Auto-IDS: Automating Wireshark-Based Packet Inspection Using Machine Learning for Real-Time Intrusion Detection

Saim Nadeem
School of Computing
FAST NUCES, Islamabad
i221884@nu.edu.pk

Fasih
School of Computing
FAST NUCES, Islamabad
i221910@nu.edu.pk

Shazer
School of Computing
FAST NUCES, Islamabad
i222043@nu.edu.pk

Ms. Sadia Saad
School of Computing
FAST NUCES, Islamabad
sadia.saad@nu.edu.pk

***Abstract*—Wireshark is a widely adopted packet inspection tool for network analysis and security investigation. Despite its effectiveness, Wireshark relies heavily on manual packet capture, filtering, and interpretation, making it unsuitable for scalable, continuous, and real-time intrusion detection.**

This paper presents *Auto-IDS*, an automated intrusion detection system that transforms traditional Wireshark-based inspection into a fully automated, machine learning–driven pipeline. Packet capture is performed programmatically using TShark, and packet-derived fields are extracted from PCAPs using PyShark (timestamp, IP addresses, ports, protocol, packet length, and TCP flags). A Random Forest classifier trained on the CICIDS2017 dataset is used to classify traffic as normal or malicious. The trained model is deployed for both offline dataset evaluation and live traffic detection, while a Streamlit dashboard supports interactive inspection and result export.

Experimental evaluation on a merged CICIDS2017 dataset (approximately 566,000 samples used in this work) demonstrates an accuracy of 99.9%, and live network testing confirms stable end-to-end execution from capture to prediction. The results show that automating Wireshark-style inspection and integrating supervised machine learning significantly improves scalability and practical usability for network security monitoring.

***Index Terms*—Intrusion Detection System, Wireshark Automation, Network Security, Machine Learning, CICIDS2017, Real-Time Traffic Analysis**

I. Introduction

Network traffic analysis is a foundational component of cybersecurity. By observing packet-level behavior and protocol semantics, defenders can identify malicious activity, policy violations, and information leakage. Tools such as Wireshark provide deep visibility into packet headers and protocol fields; however, they are primarily designed for interactive, analyst-driven workflows.

Wireshark-based inspection is typically manual: an analyst selects an interface, starts capture, applies filters, and interprets traffic. While effective for demonstrations and forensics, manual inspection does not scale to continuous monitoring due to high traffic volume and its dependence on human attention.

The base paper, *"The Role of Wireshark in Packet Inspection and Password Sniffing for Network Security"* [1], demonstrates Wireshark-based packet inspection and highlights how insecure traffic can expose sensitive information. However, it does not provide automation, dataset-based evaluation, supervised learning, or real-time intrusion detection.

To address these limitations, this paper proposes *Auto-IDS*, which preserves Wireshark's inspection philosophy but automates capture and analysis using an end-to-end machine learning pipeline. Auto-IDS integrates: (i) TShark for non-interactive packet capture [3], (ii) PyShark for programmatic parsing of packet fields [4], (iii) a Random Forest classifier for supervised detection [24], and (iv) a Streamlit dashboard for operational usability [29]. Auto-IDS is implemented as a practical system (the codebase labels the implementation as "Wired Sharks") that supports both offline evaluation and live inference.

II. Base Paper Overview and Limitations

The base paper [1] focuses on manual packet inspection using Wireshark, including interface selection, capture, filtering, and packet-level inspection. This approach is useful for understanding protocol behavior and illustrating risks of unencrypted traffic, but it is limited for operational intrusion detection due to:

(1) Lack of automation and continuous operation: the workflow is not designed to run unattended at scale.

(2) No dataset-driven evaluation: without labeled data, standard IDS metrics (accuracy, precision, recall, F1-score) cannot be reported.

(3) No learning-based or real-time detection pipeline: no model is trained, deployed, or evaluated for online prediction.

Auto-IDS extends the base-paper idea (inspection for security) by automating the workflow and adding supervised classification and visualization.

III. Proposed System: Auto-IDS

A. Design Philosophy

Auto-IDS does not replace Wireshark; instead, it automates the inspection workflow and augments it with supervised machine learning. Tasks that are typically manual (capture, extraction, interpretation) are converted into a repeatable pipeline.

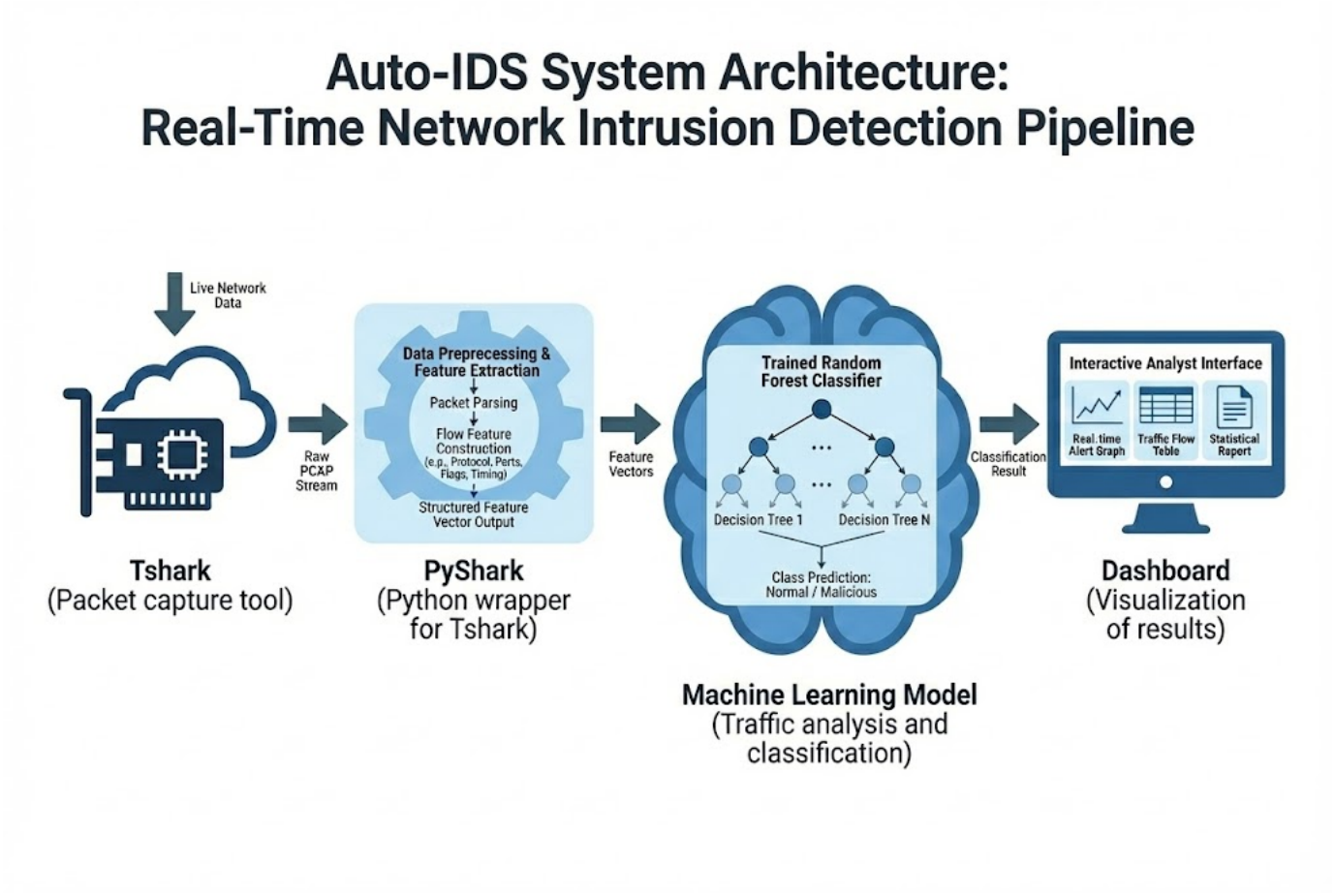

Fig. 1. Auto-IDS system architecture illustrating automated packet capture, feature preparation, machine learning classification, and visualization.

B. System Architecture

The Auto-IDS pipeline consists of:

1) Automated packet capture using TShark [3]
2) PCAP parsing and export of packet-derived fields using PyShark [4]
3) Feature preparation and schema alignment for inference
4) Machine learning–based classification using a trained Random Forest model [24]
5) Visualization and reporting through a Streamlit dashboard [29]

IV. Threat Model and Scope

Auto-IDS targets network-based intrusion detection where observable packet metadata and traffic behavior provide evidence of malicious activity. In this work, the classification objective is binary:

- **Normal (0):** traffic labeled as benign/normal,
- **Attack (1):** all non-benign traffic labels.

Auto-IDS does not decrypt encrypted payloads. When payloads are encrypted, the system can only use observable fields available from packet headers and protocol metadata.

V. Dataset Preparation

A. Dataset

Auto-IDS uses the CICIDS2017 dataset for supervised training and offline evaluation [5], [6]. The dataset contains labeled benign and attack traffic across multiple attack scenarios.

B. Merging CICIDS2017 CSV Files

To build a single consolidated training file, Auto-IDS includes a merge script that loads all CSV files from `MachineLearningCVE/` and concatenates them into `CICIDS2017.csv`. This merged file is then placed at `data/CICIDS2017.csv` to match the training script's expected path.

C. Column Name Normalization

Auto-IDS includes a preprocessing step that strips whitespace from column names and verifies the dataset contains a label column named exactly `Label`. The cleaned CSV is saved back to `data/CICIDS2017.csv` to avoid schema inconsistencies during training.

VI. Packet Parsing and Feature Extraction

A. PCAP-to-CSV Export

Auto-IDS extracts packet-derived fields from PCAP files using PyShark. For each parsed packet, the exported CSV includes:

- `timestamp` (capture timestamp)
- `src_ip`, `dst_ip` (if an IP layer exists)
- `protocol` (highest-layer protocol, lowercased)
- `length` (packet length)
- `src_port`, `dst_port` (TCP/UDP when available)
- `flags` (TCP flags when available)

Malformed packets are ignored to ensure robustness during live operation.

B. Why Schema Alignment is Required

Because the Random Forest model is trained on the CICIDS2017 feature space, Auto-IDS enforces input schema consistency at inference time. The system stores the training feature column list and aligns any inference CSV to that schema before prediction.

VII. Machine Learning Model

A. Model Selection

A Random Forest classifier [24] was selected due to its strong performance on structured/tabular data and its fast inference, making it suitable for real-time prediction.

B. Decision Function

Let $\mathbf{x} \in \mathbb{R}^n$ represent an input feature vector. Each tree T_i outputs a class prediction:

$$T_i(\mathbf{x}) \in \{0, 1\}.$$

The forest prediction is the majority vote:

$$\hat{y} = \arg \max_{c \in \{0,1\}} \sum_{i=1}^{N} \mathbb{I}(T_i(\mathbf{x}) = c),$$

where 0 denotes Normal and 1 denotes Attack.

VIII. Training Pipeline (Implementation-Accurate Description)

Auto-IDS trains the Random Forest model using Scikit-learn [25] on `data/CICIDS2017.csv`.

A. Binary Label Mapping

The training script maps the target column `Label` into a binary label:

- if `Label` is `BENIGN` or `NORMAL` (case-insensitive after trimming), it is mapped to 0,
- otherwise it is mapped to 1.

B. Feature Preparation

All non-label columns are used as input features. Columns with dtype `object` are one-hot encoded, and infinite values are replaced and missing values are filled with 0.

C. Train/Test Split and Class Weights

The script splits the dataset into train/test subsets with `test_size = 0.2`, `random_state = 42`, and `stratify = y`. Class weights are computed using balanced weighting on the training labels and passed into the classifier.

D. Random Forest Configuration and Artifacts

The trained Random Forest uses:

- `n_estimators = 200`
- `min_samples_split = 4`
- `min_samples_leaf = 2`
- `random_state = 42`
- `n_jobs = -1`
- `class_weight = (computed balanced weights)`

IX. Live Detection Pipeline (TShark + PyShark + Inference)

A. TShark Discovery and Interface Selection

Auto-IDS locates TShark either from the system PATH or from common Windows installation paths. It lists interfaces using `tshark -D` and prompts the user to select an interface number for capture.

B. Timed Capture and PCAP Storage

The live pipeline captures traffic for a user-provided duration using TShark's timed capture option and saves a timestamped PCAP at:

`captures/live_YYYYMMDD_HHMMSS.pcap`.

C. CSV Export and Prediction

After capture, the PCAP is parsed into a timestamped CSV under `data/`. Predictions are generated by aligning the live CSV to the training feature schema and applying the trained model. The pipeline appends: `Prediction` and `PredictionLabel` (Normal/Attack) and prints summary counts and a tail-view of results.

X. Feature Alignment for Inference

Auto-IDS aligns inference CSVs (live or user-uploaded) using the following steps:

- one-hot encode object/string columns,
- replace $\pm\infty$ with 0 and fill missing values with 0,
- add any missing training feature columns and set them to 0,
- reorder columns to match the stored training feature list.

This guarantees that inference inputs match the exact dimensionality expected by the trained model.

XI. Experimental Setup

A. Environment

- Operating System: Windows 11
- Python: 3.10
- Tools: TShark 4.4.9, PyShark
- Libraries: Scikit-learn, Pandas, NumPy

B. Evaluation Metrics

Performance is reported using accuracy, precision, recall, and F1-score [28].

XII. Results and Performance

A. Dataset Evaluation

TABLE I
Auto-IDS Performance on CICIDS2017

Metric	Value
Accuracy	0.99909
Precision	0.99680
Recall	0.99860
F1-score	0.99770

B. Live Network Testing

Live testing was conducted on a Wi-Fi interface, where 6,955 packets were captured and processed in real time. The goal of live testing in this work is system stability and operational execution (capture $\rightarrow$ export $\rightarrow$ alignment $\rightarrow$ prediction). Ground-truth accuracy for live traffic is not directly measurable without controlled labeling.

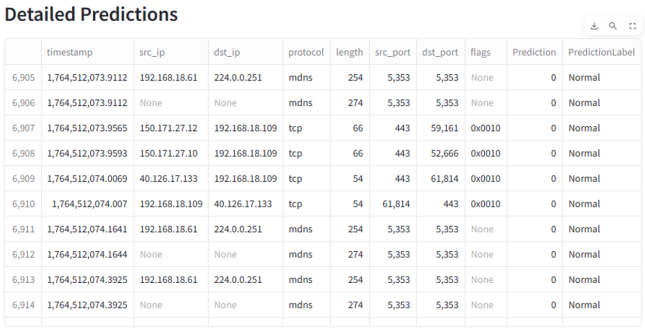

Fig. 2. Detailed predictions from live network traffic captured by Auto-IDS.

XIII. Visualization Dashboard

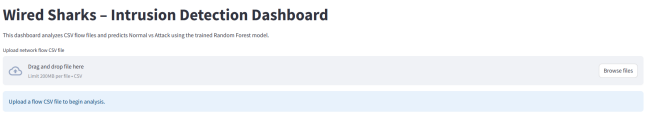

Fig. 3. Auto-IDS Streamlit dashboard for traffic analysis and prediction visualization.

Auto-IDS provides a Streamlit dashboard that loads the trained model and accepts a user-uploaded CSV. The dashboard:

- previews uploaded data,
- aligns features using the same alignment routine as the live pipeline,
- predicts Normal vs Attack,
- displays summary counts and a bar chart,
- shows a detailed table,
- enables downloading predictions as a CSV.

XIV. Comparison with Base Paper

TABLE II
Comparison Between Base Paper and Auto-IDS

Feature	Base Paper	Auto-IDS
Packet Capture	Manual	Automated (TShark)
Inspection	Manual	Automated (PyShark + ML)
Machine Learning	No	Yes (Random Forest)
Dataset Evaluation	No	CICIDS2017
Real-Time Detection	No	Yes (Live mode)
Visualization	Wireshark GUI	Streamlit Dashboard

XV. Limitations

(1) Dataset-to-reality gap: Performance measured on CICIDS2017 reflects dataset conditions and labels; real networks may differ in distribution and background traffic.

(2) Encrypted traffic: Auto-IDS does not decrypt payloads and therefore cannot inspect encrypted application content.

(3) Live labeling: Live operation demonstrates end-to-end execution; accuracy on live traffic requires controlled experiments with known ground truth.

XVI. Conclusion

This paper presented Auto-IDS, a system that automates Wireshark-style inspection by integrating TShark-based capture, PyShark-based parsing, supervised Random Forest classification, and a Streamlit dashboard for practical usage. Auto-IDS addresses the scalability limitations of manual inspection by providing an end-to-end pipeline for offline evaluation and live inference.

Future work includes adding aggregated flow/statistical features, expanding evaluation to additional datasets, and incorporating models specialized for encrypted-traffic behavior.

References

[1] M. Mishra, M. S. Hussain, N. Chaubey, and P. Gupta, "The Role of Wireshark in Packet Inspection and Password Sniffing for Network Security," IGI Global, 2025, Chapter 9, doi: 10.4018/979-8-3693-9225-6.ch009. https://doi.org/10.4018/979-8-3693-9225-6.ch009

[2] G. Combs *et al.*, "Wireshark User's Guide," Wireshark Project. https://www.wireshark.org/docs/wsug_html_chunked/

[3] Wireshark Project, "tshark(1) Manual Page," Wireshark. https://www.wireshark.org/docs/man-pages/tshark.html

[4] KimiNewt, "PyShark: Python Wrapper for TShark (Wireshark)," GitHub repository. https://github.com/KimiNewt/pyshark

[5] Canadian Institute for Cybersecurity (CIC), University of New Brunswick, "Intrusion Detection Evaluation Dataset (CIC-IDS2017)," 2017. https://www.unb.ca/cic/datasets/ids-2017.html

[6] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *Proc. ICISSP*, 2018. https://www.scitepress.org/publishedPapers/2018/66398/66398.pdf

[7] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," NIST Special Publication 800-94, 2007. https://csrc.nist.gov/publications/detail/sp/800-94/final

[8] D. E. Denning, "An Intrusion-Detection Model," *IEEE Trans. Softw. Eng.*, vol. 13, no. 2, pp. 222–232, 1987. https://doi.org/10.1109/TSE.1987.232894

[9] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion-Detection Systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999. https://doi.org/10.1016/S1389-1286(98)00017-6

[10] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks,” in *Proc. 13th USENIX LISA*, 1999, pp. 229–238. https://www.usenix.org/legacy/event/lisa99/full_papers/roesch/roesch.pdf

[11] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, 1999. https://doi.org/10.1016/S1389-1286(99)00112-7

[12] Zeek Project, “Zeek Documentation (The Book of Zeek).” https://docs.zeek.org/

[13] Open Information Security Foundation (OISF), “Suricata User Guide.” https://docs.suricata.io/en/latest/

[14] B. Trammell and B. Claise, “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information,” RFC 7011, 2013. https://www.rfc-editor.org/rfc/rfc7011.html

[15] B. Claise and B. Trammell, “Information Model for IP Flow Information Export (IPFIX),” RFC 7012, 2013. https://www.rfc-editor.org/info/rfc7012

[16] B. Trammell, A. Wagner, and B. Claise, “Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol,” RFC 7015, 2013. https://www.rfc-editor.org/info/rfc7015

[17] The Tcpdump Group, “libpcap,” GitHub repository. https://github.com/the-tcpdump-group/libpcap

[18] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A Detailed Analysis of the KDD CUP 99 Data Set,” in *Proc. IEEE CISDA*, 2009, pp. 53–58. https://doi.org/10.1109/CISDA.2009.5356528

[19] N. Moustafa and J. Slay, “UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set),” in *Proc. MilCIS*, 2015, pp. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

[20] A. Thakkar and R. Lohiya, “A Review of the Advancement in Intrusion Detection Datasets,” *Procedia Computer Science*, vol. 167, pp. 636–645, 2020. https://doi.org/10.1016/j.procs.2020.03.330

[21] R. Sommer and V. Paxson, “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection,” in *Proc. IEEE Symp. Security and Privacy*, 2010, pp. 305–316. https://doi.org/10.1109/SP.2010.25

[22] A. L. Buczak and E. Guven, “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection,” *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2016. https://doi.org/10.1109/COMST.2015.2494502

[23] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges,” *Computers & Security*, vol. 28, no. 1–2, pp. 18–28, 2009. https://doi.org/10.1016/j.cose.2008.08.003

[24] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001. https://doi.org/10.1023/A:1010933404324

[25] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. https://jmlr.org/papers/v12/pedregosa11a.html

[26] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proc. 9th Python in Science Conf. (SciPy)*, 2010. https://doi.org/10.25080/Majora-92bf1922-00a

[27] C. R. Harris *et al.*, “Array Programming with NumPy,” *Nature*, vol. 585, pp. 357–362, 2020. https://doi.org/10.1038/s41586-020-2649-2

[28] M. Sokolova and G. Lapalme, “A Systematic Analysis of Performance Measures for Classification Tasks,” *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009. https://doi.org/10.1016/j.ipm.2009.03.002

[29] Streamlit, Inc., “Streamlit Documentation.” https://docs.streamlit.io/

[30] J. Postel, “Transmission Control Protocol,” RFC 793, 1981. https://www.rfc-editor.org/rfc/rfc793