanpong, and W. Jaratmanachot, “A Rule-Based Approach for Port Scanning Detection,” in Proceedings of the 23rd Electrical Engineering Conference, Chiang Mai, Thailand, 2000. https://web.archive.org/web/20240414100601/https://citeseerx.ist.psu.edu/document?repid=rep

1&type=
pdf&doi=56ad61791b0a06b9ba99ba787304161
b030f6d8b

[5] P. Goldschmidt and D. Chudá, "Network Intrusion Datasets: A Survey, Limitations, and Recommendations," Computers & Security, vol. 156, 104510, 2025. https://arxiv.org/pdf/2502.06688

[6] Z. Xu, Y. Wu, S. Wang, J. Gao, T. Qiu, Z. Wang, H. Wan, and X. Zhao, "Deep LearningBased Intrusion Detection Systems: A Survey," arXiv, 2025. https://arxiv.org/pdf/2504.07839

[7] N. Kalpani, N. Rodrigo, D. Seneviratne, S. N. Ariyadasa, and J. Senanayake, "CuttingEdge Approaches in Intrusion Detection Systems: A Systematic Review of Deep Learning, Reinforcement Learning, and Ensemble Techniques," Iran Journal of Computer Science, vol. 8, no. 2, pp. 303–333, 2025. https://doi.org/10.1007/s42044-025-00246-8

[8] M. Wahal, T. Choudhury, and M. Arora, "Intrusion Detection System in Python," in 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 348–353, IEEE, 2018. https://doi.org/10.1109/CONFLUENCE.2018.8442909

[9] C. Birkinshaw, E. Rouka, and V. G. Vasilakis, "Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks," Journal of Network and Computer Applications, vol. 136, pp. 71–85, 2019. https://doi.org/10.1016/j.jnca.2019.03.005

[10] M. Ahmed, R. Pal, M. M. Hossain, M. A. N. Bikas, and M. K. Hasan, "NIDS: A network based approach to intrusion detection and prevention," IEEE Conference Proceedings, pp. 141–144, 2009. https://doi.org/10.1109/IACSITSC.2009.96

[11] M. A. S. Arifin, D. Stiawan, B. Y. Suprapto, S. Susanto, T. Salim, M. Y. Idris, and R. Budiarto, "Oversampling and undersampling for intrusion detection system in the supervisory control and data acquisition IEC 60870-5-104," IET Cyber-Physical Systems: Theory & Applications, vol. 9, no. 3, pp. 282–292, 2024. https://doi.org/10.1049/cps2.12085

[12] D. Oroian, R. Bolboaca, A.-S. Roman, and V. Dobrota, "Network intrusion detection system using anomaly detection techniques," in 2024 IEEE 20th International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 93–100, IEEE, 2024. https://doi.org/10.1109/ICCP63557.2024.10793 023

[13] J. M. Pittman, "Machine Learning and Port Scans: A Systematic Review," arXiv, January 2023. https://doi.org/10.48550/arXiv.2301.13581

[14] R. Guide, E. Pauley, Y. Beugin, R. Sheatsley, and P. McDaniel, "Characterizing the Modification Space of Signature IDS Rules," in 2023 IEEE Military Communications Conference (MILCOM), pp. 536–541, IEEE, 2023. https://doi.org/10.1109/MILCOM58377.2023.10356225

[15] S. K. Kim, S. H. Lee, and S. W. Seo, "An Automatic Portscan Detection System with Adaptive Threshold Setting," Communications and Networks, vol. 12, no. 1, pp. 74–85, 2010. https://ieeexplore.ieee.org/document/6388436