International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 11 November-2026 | Impact Factor: 3.5



Design and Implementation of a Rule-Based Automated Network Threat Detection System

Mrs. ASHA G
Department of CSE
Akshaya Institute of Technology, Tumakuru, Karnataka, India
ashamanoj204@gmail.com

DEVARAJ A NAIK
Department of CSE
Akshaya Institute of Technology, Tumakuru, Karnataka, India
naikdevaraj234@gmail.com

DHANYASHREE A S
Department of CSE
Akshaya Institute of Technology, Tumakuru, Karnataka, India
dhanyadhanyashree02@gmail.com

GAGANDEEP H N
Department of CSE
Akshaya Institute of Technology, Tumakuru, Karnataka, India
gagandeepu2005@gmail.com

HALAPPA V D
Department of CSE
Akshaya Institute of Technology,Tumakuru, Karnataka,India
haleshdone@gmail.com

Abstract—

Network monitoring is an important part of cybersecurity because suspicious traffic patterns can indicate scanning, excessive traffic generation, or other harmful activity. This paper presents a lightweight, rule-based Automated Network Threat Detection System built with Python and Scapy. The system captures packets, records selected packet metadata, extracts source-level traffic features, applies configurable threshold rules, writes alerts to a CSV log, and shows results in a Flask-based dashboard. The rules check high packet volume, the number of unique destination IP addresses, the number of unique destination ports in TCP traffic, and high ICMP volume. We compare the design with ten recent studies (2024–2026) and find that all of them rely on learning-based components, mostly large language models, while only half concern rule-based detection. In a functional test on captured traffic, feature extraction processed 563 packets across TCP, UDP, ICMP, ARP, and IPv6 categories, the detector evaluated 13 source-IP entries, and two medium-severity high-packet-volume alerts were raised. This run shows that the pipeline works end to end. It does not establish detection accuracy or show that the flagged sources were malicious. The system is an educational prototype and a base for controlled evaluation and threshold tuning.

The system offers a simple and transparent approach to network traffic monitoring. Its modular design supports future improvements in threat detection and analysis. Further testing with labelled datasets will help evaluate its accuracy and reliability.

How to Cite this Article:

G, A., NAIK, D. A., S, D. A., N, G. H. & D, H. V. (2026). Design and Implementation of a Rule-Based Automated Network Threat Detection System. International Journal of Creative and Open Research in Engineering and Management, 02(11), 1-9. https://doi.org/10.55041/ijcope.v2i11.174

License:

This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.





https://doi.org/10.55041/ijcope.v2i11.174

INDEX TERMS : network security, intrusion detection, Scapy, packet capture, rule-based detection, traffic analysis, Flask dashboard.

I. INTRODUCTION

Computer networks carry traffic from many protocols, applications, and devices. Monitoring this traffic helps administrators find unusual activity and investigate possible security incidents. Intrusion detection systems (IDSs) analyze network or host activity and raise alerts when observations match known signatures or suspicious behavior. Existing research covers signature-based, anomaly-based, and machine-learning-based detection, and surveys point to continuing problems with false alarms, dataset suitability, and detection of previously unseen activity [1]–[3], [15]–[18].

This project takes a transparent, rule-based approach instead of machine learning. It uses packet metadata and configurable thresholds to flag selected traffic patterns. Rule-based detection is easy to explain and suits a small prototype, but its effectiveness depends on rule quality,



traffic context, and threshold calibration. An alert is a reason to investigate, not proof of an attack.

The contributions of this paper are: (i) a modular pipeline of packet capture, feature extraction, rule-based detection, CSV logging, and a browser dashboard that can be demonstrated in a controlled academic environment; (ii) a comparison of the design with ten recent studies (Section III); (iii) a first functional run with its measurements (Section VII); and (iv) a plan for the controlled evaluation that is still needed (Section IX).

II. LITERATURE REVIEW

[1] Abdulganiyu et al. (2023) presented a systematic literature review of Network Intrusion Detection Systems, examining signature-based, anomaly-based and hybrid detection approaches. Their study discussed the strengths and limitations of different IDS techniques and identified challenges in detecting evolving network threats. This research provides a foundation for understanding different intrusion detection approaches and selecting suitable detection mechanisms for network security applications.

[2] Yang et al. (2022) conducted a systematic literature review of anomaly-based network intrusion detection methods and datasets. Their research examined 119 highly cited studies and discussed different detection techniques, datasets and evaluation approaches. The study highlights the importance of identifying unusual network behaviour and evaluating detection systems using appropriate network traffic data.

[3] Moustafa and Slay (2015) introduced the UNSW-NB15 dataset for evaluating network intrusion detection systems. The dataset contains normal network activities and multiple categories of simulated attacks. It provides network traffic features that researchers can use to evaluate intrusion detection approaches. Their work demonstrates the importance of structured network traffic data for security research.

[4] Ahmad et al. (2022) reviewed machine-learning and deep-learning techniques used in Network Intrusion Detection Systems. Their study discussed different approaches, datasets and evaluation methods for identifying malicious network activities. The research highlights the increasing use of intelligent detection techniques to improve network security, while also identifying challenges related to false alarms and detection performance.

[5] Asad et al. (2024) analysed the diversity of signature-based open-source Network Intrusion Detection Systems. Their research examined how these systems depend on predefined attack signatures and the importance of maintaining and updating signature information to identify known threats. This provides useful background for understanding the strengths and limitations of rule- and signature-driven detection mechanisms.

[6] Issa et al. (2024) presented a systematic review of recent intrusion detection research, covering machine-learning methods, deep-learning techniques, datasets and attack classifications. Their findings describe the growing use of advanced computational techniques in intrusion detection and highlight continuing research challenges in identifying malicious network activities.

III. METHODOLOGY

The system is a sequence of modules (Fig. 1):

1) Packet capture. Scapy [4] observes packets on the selected interface and extracts the timestamp, source and destination IP, protocol, ports where available, packet size, and TCP flags.

2) Packet logging. Captured metadata is stored in packet_log.csv for repeatable offline processing.

3) Feature extraction. feature_extraction.py reads the packet records and aggregates traffic by source, producing features.csv.

4) Threat detection. threat_detection.py evaluates each source-level record against the configured thresholds and writes alerts to threat_log.csv.

5) Dashboard and status API. app.py serves a local Flask application whose dashboard presents summaries and recent records, while API endpoints give overview and component-status information.

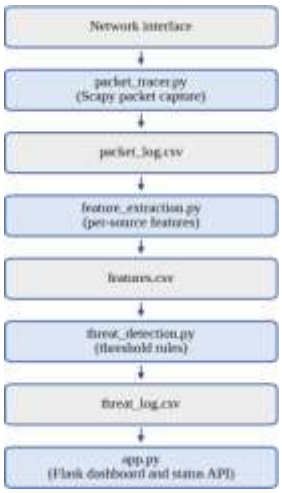


Fig. 1. Modules (blue) and intermediate CSV files (gray) of the pipeline.

The modules are separate so packet capture and analysis can be tested independently. The CSV files make intermediate outputs inspectable, although they are not meant as a high-throughput production store.

The implementation uses explicit threshold rules. The thresholds configured during testing are listed in Table I.



Rule	Condition	Severity
High packet volume	Total packet for a source ≥ 100	Medium
Possible network scanning	Unique destination IP's ≥ 20	High
Possible TCP port scan	Unique destination ports ≥ 10 and at least one TCP packet	High
High ICMP traffic	ICMP packet count ≥ 50	Medium

For each aggregated source record, the detection module checks the rules and emits an alert when a condition is met. If no rule fires, the source is reported as normal. "Normal" means only that none of the implemented rules fired; it does not guarantee that the source is benign. A threshold alert can also be caused by legitimate high-volume traffic. Counts are taken over state whether the counts cover the whole capture or a fixed time window, so the same threshold means different traffic rates for different capture lengths. A deployment should tune thresholds to its environment and combine alerts with context and investigation.

The threat log holds a timestamp, source IP, threat description, and severity. The program initializes the live alert log at the start of each detection run, so earlier alerts may be overwritten unless log rotation or append-only storage is added.

The prototype is written in Python, with Scapy for packet observation and Flask for the local dashboard. CSV files hold packet records, extracted features, and alerts. The main modules are packet_tracer.py, feature_extraction.py, threat_detection.py, and app.py. The dashboard includes an overview API and a component-status API.

The application was tested on Kali Linux. Packet capture ran with the required privileges, observed network traffic, and was stopped with Ctrl+C, after which the terminal showed a clean stop message. The packet-processing stages were then run in sequence. This shows the main pipeline from captured metadata to feature extraction and rule evaluation.

IV. RESULTS AND DISCUSSION

A functional run produced the following measurements from the team's terminal output. Feature extraction processed 563 packets with protocol categories TCP = 426, UDP = 114, ICMP = 10, IPv6 = 7, and ARP = 6, which sum to 563 (Fig. 3). The detector evaluated 13 source-IP entries and generated 2 alerts. Both were medium-severity "High packet volume" alerts, for source IPs 10.0.2.15 and 151.101.157.91 (Table I I).

TABLE I I

SUMMARY OF THE OBSERVED FUNCTIONAL RUN

Metric	Observed result
Packets processed	563
Source-IP entries evaluated	13
Alerts generated	2
Alert category	High packet volume
Alert severity	Medium
Ground-truth labels available	No
Accuracy / precision / recall	Not measured

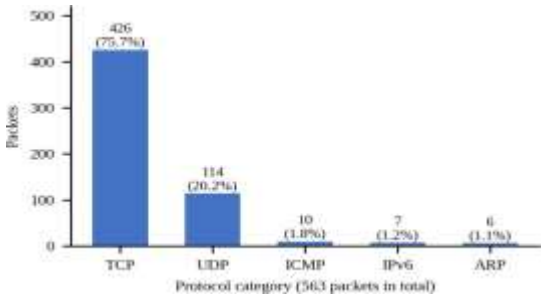


Fig. 3. Protocol categories among the 563 packets processed in the

The output shows that captured records were processed, aggregated, and evaluated by the configured rules. TCP and UDP together make up about 96% of the packets. Because only 10 ICMP packets were seen in total, the high-ICMP rule (threshold 50) could not have fired for any source in this run. The output does not show that the two flagged sources were malicious: a source can produce a high packet count during normal browsing, content delivery, or other legitimate activity. The experiment used no labeled benign or malicious ground truth and did not calculate precision, recall, false-positive rate, or accuracy. Those metrics must not be inferred from the alert count



Fig. 2: The Dashboad image that shows the captured packets with risks

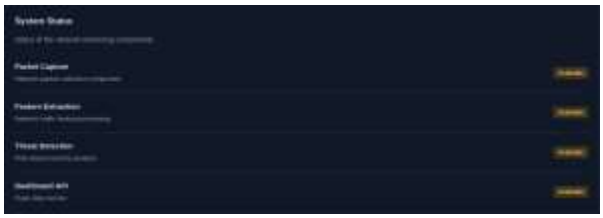


Fig. 3: Shows the system status



Fig. 4: The database image that stores the previous activity

The main demonstrated contribution is integration. The prototype captures packet metadata, extracts source-level features, applies interpretable rules, records alerts, and provides a dashboard, and it needs no trained model, external dataset, or inference service. Compared with the LLM-based studies in Section III, this keeps the system cheap and transparent, but it also means it cannot adapt to new attack patterns unless its rules are changed by hand.

There are important limitations. First, fixed thresholds may behave differently across networks and can cause false positives or miss low-and-slow activity. Second, the test is a small functional observation, not a statistically controlled benchmark. Third, aggregation by source IP may be distorted by NAT, shared addresses, and server or content-delivery traffic; ARP and IPv6 packets also need a defined way of being attributed to a source state how ARP and IPv6 packets are attributed to sources. Fourth, the status API's component checks show that required files or endpoints exist, not that capture is actively running. Fifth, the comparison in Section III-C relies on retrieved abstracts and summaries and on a keyword-based selection of studies, so it may miss capabilities described only in full texts. Finally, CSV storage and a local dashboard suit a prototype but need hardening for sustained or multi-user operation.

SECURITY, PRIVACY, AND ETHICS

Packet metadata can reveal user activity and network destinations. Testing should be done only on networks and devices the project team is authorized to use. Captured logs should be protected, kept only as needed, and sanitized before publication or demonstration. The prototype monitors and alerts; it does not block traffic or take automatic offensive action. Any future active response should include safeguards, authorization, and a review process

V. CONCLUSION

This paper described a rule-based Automated Network Threat Detection System built with Python, Scapy, CSV logging, and a Flask dashboard. A comparison with ten recent studies shows that all of them use learning-based components, while this prototype relies on explicit, inspectable rules. The functional run processed 563 packets, evaluated 13 source-IP entries, and raised two medium-severity high-packet-volume alerts. This verifies that the pipeline processes observed traffic and produces alerts, and leaves detection performance unmeasured.

ACKNOWLEDGMENT

This paper described a rule-based Automated Network Threat Detection System built with Python, Scapy, CSV logging, and a Flask dashboard. A comparison with ten recent studies shows that all of them use learning-based components, while this prototype relies on explicit, inspectable rules. The functional run processed 563 packets, evaluated 13 source-IP entries, and raised two medium-severity high-packet-volume alerts. This verifies that the pipeline processes observed traffic and produces alerts, and leaves detection performance unmeasured.

REFERENCES

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, Art. no. 20, 2019, doi: 10.1186/s42400-019-0038-7.

[2] Z. Yang et al., "A systematic literature review of methods and datasets for anomaly-based network intrusion detection," *Comput. Secur.*, vol. 116, Art. no. 102675, 2022, doi: 10.1016/j.cose.2022.102675.

[3] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, Art. no. e4150, 2021, doi: 10.1002/ett.4150.

[4] Scapy Project, "Welcome to Scapy's documentation." [Online]. Available: https://scapy.readthedocs.io/en/latest/. Accessed: Oct. 23, 2026.

[5] T. Cui et al., "TrafficLLM: Enhancing large language models for network traffic analysis with generic traffic representation," arXiv:2504.04222, 2025.

[6] S. K. Nandi, R. Ratti, S. R. Singh, and S. Nandi, "Prompt engineering-based network intrusion detection system," *IEEE Access*, vol. 13, pp. 190859–190871, 2025, doi: 10.1109/ACCESS.2025.3629761.

[7] S.-P. Paukkeri and T. Frantti, "Evaluation of local LLMs for network traffic packet analysis," *J. Cyber Secur. Technol.*, vol. 10, no. 1, Art. no. 2671738, 2026, doi: 10.1080/23742917.2026.2671738.

[8] I. Zziwa et al., "GUARD: A guided AI



system for intrusion detection and automated response in critical infrastructure environments," in *Proc. Int. Conf. Knowledge-Based Organization*, vol. 31, no. 1, 2025, pp. 219–227, doi: 10.2478/kbo-2025-0028.

[9] J. Li et al., "GRIDAI: Generating and repairing intrusion detection rules via collaboration among multiple LLM-based agents," arXiv:2510.13257, 2025.

[10] H. J. Hadi, R. Yasmin, and A. Shoker, "GenTI: Benchmarking LLMs for autonomous IDPS rule generation for unseen attacks," arXiv:2606.05844, 2026.

[11] P. Balasubramanian, T. Ali, M. Salmani, D. KhoshKholgh, and P. Kostakos, "Hex2Sign: Automatic IDS signature generation from hexadecimal data using LLMs," in *Proc. IEEE Int. Conf. Big Data*, 2024, pp. 4524–4532.

[12] X. Hu et al., "A LLM-based agent for the automatic generation and generalization of IDS rules," in *Proc. IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2024, pp. 1875–1880.

[13] L. Du et al., "Harnessing large language models for automated intrusion detection rule generation in cyber range," *IEEE Netw.*, 2025.

[14] X. Zhang et al., "Large language models powered malicious traffic detection: Architecture, opportunities and case study," arXiv:2503.18487, 2025.

[15] R. Chinnasamy, M. Subramanian, S. V. Easwaramoorthy, and J. Cho, "Deep learning-driven methods for network-based