Network Monitoring using Machine Learning

Charles Bowen-Rayner*
cdbowen21@earlham.edu
Earlham College
Richmond, Indiana, USA

ABSTRACT

Machine learning is gaining prominence for network monitoring, yet current tools are often complex to understand and use. This proposal attempts to address this by developing an algorithm for network anomaly detection using libpcap and the random forest algorithm. This approach provides real-time anomaly detection by analyzing past network traffic from real-life datasets and employing machine learning techniques. Through various tested methods, the effectiveness of identifying various network anomalies will be analyzed. This study will highlight the potential of integrating libpcap and machine learning for scalable and adaptable network security solutions, contributing to improved threat detection in modern computing environments.

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

Machine Learning, Random Forest, libpcap, Network Monitoring, Packet Capture, Packet Analysis

1 INTRODUCTION

In the field of computer networking and cybersecurity, robust monitoring tools are vital. These tools are key in maintaining network integrity by continuously scanning for anomalies, potential threats, and performance bottlenecks. Challenges such as evolving attack vectors, increasing network traffic volumes, and the need for real-time analysis require innovative solutions to enhance the effectiveness of network monitoring capabilities. In response to these challenges, the integration of machine learning techniques into network monitoring has emerged as a method for enhancing detection capabilities and improving response times. Machine learning, with its ability to analyze vast amounts of data and discern complex patterns, offers a new paradigm for network monitoring that is adaptable and dynamic. Central to the task of network monitoring is the capture and analysis of network traffic data. Here, libpcap stands out as a foundational library, providing powerful capabilities for capturing packets traversing the network interface. Its versatility and efficiency make it a preferred choice for network monitoring applications.

In this paper, I discuss the implementation of a tool designed to capture and analyze network traffic for anomaly detection. The tool utilizes packet capture techniques facilitated by libpcap to collect essential network traffic data, including source and destination IP addresses, ports, protocols, and payload data. Building upon this data collection framework, the implementation harnesses machine learning techniques to analyze captured network traffic and identify anomalies indicative of security threats. The random forest algorithm serves as the core of the detection model. The CIC-IDS2017 dataset—which contains real-world attack scenarios—was used to train the model. To address the inherent class imbalance within the dataset, I applied Synthetic Minority Oversampling Technique (SMOTE) [1] during preprocessing, ensuring the model is exposed to a balanced representation of both normal and anomalous network traffic. Additionally, class weighting was incorporated to further mitigate any residual imbalance. For feature selection, I utilized Pearson correlation matrix analysis and Recursive Feature Elimination (RFE) to identify the most relevant features from the dataset. This step significantly enhanced the model's efficiency by removing redundant or less significant features, optimizing its ability to detect malicious activity accurately. Once trained, the random forest model is integrated into a live packet-capturing system, where the tool continuously monitors network activity in real-time. The captured packets, stored in .pcap files, are fed into the detection system, which analyzes the traffic using the pre-trained model. Upon identifying any anomalous or malicious network behavior, the system immediately alerts network administrators, ensuring timely responses to potential security threats.

2 DATASET: CICIDS-2017

The CICIDS-2017 dataset [6] is a comprehensive and widely recognized dataset designed for evaluating intrusion detection systems (IDS) in the context of network security. Developed by the Canadian Institute for Cybersecurity (CIC), this dataset replicates realistic traffic scenarios through a combination of both benign and malicious network activity. The dataset was captured over five days in a controlled environment that closely mirrors a real-world corporate network.

The CICIDS-2017 dataset contains a wide variety of attack types categorized into 14 attack methods:

- **DDoS**: Floods a system with traffic from multiple sources to overwhelm it.
- **DoS Hulk**: Sends massive traffic to exhaust server resources.
- **DoS GoldenEye**: Floods a server with HTTP requests.
- **DoS slowloris**: Opens many slow connections to exhaust server resources.
- **DoS Slowhttptest**: Sends incomplete or slow HTTP requests to keep server resources busy.
- **PortScan**: Probes for open ports to find potential vulnerabilities.
- **FTP-Patator**: Brute force attack on FTP servers by guessing login credentials.
- **SSH-Patator**: Brute force attack on SSH services by guessing login credentials.
- **Bot**: Infects machines to create a botnet for large-scale attacks.

- **Web Attack — Brute Force**: Attempts to guess login credentials on a web application.
- **Web Attack — XSS**: Injects malicious scripts into web pages viewed by users.
- **Web Attack — SQL Injection**: Exploits vulnerabilities in SQL queries to access or manipulate databases.
- **Infiltration**: Gains unauthorized access to internal networks.
- **Heartbleed**: Exploits a vulnerability in OpenSSL to steal sensitive data.

A common approach when working with this dataset is to group some of the categories to help reduce the class imbalance problem. For this project, the 14 categories were grouped to a subset of 8:

- **DDoS**: DDoS attack traffic.
- **DoS**: This includes all Denial of Service attacks such as DoS Hulk, DoS GoldenEye, DoS slowloris, and DoS Slowhttptest.
- **Port Scan**: Traffic resulting from port scanning activities.
- **Brute Force**: This includes both FTP-Patator and SSH-Patator attacks.
- **Bot**: Malicious traffic associated with botnet activity.
- **Web Attack**: This category includes Web Attack — Brute Force, Web Attack — XSS, and Web Attack — SQL Injection.
- **Infiltration**: Malicious traffic that results from network infiltration attempts.
- **Heartbleed**: Traffic related to the Heartbleed vulnerability.

In addition to malicious activity, the dataset also includes normal (benign) traffic, making it suitable for binary as well as multi-class classification tasks. Each network flow is represented by 80 network traffic features that capture detailed information about the packet headers, payload size, and other flow-level characteristics. This allows for in-depth analysis and identification of subtle differences between malicious and normal behavior.

3 METHODS

In developing this network monitoring tool, I employed a hybrid approach combining machine learning-based anomaly detection with live network traffic capture for real-time monitoring. The methodology is divided into three major components: model training, packet capture, and real-time classification. Figure 1 illustrates the overall architecture of this network monitoring tool, highlighting the data flow from offline training using the CICIDS-2017 dataset to real-time detection.

3.1 Preprocessing with CICIDS-2017

Effective preprocessing of the CICIDS-2017 dataset was essential for ensuring data quality and enhancing model performance. The following steps were undertaken:

3.1.1 Data Cleaning. First, whitespace between column names was removed to standardize the dataset structure. This step facilitated subsequent data manipulation and prevented issues related to inconsistent column references. Next, duplicate records were identified and eliminated to avoid overfitting and ensure that each instance in the dataset contributed unique information to the learning process. Furthermore, the columns representing *Flow Bytes* and *Flow Packets* contained infinity values, which could adversely affect the model's training. These infinity values were replaced with the median values of their respective columns, ensuring that data distributions were maintained while mitigating the impact of outliers.

3.1.2 Label Reduction. The dataset originally contained 15 distinct class labels, representing various types of network attacks and benign traffic. To improve the tractability of the classification task, these labels were consolidated into 9 broader categories. This reduction in class labels simplified the learning problem while retaining the essential distinctions between major attack types.

3.1.3 Removal of Low-Variance Features. Columns with only one unique value—such as session identifiers and other features offering little predictive power-were removed from the dataset. Features with low variance often contribute little to classification tasks and can introduce noise, so their exclusion enhances the model's focus on relevant data.

3.1.4 Initial Feature Selection. To address multicollinearity in the dataset, a Pearson correlation matrix was used to identify and remove highly correlated features. A threshold of 0.8 was applied to the correlation values, meaning that any pair of features with a correlation higher than this threshold were considered too similar, and one was removed. By focusing on the upper triangular portion of the correlation matrix, this process efficiently identified features to be dropped. After applying this method, the number of features was reduced to 32, ensuring that only those features providing distinct and non-redundant information were retained. This reduction in feature space contributed to improving the model's performance by removing unnecessary complexity and focusing on the most relevant information for classification.

3.1.5 Class Imbalance Handling. Class imbalance is a known challenge with this dataset, as certain attack types are underrepresented. To mitigate this issue, two techniques were applied:

(1) SMOTE (Synthetic Minority Oversampling Technique) was used to generate synthetic instances of minority class samples, ensuring a more balanced dataset for model training.
(2) Class weighting was also employed during training to assign higher misclassification penalties to minority classes. This technique complemented SMOTE by emphasizing the importance of correctly classifying underrepresented attack types, thus preventing the model from being biased towards majority classes.

3.1.6 Recursive Feature Elimination (RFE). After addressing the class imbalance, Recursive Feature Elimination (RFE) was applied for the final feature selection. RFE recursively removed irrelevant features by training a base model and discarding the least important predictors in each iteration. This process was repeated until an optimal subset of features was identified. By reducing dimensionality, RFE enhanced the efficiency of the classification model and minimized overfitting. Figure 2 illustrates this RFE process.

Through this preprocessing pipeline, the dataset was transformed into a clean and balanced format, optimized for training a random forest classifier. Each preprocessing step was designed to ensure that the model could generalize effectively while maintaining high accuracy across all classes [1].

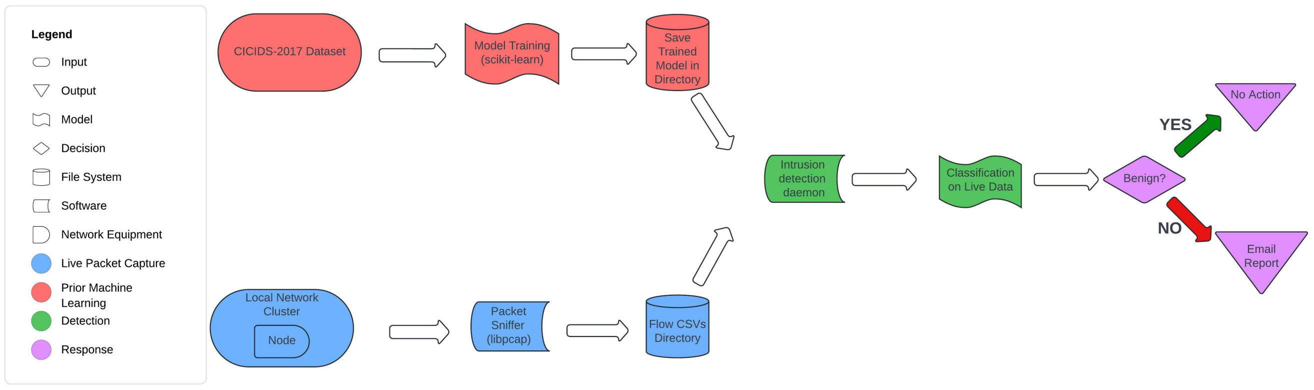


Figure 1: Data architecture of the network monitoring tool, highlighting the flow from training to real-time classification.

3.2 Model Training

To determine the optimal number of features to include in the model, the ranked features from RFE were tested across various sizes, specifically evaluating the impact on model performance. Using mean cross-validation accuracy as the evaluation metric, it was found that 10 features yielded the best results with the minimum number of features, providing a balance between model complexity and predictive accuracy. The subset of 10 crucial features identified for model training were: *Bwd Header Length*, *Destination Port*, *Total Length of Fwd Packets*, *Bwd Packet Length Max*, *Init Win bytes backward*, *Fwd Packet Length Max*, *Total Fwd Packets*, *Flow Duration*, *Flow IAT Std*, and *Init Win bytes forward*.

With the selected features established, hyperparameter tuning for the Random Forest classifier was conducted. Two hyperparameters were considered:

- **Max depth:** The maximum depth of the decision trees, which controls model complexity and the risk of overfitting. Figure 3.
- **N estimators:** The number of trees in the forest, impacts both model accuracy and computation time. Figure 4.

A loop was implemented to iterate over different combinations of these hyperparameters, with mean cross-validation accuracy used to evaluate each configuration. The hyperparameters were selected at a point where the accuracy began to plateau. This plateau suggested that additional increases would yield diminishing returns in terms of model performance, prompting the selection of the hyperparameter values just before this point to achieve optimal performance without unnecessary complexity. Therefore, max depth showed a value of 8 at this point and at 14 for n estimators. Once these hyperparameters were determined, the Random Forest model was trained on the selected features, and the trained model was saved within a directory to be called upon for live testing.

3.3 Live Packet Capture

To enable real-time network monitoring, I implemented live packet capture using libpcap in C [9]. Libpcap allows for low-level network traffic capture directly from the network interface, and in this project, it was used to capture traffic between two nodes on

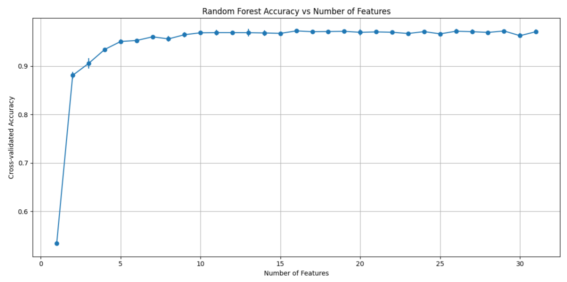


Figure 2: Visualization of recursive feature selection on trained model

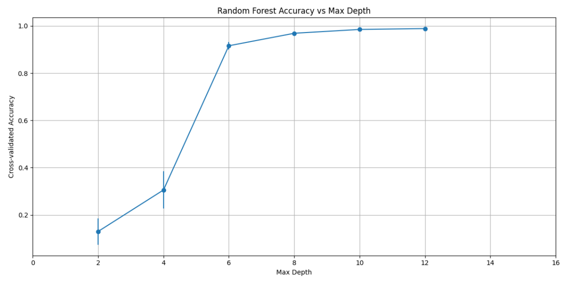


Figure 3: Visualization of max depth fine-tuning process

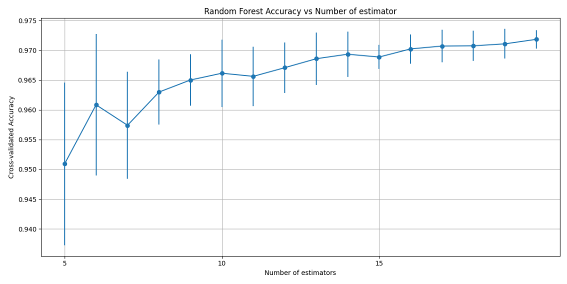


Figure 4: Visualization of n estimators fine-tuning process

my Earlham College's network. I utilized multiple nodes from a CS cluster, with one acting as a server and the others generating a variety of malicious and benign traffic.

The packet capture tool, developed using libpcap, intercepted and logged the incoming traffic on the destination node using flow-based monitoring with specific timeouts in line with the CICIDS-2017 dataset method. For TCP flows, the capture process was terminated either upon receiving a FIN packet or after 600 seconds of inactivity. For UDP flows, I set a 600-second timeout. The captured packets were aggregated into flows by utilizing 5 metrics: source IP, destination IP, source port, destination port and protocol; which were then saved in a CSV file for analysis. The flows were comprised of the features that were selected during the feature selection process in model training. These key features, along with their descriptions, are as follows:

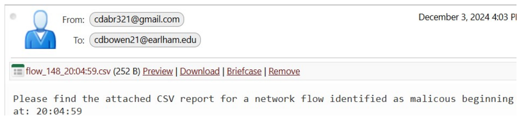
- **Bwd_Header_Length** – The total number of header bytes in the backward direction.
- **Destination_Port** – The destination port number, identifying the service or application.
- **Total_Length_of_Fwd_Packets** – The total size of packets sent in the forward direction.
- **Bwd_Packet_Length_Max** – The maximum size of any backward packet.
- **Init_Win_bytes_backward** – The initial TCP window size in the backward direction.
- **Fwd_Packet_Length_Max** – The maximum size of any forward packet.
- **Total_Fwd_Packets** – The total number of forward packets.
- **Flow_Duration** – The total duration of the flow.
- **Flow_IAT_Std** – The standard deviation of the inter-arrival time between packets.

This method of converting packets into flows and storing them in a CSV format ensured that the real-time data could be sent to the machine learning model in the appropriate input format. Once stored in the CSV file, the flow data was sent to the pre-trained model, which analyzed the data in real-time to detect potential threats.

3.4 Real-Time Response

To achieve real-time detection of network anomalies, I implemented a daemon process that operates continuously in the background. This daemon is responsible for fetching the pre-trained machine learning model and the saved flow data stored in the CSV files. By utilizing this architecture, the system can dynamically respond to live network traffic and assess its behavior against the trained model. The daemon begins by loading the saved random forest classifier model into memory, ensuring that it is ready to make predictions. Simultaneously, the daemon retrieves the latest flow data from the CSV files generated during the packet capture phase. This setup allows for immediate access to the most recent network behavior data.

During the testing phase, the daemon feeds the new flow data into the trained random forest model. This process involves pre-processing the flow data to match the format used during training, ensuring consistency in feature representation. The model then analyzes the incoming flows and outputs predictions, identifying whether the traffic is normal or indicative of an anomaly. Upon detecting an anomalous pattern, the daemon triggers an alert system that notifies network administrators of potential security threats in real-time. Figure 5.


From: cdabr321@gmail.com
To: cdbowen21@earlham.edu
December 3, 2024 4:03 P

flow_148_20:04:59.csv (252 B) Preview | Download | Briefcase | Remove

```
Please find the attached CSV report for a network flow identified as malicous beginning
at: 20:04:59
```

Figure 5: Email alert generated using Python smtplib Module

In summary, the integration of a daemon process for real-time detection enhances the effectiveness of the network monitoring system by ensuring continuous analysis of live traffic against the pre-trained model. This proactive approach enables rapid response to possible intrusions or abnormal behaviors within the network, facilitating timely interventions to mitigate risks.

4 RESULTS

4.1 Model Training

The Random Forest model was trained using the CICIDS-2017 dataset, focusing on the 10 most relevant features selected through Pearson Correlation Matrix and Recursive Feature Elimination (RFE). These features were chosen as they provided optimal classification accuracy, aligning closely with findings in prior research outlined in Table 3 of [7], which highlighted the importance of attributes such as flow duration, packet length statistics, and inter-arrival times for intrusion detection. My results reaffirm the significance of these features, demonstrating that reducing the dataset to this subset maintains high accuracy while improving computational efficiency.

4.2 Packet Capture

During the packet capture stage, a dynamic memory allocation strategy was employed to store network flows. By starting with an initial chunk size of 1 and doubling the allocation as needed, I effectively prevented memory exhaustion while maintaining performance. This approach proved superior to static allocation, especially under heavy traffic conditions, as it allowed for efficient utilization of memory resources and seamless handling of large volumes of flow data. This method ensured the stability and reliability of the packet capture process, even during prolonged network monitoring sessions.

4.3 Live Classification

During live classification, the model was tested using SSH brute force attacks, port scans, and web-SQL injection attacks. The model demonstrated a mixed ability to accurately classify these attacks. While it successfully identified the majority of brute force attacks, it struggled with port scan and web-SQL injection attacks. A notable issue was observed during port scans that targeted port 22, as these were sometimes misclassified as brute force attacks due to overlapping features between the two attack types. Furthermore, misclassifications were amplified by the model identifying multiple traffic types using the same features within the 10 that were selected, for example, Brute Force, Bot, and Infiltration traffic, leading to difficulties in accurately distinguishing between them.

These limitations highlight the challenges in accurately identifying certain attack types with individual packet-level analysis. For instance, the subtle and distributed nature of web-SQL injection traffic often lacks distinct characteristics that the model can leverage. Similarly, port scans involving short-duration packets or low rates of connection attempts can resemble benign traffic patterns.

To address these challenges, the tool must incorporate higher-level traffic behavior analysis. By aggregating traffic over time and monitoring for repeated connection attempts to various ports or anomalous patterns (such as a sustained high volume of traffic from a single IP address), the system can detect behaviors consistent with port scans or SQL injection attacks. Enhancing the tool with such behavioral analysis will improve its ability to detect a broader range of attacks and ensure timely notifications to administrators, increasing its effectiveness in real-world scenarios.

5 FUTURE WORK

Future work will focus on enhancing the tool to better identify specific attack patterns such as port scans and Denial of Service (DoS) attacks, as the current model struggled with these. Port scans, characterized by a high volume of connection attempts to various ports, and DoS attacks, which involve repetitive traffic targeting a specific service or resource, require a more robust analysis of aggregated traffic behavior. By implementing these capabilities, the packet capture system will be able to detect such behaviors in real-time and notify an administrator, making the tool more effective in handling practical network threats.

To address the observed misclassification issue, I aim to investigate whether increasing the number of features or removing the Recursive Feature Elimination (RFE) process could enhance the model's ability to distinguish between traffic types. By incorporating all non-highly correlated features, the model may gain additional discriminatory power, potentially reducing overlap and improving classification accuracy for challenging cases such as distinguishing between port scans and brute force attacks. Future experiments will evaluate the trade-offs between feature set size, model complexity, and performance.

Additionally, testing will be expanded to include live data from external networks beyond the currently monitored local network. This approach will provide a more comprehensive evaluation of the tool's performance across diverse network environments, validating its effectiveness and scalability. By leveraging insights from this expanded testing and fine-tuning, the tool will be better equipped to detect and classify malicious traffic in real-world scenarios.

REFERENCES

[1] Mustafa Al Lail, Alejandro Garcia, and Saul Olivo. 2023. Machine learning for network intrusion detectionmdash;a comparative study. *Future Internet*, 15, 7. DOI: 10.3390/fi15070243.

[2] K.G. Anagnostakis, S. Ioannidis, S. Miltchev, M. Greenwald, J.M. Smith, and J. Ioannidis. 2002. Efficient packet monitoring for network management. In *NOMS 2002. IEEE/IFIP Network Operations and Management Symposium. ' Management Solutions for the New Communications World'(Cat. No.02CH37327)*, 423–436. DOI: 10.1109/NOMS.2002.1015599.

[3] Ibrahim Ghafir, Václav Přenosil, Jakub Svoboda, and Mohammad Hammoudeh. 2016. A survey on network security monitoring systems. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 77–82. https://api.semanticscholar.org/CorpusID:15021357.

[4] Azidine Guezzaz, Ahmed Asimi, Yassine Sadqi, Younes Asimi, and Zakariae Tbatou. 2016. A new hybrid network sniffer model based on pcap language and sockets (pcapsocks). *International Journal of Advanced Computer Science and Applications*, 7, 2.

[5] V. Mohan, Y. R. Janardhan Reddy, and K. Kalpana. 2011. Active and passive network measurements: a survey. In https://api.semanticscholar.org/CorpusID:16484529.

[6] Shailesh Singh Panwar, Y. P. Raiwani, and Lokesh Singh Panwar. 2022. An intrusion detection model for cicids-2017 dataset using machine learning algorithms. In *2022 International Conference on Advances in Computing, Communication and Materials (ICACCM)*, 1–10. DOI: 10.1109/ICACCM56405.2022.10009400.

[7] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*. https://api.semanticscholar.org/CorpusID:4707749.

[8] Jakub Svoboda, Ibrahim Ghafir, Vaclav Prenosil, et al. 2015. Network monitoring approaches: an overview. *Int J Adv Comput Netw Secur*, 5, 2, 88–93.

[9] tcpdump. 2017. Tcpdump/libpcap public repository. *Tcpdump.org*. DOI: https://www.tcpdump.org/.