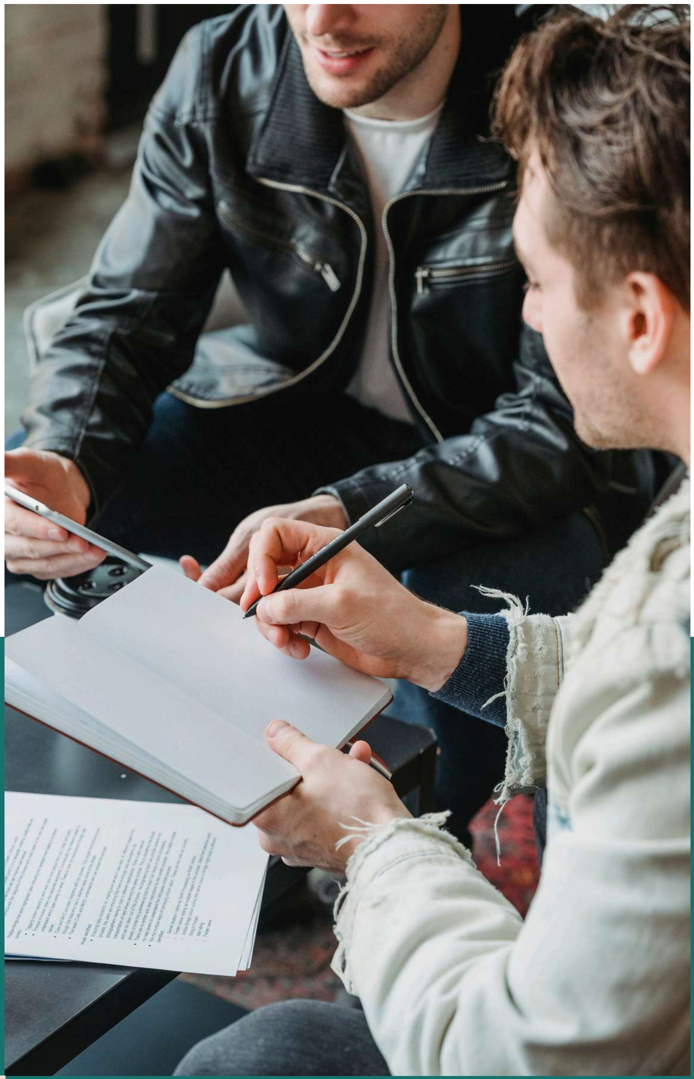
TABLE OF CONTENT

Group-12

PROBLEM STATEMENT

Network security professionals frequently face the challenge of analyzing massive volumes of packet capture (PCAP) data, which is low-level and difficult to interpret manually. Tools like Wireshark offer packet-level inspection but become inefficient at scale. The critical need is to extract high-level, flow-based features that summarize network activity for use in traffic classification, anomaly detection, and machine learning applications. Without automated extraction of meaningful features such as flow duration, packet count, and average packet size, timely threat detection becomes impractical. This project addresses the problem by developing a Python-based pipeline to convert raw packet data into structured, analyzable flow-level features.

Group-12



BACKGROUND & PREVIOUS RESEARCH

BACKGROUND

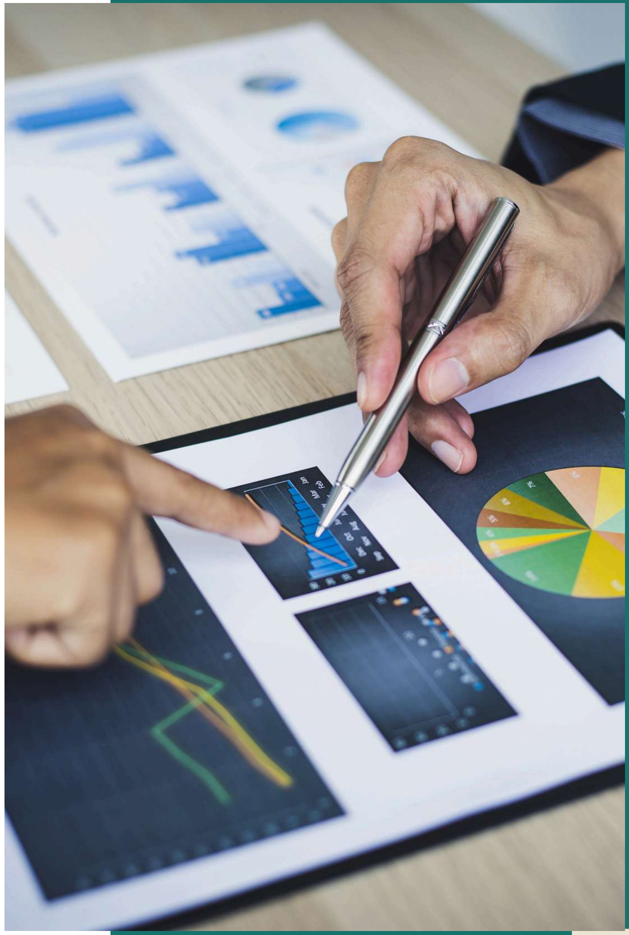
Analyzing network traffic at the packet level is a traditional approach used by network analysts. While tools like Wireshark help with manual inspection, they become inefficient when dealing with high volumes of traffic. To address this, flow-based tools like NetFlow and CICFlowMeter were developed, which summarize packet data into flow-level statistics. These flow features are crucial for modern techniques like machine learning, which rely on structured data for accurate traffic analysis and threat detection.

PREVIOUS RESEARCH

Previous studies have utilized tools like NetFlow and CICFlowMeter to extract flow-level features from packet data. These tools convert raw packets into structured information useful for network monitoring and intrusion detection. Researchers have also explored using commercial tools like Wireshark for detailed packet inspection, though they are limited in scalability. More recent research focuses on integrating flow-based features with machine learning models for automated classification and anomaly detection. However, many of these tools are either closed-source or not easily customizable for specific ML pipelines.

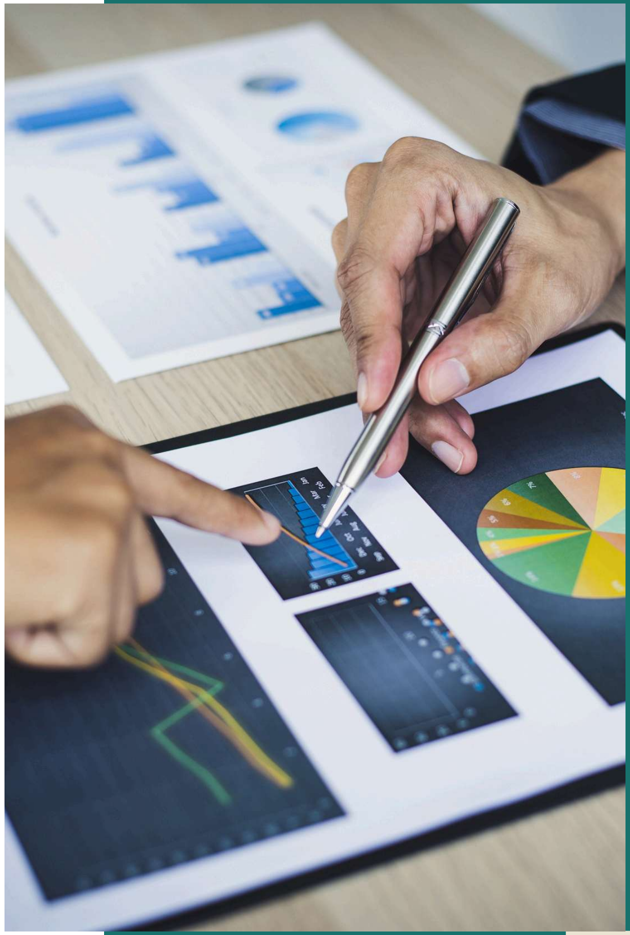
METHODOLOGY (TECHNICAL ANALYSIS)

- **Tools Used**: Python, PyShark, Wireshark, Tshark, Pandas
- **Workflow Steps**:
- Load raw packet data from .pcap files using PyShark.
- Extract 5-tuple identifiers (Source IP, Destination IP, Source Port, Destination Port, Protocol).
- Group packets into flows based on the 5-tuple.
- For each flow, compute: Flow duration, Total number of bytes and packets, Average packet size
- Export the computed flow-level features into a structured .csv file for downstream analysis or machine learning applications.



METHODOLOGY (TECHNICAL ANALYSIS)

- **Dataset**: http://cicresearch.ca/CICDataset/CIC-IDS-2017/Dataset/CIC-IDS-2017/CSVs/
- Preprocess the dataset (e.g., handle missing values, normalize features, encode labels if necessary).
- Train a machine learning model — Random Forest classifier — using the processed flow-level dataset.
- Save the trained model for reuse with real-time inputs.
- Created a Streamlit web interface for analyzing the data flow using pyshark and Tshark to monitor the live traffic



COMPARISION OF METHODS

01 XGBoost without SMOTE

- Without the application of SMOTE, XGBoost's performance dropped noticeably. The model achieved an accuracy of 60%, recall of 76%.
- The lower recall suggests that XGBoost, when trained on an imbalanced dataset, tends to misclassify more minority class instances.

02 XGBoost with SMOTE

- XGBoost with SMOTE also showed performance among all the methods, It achieved an accuracy of 79%.
- The results suggest that in this case, it might be more sensitive to noise or overfitting, especially when synthetic samples are introduced by SMOTE

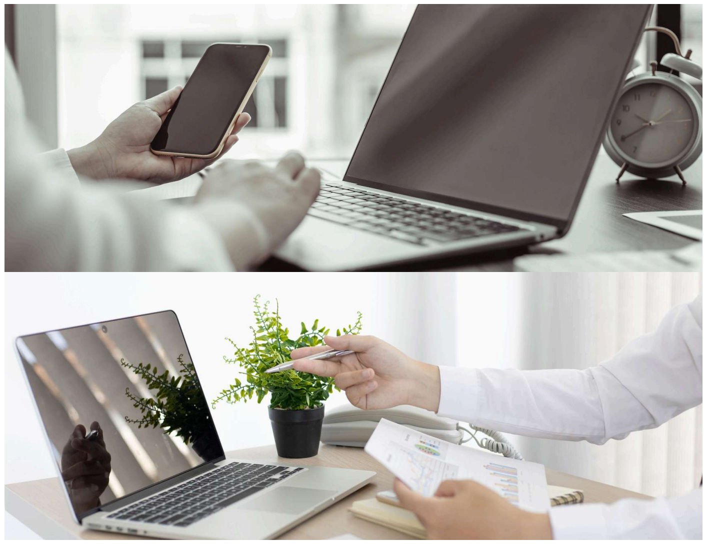
03 Random Forest without SMOTE

- Random Forest without SMOTE yielded the best performance. It achieved an accuracy of 75%, but precision, recall and F1 score are less compared.
- But we consider more of precision, recall and F1 score in classification task.

04 Random Forest with SMOTE

- Random Forest combined with SMOTE achieved the highest overall performance among all four methods. It recorded an accuracy of 91%, recall of 97%, precision of 83%, F1 score of 89%.
- The combination of Random Forest's ensemble learning and SMOTE's ability to handle class imbalance results in a strong, well-generalized model.

FINDINGS



01 The Random Forest model achieved an accuracy of approximately 91% in classifying network traffic.

02 Real-time packet capture and analysis were successfully implemented using Scapy and Streamlit.

03 The system effectively identifies suspicious traffic patterns, triggering alerts as designed.

04 Integration with a Streamlit UI simplified real-time PCAP analysis and provided a user-friendly interface for security professionals and researchers.

05 Model performance varied with packet volume — small batch sizes (e.g., 10 packets) limited context for accurate classification, suggesting potential gains with larger flow samples or temporal aggregation.

GROUP-12

CONCLUSION

:

- We have two types of outputs namely BENIGN and ERROR in our project. If the output contains label as BENIGN, it means that there is no network intrusion. But if the output contains label as an error it means that suspicious network traffic has been detected.
- We developed a functional prototype for real-time network traffic analysis using Scapy, Streamlit, XGB and Random Forest classifiers.
- Demonstrated the feasibility of live packet capturing, feature extraction, and prediction with streamlit interface.
- We have observed that XGB classifier had given more accuracy with smote than without it, meaning there is balance in minority class prediction.
- Achieved around 91% classification accuracy using Random Forest Classifier, validating the model’s potential in practical scenarios.
- The system was able to trigger alerts effectively when suspicious traffic was detected.

REFERENCES

01

- Sahingoz, O. K., Buber, E., Demir, O., & Diri, B. (2019).
- Machine-learning-based phishing detection from URLs
- Discusses URL-based phishing detection using ML techniques including Random Forest and XGBoost.

02

Abdelhamid, N., Ayesh, A., & Thabtah, F. (2020).

- Phishing detection based on hybrid feature selection and machine learning.
- Focuses on hybrid feature selection methods to improve phishing detection accuracy.

03

Verma, R., & Das, A. (2020).

- What's in a URL: Fast feature extraction and classification of phishing websites.
- Explores fast and scalable ML-based phishing detection via URL features.

04

- Alzahrani, A., & Agrawal, A. (2021).
- Intelligent phishing detection system using deep learning techniques.
- Applies deep learning and compares its performance with traditional ML models.

COMPUTER SECURITY CASE STUDY

THANK YOU

FOR YOUR NICE ATTENTION

GROUP-12