CyberMALT: Machine Learning-Assisted Traffic Analysis for Cyber Threat Detection and Classification

Domenico Ditale, Massimiliano Albanese, Kun Sun, Jianli Pan
George Mason University, Fairfax, VA, USA
{dditale, malbanes, ksun3, jpan22}@gmu.edu

2025 IEEE 22nd Consumer Communications & Networking Conference (CCNC) | 979-8-3315-0805-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/CCNC54725.2025.10976113

***Abstract*—Traditional methods for identifying and mitigating cyber attacks are becoming inadequate due to ever-increasing volumes of network traffic, the complexity of modern cyber threats, and the use of encryption to protect payloads. This paper presents CyberMALT, a novel approach designed to address these challenges through machine learning-assisted analysis of traffic metadata, which provides valuable insights into network behavior without examining payloads. Our proposed solution utilizes a two-stage approach. First, we employ unsupervised machine learning techniques to study typical network behavior. This initial stage allows CyberMALT to establish a baseline understanding of typical traffic characteristics, enabling it to identify deviations indicative of potential threats. Leveraging this knowledge, CyberMALT computes an anomaly score for each observed traffic instance, thereby pinpointing suspicious activity for further investigation. In the second stage of processing, these identified anomalies undergo a comprehensive analysis to classify the types of attacks accurately and efficiently and rule out false positives. By leveraging machine learning for traffic metadata analysis, CyberMALT offers a proactive and adaptive solution for cyber threat detection and classification. Our experiments demonstrate the effectiveness of CyberMALT in identifying and classifying diverse cyber threats while minimizing false positives, thus enhancing the security posture of networked systems.**

I. Introduction

With the increasing complexity of cyber threats and the ever-growing volumes of network traffic, traditional methods of cyber threat detection and mitigation are proving to be inadequate. The rise of encrypted [1] and voluminous payloads [2] has further complicated conventional inspection methods, making it challenging to effectively identify and address a wide range of cyber threats. Traditional methods also need to filter features in order to increase their ability to detect anomalies in the network [3]. To tackle these challenges, we present CyberMALT, a novel approach that leverages machine learning to analyze traffic metadata, offering a robust solution for cyber threat detection and classification.

The core idea behind CyberMALT is to focus on traffic metadata rather than payloads, allowing for the detection of anomalies in network behavior without the need to decrypt or deeply inspect payload content. This method not only preserves privacy but also enhances the efficiency of threat detection. Our approach involves a two-stage methodology. In the first stage, CyberMALT employs unsupervised machine learning techniques to learn typical patterns of network behavior. By establishing a baseline understanding of normal traffic characteristics, the system can detect deviations that may indicate potential threats. Each observed traffic instance is then assigned an anomaly score, pinpointing suspicious activity for further investigation. The second stage involves a comprehensive analysis of potential anomalies to identify false positives and accurately classify the types of attacks. This two-phase approach ensures that CyberMALT not only detects but also classifies diverse cyber threats efficiently, minimizing the number of false positives and enhancing the security posture of networked systems. Our experiments demonstrate the effectiveness of CyberMALT in identifying and classifying various cyber threats, showcasing its potential to significantly improve cyber threat detection and mitigation compared to traditional threat detection methods.

The remainder of the paper is organized as follows. Section II provides an overview of related work. Then, Section III details the proposed solution, and Section IV presents our experimental evaluation. Finally, Section V concludes the paper, summarizing our findings and discussing potential future work.

II. Related Work

Network Intrusion Detection Systems (NIDS) play a critical role in network security, monitoring traffic for potential threats.

A. *Traditional Intrusion Detection*

Traditional intrusion detection methods can be classified into signature-based and anomaly-based detection. Signature-based systems, such as Snort [4], rely on predefined rules and patterns to identify known threats. They compare network traffic against a database of attack signatures and generate alerts upon finding matches. These systems are effective at detecting known attacks due to their accuracy and low false positive rates. However, they cannot detect new or evolving threats (zero-day attacks) as they rely on known signatures.

Anomaly-based detection establishes a baseline of normal network behavior and flags deviations as potential threats. These methods may use statistical or rule-based techniques rather than machine learning [5]. They monitor network parameters such as traffic volume, connection patterns, and protocol usage to detect anomalies. While these systems can identify novel attacks, they often produce higher false positive rates, as unusual but legitimate behavior may be flagged.

Heuristic-based methods add another layer of detection by using expert-defined heuristics to identify suspicious activities [6]. Their effectiveness depends on the quality and breadth of the heuristics, making them suitable for specific scenarios.

B. ML-Based Intrusion Detection

In [7], the authors employed various supervised learning models for intrusion detection, using the CIC-IDS2017 Dataset [8] for testing and achieving excellent results overall. The models included GaussianNB, BernoulliNB, Decision Tree, KNN, Logistic Regression, SVM, SGD, and Random Forest. In [9], unsupervised learning models such as expectation-maximization (EM), k-means, and self-organizing maps (SOM) were evaluated for anomaly detection. The study showed that supervised learning techniques generally outperform unsupervised ones. Among unsupervised methods, EM performed best but only achieved 60% accuracy.

The work in [10] used an unsupervised autoencoder model to detect attacks by learning normal behavior and assigning anomaly scores. If the score exceeded a threshold, an attack was predicted. Lower thresholds improved detection but increased false positives, while higher thresholds reduced false positives at the cost of detection accuracy. Other studies highlight the effectiveness of deep learning in intrusion detection. For example, [11] combined a Convolutional Neural Network (CNN) with the Sigmoid Pigeon Optimization Algorithm for feature selection, achieving high accuracy. In [12], a transformer-based transfer learning method was used to enhance attack detection. The process included extracting detailed attack data, balancing the dataset with SMOTE, and employing a hybrid CNN-LSTM model for classification.

Although these methods are effective, they can be computationally expensive. Our research focuses on improving classification accuracy without additional deep learning models, emphasizing accurate true positive detection. By simplifying the feature selection process, we achieve high accuracy with lower computational costs, demonstrating that complex preselection, as in [11], is unnecessary for our approach.

III. Proposed Solution

The proposed framework, illustrated in Fig. 1, consists of three modules: *Preprocessing*, *Anomaly Detection*, and the *Classification*. The Preprocessing Module takes network flows as input, each containing metadata derived from capturing and aggregating packets exchanged during TCP sessions. We argue that a flow's destination address and port are crucial for threat detection, as normal traffic patterns vary by service type. For instance, *normal* traffic to an FTP server differs significantly from that to a web server, and traffic to different web servers on the same network can also vary. Our solution relies on this key observation to achieve more granular anomaly detection and reduce false positives.

A. Preprocessing

The Preprocessing Module performs several steps to ensure data integrity and usability. These steps include handling

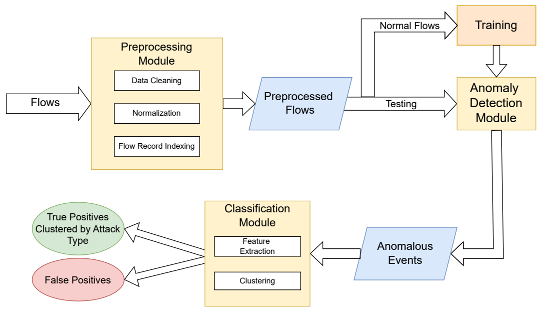

Fig. 1. Architecture of the proposed framework.

empty or otherwise unusable values to improve data quality and removing constant attributes to facilitate effective feature normalization. Data normalization, achieved through min-max normalization, prepares the dataset for model training. Flows are sorted based on timestamps – to enable sequential analysis – and grouped into small sets of consecutive flows for further processing – to reduce the sensitivity to small variations in normal traffic. Thus, the dataset can be modeled as a sequence $W = \langle w_0, w_1, \ldots, w_n \rangle$ of sets of network flow records falling within fixed-size temporal windows, as illustrated in Fig. 2.

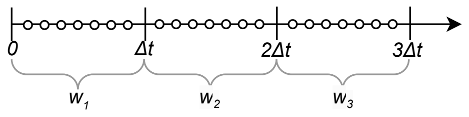

Fig. 2. Example of flow record grouping and indexing.

B. Training & Detection

We adopted an autoencoder-based approach to identify anomalies in network traffic. The preprocessed flows are split into training and testing sets, with the training set containing normal traffic (e.g., data collected on Monday in the case of the CIC-IDS2017 dataset [8] used in our evaluation).

Recognizing that different types of network services exhibit distinct traffic patterns and are uniquely mapped to specific ports, we group data flows by their destination ports during training. This allows the autoencoder to learn a tailored model for each type of service, enhancing its ability to identify network traffic anomalies accurately.

The Anomaly Detection Module evaluates flow groups against the model learned during training. The reconstruction error from the autoencoder is used as the anomaly score. However, traffic anomalies are not necessarily an indicator of malicious behavior because (i) normal behavior may exhibit a great level of variance over time and across settings, and (ii) other events, including software and hardware failures or unusual workloads might lead to anomalies in network traffic. To capture the temporal nature of anomalies, while limiting the sensitivity of the approach, anomaly scores are calculated for each fixed-size temporal window using a sliding window technique, where both the pace and the size of the window are tunable parameters. To reduce the processing time and without loss of generality, we set the window size and the pace to the same value Δt, i.e., consecutive windows do not overlap.

To leverage the relative nature of *anomaly* (i.e., a pattern's anomalousness depends on context), we use a threshold to convert anomaly score sequences (pairs of temporal windows and anomaly scores) into anomalous traffic segments. To consider the distinct traffic patterns of different network services, the threshold function, $\tau : P \rightarrow \mathbb{R}^+$, maps port numbers to distinct thresholds. Each port's threshold is set based on the statistical properties of its anomaly score distribution, commonly as $\tau(p) = \mu_p + k \cdot \sigma_p$, where k controls the sensitivity. To prioritize recall over false positives, we select lower k values.

Before feeding data to the autoencoder for testing, the Detection Module groups flows in each window $w \in W$ based on their source IP, destination IP, and destination port to study the traffic behavior on each network link with respect to the expected behavior for the specific service running on the destination port. Eq. 1 defines the set of flows in window $w \in W$ that share the same source IP $s \in S$, destination IP $d \in D$, and destination port $p \in P$.

$$w(s,d,p) = \{f \in w_i \mid f.s = s \wedge f.d = d \wedge f.p = p\} \quad (1)$$

Each set of flows $w(s,d,p)$ is fed into the autoencoder to obtain the corresponding reconstruction error. Since this error increases for data that deviates from normal patterns, we use the reconstruction error as the anomaly score. Formally, the anomaly score is defined as a function $\alpha : W \times S \times D \times P \rightarrow \mathbb{R}^+$ that maps a window w and a triple (s,d,p) to a real number $\alpha(w,s,d,p)$, calculated as the Root Mean Squared Error (RMSE) between the autoencoder's inputs and outputs, as defined by Eq. 2:

$$\alpha\left(w,s,d,p\right) = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} \left(y_i - x_i\right)^2} \quad (2)$$

where $m = |w(s,d,p)|$ is the number of flow records in $w(s,d,p)$, x_i is the feature vector of the i-th flow record, and y_i is the corresponding reconstructed output. If $w(s,d,p) = \emptyset$, we set $\alpha\left(w,s,d,p\right) = 0$.

If the anomaly score for a set $w(s,d,p)$ exceeds the threshold $\tau(p)$, it is marked as anomalous; otherwise, it is marked as normal. A lower threshold improves recall but increases the risk of false positives, while a higher threshold reduces false positives but may result in missed detections.

For fixed values of s, d, and p, α defines a mapping from windows $w \in W$ to corresponding values of the anomaly score, as illustrated by the conceptual example of Fig. 3, where anomalous data points (i.e., windows with an anomaly score above the threshold) are represented by red dots and normal data points are represented by green dots.

Before feeding this data to the Classification Module, we perform an additional level of data aggregation based on the observation that anomalies corresponding to cyber attacks may extend beyond a single temporal window if the attack

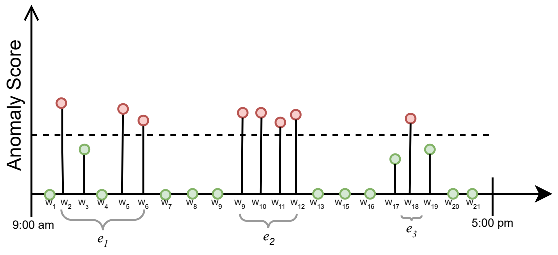

Fig. 3. Using a threshold to identify anomalous windows and group windows into events, with $g = 3$.

duration is larger than Δt. To do so, we use algorithm $getEvents(W, \alpha, \tau, s, d, p, g)$ (Algorithm 1), which identifies anomalous events as sequences of one or more consecutive windows in W, with consecutive events separated by at least g windows with an anomaly score below the port-specific threshold $\tau(p)$. Intuitively, consecutive anomalous data points are likely part of the same malicious event, even when interleaved with a few data points with below-threshold scores. A malicious event is considered to have ended after at least g windows with scores below the threshold.

The algorithm takes as input the set of windows W, the anomaly score function α, the threshold mapping τ, a source IP s, a destination IP d, a destination port p, and the minimum gap g between distinct anomalous events, and returns a set E of anomalous event. The set of events E and the variable $current\_event$, which is used to track an individual event instance, are initialized as empty sets and the $count$ of consecutive data points with scores below the threshold is set to 0 (Lines 3-5). The algorithm iterates over each window $w \in W$ (Lines 6-23). If the anomaly score $\alpha(w,s,d,p)$ is above the threshold $\tau(p)$, the window w is added to the $current\_event$ and the $count$ of consecutive data points below the threshold is reset (Lines 8-9). Otherwise, if the score is below the threshold and $current\_event$ is not empty (i.e., the algorithm is tracking an anomalous event), the algorithm evaluates whether $count$ has already reached the maximum gap g. If not, window w is added to $current\_event$ and $count$ is incremented (Lines 12-13). Otherwise, if $count$ has reached g, $current\_event$ is trimmed to remove the tailing data points with anomaly scores below the threshold, and $current\_event$ and $count$ are reset (Lines 16-19). Finally, if the algorithm is tracking an event when the last window is processed, $current\_event$ is trimmed and added to the set of events.

C. Classification

In the final stage of the proposed framework's processing pipeline, the Classification Module processes the set E of events identified by the Anomaly Detection Module. Its primary objective is to distinguish between false positives and true positives and classify the true positives based on the nature of the threat. To achieve this, the Classification Module performs a detailed analysis of the events' characteristics to identify similarities and differences, grouping the events into clusters, with each cluster representing similar types of threats.

Algorithm 1 $getEvents(W, \alpha, \tau, s, d, p, g)$

```
1: Input: the set of windows $W$, the anomaly score function $\alpha$, the anomaly score threshold mapping $\tau$, the source IP $s$, the destination IP $d$, the destination port $p$, and the minimum gap between events $g$.
2: Output: the set of anomalous events $E$.
3: $E \leftarrow \emptyset$                                  ▷ Initialize the set of events
4: $current\_event \leftarrow \emptyset$                      ▷ Initialize the current event
5: $count \leftarrow 0$    ▷ Count of consecutive points with below-threshold scores
6: for $w \in W$ do
7:     if $\alpha(w, s, d, p) \geq \tau(p)$ then
8:         $current\_event \leftarrow current\_event \cup \{w\}$
9:         $count \leftarrow 0$                                ▷ Reset the count
10:    else
11:        if $current\_event \neq \emptyset \wedge count < g$ then
12:            $current\_event \leftarrow current\_event \cup \{w\}$
13:            $count \leftarrow count + 1$
14:        else
15:            if $current\_event \neq \emptyset$ then
16:                $current\_event \leftarrow trim(current\_event)$
17:                $E \leftarrow E \cup \{current\_event\}$
18:                $current\_event \leftarrow \emptyset$
19:                $count \leftarrow 0$
20:            end if
21:        end if
22:    end if
23: end for
24: if $current\_event \neq \emptyset$ then
25:     $E \leftarrow E \cup trim(current\_event)$
26: end if
27: return $E$
```

The process starts with feature extraction from the identified events, creating compact representations that capture key information for clustering. A clustering algorithm is then applied to group events based on underlying patterns, effectively isolating false positives and accurately classifying true positives. The features extracted for each anomalous event $e \in E$ include the following. The **Anomaly Score Count** (a_count) represents the number of windows $w \in e$ with anomaly scores exceeding the threshold. The **Largest Anomaly Score** (a_max) is the highest anomaly score among windows $w \in e$. The **Mean Score** (a_mean) is the average anomaly score of windows $w \in e$ with scores above the threshold. The **Median Score** (a_median) represents the median anomaly score of windows $w \in e$ that exceed the threshold. Lastly, the **Standard Deviation** (a_stdev) is the standard deviation of anomaly scores for windows $w \in e$ with scores above the threshold.

We use the conceptual example in Fig. 3 to demonstrate how to calculate these features for the identified anomalous events. In this example, there are three events $E = \{e_1, e_2, e_3\}$, with $e_1 = \langle w_2, w_3, w_4, w_5, w_6 \rangle$, $e_2 = \langle w_9, w_{10}, w_{11}, w_{12} \rangle$, and $e_3 = \langle w_{18} \rangle$. Table I shows how the value of all the features are computed for these 3 events.

We use algorithm $clusterEvent(E, K, D)$ (Algorithm 2) to compute the event features and perform classification, utilizing the K-means elbow method to determine the optimal number k of clusters. The algorithm takes as input the set of anomalous events E, a set K of candidate values for the optimal number of clusters k, and a family D of distance metrics, returning a mapping $optimal\_clusters$ that associates each metric with an optimal set of clusters based on that metric.

For each event $e \in E$, a feature vector is computed (Lines 7-13), followed by feature normalization (Line 15). The algorithm iterates over the family D of distance metrics and the set K of candidate values of k. We applied K-means clustering using two distance metrics – Euclidean and Manhattan – to validate the clustering of anomalous events. For each metric and value of k, K-means determines a set of k clusters (Line 18) and calculates the corresponding inertia (Line 19), which sums the squared distances between each data point and its centroid. Using the elbow method, the optimal value of k is identified as the point where the rate of decrease for the inertia slows sharply, thus forming an *elbow* in the curve (Line 21). Finally, the set of optimal clusters for each metric is added to $optimal\_clusters$ (Lines 22-23).

Algorithm 2 $clusterEvent(E, K, D)$

```
1: Input: the Set of anomalous events $E$, the set of candidate values for the optimal $k$, $K$, and the family of distance metrics $D$ to use for clustering.
2: Output: the mapping, $optimal\_clusters$, of metrics to sets of clusters.
3: $optimal\_clusters \leftarrow \emptyset$            ▷ Optimal Clusters for each distance
4: $features \leftarrow \emptyset$                     ▷ Features extracted from each event
5: $inertias \leftarrow \emptyset$      ▷ Set of var. {Calculate_Inertia(Clusters,d),k)}
6: for $e \in E$ do
7:     $a\_count \leftarrow |\{w \in e \mid \alpha(w) > \tau(p)\}|/\tau(p)$
8:     $a\_max \leftarrow \max_{w \in e} \alpha(w)$
9:     $a\_mean \leftarrow avg_{w \in e \mid \alpha(w) > \tau(p)} \alpha(w)/\tau(p)$
10:    $a\_median \leftarrow median_{w \in e \mid \alpha(w) > \tau(p)} \alpha(w)/\tau(p)$
11:    $a\_stdev \leftarrow stdev_{w \in e \mid \alpha(w) > \tau(p)} \alpha(w)/\tau(p)$
12:    $vector \leftarrow \{a\_count, a\_max, a\_mean, a\_median, a\_stdev\}$
13:    $features \leftarrow features \cup vector$
14: end for
15: $features \leftarrow normalize(features)$
16: for $d \in D$ do                                   ▷ Iterate over metrics
17:     for $k \in K$ do                 ▷ Iterate over candidate $k$ values
18:         $clusters \leftarrow k\text{-}means(features, k, d)$
19:         $inertias \leftarrow inertias \cup \{(k, computeInertia(clusters))\}$
20:     end for
21:     $optimal\_k \leftarrow findElbow(inertias)$
22:     $clusters \leftarrow k\text{-}means(features, optimal\_k, d))$
23:     $optimal\_clusters \leftarrow optimal\_clusters \cup \{(d, clusters)\}$
24: end for
25: return $optimal\_clusters$
```

Once the optimal value of k is identified for each of the distance metrics, we expect to obtain two types of clusters after clustering the set E of anomalous events: clusters of the first type are expected to include false positive events, and clusters of the second type are expected to include attack instances with similar features. Since datasets of traffic flows typically contain much more normal traffic than attack traffic, setting the threshold τ to prioritize recall over precision can lead to a high rate of false positives. Consequently, clusters of the first type may be larger and more numerous, while clusters of the second type may be fewer and smaller.

IV. Experimental Evaluation

In this section, we present the results of our evaluation. These results demonstrate that the proposed approach effectively decomposes the complex task of traffic metadata analysis into two manageable sub-problems, ensuring accurate identification of malicious traffic while minimizing false positives. To evaluate the approach, we used the Canadian Institute for Cybersecurity Intrusion Detection Systems 2017 (CIC-IDS2017) dataset, a comprehensive collection of network traffic data designed for cybersecurity research, particularly focusing on intrusion detection. The dataset, created by the Canadian Institute for Cybersecurity at the University of

TABLE I
EVENT FEATURES

Feature	**Event e_1**	**Event e_2**	**Event e_3**
Anomaly Score Count	3	4	1
Largest Anomaly Score	$\alpha(w_2)$	$\alpha(w_{10})$	$\alpha(w_{18})$
Mean Score	$\frac{\alpha(w_2)+\alpha(w_5)+\alpha(w_6)}{3}$	$\frac{\alpha(w_{10})+\alpha(w_{11})+\alpha(w_{12})+\alpha(w_{13})}{4}$	$\alpha(w_{18})$
Median Score	Median$(\alpha(w_2), \alpha(w_5), \alpha(w_6))$	Median$(\alpha(w_{10}), \alpha(w_{11}), \alpha(w_{12}), \alpha(w_{13}))$	$\alpha(w_{18})$
Standard Deviation	Stdev$(\alpha(w_2), \alpha(w_5), \alpha(w_6))$	Stdev$(\alpha(w_{10}), \alpha(w_{11}), \alpha(w_{12}), \alpha(w_{13}))$	0

Algorithm 3 $analyzeData(W, \alpha, g, K, D)$

```
1: Input: the set of windows $W$, the anomaly score function $\alpha$, the minimum gap between events $g$, the set of candidate values for the optimal value of $k$, $K$, and the family of distance metrics $D$ to use for clustering.
2: Output: a mapping of metrics to sets of clusters.
3: $E \leftarrow \emptyset$
4: for $(s, d, p)$ s.t. $\exists f(f.s = s \wedge f.d = d \wedge f.p = p)$ do
5:     $E \leftarrow E \cup getEvents(W, \alpha, s, \tau, d, p, g)$
6: end for
7: return $clusterEvents(E, K, D)$
```

New Brunswick, contains traffic data captured in a controlled environment to simulate various types of cyber-attacks and normal activities.

CIC-IDS2017 includes both benign and malicious traffic generated from modern attack tools, saved as PCAP files. The CICFlowMeter tool [13] generates bidirectional network flows and extracts over 80 statistical features, including duration, packet counts, byte counts, and packet lengths. These features are calculated separately for forward and backward traffic. Data in the CICIDS2017 dataset was captured from July 3–7, 2017, with simulated attacks – Brute Force FTP/SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS – occurring from Tuesday to Friday [8], [14].

We processed the CIC-IDS2017 network flows as described in Section III-A, indexing them into 20-second windows ($\Delta t = 20s$). Normal traffic captured on Monday was used to train the autoencoder to model benign traffic on ports 80, 21, and 22. The observed attacks include FTP-Patator, SSH-Patator, DoS Slowloris, DoS SlowHTTPTest, DoS Hulk, DoS GoldenEye, web brute force attack, XSS attack, SQL injection, DDoS, and port scan. For each window from Tuesday to Friday, flows were grouped by their source IP, destination IP, and port, forming sets $w(s, d, p)$. For each triple (s, d, p), anomaly scores α were computed for all windows $w \in W$ using Eq. 2. Algorithm $getEvents$ was then applied to identify anomalous events, using $g = 30$ in our evaluation. After identifying anomalous events for all triples (s, d, p), we applied algorithm $clusterEvent$ to cluster these events. The complete process is outlined in Algorithm $analyzeData$ (Algorithm 3).

Figs. 4 and 5 show the computed anomaly scores for all triples (s, d, p) on ports 21 and 80, respectively. All anomaly scores have been normalized by dividing the result of Eq. 2 by the threshold $\tau(p)$, enabling comparisons of anomalies across different ports. A window w is considered anomalous if its normalized score exceeds 1.

Setting a relatively low threshold maximizes recall at the cost of a higher false positive rate, which is mitigated during the clustering stage. The only undetected attack is the web attack, which exhibited a low anomaly score, likely due to its host-based nature rather than being network-based.

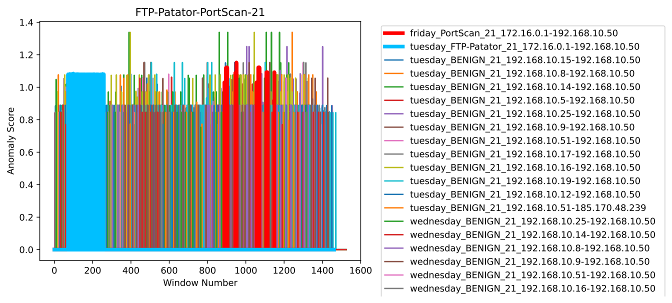

Fig. 4. Results for traffic to Port 21

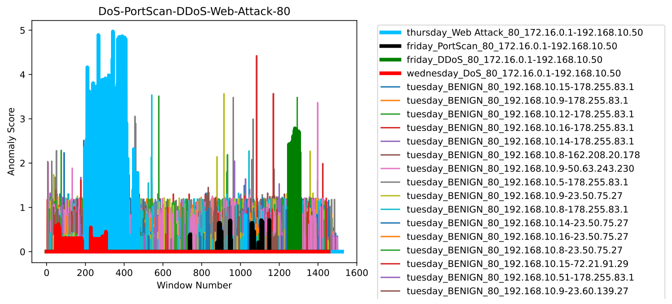

Fig. 5. Results for traffic to Port 80

In the classification phase, we computed features for each anomalous event $e \in E$, as described in Section III-C. Intuitively, analyzing multiple features of each anomalous event aids in filtering out false positives. While many data points in Figs. 4 and 5 exceed the threshold, only a few exhibit characteristics of potential attacks, such as temporal persistence and consistently exceeding the threshold. We applied K-means clustering using two distance metrics: Euclidean and Manhattan. The optimal number of clusters k for each metric was determined using the elbow method.

We clustered the set E of anomalous events using $k = 15$ and assigned a numerical label to each cluster. Tables II and III show the number of events assigned to each cluster. Each cluster predominantly contains either benign or attack events. If the majority of events in a cluster are benign, the flows in that cluster are classified as benign; otherwise, they are classified as part of attacks. For each cluster, we calculated the purity, which measures the proportion of the majority class (benign or attack) within the cluster.

These results confirm the promise of the proposed approach.

TABLE II
CLUSTERING RESULTS USING THE EUCLIDEAN DISTANCE.

Euclidean			
Label	**# Events**	**Description**	**Purity**
4	510	FPs	100%
13	312	FPs & 3 PortScans	96%
0	264	FPs & 3 PortScans	98%
11	99	FPs & 4 PortScans	96%
2	57	FPs	100%
8	39	FPs	100%
7	25	FPs	100%
5	15	FPs	100%
9	9	FPs	100%
3	8	FPs	100%
14	7	FPs	100%
1	5	FPs	100%
10	4	DDoS, FTP-Patator, SSH-Patator, FP	75%
6	2	DoS Event & FP	50%
12	2	FPs	100%

TABLE III
CLUSTERING RESULTS USING THE MANHATTAN DISTANCE.

Manhattan			
Label	**# Events**	**Description**	**Purity**
0	510	FPs	100%
3	252	FPs & 2 PortScans	99%
8	247	FPs & 2 PortScans	99%
5	105	FPs & 4 PortScans	96%
10	86	FPs & 2 PortScans	97%
14	56	FPs	100%
11	49	FPs	100%
7	22	FPs	100%
13	9	FPs	100%
9	9	FPs	100%
1	5	FPs	100%
6	4	DDoS & FPs	75%
2	2	FTP-Patator & SSH-Patator	100%
4	1	DoS	100%
12	1	FP	100%

Although the anomaly detection stage generates numerous false positives, the high purity levels indicate that the clustering process effectively partitions false and true positives. Notably, false positives are distributed across multiple clusters. While it would be ideal to group all false positives into a single cluster, this outcome is acceptable given the feature spaces used for clustering. Small clusters are predominantly composed of attack events.

We successfully separated DoS and DDoS attacks into distinct clusters and grouped the two Patator attacks together, as they target different protocols but represent the same type of attack. Additionally, clustering with Manhattan Distance provided better segregation of attack events into distinct clusters, demonstrating the approach's potential for accurately identifying and grouping similar attack patterns.

V. CONCLUSIONS

In this paper, we introduced CyberMALT, a machine learning-assisted framework for enhancing cyber threat detection and classification through network traffic metadata analysis. Our approach overcomes the limitations of traditional methods, which often struggle with the increasing volume and complexity of modern network traffic. CyberMALT employs a two-stage process: an unsupervised learning phase to establish baseline network behavior and detect anomalies, followed by a classification phase to eliminate false positives and identify specific attack types. Experiments using the CIC-IDS2017 dataset showed that CyberMALT effectively detects and classifies diverse cyber threats, such as FTP-Patator, SSH-Patator, DoS, DDoS, and port scan attacks, with high recall and acceptable precision. The clustering process further improves detection by distinguishing true positives from false positives, minimizing the misclassification of benign events. Further details on this work are available in the companion GitHub repository: https://github.com/maxalbanese/CyberMALT.

Future work will enhance the framework's adaptability to evolving threats by incorporating advanced machine learning techniques and expanding the scope of metadata analysis. We plan to extend CyberMALT to cover more attack types, including those exploiting novel vulnerabilities and sophisticated evasion tactics. Additionally, we aim to explore real-time implementation and integration with existing network security infrastructure for comprehensive, proactive defense.

REFERENCES

[1] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security (ACM ASIACCS 2017)*, pp. 561–574.

[2] S. Zwane, P. Tarwireyi, and M. Adigun, "Ensemble learning approach for flow-based intrusion detection system," in *Proc. of IEEE AFRICON 2019*.

[3] L. Chen, X. Kuang, A. Xu, S. Suo, and Y. Yang, "A novel network intrusion detection system based on CNN," in *Proc. of the 8th Intl. Conf. on Advanced Cloud and Big Data (CBD 2020)*, pp. 243–247.

[4] Z. Zhou, Z. Chen, T. Zhou, and X. Guan, "The study on network intrusion detection system of Snort," in *Proc. of the 2010 Intl. Conf. on Networking and Digital Society*, 2010, pp. 194–196.

[5] S. Rastegari, C.-P. Lam, and P. Hingston, "A statistical rule learning approach to network intrusion detection," in *Proc. of the 5th Intl. Conf. on IT Convergence and Security (ICITCS 2015)*.

[6] W. Kehe, D. Tao, and H. Jianping, "The research of intrusion detection technology based on heuristic analysis," in *Proc. of the 6th IEEE Joint Intl. Information Technology and Artificial Intelligence Conf. (ITAIC 2011)*, pp. 164–166.

[7] S. S. Panwar, Y. P. Raiwani, and L. S. Panwar, "An intrusion detection model for CICIDS-2017 dataset using machine learning algorithms," in *2022 Intl. Conf. on Advances in Computing, Communication and Materials (ICACCM 2022)*.

[8] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. of the 4th Intl. Conf. on Information Systems Security and Privacy (ICISSP 2018)*, pp. 108–116.

[9] Z. K. Maseer, R. Yusof, N. Bahaman, S. A. Mostafa, and C. F. M. Foozy, "Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset," *IEEE Access*, vol. 9, pp. 22 351–22 370, 2021.

[10] K. Roshan and A. Zafar, "An optimized auto-encoder based approach for detecting zero-day cyber-attacks in computer network," in *Proc. of the 5th Intl. Conf. on Inf. Systems and Computer Networks (ISCON 2021)*.

[11] L. Zhang and C. Xu, "An intrusion detection model based on convolutional neural network and feature selection," in *Proc. of the 5th Intl. Conf. on Artificial Intell. and Big Data (ICAIBD 2022)*, pp. 162–167.

[12] F. Ullah, S. Ullah, G. Srivastava, and J. C.-W. Lin, "IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic," *Digital Communications and Networks*, vol. 10, no. 1, pp. 190–204, 2024.

[13] "CICFlowMeter," https://www.unb.ca/cic/research/applications.html, retrieved: 2024-11-11.

[14] "Intrusion detection evaluation dataset (CIC-IDS2017)," https://www.unb.ca/cic/datasets/ids-2017.html, retrieved: 2024-11-11.